A Specification and Analysis Framework for Provenance Awareness of Service Compositions

Zerva, Paraskevi

Awarding institution: King’s College London

The copyright of this thesis rests with the author and no quotation from it or information derived from it may be published without proper acknowledgement.

END USER LICENCE AGREEMENT

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International licence. https://creativecommons.org/licenses/by-nc-nd/4.0/

You are free to:
  • Share: to copy, distribute and transmit the work

Under the following conditions:
  • Attribution: You must attribute the work in the manner specified by the author (but not in any way that suggests that they endorse you or your use of the work).
  • Non Commercial: You may not use this work for commercial purposes.
  • No Derivative Works - You may not alter, transform, or build upon this work.

Any of these conditions can be waived if you receive permission from the author. Your fair dealings and other rights are in no way affected by the above.

Take down policy

If you believe that this document breaches copyright please contact librarypure@kcl.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
A Specification and Analysis Framework for Provenance Awareness of Service Compositions

Author: Paraskevi Zerva

Supervisors: Dr. Steffen Zschaler
Dr. Simon Miles
# Contents

1 Introduction .................................................. 22  
1.1 Motivation .................................................. 23  
1.2 Research Gap ............................................... 28  
1.3 Problem Definition ....................................... 30  
1.4 Motivating Use Case Example ............................ 35  
1.5 Research Questions ....................................... 37  
1.6 Contributions .............................................. 38  
1.7 Expected Benefits ......................................... 41  
1.8 Organization of the Thesis ............................... 42  
1.9 Declarations ................................................ 43  

2 Background .................................................... 46  
2.1 Fundamentals of Services and Compositions ............. 47  
2.1.1 Service-Oriented Architecture and Services ........... 47  
2.1.2 Service Properties and Principles .................... 51  
2.1.3 Service-oriented Computing Paradigm ................. 56  
2.2 Functional and Non-Functional Requirements ............ 59  
2.2.1 Functional Requirements ............................... 60  
2.2.2 Non-Functional Requirements ......................... 61  
2.2.3 Non-functional Properties ............................. 64  
2.2.4 Specification Languages for NFPs ..................... 69  
2.3 An introduction to Provenance ........................... 76  
2.3.1 Provenance Definitions ............................... 77  
2.3.2 Provenance Drivers and Usability ..................... 79  
2.3.3 Provenance Awareness and Accountability .......... 81
2.3.4 Provenance Awareness as an NFP 83
2.4 Provenance Data Models 84
2.5 Provenance Querying Languages 89

3 Related Work 92
3.1 Provenance for SOA and Services 92
3.2 Provenance and Trust for Cloud Services 94

4 Problem Statement 96
4.1 Use Case Scenario 97
4.1.1 Travel Planner Composition 97
4.1.2 Example Provenance Questions 100
4.2 Research Questions and Objectives 103

5 Approach and Architecture 112
5.1 Goals and Challenges 113
5.2 Approach 114
5.3 Research Methodology 121
5.4 Proposed Architecture 123
5.5 Research Contributions 131

I Formal Models for the Specification and Analysis
of Provenance Awareness 134

6 Provenance Facets 135
6.1 Provenance Questions 136
6.1.1 Travel Planner Use Case Scenario 136
6.1.2 Questions for the Travel Planner Scenario 138
6.2 Faceted Classification of Provenance Questions 144
6.2.1 Provenance Facets for Provenance Awareness 145
6.2.2 Multi-Facet Provenance Questions 150
6.2.3 Genericity of Facets and Impact on other NFPs 151
7 A Provenance Data Model - ServiceProv

7.1 Context and Motivation ........................................ 155
7.2 A Provenance Data Model for SOC Systems .................. 158
  7.2.1 PROV-O Background ........................................ 159
  7.2.2 ServiceProv Ontology by Example ........................ 160
7.3 The ServiceProv Ontology ..................................... 168
7.4 Extensibility vs Completeness ................................. 176

8 Provenance Question Type Patterns ............................ 178

8.1 Context and Motivation ...................................... 179
8.2 Provenance Graphs and Templates ............................ 181
8.3 Provenance Question Type Patterns .......................... 185
  8.3.1 Service and Provider Identity Patterns (Facet 1) ........ 190
  8.3.2 Data Flow Patterns (Facet 2) .............................. 208
  8.3.3 Resource/Physical Deployment Patterns (Facet 3) ....... 212
  8.3.4 Time Patterns (Facet 4) .................................... 216
  8.3.5 Routes (not) followed Patterns (Facet 5) ................. 220
  8.3.6 Past History Patterns (Facet 6) ........................... 222
  8.3.7 NFPs and QoS Patterns (Facet 7) .......................... 223
  8.3.8 Design Information Patterns (Facet 9) .................... 232

9 A Behavioural Model for Provenance Infrastructure ............ 238

9.1 Context and Motivation ...................................... 239
9.2 Background ...................................................... 241
  9.2.1 Themes/UML ................................................ 242
  9.2.2 UML Profiles and Stereotypes ............................ 244
  9.2.3 Specifying Services and Service Compositions ............ 253
9.3 Provenance Infrastructure Design ............................ 258
9.4 Provenance throughout the SOC Lifecycle .................... 261
9.5 Modelling Provenance Infrastructure ........................ 264
  9.5.1 Provenance Infrastructure Template Model ................ 264
  9.5.2 Template Binding .......................................... 271
III Conclusions

12 Conclusions and Future Research

12.1 Thesis Summary

12.2 Assessment of Research Questions Against Research Contributions
IV Appendixes

13 Appendix A
13.1 SOA and SOC Glossary

14 Appendix B
14.1 ServiceProv Ontology Documentation

15 Appendix C
15.1 Queries - Travel Planner Case Study
15.2 Queries - Industrial Case Study

16 Appendix D
16.1 Query Patterns - Prototype Tool Implementation

17 Appendix E
17.1 Travel Planner WSDL Specification
17.2 Travel Planner BPEL Specification
17.3 ORBI WSDL Specification
17.4 ORBI BPEL Specification

Bibliography
## List of Figures

1.1 Motivating Example: Travel Planner Scenario Workflow . . . . . . 35
1.2 Provenance Awareness Specification and Analysis Framework . . 38

2.1 SOC pyramid - Extended Service-Oriented Architecture [119] . . 57
2.2 Service Layer QoS Taxonomy [132] . . . . . . . . . . . . . . . . 66

4.1 Travel Planner Scenario: Operations and Messages . . . . . . . . 98
4.2 Service Aggregator Role . . . . . . . . . . . . . . . . . . . . . . 99
4.3 Service Broker Role . . . . . . . . . . . . . . . . . . . . . . . . 100
4.4 Provenance Data Graph . . . . . . . . . . . . . . . . . . . . . . 101

5.1 Provenance Awareness Framework: Approach and Architecture . 114
5.2 Provenance Awareness Framework: Architecture and Roles . . . 124
5.3 Provenance Awareness Design and Analysis Workflow . . . . . . 129

7.1 Provenance of Atomic Service Execution . . . . . . . . . . . . . . 161
7.2 Provenance of Orchestrated Services . . . . . . . . . . . . . . . . 162
7.3 Provenance of Choreographed Services . . . . . . . . . . . . . . . 163
7.4 Provenance of Service Discovery and Selection . . . . . . . . . . 164
7.5 Provenance of Service Description Publishing and Registries . . 165
7.6 QoS Provenance . . . . . . . . . . . . . . . . . . . . . . . . . . . 167
7.7 Time Provenance . . . . . . . . . . . . . . . . . . . . . . . . . . 168
7.8 Resource Provenance . . . . . . . . . . . . . . . . . . . . . . . . 168

8.1 Provenance Templated Model for the Travel Planner Use Case
Scenario . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 184
8.2 Structure of Provenance Templates and Repetition Patterns . . . 188
8.3 Service Provider Identity Provenance Template . . . . . . . . . . . 192
List of Tables

2.1 Contract Specification Languages for Services [142] ................ 70
2.2 Comparison of Specification Languages for Services and Service Compositions ........................................ 75

6.1 SOC and SOA Evaluation Criteria Checklist (I) ............... 139
6.2 SOC and SOA Evaluation Criteria Checklist (II) ............... 140
6.3 Facets Ranking according to Provenance Recording Impact (I) . 153
6.4 Facets Ranking according to Provenance Recording Impact (II) . 154

7.1 Service Provenance Model Activity Type Concepts (I) ........ 171
7.2 Service Provenance Model Activity Type Concepts (II) ........ 172
7.3 Service Provenance Model Agent Type Concepts ............... 172
8.1 Service Provider Identity Provenance Query Pattern . . . . . . . . 193
8.2 Service Execution Identity Provenance Query Pattern . . . . . . . 196
8.3 Service Execution Server Identity Provenance Query Pattern . . 198
8.4 Registry Identity Provenance Query Pattern . . . . . . . . . . . . 200
8.5 Service Broker Provenance Query Pattern . . . . . . . . . . . . . 202
8.6 Aggregator Identity Provenance Query Pattern . . . . . . . . . . . 205
8.7 Selected Service Identity Provenance Query Pattern . . . . . . . . 207
8.8 Input Message Parameters Provenance Query Pattern . . . . . . . 210
8.9 Output Message Parameters Provenance Query Pattern . . . . . . 213
8.10 Resource Availability Provenance Query Pattern . . . . . . . . . . 215
8.11 CPU Resource Availability Provenance Query Pattern . . . . . . . 216
8.12 Memory Resource Availability Provenance Query Pattern . . . . 217
8.13 Operating System Resource Availability Provenance Query Pattern 218
8.14 Service Execution Timestamps Provenance Query Pattern . . . . 220
8.15 Service Invocation Timestamps Provenance Query Pattern . . . . 220
8.16 Service Availability Provenance Query Pattern . . . . . . . . . . . 226
8.17 Service Reliability Provenance Query Pattern . . . . . . . . . . . 227
8.18 Service Security Provenance Query Pattern . . . . . . . . . . . . . 228
8.19 Service Reputation Provenance Query Pattern . . . . . . . . . . . 229
8.20 Service Cost Provenance Query Pattern . . . . . . . . . . . . . . 230
8.21 Service Penalty Provenance Query Pattern . . . . . . . . . . . . . 231
8.22 Service Execution Time Provenance Query Pattern . . . . . . . . 232
8.23 Service Response Time Provenance Query Pattern . . . . . . . . 233
8.24 QoS Requirements Provenance Query Pattern . . . . . . . . . . . 234
8.25 QoS (Descriptions) Offered Provenance Query Pattern

9.1 Template Stereotype Notation
9.2 Bound Element Notation
9.3 Binding Information Notation
9.4 Template Parameter Notation
9.5 Template Binding Specification
9.6 Provenance Awareness Generation Analysis Algorithm

11.1 Querying Data Flow’s Provenance
11.2 Querying Service Provider’s Identity Provenance
11.3 Querying Data Flow’s Provenance
11.4 Querying NFP’s Provenance (Availability)
11.5 Querying Service & Provider Identity Provenance
11.6 Querying Data Flow’s Provenance
11.7 Querying NFPs and QoS Provenance
15.1 Querying Service Provider’s Identity
15.2 Querying Service Execution Identity
15.3 Querying Service Execution Server Identity
15.4 Querying Registry Identity
15.5 Querying Broker Identity
15.6 Querying Aggregator Identity
15.7 Querying Selected Service Identity
15.8 Querying Input Message Parameters
15.9 Querying Output Message Parameters
15.10 Querying Resource Availability
15.11 Querying Resource CPU Availability
15.12 Querying Time Provenance
15.13 Querying Service Availability
15.14 Querying Service Reliability
15.15 Querying Service Security
15.16 Querying Service Reputation
15.17 Querying Service Cost
15.18 Querying Service Penalty
15.19 Querying Service Execution Time
15.20 Querying Service Response Time
15.21 Querying QoS (Descriptions) Offered
15.22 Querying Service Description
15.23 Querying QoS Requirements
15.24 Querying Service Provider’s Identity - Request Contact Service
15.25 Querying Server Executor’s Identity - Create Appointment Service
15.26 Querying Server Executor’s Identity - Calendar Service
15.27 Querying Server Executor’s Identity - RSS Service
15.28 Querying Server Executor’s Identity - Weather Service
15.29 Querying Server Executor’s Identity - Update Appointment Service
15.30 Querying Server Executor’s Identity - Item Service
15.31 Querying Server Executor’s Identity - TODO Service
15.32 Querying Server Executor’s Identity - Collections Service
15.33 Querying Server Executor’s Identity - JOURNAL Service
15.34 Querying Server Executor’s Identity - TENANT COMPANIES Service
15.35 Querying Server Executor’s Identity - Create Purchase Service
15.36 Querying Server Executor’s Identity - Create Sales Service
15.37 Querying Server Executor’s Identity - Create Projects Service
15.38 Querying Server Executor’s Identity - Create Payments Service
15.39 Querying Server Executor’s Identity - Create Order Service
15.40 Querying Server Executor’s Identity - Make Payment Service
15.41 Querying Service Provider’s Identity - Make Payment Service
15.42 Querying Service Identity - Create Order Service
15.43 Querying Input Data Flow - Request Contact Service
15.44 Querying Output Data Flow - Request Contact Service
15.45 Querying Input Data Flow - Arrange Appointment Service
15.46 Querying Input Data Flow - Calendar Request Service
15.47 Querying Data Flow - RSS Service
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.79</td>
<td>Querying Timestamps - Create Payments Service</td>
<td>420</td>
</tr>
<tr>
<td>15.80</td>
<td>Querying Timestamps - Create Order Service</td>
<td>420</td>
</tr>
<tr>
<td>15.81</td>
<td>Querying Timestamps - Make Payment Service</td>
<td>420</td>
</tr>
<tr>
<td>15.82</td>
<td>Querying Past History Data - Availability</td>
<td>421</td>
</tr>
<tr>
<td>15.83</td>
<td>Querying Past History Data - Reliability/Failure Rate</td>
<td>421</td>
</tr>
<tr>
<td>15.84</td>
<td>Querying NFPs - Execution Time</td>
<td>421</td>
</tr>
<tr>
<td>15.85</td>
<td>Querying NFPs - Response Time</td>
<td>422</td>
</tr>
<tr>
<td>15.86</td>
<td>Querying NFPs - Latency</td>
<td>422</td>
</tr>
<tr>
<td>15.87</td>
<td>Querying Broker’s Identity - Make Payment Service</td>
<td>423</td>
</tr>
<tr>
<td>15.88</td>
<td>Querying Orchestrator’s Identity - Payment Service</td>
<td>423</td>
</tr>
<tr>
<td>15.89</td>
<td>Querying Service Provider’s Identity - Payment Service</td>
<td>423</td>
</tr>
<tr>
<td>16.1</td>
<td>Querying Service Execution Identity</td>
<td>424</td>
</tr>
<tr>
<td>16.2</td>
<td>Querying Service Provider Identity</td>
<td>425</td>
</tr>
<tr>
<td>16.3</td>
<td>Querying Service Registry Identity</td>
<td>426</td>
</tr>
<tr>
<td>16.4</td>
<td>Querying Service Broker Identity</td>
<td>427</td>
</tr>
<tr>
<td>16.5</td>
<td>Querying Service Aggregator Identity</td>
<td>428</td>
</tr>
<tr>
<td>16.6</td>
<td>Querying Service Selected Identity</td>
<td>429</td>
</tr>
<tr>
<td>16.7</td>
<td>Querying QoS Requirements</td>
<td>430</td>
</tr>
<tr>
<td>16.8</td>
<td>Querying Output Messages of Executed Services</td>
<td>430</td>
</tr>
<tr>
<td>16.9</td>
<td>Querying Input Messages of Executed Services</td>
<td>431</td>
</tr>
<tr>
<td>16.10</td>
<td>Querying Service Availability</td>
<td>431</td>
</tr>
<tr>
<td>16.11</td>
<td>Querying Service Reliability</td>
<td>432</td>
</tr>
<tr>
<td>16.12</td>
<td>Querying Service Security</td>
<td>432</td>
</tr>
<tr>
<td>16.13</td>
<td>Querying Service Reputation</td>
<td>433</td>
</tr>
<tr>
<td>16.14</td>
<td>Querying Service Cost</td>
<td>434</td>
</tr>
<tr>
<td>16.15</td>
<td>Querying Service Penalty</td>
<td>435</td>
</tr>
<tr>
<td>16.16</td>
<td>Querying Service Execution Time</td>
<td>436</td>
</tr>
<tr>
<td>16.17</td>
<td>Querying Service Response Time</td>
<td>437</td>
</tr>
<tr>
<td>16.18</td>
<td>Querying Time</td>
<td>438</td>
</tr>
<tr>
<td>16.19</td>
<td>Querying QoS Offered</td>
<td>438</td>
</tr>
<tr>
<td>16.20</td>
<td>Querying Discovered Services’ Identity</td>
<td>439</td>
</tr>
</tbody>
</table>
Science is competitive, aggressive, demanding.

It is also imaginative, inspiring, uplifting.

VERA RUBIN
Acknowledgments

This research has been funded by the European Union under FP7 ITN RE-LATE.

A PhD thesis is never the result of a single person’s effort and it would not have been possible without the support of many different people. First and foremost, I would like to thank my supervisor Prof. Steffen Zschaler for giving me the chance to do a PhD thesis in the space of non-functional properties (NFPs) for compositions of services, taking a model-driven engineering approach, and specialising on the quality aspect of provenance awareness as an explicit NFP. He created an environment that allowed me to work on new ideas regarding provenance, on the data, application and infrastructure level. I achieved this by working and combining the aspects of service-oriented computing systems specification, provenance infrastructure modelling and design, but also by independently focusing on the specification and analysis of the provenance data structures through the use of ontology-based graph models (including PROV-O, ServiceProv and SEALs ontologies) and semantic web standards (including OWL, RDF and SPARQL). Additionally, I would like to thank Prof. S. Miles for being my second supervisor and his valuable contribution and experience with regard to provenance and provenance awareness aspects.

I would also like to thank my colleagues in RELATE group, especially the postdoc Kahina Hamadache, and the core members of RELATE for their feedback during our numerous workshop sessions that took place in the different locations where the university and industrial partners of RELATE were based. It was a great time working together with you on various research papers and having fruitful discussions on related aspects that enhanced my domains of knowledge and my research skills on a broader spectrum. I am also very grateful for the great time I had during my two placements: 1) my research placement at the Karlsruhe
Institute of Technology (KIT) in Germany and 2) my industrial placement in Singular Logic in Athens Greece. The Palladio and Descartes research groups in KIT, provided a great environment for constructive criticism that enabled me to further advance my research perspective while the research team in Singular Logic, and especially Kahina Hamadache was a main source of inspiration for my industrial case study and the exploration of research paths, including reputation and qualities for cloud-based services, cross-cutting the provenance awareness concerns of this thesis. In addition, George Angouras gave me his full support on providing real-system provenance and logging data that allowed me to successfully conduct my industrial case study.

My deepest and sincere thanks goes to my parents Evangelos and Vasiliki and my brother Thomas who supported me emotionally and financially throughout my whole life and my beloved friends who were always there to listen to my worries and thoughts. All things we have experienced in different places of the world so far will never be unforgotten. I dedicate my thesis to all the people in my life journey till now that I love and have experienced life-long learning.
Abstract

The Service-Oriented Computing (SOC) paradigm, realizing a software design philosophy and architecture – the service-oriented architecture (SOA) – provides the means to develop service-oriented systems by exploiting and composing loosely coupled services. As service-oriented systems become increasingly large-scale and infrastructure-heterogeneous, their execution leads to excessive data production stressing the need for service providers to exhibit accountability about the systems’ qualities and actions. Accountability requires that systems faithfully document their execution, being able to answer questions about how the data was produced and processed, referred to as data provenance.

Provenance awareness is the functionality providing access to provenance by allowing users to query data about past processes and answer provenance questions. The SOC paradigm promotes composability. For a composite service, the independent audits of atomic services do not simply compose to a connected queryable picture of the provenance across the end-to-end pathways of a composition’s execution. Taking into account the inherent complexity of service discovery, selection and dynamic composition aspects of SOC, we realize that we need to carefully design for provenance awareness support by modeling explicitly the provenance data structures and infrastructure independently of specific applications.

This thesis proposes a set of formal models, analysis methods, and tools to address this need. We contribute a formal specification and analysis framework for provenance-awareness of service compositions introducing: a faceted classification of provenance questions to formally express provenance requirements acting as analytical metrics for provenance-awareness, a provenance data model capturing the provenance structures of service compositions, a template provenance infrastructure metamodel allowing one to design the provenance infrastructure of atomic and composite services, and a modeling and analysis environment that
verifies through simulation and analysis whether the provenance infrastructure system design satisfies a set of provenance requirements. As a proof of concept, we provide a prototype tool implementation of our framework.
Chapter 1

Introduction

In this chapter, we briefly set the research context, motivate and identify the research gap for this thesis. In particular in Section 1.1 we briefly set the background and provide motivation about conducting research in the areas of service-orientation, specification languages for non-functional properties (NFPs) of service-oriented systems and service compositions, with a particular focus on provenance. We then discuss the open challenges in the space of modelling provenance awareness as a non-functional property for compositions of services which enables us to define the research problem, identify the research questions and outline the main contributions of this thesis.

Contents

1.1 Motivation .............................................. 23
1.2 Research Gap ........................................... 28
1.3 Problem Definition ................................. 30
1.4 Motivating Use Case Example ......................... 35
1.5 Research Questions ................................. 37
1.6 Contributions ........................................ 38
1.7 Expected Benefits ................................. 41
1.8 Organization of the Thesis ......................... 42
1.9 Declarations ........................................ 43
1.1 Motivation

Over the last ten years, the Web [20] has led to the availability of software services and the possibility of developing highly distributed service-based systems that enable software reuse and system integration across different organisations [139]. The distribution of software systems – including service-based systems – has led to a significant rise in system and application complexity. The system and application complexity has been primarily caused by an increased demand for integrating and aggregating heterogeneous services and resources to achieve higher-level functionality and address more complex business requirements. Those requirements have given rise to the adaptation of new paradigms and standards, such as the service-oriented architecture (SOA) [95] and the service-oriented computing (SOC) paradigm [119], that can cope with the challenges of highly distributed services and the heterogeneous nature of the infrastructure that services rely on.

Service-orientation has sparked a movement that has positioned SOA as the driver for the next phase in evolution and success of traditional distributed architecture on the global scale, establishing a universal logical model and introducing a globally accepted standard for the architecture of computer software design. SOA aims to provide services to be consumed either by end-user applications or by other services (clients) or business processes distributed in a network. Those services are well-defined, self-contained, and platform-independent computational entities that perform certain business functions [118]. They use a set of published and discoverable interfaces and message-exchanging protocols to communicate with each other and they can be dynamically located, discovered and invoked through service brokers that publish their service descriptions along with additional information such as the services’ reliability, trustworthiness, quality of service (QoS) and service level agreements (SLAs) [154]. Web Services in particular utilise standardized protocols for describing and publishing new available services, such as Web Services Description Language (WSDL) [107], Simple Object Access Protocol (SOAP) [106], and Universal Description, Discovery and
Integration registry (UDDI) [17], in order to support the processes of service publication and discovery. Web services, driven by the Web Services Architecture [7], are mentioned here as they implement a web-based SOA, and they provide a standard implementation of the service-oriented computing (SOC) paradigm.

One of the primary assets of service-orientation is composability, defined as the ability to build new composite services by re-using well-known functionality provided by other locally and remotely available services [116]. Service composition is an important aspect of building software services when no single service is sufficient to satisfy all the required functionality. In particular, Service-oriented computing (SOC) [119] comprises a software design philosophy and a system architecture – the service-oriented architecture (SOA) – which promotes the idea of assembling application components into a network of services. Those services can be loosely coupled in order to create flexible, dynamic business processes and compositions of services, and agile applications that span across large-scale organizations and computing platforms [119].

Service-oriented systems aim to satisfy primarily a set of functional, but also a set of non-functional requirements (NFRs) [59], with functionality being the key for selecting the appropriate candidates for forming a composition of services. However, different service providers may make available services with similar or identical functionality, and thus in those cases, service clients will need to consider and choose and decide among services based on other quality attributes, such as performance or reliability, which would better satisfy the users’ non-functional requirements. Those qualities are defined as non-functional properties (NFPs) and take the form of quantitative quality attributes specified as constraints over quality values, as measured by specified metrics [132].

As services increase in complexity with regards to compositional processing and data integration, service providers have to come up with new techniques for data management and data processing to satisfy non-functional requirements and provide guarantees on the different system or service qualities the end users may
ask for. Those requirements include guarantees about service properties such as: the performance or availability of services, the security and trustworthiness of the executed processes and orchestrations that manage and produce new data results as part of their execution, the reliability of the sources that produced a data item, the validity of the final data generated as a result of compositional processing and service execution.

Among other quality guarantees, an important aspect that is becoming crucial for service providers offering composite-service solutions is accountability [151]. Dynamically assembled systems - where individual services are dynamically discovered, bound and executed as compositions of services - need to be made accountable for users to gain trust in them [108]. Accountability enables service consumers to be more confident of the validity and quality of the execution of services they use, provides an understanding of how the data results routed through the service execution cycle came to be as they are, therefore creating confidence that the composite services exchanging and using the data are performing as expected.

In particular providers of composed services need to provide accountability support with regards to:

- the quality properties of the system and the data elements processed and produced, including data items such as service level agreements (SLAs) [90] and quality contracts in cases where there are cross-dependencies in the services' execution; this aspect of accountability involves the requirement for information usage to be transparent, so it is possible to determine whether the use of information is appropriate for a given set of rules and to assert/-explain compliance with or violation of policies,

- past execution processes, system actions, and decisions made, and routes followed during the service-oriented (computing) execution cycle phases (including phases of service discovery, service selection, orchestration and choreography),
• the actual data elements processing by providing auditing trails fully tracing the source and generation process of data products on the end-to-end multiple pathways taken as composite services execute; that is, when and where which data was processed to generate which new data.

Accountability has been related to the issue of maintaining provenance for scientific data as part of scientific processes’ execution [56] and data provenance in web and grid services [151]. In both cases, accountability came as a requirement for systems that needed to document their execution faithfully. In web-based systems, this gives the ability to look down on how, when and by whom the data was produced and processed while with regards to scientific workflow systems it acts as a logbook that enables the reproduction of the same processing steps followed during experimental computations. Provenance is the key enabler for accountable systems providing the necessary logs to reason about the system’s history processing since it consists of an explicit auditing of past processes, which allows tracing the origin of data, actions and decisions (whether automated or human-driven) [151]. In the case of service compositions, the requirement about accountability support is becoming a requirement for being able to answer questions with regards to the data that were produced and processed during the system’s execution; the system’s data provenance [150]. Data provenance is therefore expressed as historical metadata describing the system’s processing during its execution cycle that allows full traceability of past execution processes, routes followed and data origins.

The ability of a system to answer any questions about the history of the system’s processing is called provenance awareness and requires that provenance data are recorded and stored at system’s execution time [104]. The need for provenance awareness becomes essential for service compositions as promoted by the SOC paradigm where several individual services are executed, and a large number of result datasets are produced and processed in the forms of data files, data values or parameters. This excess of data exchange and production over systems running on distributed and heterogeneous infrastructures requires capturing
the derivation history of data products associated with the various activities/processes and agents incorporated into the service-oriented computing (SOC) execution cycle. This historical metadata is then used as the foundational piece of information to provide accountability support for a composite service system.

Considering the inherent complexity of service discovery, selection and dynamic composition aspects in SOC systems, along with the requirements for a connected picture of the provenance data information capture that would allow querying over end-to-end operations executing within the SOC execution cycle, we realize that we can no longer rely on ad-hoc solutions for provenance. Complexity here refers to two main aspects. First, there is complexity on the process of identifying and modelling all provenance concepts representing critical provenance information that are required to be captured across the different phases of the SOC execution cycle. This process incorporates the need to identify through a thorough literature review all the concepts defined as part of the SOC execution cycle such as the different roles involved, their activities and the end products of those activities for which it is critical to capture provenance. This thorough review prevents us from missing out any important information in the provenance trails constructed in order to get a sufficient provenance graph generated as a final result. Second, the complexity of representing the connections of those provenance concepts across all SOC execution cycle phases for service compositions that are formed when dynamically aggregating third party services made available by different service providers. The complexity here originates from the formulation of a comprehensive specification language that provides the notation and tools to model i) the dynamic connections of all the provenance information for the services and service compositions across the different phases of the SOC execution cycle (e.g., around service publishing, discovery, orchestration, choreography, non-functional qualities and SLAs, resources availability) where sequential, parallel or iterative provenance activities need to be captured, ii) the connections between the various provenance concepts captured at three different levels of granularity: the data, service and service composition level while satisfying the need for all this
information to be represented in a connected provenance data structure. The need to incorporate the concepts of time (timestamps), the sequence/parallelism of services executed as part of the composition workflow, and the design-time information adds even more complexity in the creation of the cross-level connected data structure.

Therefore, we need to carefully design provenance support – as is possible for other non-functional properties (NFPs) such as performance or reliability – by modelling the provenance data collected in the different phases of the SOC execution cycle and the provenance recording infrastructure activity that is accountable to generate this information as part of an integrated architectural solution. Modelling the provenance properties of a service-oriented system will enable measurement, analysis, and validation at design time of whether the provenance infrastructure incorporated into the system’s design would be sufficient for capturing the provenance data required to satisfy a set of provenance requirements. This model-driven engineering provenance approach will enable measurement, analysis, revision and changes in the provenance infrastructure design beforehand the system’s actual implementation.

This thesis addresses the design and development of the required models/specifications, analysis methods, and tools to cope with these issues. In particular, it proposes a specification and analysis framework for provenance awareness – including a set of proposed models for provenance awareness properties and requirements, and an implementation toolset to realise the proposed specification and analysis approach – of SOC systems and service compositions. Next, we identify the research gap for a formal specification for provenance awareness of service compositions.

1.2 Research Gap

Having provided motivation and set the background briefly for our research, in this section, we identify the research gap and open issues about specification
languages for non-functional properties, with a particular focus on modelling and formalising provenance awareness for compositions of services. Modelling provenance awareness for service compositions is the core aspect we are trying to address through this thesis.

Services have associated non-functional properties (NFPs) such as performance (measured by metrics such as response time and throughput), reliability or security [132] that should be considered to best satisfy the overall non-functional requirements (NFRs). NFPs is an important aspect of service compositions since they can serve the role of discriminators for selecting the appropriate candidate services with similar functionality to implement quality optimal composite-service designs.

Composition according to NFPs is not a trivial task as there are dependencies between NFPs and the properties of the execution environment (e.g., workload, physical resources that influence NFPs) [69]. Formal modelling of those properties is crucial for enabling precise specification and analysis at design time a priori to the system’s implementation which prevents unnecessary re-engineering costs. Modelling languages for expressing NFPs of individual services, and accompanying approaches to specify and analyse the behaviour of NFPs for composite services have been developed as shown in Table 2.2. Those only focus on properties such performance [6,51] or reliability [43] and do not allow the specification of other properties such as provenance awareness which remains unexplored. Provenance awareness forms a relatively recent concern for NFPs of service-oriented systems and their compositions that need to provide support for answering questions about the documented history of a service composition’s processing and what processes led to the data produced, namely the data’s provenance [64,105].

Composite-service systems need to be accountable, answering questions with regards to provenance (provenance questions), about past executions of individual services and their interactions, the data items exchanged and produced during their execution and the dependencies among those items. We define the ability of
a service-oriented system to answer different types of questions, with regards to the documented history of the services’ processing and execution, as provenance awareness, where services may be compositions of other services. Those are being referred to as atomic services. Provenance awareness is, therefore, the non-functional quality of a system providing access to provenance.

In this research our focus is on filling the gap with regards to designing provenance awareness for composite services, making this accessible as a measurable non-functional property design, similarly to performance or reliability. We aim into providing specification languages for expressing both the provenance properties of service compositions (including the provenance data structures and the provenance infrastructure) and the provenance awareness requirements for composite service designs. We are also interested in developing the necessary tools that will enable modelling and analysis of provenance awareness, incorporated as part of the composite service design, where the results of such an analysis are beneficial to:

1. drive decisions about the required provenance infrastructure and choice of an efficient architectural design to support a set of provenance awareness requirements a priori to system’s implementation,

2. identify the trade-offs of incorporating provenance awareness into a composite service-based system’s design with regards to other non-functional properties such as performance or scalability.

Having identified the research gap for this thesis, we next present and define the research problem this thesis is trying to address.

1.3 Problem Definition

In the previous section we discussed open issues with regards to providing a language for formally specifying provenance awareness of service compositions. Next, we express the problem statement for the proposed research which is twofold:
i) the need to provide design support and a language to express, model and specify provenance awareness for composite services as a non-functional property (NFP)

ii) the need to incorporate provenance awareness in the design of composite service-based systems to enable analysis/validation and measurement of the system’s provenance awareness support a priori to its implementation against a set of given provenance awareness requirements

Modelling Provenance Awareness as an NFP for Composite Service Specifications. Composition of services is an important aspect in the way online systems are structured today since in most cases a single service would not be able to satisfy the complete set of requirements consumed by numerous users. The success of modern tech companies is greatly relying on covering the users’ needs by creating new high-level composite services (service compositions or mashups) in a fast pace rhythm and by assuring certain quality guarantees. This makes the selection of candidate services a rather challenging task as both functional and non-functional requirements need to be taken into consideration. Also, users may have different requirements regarding the non-functional aspects of services, and therefore the interdependency between NFPs and the associated trade-offs between those are adding extra complexity into the service design process. In such cases, the precise specification and explicit representation of NFPs is important, as ad-hoc solutions for ensuring certain quality guarantees with regards to users non-functional requirements at system’s run-time are hard to be met or concretely analysed and measured.

At design time the purpose of formal modelling of NFPs enables taking better design decisions by comparing different design alternatives and analysing which of these alternative designs would meet the corresponding non-functional requirements (NFRs). Where services are composed, the service composition’s NFPs will depend on those of the individual services composed as well as on the details (including architecture and infrastructure details) of the composition and its in-
terdependencies with the individual services fitting into the bigger picture. All those aspects need to be taken into consideration when comparing and making decisions with regards to non-functional properties of a composite service system at design-time; therefore they create a need and a challenge for developing expressive specification languages that provide the modelling notation to achieve so.

With regards to providing a specification language that allows modelling provenance awareness as an explicit NFP, there are additional challenges to the composition details (architecture and infrastructure) and interdependencies with individual services and other NFPs. First, there is a challenge to provide appropriate mechanisms that will allow to model and specify explicitly the provenance data structures at different levels of granularity (e.g., system/application level, composition level, service level and data element level), as well as the provenance infrastructure (activity) that would capture the corresponding provenance information. Second, there is a challenge in identifying the key aspects that could act as the binders between those two types of specifications to provide a connected provenance model picture for atomic services and their composition. The third challenge is into looking how the provenance infrastructure design affects the overall system infrastructure design and the associated trade-offs that may be caused with other NFPs such as performance or scalability. Fourth, with the SOC paradigm promoting compositionality, there is an inherent complexity in providing a specification language that allows to model provenance across the different phases of SOC execution cycle including service discovery, selection and orchestration execution phases.

A number of approaches have been developed to provide a framework for capturing and recording provenance in SOAs [101,125,149,150]. Those works do not address how provenance awareness can be defined and expressed as an explicit NFP addressing the challenges mentioned in the specific area. With our work, we aim to address this gap by developing the necessary modelling notation and tools.
Incorporating Provenance Awareness in Composite Service Designs to enable analysis of system’s accountability support. Second, as on-line services (e.g., web services) become more and more complex, service-oriented systems need to be accountable for their system actions and execution being able to answer questions (provenance questions) about how they have processed and produced data (provenance data) at run-time. Accountability is an important quality for service providers that enable them to showcase guarantees for performance, reliability and trustworthiness both for the process – service execution – and for the data items that will be produced during execution, through simulating and analysing the support of those properties at system’s design time.

The need for accountability and developing provenance-aware systems with the ability to answer provenance questions is even more evident in compositions of services, where audits of each atomic service’s use taking part in the composition, do not provide a connected picture of the composition’s processing history. Atomic provenance, referring to the provenance information captured for each individual services being part of a service composition, does not simply compose into composite provenance, referring to the provenance information captured about service composition execution. We, therefore, can not rely on ad-hoc solutions to provenance; we need to carefully design provenance awareness support for service compositions in a form that allows analysis and measuring of the provenance awareness properties at design time. A number of works on provenance have been proposed to implement provenance recording and providing provenance storage [112, 128] utilities as part of the web or cloud-based systems. Yet, those do not allow provenance-awareness to be specified as part of a composite service system design considering the foundations of SOA and the SOC paradigm, so that the provenance support can be analysed and measured a priori to the system’s implementation.

The challenge here is twofold. First, to identify the mechanisms for simulating the provenance captured by the specified provenance infrastructure design, where the results of this simulation can be expressed in a modelling representation that
exhibits provenance as a unified knowledge graph structure we could query. Simulation allows predicting at design time whether the system’s associated provenance infrastructure design can capture the desired provenance information for service compositions before implementation. This is achieved by simulating the composition’s provenance recording activity, where the generated outcome of the simulation is a provenance data graph structure on which we can apply analysis checks to validate the system’s provenance awareness support. A simulator is needed to allow application and infrastructure designers to compare different design alternatives and determine which design would best meet the provenance awareness requirements of the system’s users. Compared to an analysis tool, at this stage we choose a simulator as simulations can be easily designed, repeated, allow rapid prototyping of different design alternatives for provenance awareness and conclude in initial results quickly. Those preliminary results give feedback on the properties an application/infrastructure designer is modelling early enough in the design and development lifecycle of a service composition. In this way, he can make better design decisions before the system’s implementation which prevents unnecessary re-engineering costs and cuts down on time and resources required in cases of reimplementation. In addition, a simulator provides a test environment for the infrastructure designers to verify how making changes in the provenance infrastructure design may affect an existing system’s implementation without putting at risk the availability of the actual system and the current system’s implementation. Overall, provenance properties of a service composition depend on the properties of the individual services composed, on the architecture and infrastructure details of the composition and the composition’s interdependencies with the individual services. Those aspects make challenging the effort of the application and infrastructure designers to take the right decisions before implementation; therefore, simulation allows them to analyse and foresee the outcome of different decisions by decomposing the complexity of modelling provenance properties of a service composition.
Second, to identify the mechanisms in place that allow us to define provenance awareness as an analytical metric and propose ways to formally express provenance awareness requirements. Those mechanisms will enable analysis and validation of whether the provenance infrastructure design is sufficient to capture the provenance data required for satisfying a set of provenance awareness requirements and therefore determine and measure the system’s accountability support at system’s design-time.

1.4 Motivating Use Case Example

Figure 1.1 depicts a service composition for a travel planner scenario. That is a standard example (use case scenario) taken from the service composition literature [40] that we use throughout our thesis in order to explain our motivation for resolving the problems defined in this research study and to present our approach and architecture in Chapter 4.

The control flow of this composition is represented as a UML activity diagram where an action corresponds to the specification of an abstract service task to be executed in which a named service operation is invoked. Arrows run from the start towards the end and represent the order in which actions happen. In a given instantiation, each task would be performed by a particular service discovered at run-time. The tasks are to book a flight (t₁), book a hotel (t₂), book an attraction to visit (t₃), calculate the driving time from the hotel to the attraction (t₄), rent a bike to travel between the hotel and attraction (t₅), rent a car to travel between the hotel and attraction (t₆), and take payment for the travel package by credit

![Figure 1.1: Motivating Example: Travel Planner Scenario Workflow](image_url)
card \( (t_\gamma) \). The composition is expressed in terms of control flows, fork/join nodes to express concurrency, and decision nodes to represent conditional branching.

A user interacts with the *travel planner* service composition provider which discovers, then invokes each of these atomic services, such as the flight booking or the hotel booking service, providing a complete service composition solution.

The users may have requirements on a number of non-functional properties (NFPs) such as performance, reliability, security or provenance awareness. Making the selection of the appropriate services for a service composition relies on both the functional and non-functional properties of the individual service candidates, as users may have different requirements regarding the non-functional aspects of services which should be taken into account. Specification languages for NFPs enable designers to model the non-functional aspects of composite service systems and analyse the system’s support for a set of non-functional requirements (NFRs). Where services are composed, the service composition’s NFPs will depend on those of the individual services composed as well as on details of the composition.

With regards to provenance awareness, the users may be interested to know answers to questions related to the provenance of the travel planner results and the travel planner composition execution such as: why was a bike booked instead of a car, why was a high-cost flight booked, which bank managed the credit card, were all the hotel services available during the hotel booking selection. These questions can be answered based on provenance data recorded during the execution of the travel planner. To ensure the right data would be collected at runtime, designers need to plan for the provenance awareness infrastructure support required at service design time.
1.5 Research Questions

In the previous sections we identified a gap with regards to the existing approaches for modelling, specifying and analysing provenance awareness as an NFP and we explained our problem definition exhibiting this through a motivating use case scenario. To the best of our knowledge, there is no formal approach and modelling framework that will allow the specification and analysis of provenance as an NFP for the composition of services. Therefore, there is the need for a set of models and a framework to effectively specify and analyse provenance awareness support of service-based systems at system’s design-time.

Having this as the principal goal this thesis is guided by the following four main research questions including a set of sub-questions that we discuss in more detail as part of Chapter 4.

- RQ 1 How can we express provenance awareness as a non-functional requirement for service-oriented (computing) systems and service compositions to be able to analyse a system’s accountability support?

- RQ 2 How can we design provenance awareness by providing support for specifying provenance properties such as the provenance infrastructure and provenance data structures for service-based systems and compositions of services, considering the architectural aspects and principles of the Service-Oriented Computing (SOC) paradigm and Service-oriented Architecture (SOA)? How considering the different phases of the SOC execution cycle and the principle SOA concepts affect architectural and modelling decisions about the provenance infrastructure and provenance data structures?

- RQ 3 What would be the method and supporting framework for simulating and analysing provenance awareness support of service-oriented (computing) systems operating across the full set of SOC execution cycle phases at system’s design-time?
• RQ 4 How can we use the proposed framework of provenance awareness in order to enable measurement in a standardised way and implement analytical metrics for the system’s provenance awareness support against a set of provenance awareness requirements?

1.6 Contributions

Having as a basis the research questions of Section 1.5, in this section, we present the contributions of this thesis, visualised as part of a simplified\(^1\) version of our architecture for the specification and analysis provenance framework proposed, that grasps the “big picture” of this thesis. The architecture as depicted in Figure 1.2 comprises of two different levels 1) the model M1 level and 2) the metamodel M2 level. We have annotated the architecture to associate the specific contributions to the different models/aspects of the corresponding layers. We next give a description of our main contributions.

![Figure 1.2: Provenance Awareness Specification and Analysis Framework](image)

**Contribution 1: Provenance Data and Infrastructure Model for Service Provenance.** With this thesis we first contribute a specification language for data provenance of composite services by identifying a set of required models:

\(^{1}\)A complete description of our architecture and approach is discussed in Chapter 5 while a diagrammatic view of the core components of our architecture is presented in Figure 5.1.
the service interface and service composition model, provenance models including models that express the provenance data structure and the provenance infrastructure behaviour of a composite service system and their interconnections as visualized in Figure 1.2. Those models provide a formal concept definition to specify 1) atomic and composite services (service interface and composition models), 2) provenance data (provenance data schema), 3) provenance recording behaviour (provenance infrastructure model).

Regarding the functional specification of composite services, including the interfaces and composition relations of atomic and composite services, this is expressed by existing specification languages for services and their compositions, including WSDL and BPEL. The provenance data model for composition of services describes the types of provenance data that may be recorded and stored when executing that service composition. We have expressed those provenance data concepts in the form of an ontology-based data model, namely ServiceProv Ontology, as an extension of PROV-O [14]. The provenance data concepts then map to a set of templated models that express the provenance infrastructure recording activity and behaviour of a system. Inspired by an approach by Danger and Curcin [33], we use the notion of provenance template to introduce a higher level abstraction of the provenance data graph records we would like our system to support. Those provenance data graph records represent the provenance data structures that could be captured regarding the overall provenance recording infrastructure activity across the service-oriented computing execution cycle. In fact multiple provenance models created by service designers, expressed as provenance templates, form instances of the service provenance meta-model that defines the core concepts and relationships of service provenance. Finally, we also introduce the modelling notation for expressing the provenance infrastructure recording activity as a set of UML stereotypes. The tagged properties of those activities (passed in the form of template parameters) then correspond to the provenance data model instances we would like our system to record.
Contribution 1 is mainly answering research question RQ 2 as presented in Section 1.5.

Contribution 2: Provenance Facets In order to define provenance awareness for service compositions in this thesis we also identify a number of elemental provenance question categories so that provenance awareness can be specified as the categories of questions a system can answer. Many realistic provenance questions require a combination of kinds of data so that those could be answered. Therefore, instead of dividing all questions into distinct categories, we define a number of provenance facets [156], each corresponding to some kind of captured data, and then show how a given question exhibits a combination of facets – meaning that it requires those facets’ corresponding data to be captured. The facets act as orthogonal axes for the space of provenance questions. We then contribute a set of formal models for the provenance questions, namely a set of provenance-question-type-patterns covered as an additional contribution (Contribution 3).

Contribution 3: Provenance Question Type Patterns. We contribute a set of formal models using the SPARQL query language for expressing the different kinds of provenance questions – representing provenance awareness requirements of users – exhibiting the provenance facets. Those formal models, namely provenance question type patterns are provided in a catalogue form. The catalogue of question type patterns is used as a basis to analyse a composite service system’s provenance awareness support, by checking at design time whether provenance data provided by the system specified in a provenance model, is captured either by one or an aggregation of those patterns. Patterns contain variables, and in the case of provenance question type patterns, those variables correspond to the key provenance data concepts required for satisfying a set of provenance awareness requirements. To express provenance awareness requirements we start with asking high-level provenance questions expressed in natural language. Then those questions are expressed formally becoming typed questions, adhering to a provenance data model (schema), where the schema allows one to
formally express the core concepts and constructs of a provenance question and the assertions that need to be made in order to query certain provenance information. The results of querying specific provenance information allow one to measure provenance awareness of a system. Therefore, the provenance question type becomes a pattern that validates support of specific provenance awareness requirements with regards to the system’s provenance-aware design.

The aforementioned Contributions 2 and 3 are answering research question RQ 1 as presented in Section 1.5.

Contribution 4: Modelling Environment for Provenance Awareness Analysis Support. This thesis also contributes a specification and modelling framework for designing and analysing provenance awareness support of service compositions. The framework allows one to perform matching between the provenance question type patterns expressing the provenance awareness requirements, and the provenance data graphs generated by simulating the end-results of the provenance infrastructure recording behaviour. To this end, we have built a modelling and analysis environment and a prototype tool implementation (PROVa) for provenance awareness. PROVa allows one to apply checks at design-time of whether the provenance data provided by the system – specified through a provenance data model – is captured either by one or a composition of provenance question type patterns in our catalog.

Contribution 4 is answering research questions RQ 3 and RQ 4 respectively as presented in Section 1.5.

1.7 Expected Benefits

The benefits of the proposed approach and architecture include the following:

- Service designers will be able to ensure that their services will meet provenance awareness requirements at run-time, by being able to specify provenance-
related properties at design time and analyse the system’s provenance support apriori to system’s implementation.

- Formal specification of provenance awareness will constitute a basis for comparing different models of provenance data recording and storage mechanisms used to satisfy a set of provenance awareness related requirements (answers to provenance questions) for composite service specifications. This is essential for evaluating/verifying whether the modelled provenance infrastructure design of a service composition would satisfy the corresponding provenance-related requirements.

- Service designers will be able to analyse the impact of provenance awareness recording and storing mechanisms on other NFPs. The challenge here is that the proposed framework will be built in such a way in order to form a basis for identifying and balancing the trade-offs between provenance awareness and other NFPs (e.g., performance or scalability).

### 1.8 Organization of the Thesis

This thesis is organised as follows: Chapter 2 first presents the background classified into sections about the fundamentals of services and service composition, functional and non-functional properties (NFPs) and state of the art in specification languages for NFPs and provenance. It concludes with an introduction on defining provenance and provenance awareness in the context of our research. We then discuss provenance related work on mechanisms for capturing provenance for SOA and service-based systems (including cloud-based services), existing provenance data models and querying languages for provenance information in Chapter 3. Chapter 4 presents our problem statement and research questions in detail based on an example use case and Chapter 5 discusses our proposed approach and overall architecture towards a formal specification and analysis framework for provenance awareness. We further present our research methodology towards fulfilling our research goals, and the expected benefits of
the research conducted in this thesis.

The thesis is then split into two major parts: *Part I* comprises the contributions related to the formalisation of provenance awareness, describing formal models for the provenance data of service compositions and provenance facets as well as a template provenance metamodel for modelling the provenance infrastructure recording activity and behaviour of service-oriented computing (SOC) systems. *Part II* comprises the contributions related to the modelling environment for analysing provenance awareness support, our prototype tool implementation PROVa and the evaluation of the proposed specification and analysis framework through two use case studies – an example and an industrial use case. We end up this thesis with presenting our conclusions and outlining our perspective for future research in the area of provenance trade-off analysis in relation to other NFPs (e.g., service reputation) and specifying provenance properties for cloud-based services.

### 1.9 Declarations

The contributions presented in Section 1.6 have been published as part of the research results of the conference work listed in this section.

In particular, in [D1] we introduce the need for defining and incorporating provenance awareness as part of composite service designs. In [D2] we define provenance awareness by introducing a set of provenance facets (Contribution 2). In [D3] we introduce a provenance ontology model for expressing the core concepts, structure and relationships of provenance data (concepts) for composite services in service-oriented environments (Contribution 1), while in [D5] we use our model in order to express provenance data, in particular feedback information of users about a service’s qualities and performance, that allows to derive conclusions about the reputation of service-oriented systems. This work makes a case study of how provenance awareness can be related to other non-functional properties (NFPs) of a service-based system.
Services, [D5], also forms one of our proposed areas of future research as discussed in Chapter 12. In [D4] we present how the provenance awareness requirements for an industrial case study scenario are expressed through a set of provenance question type patterns (Contribution 3). Finally, in [D6] we present the mechanics of our proposed framework for modelling and analysing provenance awareness of service-oriented computing (SOC) systems and its prototype tool implementation PROVa (Contribution 4).


D2 P. Zerva, S. Zschaler, S. Miles, Towards Design Support For Provenance Awareness: A Classification Of Provenance Questions, BIGProv 2013 (EDBT 2013), International Workshop on Managing and Querying Provenance Data at Scale, Genova Italy.


D6 P. Zerva, S. Zschaler, S. Miles: Modelling and Analysing Provenance Awareness Infrastructure For SOC Systems, Proceedings of the 9th IEEE Symposium On Service-Oriented System Engineering (SOSE 2015), San Francisco US.
Summary: In this chapter we have made an introduction on the motivation, research gap/questions, and main contributions and organisation of this thesis. Next, we present in detail the background and related work to our research problem.
Chapter 2

Background

Before discussing details on our research problem and proposed approach, we need to introduce the essential concepts for services, service composition and Service-Oriented Architecture (SOA). The main discussion of this chapter is on non-functional properties (NFPs) and why specifying those properties is necessary for service compositions and for analysing if the user non-functional requirements (NFRs) are satisfied. We first point out the differences between functional and non-functional requirements. We then discuss the different NFPs for individual and composite services, and we present existing specification languages for them. We identify a gap on specifying provenance awareness as an additional NFP for services and their compositions and also show the background with regards to how data provenance, accountability and provenance awareness are defined and managed in different contexts.

Contents

2.1 Fundamentals of Services and Compositions .......................... 47
   2.1.1 Service-Oriented Architecture and Services .................... 47
   2.1.2 Service Properties and Principles .............................. 51
   2.1.3 Service-oriented Computing Paradigm .......................... 56
2.2 Functional and Non-Functional Requirements ..................... 59
   2.2.1 Functional Requirements ...................................... 60
   2.2.2 Non-Functional Requirements ................................. 61
2.1 Fundamentals of Services and Compositions

In this section we discuss the fundamentals of atomic and composite services, their properties and principles related to our research and how those affect modelling NFPs, such as provenance awareness.

2.1.1 Service-Oriented Architecture and Services

SOA In this research we propose an approach to design and formally specify provenance awareness as an NFP for services, to be applied on the Service-oriented Architecture (SOA) [94] and Service-oriented Computing (SOC) [118] paradigm. SOA is an architectural approach to loosely coupled, protocol independent, standard-based distributed computing where software resources available on a network are considered as services [120]. Those services are used to build software applications and end-user services distributed in the network, via published and discoverable interfaces [119, 120]. SOA is the evolution of well-known distributed architectures and integration methods, yet it extends the flexibility of previous architectural approaches to focus on: reuse of applications, efficient interoperability, composition and unification of business processes by modularizing large applications into services [4]. “It promotes the loose coupling
principle, which facilitates reuse and composition among cross-enterprise applications” [120], where the notion of composition is of first importance for our research. In service compositions the complexity of the recorded history of the data produced and processed during its execution increases significantly compared to that of atomic services, leading to the need to provide a connected picture of the composition’s processing history, which can not be provided by only modelling the audit trails of atomic services.

SOA requires thinking about the system in terms of i) Services (well-defined business functionality), ii) Components (discrete pieces of code and/or data structures) and iii) Processes (Service orchestrations - generally using BPEL). Being able to compose new higher level services or business processes is a basic feature of a good SOA. XML, SOAP [106] based (web) services and related standards are good fit for realizing SOA. Also SOA has a few accepted principles as discussed later in detail in Section 2.1.2, while a SOA based architecture is expected to have a service definition. We later discuss and compare alternative realizations of web services such as REST-style services.

Service  Service-oriented architecture (SOA) uses services as its core constructs to build and compose applications through discovering and invoking network available services [117]. A service is an implementation of well-defined business functionality, self-contained (maintains its state), platform-agnostic computational element or process, which performs a specific task function described, published, located and invoked over a network. Those task functions can range from simple requests, business functions and business transactions composed of lower-level functions, to sophisticated business processes [40,95].

Web Services  For the needs of this thesis we also define the properties of web services as we use a use case scenario of web services to state our research problem, present and evaluate our approach. Web services provide the basis for the development and execution of business processes that are distributed over the network and are made available via standard interfaces and protocols. Web ser-
vices are modular, self-aware, self-describing applications. A web service knows how to perform its discrete tasks and how to interwork with other services. They are characterised by great interoperability and extensibility thanks to service descriptions agreed between the service requester and the service provider. Web services description and specification of their properties are crucial for the successful composition of their functionality, which leads into defining new composite services. Therefore, different aspects need to be considered including functional and non-functional, with non-functional properties being the main focus of this research.

Although SOA is an architectural style independent of specific technologies, such as web services, it seems that web services has become the preferable way to implement SOAs [31]. Web Services communicate through the internet, therefore they comply with the web services standards such as the Simple Object Access Protocol (SOAP) [106], the Web Services Description Language (WSDL) [107] for defining services, and the Business Process Execution Language for Web Services (BPEL4WS) [4] for orchestrating services.

In a 2004 document [20], the W3C extended the definition of a web service identifying two major classes of web services: REST-compliant web services, in which the primary purpose of the service is to manipulate XML representations of web resources using a uniform set of “stateless” operations; and arbitrary web services, in which the service may expose an arbitrary set of operations. Many web services nowadays use JSON serialisation over self-describing representational state transfer (REST) APIs. Restful APIs do not require XML-based web service protocols (SOAP and WSDL) to support their interfaces [135].

RESTful Web Services exhibit a lightweight implementation compared to its competitor (WSDL/SOAP). However, REST lacks a definitive service definition. Therefore, it is difficult for a REST based system to fulfill most of the service principles (Section 2.1.2) required for a SOA architecture. As the focus of this research is to model provenance awareness for service compositions where Business
Process Execution Language (WS-BPEL) is the current standard language for web service composition, from a syntactical perspective, the service abstraction assumed by BPEL is the one provided by the Web service description language (WSDL). Since most RESTful Web services are not described using the standard Web service description language (WSDL), it is not possible to reuse existing languages and tools that require the presence of WSDL interface contracts like BPEL. REST requires clients to interact with many resources identified by URIs. BPEL instead assumes to interact with services bound to a few fixed communication endpoints. Thus, it does not handle well the variable set of URIs that make up the interface of a RESTful Web service. In addition, REST-style services have been created for different audiences and have different goals. They are good in querying resources but in our research we required more structure in information that was used as input to our template provenance infrastructure model. Coordinating multiple requests in order to get the provenance information required for dynamic service compositions will end up in a lot of expensive round trips. In addition, each request has a slightly different response shape and not necessarily the shape we want to use in our Template Provenance Infrastructure Model, therefore this would create the need for a lot of customizations in implementing different template binding specifications.

In order to support REST we would need to define and implement a set of different template binding specifications to support bindings with a REST style specification, but the rest of the components of our framework will remain independent of specific protocols. REST web services introduce a new kind of abstraction, the resource, and therefore REST requires clients to interact with many resources identified by URIs which does not fit well with the message-oriented paradigm of the Web service description language (WSDL). BPEL as mentioned already interacts with services that bind to fixed communication endpoints, thus the interworking with the RESTful Web service interfaces is hard to handle. In order to be able to support REST-style services we would need to look into a way to extend BPEL for REST, a theme that falls outside the focus
of this research and can be considered as future work. For example, C. Pautasso in his work [121] introduces extensions to enable processes that natively invoke RESTful Web service APIs and publish a view over their execution state through a RESTful Web service interface.

(Web) Services follow a number of principles that characterise the service-oriented (computing) context where they compose and execute. Next, we discuss on those principles to enable a better understanding of the service-oriented framework. A good knowledge of service properties and principles is essential for our research goals towards modelling provenance, as we need to consider the effect of atomic and composite services structure on provenance-awareness, e.g., the inter-dependencies between the data that are produced and processed by atomic services that participate in a service composition execution.

2.1.2 Service Properties and Principles

In the previous section we discussed SOA and services, as the proposed field of application for the specification of provenance awareness as an NFP. To better explain how services function and interact in the service-oriented framework, in this section we discuss their fundamental elements and characteristics, including well-defined interfaces, loose-coupling and composition. We extend our discussion on types of composition, including orchestration and choreography, an important topic that drives our research. Composition requires the consideration of different actors of services (e.g., orchestrators, aggregators) and activities (e.g., service discovery, service selection), which imply the need for various recording provenance mechanisms to ensure provenance awareness of service-oriented systems.

Well defined interfaces A service in an SOA is a bound pair of a service interface and a service implementation. The service is described using a well-defined interface that exposes the functionality and hides the underlying implementation details. Interfaces are used by service consumers to get access to services. The interface defines a set of operations, which are accessed through a set of messages
that specify the data to be exchanged. Regarding the definition of transactions in business processes we should consider that in SOA, services are following the coarse granulation paradigm. Granularity refers to the extent to which a system is broken down into smaller parts. Coarse-grained systems consist of fewer, larger components than fine-grained systems. The most frequent usages of SOA (e.g., web services) imply a relatively coarse and fixed granularity for services [30].

In the service-oriented world an interface is a contract between the service provider and the service consumer, platform-independent and self-described, separated from its implementation, that is used as a means to communicate [4]. Information on the provided and the required interfaces (e.g., provided and required operations), as well as on the way of how services communicate (e.g., protocols, message format) constitute valuable design information that affects the process and production of data during service composition execution.

**Dependencies** Any dependencies between services should be defined regarding common business processes, functions and data models. Similarly to the object-oriented paradigm, services try to provide a higher level of abstraction through encapsulation and reuse of existing services [21]. In services, there are no internal dependencies. Services are agnostic about other services, nor do they have knowledge about the services they want to interact with (e.g., orchestration). This makes provenance of composite services a challenging, yet interesting problem, as the traces of atomic services taking part in the composition can not just compose to provide a connected picture of the service composition’s processing history.

**Service Discovery** We already mentioned that services have neither intrinsic dependencies nor knowledge on other services they require to interact with (e.g., orchestration paradigm) [30]. Therefore, service discovery mechanisms are of particular importance. Service discovery is the process of finding a suitable service for a given task. Three different roles are involved in the process of discovery: the service provider, the service requestor and the service broker, as described more
extensively in [114]. The service provider provides a number of services that publishes through the service broker. The service broker receives descriptions of services from the service provider that act as advertisements for the service requestor. Finally, the service requestor can discover the appropriate services, using their service descriptions, which satisfy the respective user requirements. The concepts of the service provider, requestor and broker are also important for provenance as these constitute additional actors in the service management and service execution process during which additional provenance information is created.

**Loose coupling** The most important principle that SOA enables is loose-coupling. Loose coupling means that one unit of software is widely independent of another. It also means that one unit of software can relatively easily substitute an already-deployed unit. In services, loose coupling refers to the degree of dependencies between them. As services externalise their functions through well-defined interfaces, which let them communicate with other services, they can minimise those dependencies to the ones of the dependant’s service’s interface. Any service that implements the interface can thus satisfy the dependency of a dependent service without the need of changing this. The last results in the minimal requirement of changes, which promotes the principle of service reuse. Service reuse is particularly important for composite service designers when they try to identify suitable services satisfying certain functional or non-functional aspects (e.g., provenance awareness).

**Service Composition** The most necessary concept of SOA, according to [4], and also important for our proposed research, is the composition of services into business processes in a flexible way, by composing services in a particular order and with certain rules. The composition of services enables us to modify business processes due to the dynamic nature of web service environments, their foundation on network-based applications, the use of well-defined interfaces. Concerning web services, in case there is the need for invoking several so as to satisfy more complex tasks, it is necessary to combine their functionality and built composite services.
The process of developing a composite service is called service composition [40]. Service composition can be static (orchestration or choreography) or dynamic. Dynamic composition requires the location of services based on their capabilities and the recognition of those services that can be matched to create a composition [12]. In this research, we are interested mainly on static composition as we focus on modelling and analysing service composition properties at design-time. We are planning to address the problem of dynamic and on-the-fly composition by pre-planning possible adaptations which take into account different configurations of NFPs such as provenance awareness. This exercise leads into statically mapping out the dynamism of service-oriented composite systems and their NFPs.

Depending on the type of requirements, composition of services can either address internal (private within the same software development organization) or collaborative (public among different organisations) processes. Also, according the existence or absence of a central coordinator in place to initiate and manage the composition two notions are used respectively: Orchestration and choreography.

**Orchestration** In orchestration there is a central orchestrator that is responsible for taking control over the associated web services, by invoking and combining their operations. Only the orchestrator is aware of the higher business process that takes place, while the associated web services are agnostic about their existence in a composite service [4]. Orchestration describes how services interact at the message level, including the business logic and execution order of interactions under control of a single end point. It is an executable business process that can result in a long-lived, transactional, multistep process model [118].

Orchestration is usually used in private business processes and is achieved through BPEL4WS [4]. The documented history of the data that orchestrators manage and process constitute valuable provenance information we are interested in considering in our specification.
**Choreography**  Choreography does not rely on a central coordinator but enables the execution of complex tasks by specifying a conversation among the involved participants, focusing on the exchange of messages, rules of interaction and agreements in public (globally visible) business processes. Those agreements usually occur between multiple business process end points rather than a particular business process executed by a single party. The participants of the choreography need to be aware of the business process, operations to run, messages to exchange, the timing of message exchanges and whom to interact with [118]. These constitute as well examples of provenance data that need recording during service composition execution. Service choreography is achieved via the Web Services Choreography Description Language (WSCDL) [81] that specifies the common observable behaviour of all participants.

There are a number of specification languages for service composition (both orchestration and choreography), but those rather focus on the behavioural than the quality aspects of services. In order for the service composition to be successful, it should be quality aware (e.g., to understand and respect policies, performance levels and security requirements, and service level agreements (SLAs) [90]), by taking into consideration quality of service (QoS) aspects [153] and non-functional properties (NFPs) [36].

**Quality of Service**  Services have associated quality-of-service attributes. Such attributes, including performance, dependability, security, reliable messaging, transaction, management, policy, and other QoS properties, appear as a result of the user non-functional requirements (NFRs) [96]. Therefore, the infrastructure of services must provide support for those attributes. In web services, QoS attributes or else non-functional properties (NFPs) are described by web service specification languages (e.g., WS-Policy [146], WS-Security [113]). These properties differ from the functional aspects of services and are defined as constraints exhibited over the functionality of the service.
Having set the background for SOA, services and service compositions and their principles, it is important to note that being aware of the user NFRs of services is crucial so as to find the best composite service candidates. Analysing NFPs is a necessary procedure, which further pioneers the need for mechanisms in place to allow their formal specification. For a formal specification to be complete, it is important to identify the related concepts and factors that affect the NFPs as well as the dependencies between those. As we focus on provenance awareness as an NFP, in our future steps we identify related concepts and dependencies for provenance awareness with 1) the service (composition) principles and properties by introducing the core aspects of service-oriented computing (SOC) in the next section, 2) other NFPs as those are discussed in detail in Section 2.2.

2.1.3 Service-oriented Computing Paradigm

Service-oriented computing (SOC) [119] leverages a software design philosophy and a system architecture of composing a collection of higher-level loosely-coupled services that may run on distributed infrastructures and be part of heterogeneous software platforms. Those services communicate with each other through published and discoverable interfaces and message-exchanging protocols and can be aggregated to form composite service structures that satisfy more complex functional and non-functional requirements.

The SOC research road map as shown in Figure 2.1, introduces an extended SOA that separates the mechanics of functioning services in three levels: service foundations, composition, and management and monitoring. This architectural structure has its causes in need to separate: basic service capabilities provided through conventional SOA from more sophisticated functionality supported through dynamically composing services, business services from system-centered services (e.g., component-based services and applications), and service composition from service management and monitoring [118]. The perpendicular axis indicates properties cutting across all three levels including semantics, non-functional properties (NFPs) and quality of service (QoS) [118]. QoS, as
already discussed, incorporates important non-functional properties such as performance related metrics (e.g., response time, execution time), security features, transactional integrity, reliability, scalability, availability, provenance awareness and accountability. Also, as services aggregate to form composite services, there is a greater requirement for continuous availability of individual services, greater complexity in accessing and managing composite services as a whole, and mechanisms for monitoring both availability and the roles/their actions managing those services.

The roles defined by the SOC research roadmap include those of the service requester or client and service provider that have to agree on the service description (WSDL definition) and business processes and protocols (BPEL) – including the sequencing of messages exchanged between interacting services and the semantics that will govern the interaction between those services so that those services can successfully function as part of composite service applications. The roadmap also defines the roles of aggregator and operator. The aggregator may act as 1) a broker facing the service provider or the service requester directly during the service discovery phase, or 2) a planner during the service selection phase.
The service operator is acting on the management and monitoring level/plane looking into tracking information with regards to the health of the system, the execution of business processes and atomic services, and the managed resources and availability/reliability of the services involved. The service operator becomes a service aggregator when starts acting on the composition level by grouping services provided by other providers into a distinct value-added composite-service. The aggregator can then act as provider itself by publishing the service descriptions of the composite service they create.

Looking into more detail at the individual levels, the service foundations plane consists of a service-oriented middleware realising the SOA infrastructure at runtime. The SOA infrastructure connects heterogeneous services running on distributed systems and platforms while it allows applications to define underlying service functionality regarding service descriptions, interface publishing, discovery, and service binding. “In a typical service-based scenario, a provider hosts an implementation of a given service while it publishes the service’s functionality through a service description through which it can be made discoverable. A client or a service broker acting on the client’s behalf discovers a service and retrieves this through a registry or repository such as UDDI. The service description is then used to bind to the provider and invoke the service” [118].

The service composition plane is focusing on the aggregation part of several atomic services into a composite service. Service aggregators accomplish this task becoming themselves service providers by publishing the service descriptions of the composite services they create. The aggregators may also enforce policies on the invocation of the services aggregated. During service composition, service or application designers must be able to assess and monitor the status and health of their system from a higher level of granularity (composition level) to a finer standard of granularity (individual services). Therefore, there is a separate plane for addressing those issues; the service management and monitoring plane.
Service management is looking into gathering information about the services and business processes managed, the status of services and the state and performance of their execution by conducting root-cause failure analysis and providing SLA monitoring and reporting utilities as part of a management life-cycle. Those service controls involve monitoring activities for tracking down events or information produced by the services and processes or monitoring instances of business processes and their qualities and non-functional requirements [118]. This is essential for creating performing and satisfactory service compositions, in combination with modelling and service-oriented engineering techniques – service-oriented analysis, design and development methodologies – the SOC paradigm leverages.

Next, we discuss on requirements of NFPs in the service-oriented domain and on existing approaches that try to describe and specify them. We also present a comparison of the corresponding specification languages for service NFPs, while we identify a research gap by exhibiting properties (e.g., provenance awareness) that are not yet covered by existing approaches.

### 2.2 Functional and Non-Functional Requirements

To understand the NFPs of services, such as provenance awareness, whose formal specification is the primary focus of this research, we will introduce this chapter by discussing software system requirements, often categorised as functional or non-functional. This categorization is an important discussion as to understand the needs for NFPs of composite services to satisfy NFRs. NFRs describe what the users or system require, while NFPs take the form of measurable quality attributes, which should meet the above requirements.

In general, software requirement specifications address the description of the required functional attributes and performance characteristics of a component (or service module) without taking into consideration its realisation, to facilitate understandability [131]. Their necessity results from imminent dangers a component’s design might involve, such as unexpected complexity due to design
limitations and severe maintenance issues [131]. The term “requirement” can be considered simply as an abstract, high-level description of the services that a system should provide, or the constraints on it called “user requirements” [139]. However, from a technical point of view the term “system requirements,” [139] is used to indicate a more detailed description of the system’s functionality and service’s operational semantics, bringing up the notion of functional requirements.

2.2.1 Functional Requirements

In a software system (including service or component based systems) functional requirements are defined as the system’s possible functionality determined by the software type and the system users. Thus, we can consider them either as user requirements, described in an abstract way that can be identified by system users or as system requirements, which are accurate and detailed descriptions of system functions, services and operations. At this point we adopt the following definition as proposed in [139] to describe functional requirements of a system:

**Definition 1 Functional requirements:** “Statements of services the system should provide, how the system should react to particular inputs, and how the system should behave in specific situations. In some cases, the functional requirements may also explicitly state what the system should not do.”

Therefore, the emphasis is given either on functions that a system must be able to perform or on behavioural aspects of a system that specify the inputs, outputs (responses) to the system, and the behavioural relationships between them. Behavioural aspects mainly refer to the system’s interaction with its environment.

Among different definitions of functional requirements in the literature considering behaviour, Glinz suggests in [59] timing requirements. Those requirements can be considered as behavioural but not functional according to Glinz. In most works of the literature timing requirements are considered as performance requirements, which in turn are classified as non-functional requirements (NFRs). This classification is quite an important statement for our research where we are
interested in modelling NFPs such as provenance. In the next section, we give a definition of NFRs, distinguishing those from the functional aspects presented above.

### 2.2.2 Non-Functional Requirements

In this section we discuss NFRs through presenting existing taxonomies and classifications, based on which we later identify NFPs that need to be specified in service-based systems.

The concept of quality is fundamental for software engineering. Thus, both functional and non-functional properties should be taken into consideration during the development of a software system. The formal definition of NFRs is a rather demanding and complex task, as there is a great diversity of NFR types that present at different phases of the development of a software system. They may be related to design solutions or may be determined at development-time or run-time. In this section, we describe NFRs by comparing various definitions presented in the literature.

Ian Sommerville in [139] gives a definition of NFRs, where they are defined mainly as constraints over functionality:

**Definition 2 Non-functional requirements:** “They are constraints on the services or functions offered by the system. They include timing constraints, restrictions on the development process, and limitations imposed by standards. Non-functional requirements often apply to the system as a whole, rather than individual system features.”

As Glinz [59] suggests mostly all definitions in the bibliography are composed of the following subterms, where there is no clear distinction of their meaning including property or characteristic, attribute, quality, constraint and performance. The terms property and characteristic are mostly used with a general sense, to
denote an asset that the system should have, a particular quality property excluding the functional ones [59]. Furthermore, in his survey Glinz argues about constraints being properties, considering research works where constraints are excluded from the NFRs definitions [5], versus others that include both terms [79]. The subterm attribute is used both with a narrow and a broad meaning [59]. The standard IEEE 830-1998 [76] narrows down its meaning by defining attributes as a group of quality properties (e.g., security) excluding performance. On the other hand, the term quality [59] narrows down to a group of specific quality types excluding any functionality.

NFRs can be considered as constraints, restricting the space of possible solutions to the ones that satisfy non-functional requirements. Yet, different definitions are given about the restrictions a constraint may include (e.g., design constraints [76], physical constraints [79], operational constraints [130]). Finally, the subterm performance is defined in [76] as an explicit category from other properties. Glinz reviews all the above definitions in [59] and gives his explanation on NFRs as: “attributes of or as constraints on a system”, that we adopt as a general term for NFRs.

Chung et al. [28] give their definition about NFRs that “constitute the justification of design decisions and constrain the way in which the required functionality may be realised”. Another work where the term of design is taken into consideration for the definition of NFRs is this of Malan and Bredemeyer [96]. In [96] they consider NFRs from both a design and developer point of view via making a useful distinction between run-time and development-time qualities. Development-time qualities are related to architecture design. Run-time qualities are related to the system execution, focusing on user goals and the operating environment. This distinction between run-time and development-time qualities has significant implications on how NFRs are defined, and as a result on how NFPs are specified, as there are trade-offs between run-time and development-time qualities that need to be considered. Hence, this distinction raises the necessity of defining factors, which influence the specification of NFPs during the design and execution phase.
to be taken into consideration.

In summary, different definitions were presented among various pieces of work that result in different sorts of classifications of NFRs. In the next section, we discuss the ones most relevant to our research.

Classification of Non-Functional Requirements

In this section we make a selection among several taxonomies and classifications of NFRs that have been proposed in the literature, to discuss the more relevant ones for our research approach. This discussion will clarify the role of the different types of existing NFRs and the scope of different NFRs. Furthermore, we identify quality metrics that are attached to various quality dimensions with respect to the service-oriented vision.

To begin with, in [131] Roman proposes one of the earliest NFR taxonomies for software systems found in the literature, composed of different types of constraints, that introduces concisely the fundamental quality dimensions: interface constraints (e.g., interaction with the environment), performance constraints (e.g., timing requirements, reliability, security), operating constraints (e.g., physical constraints, personnel availability, environmental conditions), life-cycle, economic and political constraints (e.g., policy and legal issues).

Sommerville’s NFR’s classification [139] is using as its fundamental criterion the actual source of requirements, such as the software product, the organisation developing the software, and external sources as well. This classification is an interesting piece of work for our research, as it categorises the NFRs according to the roles that affect them in the software process, therefore it clarifies which quality dimensions arise from the product and the software system’s development, and which ones are related to the user. Product requirement’s role is to restrict (constrain) the behaviour of the software or system; therefore it is all about requirements including performance, reliability, security and usability. This category presents similarities with performance constraints in Roman’s taxonomy, as
presented in [131], but also contains dependability requirements, which are separated from the actual performance dimension. On the other hand, Sommerville’s classification also contains organisational requirements that correspond more or less to interface, life-cycle, and operating constraints of Roman’s taxonomy, and external requirements, which cover all aspects derived from external factors of the system and are similar to political constraints in Roman’s taxonomy. Another concern pointed out by Sommerville [139] is the lack of specified metrics for most of the attributes corresponding to the respective NFRs. NFRs also constitutes the primary concern for the specification of NFPs. We consider NFPs as quality properties with specific metrics for measuring their corresponding values.

Finally, there exist a number of hierarchical models (Boehm’s model [19], McCall’s model [99], FURPS model [61], ISO9126 [78], ATAM evaluations [11], Dromey’s model [39]), that model software quality in a hierarchical and structured manner by using a hierarchy of factors, subfactors and criteria structured from higher to lower level. The subfactors are further described by relevant quality attributes that can be measured by specified quality metrics in the lowest levels of their hierarchy. Among the already mentioned hierarchical models, ATAM evaluations [11] provide a classification scheme of NFRs in the area of architectural design, where a clear distinction is made between runtime qualities (e.g., availability, performance) and development time qualities. This distinction has important implications on how NFRs are specified; therefore we need to be clear about which of the two are in the scope of our research towards our goal for modelling NFPs for services. In this research, we focus on runtime qualities.

2.2.3 Non-functional Properties

In the previous section we presented various definitions and compared a number of classifications related to NFRs. In our research study, we take into account those classifications to help us to be explicit about what kind of NFPs we address in this research. We consider NFPs as quantitative quality attributes specified as constraints over quality values, as measured by specified metrics. As the
The perspective of this research is to propose a formal specification of provenance awareness as an NFP for service compositions, in this section, we discuss how existing NFPs are defined and measured for services and what quality aspects are related to SOA and service-oriented computing systems.

Firstly, we give a general definition of QoS characteristics, as presented in [77], where QoS characteristics are used to represent QoS aspects of a system, a service or a resource, that can be identified and quantified. In this research, we should consider those quantitative characteristics as an alternative way to name NFPs. In [77], QoS characteristics are grouped as follows: time-related, coherence characteristics, capacity-related, integrity-related, safety-related, security-related and reliability-related. Another similar categorization is presented in the UML QoS-Profile [66] which provides a narrowed set of common QoS categories, as it subsumes time-related and capacity-related properties as performance-related properties. However, a main drawback of the UML QoS-Profile is that it does not explicitly apply on SOAs.

NFPs bring up the notion of “metrics”. In [133] a taxonomy is presented for specifying NFPs of components, where metrics are used to specify security (policies and mechanisms related to data security) and performance (by defining performance parameters such as timeliness, precision, and accuracy).

In [114] a definition of NFPs is given under the service-oriented vision, where the former are considered to be constraints over the functionality of a service. In this work services may contain other services, introducing the notions of service composition and aggregation. O’Sullivan’s approach in [114], is extended in [115], with a set of proper definitions of NFPs for web services. Yet, those approaches do not cope with NFPs specification for composite services.

Ran in [127], introduces a QoS model for Web services that provides among others run-time related attributes involving scalability, capacity and performance (e.g., response time, latency and throughput), reliability (containing the charac-
The characteristics include mean time between failure (MTBF), mean time to failure (MTTF) and mean time to transition (MTTT) that show a strong connection to availability, robustness, and accuracy. An extended version of these metrics for reliability and availability is presented in [139]. Another work is that of Zeng et al. [154], which presents a framework for QoS service composition where they introduce reputation property.

We have mentioned a number of works on the definition of NFPs for services and service composition. Although composed services require specific mathematical functions to calculate NFPs, they are based on the same parameters the single service NFPs need [27]. Important work is that of Rosenberg et al. [132] where they introduce a framework for the specification of NFPs. This work consists of the only work to our knowledge that specifies NFPs for atomic web services, by providing specific mathematical formulas for the metrics of the measurable quality attributes it identifies. More specifically, Rosenberg et al. identify four main QoS groups, namely: Performance, Dependability, Security and Trust, and Cost and Payment with their relevant attributes as shown in Figure 2.2. Next, we present a description of those properties along with an example of their metrics as those were presented in [132].

![Figure 2.2: Service Layer QoS Taxonomy [132]](image)
Performance  The first quality category, performance, is related to a group of observable and measurable QoS attributes related to the runtime performance of a service. More specifically the quality attributes that fall in this category are processing time, execution time, latency, response time, round-trip time, throughput and scalability. We give an example of the metric processing time as this was defined in [132]:

- **Processing Time:** Given a service $S$ and an operation $o$, the processing time $t_p(S, o)$ of a service defines the time needed to actually carry out the operation for a specific request $R$. The value is calculated by using the timestamps $t_{p1}$ and $t_{p2}$ taken before and after the processing phase. The processing of the operation $o$ does not include any network communication time. This value is affected by the service implementation and the corresponding operation $o$. The processing time $t_p(S, o)$ is calculated as:

\[
t_p(S, o) = t_{p2} - t_{p1}
\]

Similar metrics for performance are defined in [132] including execution time, latency, response time, wrapping time, round trip time, throughput and scalability.

Dependability  Another quality dimension as presented and described in [132] is dependability. Dependability takes into account attributes such as availability, accuracy, robustness and reliable messaging. Sommerville’s approach on dependability as presented in [139], is composed of the dimensions of availability, reliability, safety and security. In [132] they enhance the dimension of dependability with the reliability attribute. Metrics that are used to measure reliability mainly focus on system failures and relate to fault, error, and failure notions. Namely, two kinds of metrics are used by Sommerville [139] to specify reliability: Probability of Failure on Demand (POFOD) and Rate of Occurrence of Failures (ROCOF) or instead (MTTF).
**Security** Security and trust quality dimensions are important to enable discovery of trusted and secure services. Currently, a number of security solutions consist of establishing a VPN (Virtual Private Network) among the providers and consumers or using SSL (Secure Socket Layer) as transport-level security. The security attribute defines the security mechanisms that can be supported by a service (e.g., security mechanisms for authentication, accessibility, confidentiality, integrity). Regarding trust, there is no existing formal specification for this but mainly trust refers to measuring the reputation of the service, as this is rated by service consumers [132]. With regards to the research presented in this thesis, we explore a research path on the use of provenance to evaluate the reputation of services based on the provenance of users’ feedback [155].

**Cost and Payment** Cost and payment represent QoS attributes that are related to monetary costs when invoking and using services. Cost refers mainly to the monetary value that is associated with a particular service operation when invoking it, while there is also the notion of penalty in case the provider of a service violates the negotiated QoS values.

In this section we have presented a definition for NFPs of services. Services have associated NFPs such as performance (measured by metrics such as response time and throughput), reliability and security. The precise specification of NFPs is necessary so that designers can ensure that their services will meet non-functional requirements at run-time. Modelling languages for expressing NFPs of atomic services, and accompanying approaches to predict NFPs of composite services, have been developed. At design time, the purpose of formal modelling and prediction of NFPs is to compare different design alternatives and to analyse whether those alternatives meet the NFRs. We present and compare existing specification languages for NFPs of services and service compositions in the next section.
2.2.4 Specification Languages for NFPs

In the previous section we discussed how NFPs are defined through presenting several classifications and taxonomies for them. As our primary focus is to form a specification language that will enable to design provenance awareness as a design NFP of services (atomic and composite ones), in this section we discuss existing specification languages for modelling services and service compositions.

We have already discussed that to select the appropriate services to be composed we need to provide information for both their functional and non-functional properties. In particular for specifying QoS guarantees, we need to introduce the notion of contract, as a formal agreement between collaborating entities that use web services and their compositions. Therefore, most specification languages for services provide a comprehensive contractual description of them. A comparison of how prominent languages can be used for the specification of different contract types has been presented in [142], including Web Services Description Language (WSDL) [107], Universal Description, Discovery and Integration (UDDI) [17], Business Process Language for Web Services (BPEL4WS) [4], Web Services Choreography Description Language (WS-CDL) [81], Web Services Policy Framework (WS-Policy) [146], Web Service Level Agreement and Web Service Offerings Language (WSLA and WSOL) [82, 143, 144], Web Service Modelling Ontology (WSMO) [140] and Web Service Modelling Language (WSML) [92], and OWL-S [98]. The results of this comparison as shown in Table 2.1 indicate that these languages enable mostly the specification of only particular types of contracts without considering the dependencies between different NFPs.

More specifically the contracts discussed in [142] are classified into three categories, namely: functional, quality and infrastructure contracts. Functional contracts mainly describe the functional characteristics of the operation of a (web) service, including the syntax that is determined by information such as names, data types, input and output parameters, the behaviour that captures semantic aspects of the service execution and functional constraints (e.g., pre-conditions,
Table 2.1: Contract Specification Languages for Services [142]

<table>
<thead>
<tr>
<th>Contract Category Type</th>
<th>Functional-Identity</th>
<th>Quality</th>
<th>Infrastructure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WSDL</td>
<td>+</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>BPEL4WS</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>WS-CDL</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>WS-Policy</td>
<td>+</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>WSLA</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>WSOL</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>OWL-S</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

post-conditions), synchronization which describes synchronization dependencies (e.g., sequence, parallelism), and composition that describes how (web) services participate in one or more service compositions and the flow of messages that are exchanged between them. Quality contracts are important to explicitly and formally specify NFPs, allowing to differentiate between services providing the same functionality. Those contracts refer to QoS (e.g., performance, reliability, availability) and pricing. Lastly, infrastructure contracts specify underlying infrastructure technologies such as communication contracts (e.g., message protocols), security (e.g., authentication, accessibility, confidentiality, integrity, non-reputation) and management (e.g., entities performing monitoring, metering, accounting and control of constraints). In the remainder of this section, we discuss on specification languages of services and their properties as presented in Table 2.1, along with other existing approaches for the specification of service compositions. In particular we try to identify their gaps and limitations regarding
the formal specification of NFPs for service compositions.

Web Services Description Language (WSDL) [107] is the standard language for describing the functionality of web services. It enables specification of web service identity, syntactical and communication aspects as shown in Table 2.1. However, WSDL does not support specification of quality contracts. UDDI [17] is used along with WSDL to define a set of NFPs for service providers (e.g., address, email and phone number) identified by customers in the network. Yet, it defines metadata information rather than measurable NFPs. Regarding service composition, WSDL is extended by BPEL and WS-CDL. Business Process Language for Web Services (BPEL4WS) [4] is used for the description of web service orchestrations and supports specification of composition and synchronisation aspects (Table 2.1). Web Services Choreography Description Language (WS-CDL) supports web service choreographies by describing mainly synchronisation and behavioural contracts (Table 2.1). However, both BPEL4WS and WS-CDL do not contain concepts for specifying NFPs.

Another extension of WSDL related to policies and security issues for services is the Web Services Policy Framework (WS-Policy) [146]. This does not contain concepts to specify metrics for NFPs. It comprises a general framework where the details of the specification of particular categories of policies are defined in specialised extensions of the public policy. Currently, WS-Policy specification focuses on security issues using a WS-Security Policy extension (e.g., message authentication, signature and encryption) and reliable messaging using WS-RM Policy, whereas other QoS related policies are currently not available. WS-Security deals with message authentication, integrity, and privacy by leveraging other existing protocols (e.g., specifications though XML Signature and XML Encryption [123]). WS-Policy also supports WS-Policy Assertions for functional aspects specification and monitoring policies through WS-CoL [9]. Yet, those are not the primary focus of this research.
The Web Service Offerings Language (WSOL) [143,144] is an XML-based specification, compatible with the WSDL, that enables formal specification of service offerings. A service offering is a formal representation of a class of service (a discrete variation of the complete service and QoS) for a web service. Combinations of various constraints determine the service offerings. Thus, WSOL enables formal specification of both functional and non-functional constraints (e.g., performance, reliability and availability) as well as access rights, pricing and relationships between service offerings. The advantage of WSMO and service offerings is the support of selecting appropriate services for particular circumstances, which enables the process of dynamic service composition and adaptation. However, WSMO can not address issues for modelling NFPs of composite services.

The Web Service Level Agreement Language (WSLA) [82] is designed to support service level agreements (SLAs) in a formal way to enable automatic configuration of both the implementation system of service providers and the supervision system that deals with guarantees of the agreed QoS. It also enables measures to be taken (action guarantees) in the case of failure to meet the service guarantees. To facilitate these actions, WSLA is based on a detailed specification of the service level parameters (SLA parameters) using specific metrics. SLA parameters are properties of a service object, each having a name, type and unit. Examples of SLA parameters are service availability, throughput and response time. WSLA also comprises a description of involved parties, their interfaces and can represent their obligations. The language is extensible and allows to derive new domain-specific or technology-specific elements from existing language elements. Yet, it does not support formalisation of NFPs for composite services. Regarding SLA parameters, those are used in many areas to formally or semi-formally describe the NFPs that an individual service should provide. However, SLAs only specify the properties required for a service as a whole. The properties of atomic services and how these compose to provide certain properties for a composite service system are out of SLAs’ scope.
Another language that provides a formal basis for comparison of SLAs is SLang [90], modelled in UML, adopts the abstract syntax of UML. The semantics of SLang are defined by the constraints imposed on a behavioural model by the presence of SLA parameters. SLAs here capture the mutual responsibilities of the service provider and its client on NFPs. Although, SLang seems to be the only formal specification language that takes a generic approach to the specification of NFPs for web services, yet it does not manage specification of NFPs for service compositions.

Web Service Modelling Ontology (WSMO) [140] does not provide alone a formal approach for modelling NFPs of a service but rather recommends a set of NFPs for each particular element of a service description. In combination with the Web Service Modelling Language (WSML) [92], the assignment of NFP values is possible onto WSMO elements. For modelling NFPs in WSMO/WSML, the proposed solution is to define ontologies for specifying non-functional domain properties that can be instantiated and used in service descriptions. However, NFPs as defined by WSML rather comprise metadata information rather than formal specifications of NFPs for atomic and composite services.

OWL-S as presented in [98] enables the specification of syntactic, behavioural, synchronisation, and some types of compositional contracts. It contains extensions (called placeholders) for the specification of other service properties, such as QoS, price, and, security. However, those placeholders are quite general and cannot be considered as formal specifications for NFPs.

As far as it concerns semantic-oriented approaches, there are a number of semantic ontologies for describing web services such as the Semantic Web Service Ontology (SWSO) [75]. Also, there are various QoS ontologies to express QoS information about services, their NFPs and their corresponding metrics e.g., OWL-Q [88], QoS-MO [55], WSMO-QoS [148], QoSOnt [38] and DAML-QoS ontology [161]. QoS ontologies are mainly used for services and service properties description to create a common vocabulary for them. Therefore, their concepts
and any QoS factors they present can be useful for formalising provenance awareness as an NFP.

Having presented the state of the art on specification languages for services and service composition, we now showcase our own comparison as shown in Table 2.2, by defining specific criteria, relevant to our approach towards a formal specification for NFPs and provenance awareness in particular. We present limitations and support of existing specification languages for services, by defining the criteria of genericity, formality and service composition support. With the term genericity, we refer to the ability of a language to specify all the relevant concepts and factors that influence NFPs and their metrics, to give the opportunity to a service designer to determine its NFPs and their metrics. Also, the notion of formality implies the existence of a formal semantics that defines the behaviour of a given model in a mathematical formulation. Such semantics constitute the means of the specification for model transformation algorithms that translate a given model into a behaviourally equivalent formal model, which can be analysed by existing analysis techniques. The last criterion we consider for our comparison refers to the ability of a language to support specification of service composition concepts.

Having made this comparison we conclude that there is a lack of real support for formal specification languages and methodologies for NFPs of composite services and especially for provenance awareness. Several standards have been developed that provide concepts for modelling services, business processes [4] and choreographies [81] but the focus is on the functional properties of services and their compositions. WSOL [143] and WSLA [82] are semi-formal languages that allow the specification of some NFPs, including performance, dependability, security and cost. SLang [90] is similar, but offers more formality. However, neither WSLA/WSOL nor SLang deal with NFPs of composed services. The model-driven approach of Gallotti et al. [6, 51] supports model-based analysis of service compositions, with a focus on the assessment of performance and reliability NFPs, by automatically transforming a service composition design model
into an analysis model, which then feeds a probabilistic model checker for quality prediction. Fillieri et al. [43] present a novel approach to reliability modelling and analysis of a component-based system that allows dealing with multiple failure modes and studying the error propagation among components.

While there are theories and models for some NFPs (most notably performance and reliability as shown in some examples above), a large number of other properties are less well studied. An insufficiently explored area in the NFP specification is that of provenance awareness. Notably, none of the languages described above provides a specification of provenance awareness for composed services. Therefore, we identify a gap regarding the specification of provenance awareness for services, and we propose a formal specification and analysis framework for provenance awareness of service compositions in Chapter 5.
In this section we discussed related work and state of the art on specification languages for services and service composition. More specifically we tried to identify how NFPs are specified in those and we have identified a gap regarding formal specification of provenance awareness as an NFP. Next, we try to define provenance as a property for atomic and composite services, and we present related work of provenance in service-oriented systems and other distributed-based system architectures such as the Cloud Computing paradigm. We also discuss related work on provenance data models and provenance query languages to compare with some of the fundamental components and contributions to the proposed research in Chapter 4.

2.3 An introduction to Provenance

Provenance, the origin or source of something, has become an important concern for several research communities, including computer and other sciences that are conducting research requiring management of experimental processes of complex scientific workflow systems, since it offers the means to verify data products or data results, to infer their quality, to analyse the processes that led to them, and to decide whether they can be trusted [108]. In fact, provenance can be considered as an intrinsic property of data, which enhances the data value and quality when accurately captured and recorded [108] and has potential benefits on a number of contexts from e-science [35], [34], [136], [83], [56], [103], [10], electronic data and records [110], [102], curated databases and the Semantic Web [160], [70], [60], [50] to food industry and artwork ownership [108].

In this section we first present a set of definitions for provenance starting from more generic and moving towards definitions within a specialized contexts, second we present the drivers for capturing provenance data under these contexts and third we focus on how provenance can enable service providers to support accountability with regards to their services and service compositions by specifying and analysing provenance awareness as an explicit non-functional property.
2.3.1 Provenance Definitions

Provenance – which etymologically originates from the french verb ‘provenir’ meaning to come forth/originate – is defined as 1) the fact of coming from some particular source or quarter; origin, derivation, 2) the history or pedigree of an object, a record of the ultimate derivation and passage of an item through its various owners (e.g., looking into the history of ownership of a valued object) [108]. It therefore encapsulates both the concept of source or derivation of an object but also may refer to a record of such a derivation.

Process Provenance  In the context of computer science, data objects (referred to as data items or data products) may be the result of computational activities carried out through executing programs that may be compositional by nature, being the result of sophisticated compositions (executing sequentially, in parallel, conditionally) of simpler computations that take input data/configurations and produce output data. In this context the definition of “process provenance” \(^1\) has been introduced to refer to an instance of an execution, or a running program; in other words a computation. Therefore, if there exists a recorded description of the past process(es) that resulted in a data item/data result, then it is possible to be explained how this data was derived by capturing various data products during execution that will enable useful analysis and reasoning.

In the context of computer science process provenance can be considered as technology-agnostic. It could apply to a record associated to a file system on the Web, a record file that may have been the result of a Java program execution or a scientific workflow system, may rely on data stored in a database, or that may have been derived from as a result of mashing up services across the Internet.

\(^1\)Provenance process is defined as the process that led to a piece of data.
**Provenance as a Directed Acyclic Graph**  Another approach to define and represent provenance is expressing this through a directed acyclic graph (DAG), explaining how a data product or event came to be produced in an execution [108]. In such a definition we assume that nodes represent data items and edges data derivations and therefore we can showcase how a data result/object was derived from another or a combination of those in a directed acyclic graph structure.

**Why-Provenance**  Initially defined in the context of databases and data lineage, why-provenance identify tuples whose presence justifies a query result [108]. In other words why provenance is considering the information records obtained by computing the transitive closure of plain edges, representing data derivations.

**Where-Provenance**  Such a notion of provenance, referred to as where provenance, was initially defined in the context of databases; it helps to identify where information was copied from [108].

**How-Provenance**  Why-provenance states which source tuples witness the existence of a result, but it does not explain how they are involved in the creation of this result, i.e. how their involvement proves the result. How-provenance consists of a polynomial representation that hints at the structure of the proof explaining how an output tuple is derived. How-provenance was first defined in the context of relational algebra and recursive datalog [108].

**Provenance as Annotations**  There exist ontologies to provide structure and semantics to metadata of resources where that semantic metadata are expressed as annotations. Aspects of these ontologies are provenance related, such as author, creation date, and version [108].

**Provenance as Query over Process Assertions**  PASOA approach [65] makes the distinction between assertions about a process and the provenance obtained by a query over process assertions. In PASOA approach the query result
is a reconciliation and composition of assertions according to the flow of information over a provenance graph capturing information sourced from distributed systems or applications (such as mashups).

Considering those different provenance definitions we realize that the nature of the data derivation, or data source/history, may take various forms, or may emphasize different properties (or assertions) according to the user’s interest. Next we are going to focus on the usability of provenance across different contexts in science and industry, its form and drivers for which provenance has become a significant concern in those contexts.

2.3.2 Provenance Drivers and Usability

In the context of e-sciences and distributed computing provenance has been defined as the process that produced/led to a data item [105], defined by properties related to the process itself, the timing of the process and the input/derived output data to the process - which is related to the definition of process provenance as described in the previous section. Researchers are becoming providers of data products, which could either take the form of raw data sets from sensors, instruments, data results produced by scientific workflow-based system computations, or databases being curated by data domain specialists. In all those cases provenance is used to provide an equivalent of the logbook for capturing the steps that were involved in the actual derivation of a result. For example, in scientific workflows, it is used to execute a scientific experiment that led to that result so as to validate and assert the trustworthiness of the evaluation methodology being used.

In curated databases the problem of attribution (who initially created a description), raises the need for questions about the source, or provenance, of such descriptions (where were the descriptions initially published, who is their owner, in which version those belong) [108]. This represents a classic data lineage problem, where data lineage is considered as a subset of full-provenance.
In the Web, information sharing, discovery or aggregation occur in an unprecedented manner. Therefore, identifying reliably the source that produced a data item on the Web, becomes quite difficult [108]. Without knowing the provenance of information, information services may not be able to undertake the necessary due diligence about their content, they may be the subject of fraud or spam, and overall they may be judged as unreliable. Provenance is in fact identified as one of the many salient factors that affect how users determine trust in content provided by web information sources [108]. The shift towards highly distributed systems and dynamic service compositions is placing emphasis on both the provenance of data and additional metadata to be captured that will enable provision of certain levels of assurance and trust with regards to the source and derivation of data and information used and exchanged during their execution.

The Semantic Web sets an even more promising context for provenance to get active since it envisions a Web of information and knowledge processable by computer systems which undertake automated reasoning [108]. Central to this effort are the Resource Description Framework (RDF) [91] and the Web Ontology Language OWL [73] that enable the expression of metadata. The vision is being deployed through Linked Data [72], an information space in which data is enriched by typed links expressed in RDF, cross-referencing data sets in a machine-processable fashion. Given the possibility for anybody or any system to publish sets of Linked Data that refer to others, reasoners will need explicit representations of provenance information to make trust judgements about the information they use.

The issue of provenance is not limited to data, information or knowledge. It also applies to physical artefacts, e.g., the food industry and understanding the provenance of food – its origin, how it is produced, transported, delivered to consumers – allows showcasing its quality. Furthermore, governments and associated regulatory authorities are looking into food safety, which requires food traceability mechanisms in place. Finally, in the context of art, the provenance of art objects is so important that available evidence is produced to showcase
ownership history of artwork.

The drivers for provenance are not limited to scientific research focused scenarios. Different businesses look into provenance as a means to provide accountability with regards to the data quality and data governance procedures (e.g., financial institutions and shipping management companies in request of regulatory reports), auditing of decisions made and practices followed as part of a treatment process in healthcare systems [145] and clinical trial management [32], or retailer companies looking into the provenance of products source, transportation and delivery to the consumers.

In summary, the usability of provenance covers a number of aspects such as:

1 determining ownership /rights over an object,
2 inferring reliability and quality of services (security, fault detection,
3 verifying that a process and its results can be trusted2
4 making systems auditable, being used to perform compliance checks (such as conformance to a process, checking that terms of data licensing are met),
5 raising issues related to data quality and privacy.

For the scope of this research we are interested in provenance information used to make service-oriented systems and compositions of services accountable for answering questions about how they have processed and produced data. Next, we define provenance regarding information and system accountability, introducing the notion of provenance-aware systems.

2.3.3 Provenance Awareness and Accountability

Accountability suggests the requirement for approaches where systems faithfully document their execution being able to answer questions about how the

\footnote{For verification of a trusted procedure, provenance is regarded as the equivalent of logbook, capturing all the steps that were involved in the actual derivation of a result.}
data was produced and processed during the system’s execution. Accountability can be accomplished through explicitly representing provenance. This historical type of metadata describing the system’s processing during its execution cycle defined as data provenance, consists of an explicit representation of past execution processes which allows full traceability of the origins of data, actions taken, decisions made and routes followed at system’s run-time. Provenance, therefore, provides the necessary logs to reason about the systems’ accountability, to derive trust in results and systems and to allow auditors to determine whether SLAs and policies are satisfied through capturing and reasoning over a full set of auditing trails that determine systems as provenance-aware.

Regarding provenance awareness, the research community has gained a relatively good understanding of how to make a single application provenance-aware [104]. A provenance-aware application can track the provenance of its data and allows for such provenance to be queried. In particular, provenance-aware applications create process assertions and store them in a provenance store, the role of which is to offer a long-term persistent, secure storage of process assertions [104], [111].

Provenance-aware systems differ from logging systems. Logging is focused on informing the system designers, while provenance is structured to answer user questions representing a set of user provenance awareness requirements [138], [63]. Therefore, the infrastructure required for provenance awareness is driven by different requirements to logging infrastructures. Second, logging concerns the operation of a single system (and whether that operation is correct/optimal), while provenance is cross-system. For example, if a service receives an external input, its provenance can be combined with service operation provenance records to answer a richer set of questions. The implications of this are that provenance data must follow a standardised queryable structure (e.g., PROV-O [14]) that allows cross-system linking (through the use of global resource identifiers and standards like RDF and OWL representation language), whereas logging can be system-specific, and often intended to be read/processed by people (designers)
rather than to be systematically and mechanically queried.

Finally, logging tends to record what happens at each system locally, whereas with provenance the location of execution often doesn’t matter and so it is abstracted away. More generally, provenance tends to describe operations at a higher level of abstraction than logs do. Yet, Ghoshal in [54] presents a study where he explores the option of deriving provenance information from existing log files under certain circumstances. However, in this work the provenance gathered may not necessarily exhibit completeness and provenance records captured in an ad-hoc way do not allow support of provenance awareness for service compositions. In our next section, we present the benefits and importance of being able to formally design and represent provenance awareness and define this explicitly as an NFP for atomic services and service compositions.

### 2.3.4 Provenance Awareness as an NFP

Provenance is becoming increasingly more important for on-line services and distributed systems where complexity has increased significantly. Service composition, constituting a workflow that invokes and aggregates services, is a particular kind of data item for which the process of derivation needs to be tracked. The need to provide support for answering questions about how they have processed and produced data is particularly evident in compositions of services, where audits of each service’s use do not provide a connected picture of the composition’s processing history.

Our intention is to analyse at design-time that service-oriented systems, realised through individual and composite services, are provenance-aware. We define provenance awareness as the ability of a service-oriented system to answer different types of queries about the documented history of a service’s or service composition’s processing. We aimed to specify provenance awareness as a non-functional property (NFPs), by identifying the requirements in storage, recording and monitoring mechanisms we need to include in our system design in
order to ensure that users of the system will be able to get answers to particular provenance question/query types; by mainly analysing and verifying the system’s provenance awareness support.

We also aim to analyse at design-time in what ways a service-oriented system is provenance-aware. Services may be compositions of other services, the latter being referred to as atomic services, and the questions may relate to how the composition has occurred. The provenance questions that a user may ask are categorised using a faceted classification approach, by introducing a set of provenance facets, where each facet corresponds to particular kinds of information being recorded in the provenance, and so particular sets of provenance questions being answerable.

Next, we discuss existing provenance data models and querying languages for provenance in the literature.

### 2.4 Provenance Data Models

Part of our research contributions is a provenance data model for expressing and capturing the provenance data structures for service-oriented (computing) systems and their compositions. Next, we present a number of provenance data models existing in the provenance literature.

**OPM Model and OPMW Ontology** In [58] Garijo and Gil extend OPM model [109] to capture the execution traces of a workflow template (process view provenance) along with the metadata of the template and execution itself (attribute provenance). This requirement was motivated by the goal of publishing workflows of scientific articles and their results, maintaining the OPMW [52] ontology quite simple in its conceptualization capability, suitable only for describing the provenance of workflows. Similarly to this work, we are interested into representing, at design-time, the structure of provenance information for the service composition’s outputs that may be recorded and stored during its exe-
cution, going one step further by considering the aspects of infrastructure and the SOC principles and processes such as the service discovery, orchestration or choreography.

**PROV-DM** The PROV Data Model [13, 57] is the W3C standard formed to represent the structure of provenance data. PROV-DM [13] is a generic data model for provenance that allows domain and application specific representations of provenance to be translated into such a data model and interchanged between systems. Heterogeneous systems can export their native provenance into such a core data model, and applications that need to make sense of provenance can then import it, process it, and reason over it. PROV-DM is a domain-agnostic model, but with clear extensibility points allowing further domain-specific and application-specific extensions to be defined.

The primary concepts of PROV-DM are entities, activities and agents.

- **Activities** represent processes that have occurred, such as the execution of a service operation. An activity is something that occurs over a period of time and acts upon or with entities; it may include consuming, processing, transforming, modifying, relocating, using, or generating entities [57].

- **Entities** are digital, physical or conceptual things that existed with some fixed aspects, such as the messages exchanged between services or part of a service’s state [57]. Activities generated new entities and used existing entities, and one entity may have been derived from another.

- An agent denotes something that was responsible for an activity having taken place, such as a service composition’s aggregator or orchestrator, where we say that the agent was associated with an activity it was responsible for. An agent is therefore something that bears some form of responsibility for an activity taking place, for the existence of an entity, or for another agent’s activity. [57].

85
• PROV-DM also allows us to express the role played by an entity or agent in an activity \(^3\), the time at which an entity was generated or used by an activity, the plan that was followed by an activity in execution, and much more.

**PROV Ontology (PROV-O)** PROV-O [14] defines the OWL2 [73] Web Ontology Language encoding of the PROV Data Model (PROV-DM) [13]. It describes a set of classes, properties, and restrictions that constitute the PROV Ontology. This ontology provides the foundation to implement provenance applications in different domains that can represent, exchange, and integrate provenance information generated in various systems (e.g., web-based applications) and under different contexts.

PROV-O is a lightweight ontology that can be adopted in a wide range of applications (e.g., service based, component based, composite). The PROV-O ontology classes and properties are defined such that they can not only be used directly to represent provenance information but also can be specialised for modelling application-specific provenance details in a variety of domains. Thus, PROV-O ontology is expected to be both directly usable in applications as well as serve as a reference model for creating domain-specific provenance ontologies and thereby facilitates interoperable provenance modelling. As part of this thesis, we specialise PROV-O to model provenance for service-oriented computing systems and compositions of services.

A number of works have been proposed to represent an abstract schema for the provenance of workflow results as discussed below. Those works though are not able to express or work as schemas for provenance data of service compositions (workflows) expressing the full set of aspects as discussed in the SOC paradigm and SOA.

\(^3\)A role is the function of an entity or agent with respect to an activity, in the context of a usage, generation, invalidation, association, start, and end.
Wf4Ever Research Object Model – RO-MODEL  The Wf4Ever Research Object Model [16] provides a vocabulary for the description of workflow-centric research objects seen as aggregations of resources relating to scientific workflows [16]. In this work Belhajjame et al. [15] propose RO-MODEL to describe the provenance of research objects aggregated as part of a workflow, with the workflow following a template 4 which describes the steps involved in the workflow’s execution.

Scientific workflows are used to describe series of activities and computations arising as part of a scientific problem-solving allowing scientists the means to specify their experimentation methodology in a structured way. From a computational viewpoint, experiments can be represented as a directed acyclic graph where the nodes correspond to analysis activities/operations and the edges specify the data flow between them. The use of workflow specifications on their own does not guarantee support of reusability, reproducibility, or a clear understanding of the semantics of the scientific methodology and experimentation introduced. Also, tools, owners and data may evolve and change. Therefore additional information including annotations to describe the operations performed by the workflow may be required, ownership and versioning information (e.g., authors, versions, citations), and links to other resources (such as the provenance of the results obtained by executing the workflow and the source/origin of datasets used as input). All those aspects may enable a comprehensive view of the scientific experimental methodology followed. Those rich annotation objects are defined as workflow-centric Research Objects (RO) – forming an aggregation of resources along with annotations on those resources themselves.

Research Objects could play the role of 1) technical objects; providing access to resources that are required to support execution and record the provenance traces of those executions, maintaining version information about the lineage/evolution

---

4A workflow template is a network in which the nodes are processes, and the edges represent data links connecting the inputs and outputs between different processes. Processes specify the software components (e.g., web services) which are responsible for undertaking a set of actions.
of those resources, 2) social objects; encapsulating the know-how and protocols followed (e.g., best practices followed that facilitate reproducibility of experiments) and information about the people involved in experimentation (actors that create, use, extend and curate the objects).

The Wf4Ever Research Object model consists of a suite of ontologies used to describe workflow-centric ROs, where the key ontologies provided are:

- **ro**: Provides basic structure for the description of aggregated resources and the annotations that are made on those resources.
- **wfdesc**: A vocabulary for the description of workflow; provides an abstraction that can be mapped to different workflow systems.
- **wfprov**: A vocabulary for the description of provenance information; provides an abstraction that can be mapped to other provenance vocabularies.

**Abstract Provenance Graphs – (APGs)** Zinn et al. [162] introduce the concept of an abstract provenance graph (APG) in a first attempt to exploit provenance information during the design process of scientific workflows. APGs are proposed as being helpful for workflows’ construction that enables spotting design bugs early in the stage of development of the scientific workflow and exhibiting the evolution of the overall data organisation of a workflow. In their approach, they specify and derive provenance information apriori to implementation based on the given workflow specification – that forms a description of the workflow input structure using XML DTDs [147] – and a set of declarative scope expressions (e.g., actor configurations).

Next, we look into existing literature about languages used for querying provenance instances conforming to provenance data models.
2.5 Provenance Querying Languages

In this section, we discuss related work on provenance query processing in existing systems. Querying provenance information is necessary for our research as we are looking into implementing a set of analytical metrics for analysing provenance awareness for service compositions at design time; therefore it is important to see how those analytical metrics can be formalised and expressed with a query language to allow reasoning over provenance data graph snippets.

**ProQL** In [80] Karvounarakis et al. propose ProQL, a query language for provenance graphs which is useful in supporting a wide variety of applications with derived data. In particular, ProQL queries can be used to assess trust and derivability or detect side effects of possible updates as well as to express more complicated provenance queries and, optionally, compute data annotations. Through [80] the authors also present a prototype implementation of ProQL over a relational database management system.

**vtPQL** VisTrails [134] captures provenance of both the workflow evolution and associated data products by a change-based mechanism and visualises the workflow evolution provenance as a version tree. VisTrails can visualise query results by highlighting workflow versions that match query conditions by using the VisTrails query language, called vtPQL [134].

**QLP** Kepler [97] implements an interactive provenance browser to visualise and query its proprietary workflow trace. Users can express complex and recursive provenance queries by the Kepler’s query language, called QLP. The QLP query language is defined on its proprietary provenance model, which explicitly supports workflow steps that process XML [123] data and employ update semantics; thus, QLP allows users to use XML data structures and XPath [29] expressions when provenance queries are formulated. In general, QLP can be useful in the situation where data is organised into nested collections like XML data, and dependencies are defined among data nodes.
SQL and XPath  Karma [137] presents an integrated provenance management architecture that supports automated data provenance collection, annotated provenance, and provenance visualisation. Karma supports provenance queries in SQL and XPath [29]. However, SQL requires the user to know the underlying storage schema as if the schema changes the SQL [26] query needs to be edited. SQL [26] is also only compatible with relational database technology which limits its use in workflow systems with non-relational storage.

GraphQL  GraphQL [71] is a graph-based query language for graph databases. GraphQL is defined over a data model representing attributes of a generic graph. A GraphQL query takes a collection of graphs as input and produces a collection of graphs using graph patterns. GraphQL is a semi-structured query language, and therefore like SQL, it requires users to directly formulate recursive queries to support provenance lineage queries.

SPARQL  SPARQL query language [74] has been used to support provenance queries by Taverna [160]. Taverna implements a semantic provenance infrastructure and visualises semantic, RDF-based provenance graphs based on the provenance ontology PROV-O. In our thesis, we pick SPARQL as the query language to express out provenance awareness requirements as this is the standard querying language for querying RDF graphs such as PROV-O or any specialisations of this.

OPQL  OPQL, as proposed in [93], is a provenance query language that enables the querying of provenance directly at the graph level for scientific workflows. OPQL queries are not tightly coupled to the underlying provenance storage strategies. OPQL is implemented as a Web service allowing users to invoke and execute OPQL queries in a provenance browser. However, OPQL implementation does not allow a designer to specify all the provenance-related properties of a service-based system and analyse those at design time, apriori to system’s implementation.
Summary: In this chapter we presented the background in service and service composition principles, specification of their NFPs and provenance in particular. We also presented background on provenance data models and languages for querying and expressing provenance requirements as those comprise critical components of our specification and analysis framework for provenance awareness of service compositions proposed in this thesis. Next, we present related work on provenance in SOA and service-based systems including systems based on the cloud-computing architecture.
Chapter 3

Related Work

3.1 Provenance for SOA and Services

The importance of provenance data for SOA has been increasingly recognised, and a number of approaches have been developed to provide a provenance framework for them. Tsai et al. [149, 150] analyse the unique characteristics of data provenance in a SOA system and provide a framework for classification and collection of provenance data. However, this work focuses on security, reliability and integrity of data as it is routed through an SOA system rather than considering the NFPs of services and their dependencies in composite service scenarios.

Michlmayr et al. [101] present an approach in which service runtime events are captured, but it is again focused on security issues such as data integrity and access control mechanisms. Rajbhandari et al. [125] propose an approach for capturing and recording provenance in SOAs, including a scalability analysis of the effects of increases in provenance data, to evaluate the performance of their recording mechanism. They extend this preliminary work in [124] with an analysis tool that makes use of provenance data to assist in evaluating the trustworthiness of workflow execution results.

Muniswamy-Reddy et al. [112, 128] define desirable properties for distributed provenance storage systems and design alternatives for storing data and prove-
nance on cloud-based web service platforms (e.g., Amazon’s Web Services platform (AWS)). They mainly propose the design and implementation of three protocols for maintaining provenance and data on the cloud, evaluating each protocol with respect to the properties they have established. Yet, their focus is mainly on properties in their design that can guarantee the availability and the scalability of the cloud, while they evaluate their approach by making a comparison of the cost and performance of the proposed three provenance storage protocols. In our work, we are also interested in considering the trade-offs of performance and storage demands with provenance awareness as an explicit NFP for service-oriented systems realised through composite services.

The research community has also been interested in methods to make applications provenance-aware. The Provenance Aware Service Oriented Architecture (PASOA) project [64,105] proposes a generic, technology and application-independent architecture that meets technical requirements of different use case scenarios. Miles et al. [104] suggest a software engineering technique for making applications provenance-aware, by adapting designs to enable their interaction with a provenance middleware layer. In [62] Groth et al. outline the user requirements that arise in designing provenance systems for the web space by focusing on three central dimensions: content, management and use. Similarly to our approach, they try to make precise these dimensions according to the requirements they imply through a number of use case scenarios. Yet, their work does not address how provenance awareness can be expressed as an NFP.

In summary, while there is a large body of work on specifying NFPs of service-based systems, this does not cover provenance awareness. At the same time, while provenance awareness of service-oriented systems has been discussed in the provenance literature, this has not been done in a way that allows provenance-awareness to be designspecified and analysed as an explicit NFP for composite service systems. With our work, we aim to address this gap.
3.2 Provenance and Trust for Cloud Services

The research community is also already aware of the importance of provenance for the Cloud based paradigm. Here we list some works and the problems those tackle.

In [1] Abbadi et al. analyse the cloud computing structure to come up with a list of challenges where provenance data collection may be beneficial for the Cloud paradigm. They particularly focus on the problems of cloud logging and auditing for the different types of resources used in the cloud (e.g., physical, virtual, application), the building of self-managed services and on trust establishment for the cloud. This work identifies the need for establishing trustworthy cloud provenance mechanisms – where by trustworthy the authors refer to the fact that both cloud users and providers can attest to provenance mechanisms ensuring that they have performed their job as expected. This work does not focus however on providing a solution that would explore the techniques and tools needed for specifying and analysing provenance awareness considering the principles of services and aspects discussed in the SOC roadmap.

In [159] Zhang et al. are emphasising the need for a more accountable and transparent cloud with regards to the data this manages. Yet, they focus on enabling provenance in supporting better integrity and safety of customer’s data. They further propose a framework for accountability and trust for the Cloud in [86, 87]. However, they make no consideration about reputation mechanisms incorporating trust as we propose for our future work perspectives as discussed in Chapter 12.

The research community has also presented a number of works on provenance and trust aspects for different application domains other than those serving the needs for cloud computing architecture, yet those are getting out of the scope of this research but are relevant with regards to our future perspectives for exploring research paths of provenance awareness with regards to to other NFPs
such as trustworthiness or service reputation. Rajbhandari et al. capture the topics of evaluation and assessing trust both for scientific workflows [126] and service-oriented applications [124]. The first illustrates how provenance information in association with a workflow can be used to evaluate trust, where the trust refers to the individual actors involved in the workflow and their past behaviours, namely *actor provenance*, while the second one mainly focuses on building an analysis tool that makes use of provenance information to assist in evaluating trust in the outcome of a workflow execution.

Bizer and Oldakowski [18] are proposing a semantic web trust architecture which considers both subjective - and task-specific policies, similarly to our idea for subjective and objective feedback as presented in Chapter 12. Finally, few works have focused on provenance-aware trust for the Semantic Web. Ding et al. [37] propose an ontology to associate trust with provenance and mechanisms to evaluate the trustworthiness of semantic association or any collection of statements obtained from multiple sources. Compared to our approach discussed in Chapter 12, as a future perspective of research on the provenance of feedback for evaluating reputation for cloud-based services, this work is applied to a different domain, the Semantic Web domain, considering no SLAs or reputation management mechanisms.

**Summary:** In this chapter we discussed in detail background on provenance definitions, existing provenance data models and querying languages for service-based and scientific workflow-based systems.
Chapter 4

Problem Statement

In the previous chapters we identified a gap regarding providing a specification language to express provenance awareness that would allow modelling and designing this as a composition’s non-functional property design. In particular, in Chapter 1 we defined the research problem of this thesis with regards to designing provenance awareness for service compositions addressing the research gap for:

1. a specification language that defines, expresses and models provenance awareness for compositions of services as an explicit non-functional property (NFP),

2. the modelling notation and tools that will enable composite service designers to incorporate provenance awareness as part of the service compositions design process, to support analysis and measurement of the system’s accountability – provenance awareness – support at system’s design time.

In this chapter we discuss the research problem as stated in Chapter 1 by exhibiting its twofold nature through our standard use case scenario as this was introduced in Chapter 1. We then discuss in detail the research questions we address through this thesis and how those decompose in a set of subquestions, by providing an explanation of how those reflect in our standard composition use case scenario. This scenario is used throughout our thesis to showcase the development of the core components of our framework and for evaluation purposes.
in Chapter 11.

Contents

<table>
<thead>
<tr>
<th>4.1</th>
<th>Use Case Scenario ........................................... 97</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.1</td>
<td>Travel Planner Composition ................................... 97</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Example Provenance Questions ................................ 100</td>
</tr>
<tr>
<td>4.2</td>
<td>Research Questions and Objectives ........................... 103</td>
</tr>
</tbody>
</table>

4.1 Use Case Scenario

Figure 1.1 depicts a service composition for booking multiple travel arrangements. That is a standard example (use case scenario) taken from the service composition literature [40] as we briefly presented in Section 1.4.

4.1.1 Travel Planner Composition

The control flow of this composition is represented as a UML activity diagram where an action corresponds to the specification of an abstract service task to be executed in which a named service operation is invoked, with some given input and output messages as shown in detail in the statechart version of the travel planner composition in Figure 4.1 – the statechart notation is presented here in addition to the control flow activity diagram following the standard notation paradigm for representing the travel planner composition use case in the existing literature (Figure 1.1). As we have already mentioned, arrows run from the start towards the end and represent the sequence of the actions occurring. In a given instantiation, each task would be performed by a particular service discovered at run-time, where the tasks here are to book a flight \((t_1)\), book a hotel \((t_2)\), book an attraction to visit \((t_3)\), calculate the driving time from the hotel to the attraction \((t_4)\), rent a bike to travel between the hotel and attraction \((t_5)\), rent a car to travel between the hotel and attraction \((t_6)\), and take payment for the travel package by credit card \((t_7)\).
Figure 4.1: Travel Planner Scenario: Operations and Messages
A user interacts with a travel planner (aggregator) which discovers, then invokes each of these atomic services, such as the flight booking or the hotel booking service, providing a complete service composition solution. In our scenario, the travel planner aggregator performs a dual role, by acting as a service provider and a service requester (as shown in Figure 4.2, [119]). As a service requester, the aggregator may request for the reservation of available atomic services from other providers. We assume that those services become visible for use through their service descriptions that become published by their service providers, and made available through a service registry. Then the corresponding services are discovered through a brokering service (Figure 4.3, [119]), which investigates the corresponding service registries and searches out the service descriptions that would best fit the composition plan.

As a service provider, the aggregator forms higher level service compositions by following an abstract business process plan – expressed through specialized composition languages such as BPEL [4] or BPMN [3]) – and by using the service descriptions and a set of QoS requirement specifications to invoke and bind the selected services together into a service composition [119]; therefore QoS specifications enable the selection of individual services among discovered services with similar functionality.

The users may have requirements on a number of non-functional properties (NFPs) such as performance, reliability, security or provenance awareness. Next, we present a set of examples with regards to provenance awareness requirements.
4.1.2 Example Provenance Questions

With regards to provenance awareness, the users may be interested to know answers to questions related to the provenance of the travel planner results and the travel planner composition execution such as:

i Why was a bike booked instead of a car?

ii Why was a high-cost flight booked?

iii Which bank managed the credit card authentication?

iv Were all the hotel services available during the hotel booking selection?

These questions can be answered based on provenance data recorded during the execution of the travel planner. To ensure the right data would be collected at runtime, designers need to plan for the provenance awareness infrastructure support required at service design time.

Provenance data required to answer questions as listed in the example Section 4.1.2 must conform to some data model or data schema. In this thesis, we adopt the W3C PROV-DM [57] (its primary concepts have been discussed in detail in Section 2.4) to represent the structure of provenance data. We later extend the PROV-DM model to provide a domain specific model for expressing data provenance of SOC systems and compositions of services.

Figure 4.4 graphically depicts a snippet of PROV data for an execution run of our travel planner. The data is in the form of a graph, where ovals are en-
Figure 4.4: Provenance Data Graph

tities, rectangles are activities, pentagons are agents, and the edges point from the future to the past denoting used, generation, derivation and association relationships. In this snippet, we exhibit two different agents: the travel planner aggregator associated with all the individual services in the composition, and a third party agent that manages the authentication of the credit card. The input and output messages (entities) correspond to the data and parameters passed from one service to another during the composition plan execution. Messages may either be generated by a service execution (activity) as output messages or may be derived from output messages (entities) of previous service executions (activities) and become input to new services. This is only an illustrative snippet of how provenance data can be expressed through a formal model such as PROV-DM and does not contain the complete set of provenance required to be captured for our scenario. Also, in this snippet both the provenance of the car and bike rental service execution is presented. Depending on the specific path taken during a specific service execution, only one of these services will be used. In our provenance model, however, we need to make provisions for collecting data from any such run.

To answer particular provenance questions, as the examples listed at the beginning of Section 4.1.2, we would have to capture the values of data such as the message contents and service/server identifiers, during service composition
execution. Different questions require different kinds and levels of detail, and it depends on what the user will or is likely to require as to what provenance data should be recorded to provide the corresponding provenance awareness for a system.

In order to ensure that the service composition we are designing has the corresponding recording and storing mechanisms to provide the required provenance data for answering different kinds of provenance questions (making the system provenance-aware), we further require to model the provenance infrastructure behaviour (recording activity) of our system throughout the different phases of the SOC execution cycle. In particular, we need to consider the fundamentals of service-oriented architecture (SOA), the service principles and the execution phases of service discovery, service selection, service planning, orchestration, etc., so that the provenance awareness support of a system can be analysed and measured in each of these phases apriori to system’s implementation.

In this thesis, we, therefore, form a problem statement of designing provenance awareness for service compositions that focuses on the following aspects:

- the need for a formal definition and a specification language for data provenance of service compositions that allows a designer to model the provenance data explicitly (structures) and provenance infrastructure activity required for capturing this information, independently of specific applications and services,

- the need to develop a suitable modelling notation that enables to express the fundamentals and principles of the service-oriented architecture (SOA) and the provenance aspects relevant to the different phases of the service-oriented computing (SOC) execution cycle (e.g., service discovery, selection, orchestration),

- the need to provide a language for expressing the provenance awareness requirements of the users, where those could take the form of patterns and act as analytical metrics for provenance awareness of a system,
the need for an overall approach and framework that will enable service designers to incorporate provenance awareness in service composition designs and enable simulation and analysis of provenance awareness support apriori to system’s implementation, at design time.

4.2 Research Questions and Objectives

In the previous section we stated our twofold research problem by discussing this based on a standard use case scenario taken from the service composition literature. In this section, we discuss the research questions we are trying to address in relation to our problem statement on providing a formal approach and modelling framework that will allow the specification and analysis of provenance awareness as a non-functional property (NFP) for the composition of services.

Having this as the core objective of our research, this thesis addresses four research main questions as introduced in Chapter 1. In this chapter, we decompose those leading research questions into a set of subquestions where required, to express the individual queries we need to address at the lowest level of granularity. We also discuss how those interrelate and explain those providing references to the use case scenario presented in the beginning of this section.

With regards to their structure, RQ 1 is decomposed into three subquestions, namely RQ 1.1, RQ 1.2, RQ 1.3, where each one of them is a research objective that supports the main argument of RQ1. RQ2 is composed of RQ2.1.1 and RQ2.1.2, that make a supportive argument to each other, and RQ2.2. RQ 3 is also decomposed into two research questions RQ3.1 and RQ3.2 that need to be addressed in a lower level of granularity so that we can answer RQ 3. Finally, RQ 4 is a separate research question that is however related to the contributions of RQ 3; RQ 4 contains no further subquestions. Their meaning and interrelations are discussed in detail in the discussion sections below.
• RQ 1 How can we express provenance awareness as a non-functional requirement of service-oriented (computing) systems and service compositions to be able to analyse a system’s accountability support?
  
  – RQ 1.1 What types of questions/queries, posed by the system users or other stakeholders, can define provenance awareness requirements of service-oriented (computing) systems and compositions of services?
  
  – RQ 1.2 What kind of categories of questions exist? How can we classify different kinds of questions into question categories to build analytical metrics for the different aspects of the system’s provenance awareness design?
  
  – RQ 1.3 How can these questions/requirements be expressed through a formal representation (e.g., formalised into a set of patterns)?

**Discussion**  
RQ 1 addresses the third aspect of our problem statement, which is about providing a definition and a way to express provenance awareness as a non-functional property (NFP) for compositions of services. Our objective is to specify formally provenance awareness in a way that will enable analysis with regards to the accountability support of a composite service system. For example, in our use case scenario, we would need to identify a specification mechanism to express provenance awareness as an NFP for the travel planner composer, enabling a service designer to analyse how accountable the travel planner composition can be with regards to the set of questions presented in Section 4.1.2.

RQ 1 encounters a set of research sub-questions (RQ 1.1, RQ 1.2, RQ 1.3) that represent narrower objectives we need to achieve to answer our leading research question. Those objectives comprise of:

  – the need to identify the types of provenance questions based on which we can define provenance awareness requirements for compositions of services (RQ 1.1) – for example, with regards to our travel planner
scenario we may be interested in questions about the routes followed in the composition workflow (e.g., *why a bike was booked instead of a car*), the availability of particular of services (e.g., *were all the hotel services available during the hotel booking selection*), or information about the agents involved in composition activities (e.g., *which bank managed the credit card authentication*),

- the need to classify different kinds of provenance questions in different (semantic) question categories in order to use those as metrics to analyse the provenance awareness support of the current system design (RQ 1.2) – for example in our travel planner scenario we may want to classify the question about the availability of services as part of a semantic category related to NFPs and QoS; in this way we can build a set of questions falling under a particular semantic category that acts as a set of metrics to analyse provenance awareness support of a system with regards to its NFPs.

- the need to formally express those query/question metrics as a set of query patterns (RQ 1.3) – for our travel planner scenario we may want to describe the question about the hotel service availability through a standard format and language and express this by making a set of specific assertions that will enable to identify answers to this particular type of question; this allows us to turn the way we ask specific questions for provenance awareness of a system into query patterns, and therefore we could then generically apply those patterns for provenance awareness analysis to other composition scenarios.

With regards to RQ 1.1 and RQ 1.2 we use a faceted classification approach for classifying questions about provenance into a set of *(semantic) provenance question categories*, exhibited by one or multiple facets; the facets are identified and correspond to particular kinds of information being recorded in provenance, and so particular aspects and sets of provenance questions being answerable. The purpose of building a faceted classifica-
tion of questions into a set of (semantic) provenance question categories is about arguing for completeness of the provenance awareness requirements of users based on the types of questions introduced and for enabling systematic derivation of more complex questions that would express additional provenance awareness requirements. Therefore, the requirement to address research question RQ 1.2 derives as a result of this condition introduced in our approach and methodology. As an example, in our travel planner scenario, a user may have a more complex question such as whether the non-availability of a car rental service was the reason for booking a bike instead of a car, that requires combining questions (i) and (iv) of Section 4.1.2.

- RQ 2 How can we design provenance awareness by providing support for specifying provenance properties such as the provenance infrastructure and provenance data structures for service-based systems and compositions of services, considering the architectural aspects and principles of the Service-Oriented Computing (SOC) paradigm and Service-oriented Architecture (SOA)? How does considering the different phases of the SOC execution cycle and the principle SOA concepts affect architectural and modelling decisions about the provenance infrastructure and provenance data structures?

  - RQ 2.1.1 What kinds of models need to be built in order to specify the structure of provenance data for service-oriented systems and their compositions, while exhibiting the provenance that could be captured through the different phases of the SOC execution cycle (identifying patterns for service discovery, service selection, orchestration/choreography or planning SOC execution phases)?

  - RQ 2.1.2 How modelling provenance for composite services (considering the orchestration or choreography composition model) is different from modelling provenance for atomic services?

  - RQ 2.2 What kinds of models and profiles need to be built to specify the source of information which is required for capturing the corre-
sponding provenance data structures of the SOC execution cycle (e.g., infrastructure, provenance recording behaviour, compositional workflow)?

As mentioned in RQ 2.2, there is a requirement for building separate models/profiles for the provenance infrastructure design of atomic services and their compositions. Modelling the infrastructure and the service parts separately is beneficial for system architects and quality designers that could now build the provenance infrastructure design independently of specific services. As a result of this separation in the design process, we need to identify the corresponding bindings (mapping rules) and binding mechanisms required between the provenance infrastructure design and the specifications of atomic services and their composition. This requirement, as a result, is reflecting on a separate research question (RQ 2.3).

- RQ 2.3 What kind of bindings and binding mechanisms do we need in place to associate/bind the provenance infrastructure modelled, with the composition’s overall design in a way that this could be analysed as an integrated and complete system design?

Discussion Our objectives for the second set of research questions (RQ 2.1 to RQ 2.3) come in place as an additional effort to support our research goal as described in RQ 1. The connecting point here is the different kinds of provenance data that need to be recorded during service composition execution, to answer the various provenance questions that the users require the system to be accountable (provenance-aware) for. To answer such provenance questions, composite service specification designers need to ensure that:

- the fundamental concepts of services, service composition (orchestration/choreography) and provenance data that need to be modelled can be specified (RQ 2.1.1, RQ 2.1.2) – for example in our travel
planner scenario we would need languages or models that allow us to express the concepts related to services/their composition (such as service provider, broker, aggregator, orchestrator, workflow) and the provenance data that need to be modelled (such as the input and output messages or service qualities like availability),

– the service-oriented system has the corresponding recording and storage mechanisms to capture the required provenance data (RQ 2.2)
– for example in our travel planner scenario we would need a language or model that allows to design the provenance infrastructure behaviour and activity required for capturing provenance data for the non-functional qualities (e.g., availability) of atomic services such as the flight booking or hotel booking services, the composition execution workflow messages derived (e.g., the travel plan booking results, tickets) and the routes followed (e.g., the rental of a bike instead of a car),

– there is a way to bind the modelled data structures with the corresponding provenance infrastructure (RQ 2.3) of a SOC system; for example in our travel planner scenario we may want to bind a BPEL or BPMN specification describing the orchestration/control flow of the travel planner respectively, and a set of WSDL specifications describing the atomic services involved in the travel planner composition, to an existing provenance infrastructure design model.

• RQ 3 What would be the method and supporting framework for simulating and analysing provenance awareness support of service-oriented (computing) systems operating across the full set of SOC cycle phases at system’s design time?

– RQ 3.1 What would be the method/framework that enables us to predict and validate through analysis whether the provenance infrastructure design will be sufficient to capture the desired provenance for
particular service implementations that are dynamically bound and composed?

– RQ 3.2 How can we simulate the provenance data captured by the design in a way that enables designers to derive conclusions about provenance awareness before the system is implemented?

Discussion While research questions RQ 1 and RQ 2 are mainly addressing the first three aspects of our problem statement, research question RQ 3 is focusing on addressing the fourth aspect, which is about providing an overall approach and framework for incorporating provenance awareness in composite service designs. The framework will enable simulation and analysis of provenance awareness support of a system at design time. Therefore, RQ 3 is questioning exactly this: How can we build an appropriate framework that will support provenance awareness simulation and analysis at system's design time? Analysing and questioning the ability of a system to support provenance awareness requires a framework where we combine a set of specifications/models to express 1) the different provenance data and infrastructure aspects and the service composition execution properties (as discussed in research objectives related to RQ 2) and the provenance requirements of users (as discussed in research objectives related to RQ 1). Then those become the fundamental components for building a modelling environment for provenance awareness by joining and integrating these pieces together in an integrated framework that allows simulation and analysis of provenance awareness support for a composite service system by applying provenance-awareness analysis checks over the simulated provenance data. Simulation and analysis checks encounter the development of an approach that will enable 1) prediction/verification of the provenance infrastructure design support being sufficient to capture the desired provenance information for particular service implementations (RQ 3.1), and 2) simulation of the provenance information that could be captured by the particular infrastructure required at design time in order to
satisfy a set of provenance awareness requirements (RQ 3.2).

For example in our travel planner scenario we would like to be able to have an integrated environment that would allow us to model all the provenance aspects for the travel planner composition at design time (including the provenance data structures, the provenance infrastructure, the services/composition workflow) and then be able to simulate the captured provenance data by the particular infrastructure design; the simulated provenance captured can then be used in order to apply analysis checks and verify the system’s provenance awareness overall support. Analysing provenance awareness in a formal and standardised way creates the need for identifying well-formed analytical metrics and forming benchmarks. We reflect this requirement as part of research question RQ 4.

- RQ 4 How can we use the proposed framework of provenance awareness to enable measurement in a standardised way and implement analytical metrics for the system’s provenance awareness support against a set of provenance awareness requirements?

Discussion With regards to RQ 4 our research objectives encounters answering the following questions: How service designers could analyse properties of their services to see whether they meet provenance awareness requirements of their users? How can we ensure that the proposed framework will form a basis for achieving this target by enabling measurement of provenance awareness in a formal and standardised way? Our objective here is to introduce a set of analytical metrics (related to RQ 1.3) for provenance awareness and integrate those as part of our framework solution to enable metric validation by comparing the modelled (predicted) provenance awareness (RQ 3.1) against the simulated/modelled provenance data measurements (RQ 3.2). The analysis results of such a framework will then provide useful feedback to composite service designers and enable them to take decisions for their system’s design enhancement and future development based on functional and other non-functional requirements.
Summary: In this chapter we analysed in detail our research questions and objectives. Next, in Chapter 5) we discuss our approach towards answering those questions and the research methodology we followed. We also present the overall architecture of our proposed framework solution.
Chapter 5

Approach and Architecture

In this RQ1 chapter we first summarize the goals and challenges from a modelling and technical perspective in our effort to address the research questions of this thesis as those were presented in Section 4.2, towards an integrated specification and analysis provenance awareness framework solution. We then present our approach, research methodology and architecture reflecting the main goal and contribution of this thesis, which is about providing a specification and analysis framework for provenance awareness of service compositions. We further list the research contributions of this thesis in detail, we associate the contributions with the proposed architecture and group them based on their significance of use for the roles involved in the system’s design work-flow.

Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Goals and Challenges</td>
<td>113</td>
</tr>
<tr>
<td>5.2</td>
<td>Approach</td>
<td>114</td>
</tr>
<tr>
<td>5.3</td>
<td>Research Methodology</td>
<td>121</td>
</tr>
<tr>
<td>5.4</td>
<td>Proposed Architecture</td>
<td>123</td>
</tr>
<tr>
<td>5.5</td>
<td>Research Contributions</td>
<td>131</td>
</tr>
</tbody>
</table>
5.1 Goals and Challenges

In order to answer the research questions of our thesis as those were presented in detail in Chapter 4 we aim to achieve the following goals:

- to provide a formal specification of provenance awareness properties and requirements for service compositions and SOC systems by introducing the set of required models,

- to provide a design-time analysis framework for provenance awareness of service compositions, where the framework offers the ability to analyse apriori to the system’s implementation whether the modelled system is provenance-aware, enabling designers to explore the impact of provenance awareness on other service properties and take the right provenance infrastructure decisions by considering possible trade-offs.

The main challenges for this research were:

- to define provenance question categories in such a way so those can exhibit the provenance data that are required to be recorded for answering those questions, by identifying elemental semantic provenance question categories, organised as a set of provenance facets.

- identifying the right formal methods in order to formalise the different aspects of provenance properties and provenance requirements,

- identifying the dependencies between the different types of models and technical challenges on integrating the models into a complete framework solution,

- introducing a set of analytical metrics for provenance awareness and integrate those as part of our framework solution to enable metric validation by comparing the modelled (predicted) provenance awareness against the simulated/modelled provenance data measurements.
Next we discuss the approach we have taken towards achieving our goals, followed by our research methodology plan and proposed architecture.

## 5.2 Approach

![Provenance Awareness Framework: Approach and Architecture](image)

The basic components we worked on with regards to our approach focusing on an overall specification and analysis framework for provenance awareness of service compositions are visualised in Figure 5.1. The steps we have taken in our thesis approach towards building our proposed framework are summarised in the following list. The following list is restating the contributions we have presented in a high-level description in Section 1.6, by considering the technicalities, methodology and sequence of steps required as part of building an architectural solution.
1. providing a specification language and models to describe service provenance concepts (considering atomic and composite services)

2. providing a method and modelling notation for expressing the provenance infrastructure activity and behaviour

3. providing models and binding mechanisms to describe the interconnections between the provenance infrastructure models and specific atomic and composite service specifications

4. identifying the kinds of provenance questions and categories, and formalising those into patterns to express and analyse provenance awareness requirements of the users

5. integrating all those specification, modelling and analysis mechanisms into a unified specification and analysis framework and ecosystem, providing a set of toolsets to realise this.

Looking into more detail in those steps we first identified the set of required models for expressing provenance awareness for composite services; namely a service interface model, a service composition model, a provenance data model, a provenance infrastructure model and formal models for provenance questions.

Those models provide concepts to specify:

- service interfaces (*Service Interface Model*) and the composition process schema/workflow (*Service Composition Model*),

- provenance data (*Provenance Data Model*),

- provenance recording behaviour (*Provenance Infrastructure Model*),

- provenance questions/queries that correspond to the user provenance awareness requirements (*Provenance Question Type Patterns*)
The *Provenance Data Model* does not limit to the definition of a conceptual model for provenance data about services and service compositions, but our main contribution is to provide a higher level abstraction of the provenance data structures including relationships between provenance data, where the data structures are parametrized as part of parameterized model elements – *provenance templates*. A number of provenance templates are required in order to reflect the provenance data structures modelled corresponding to the overall provenance infrastructure activity of the service composition execution cycle (e.g., including phases of service discovery, service selection, service execution etc.).

Next, we describe the structure and scope of those models in detail as part of our overall approach and architecture.

**Service Interface Model**  This model is required to describe functional properties of a service, corresponding to atomic or composite services. By functional properties, we mainly refer to the functionality that every service (atomic or composite one) provides and could be potentially used to cover part of the functional requirements for a service composition. Such a model specifies the provided and required interfaces of every service, including the operations (actions supported by a service) corresponding to them, the input and output messages (data being communicated) and parameters passed to them for every operation. It also provides a way to identify the different services that provide a particular functionality (e.g., URI, binding port, SOAP address). In order to specify this model, we are using an existing specification language for describing services such as WSDL [107].

For example, for the travel planner scenario as presented in Figure 1.1, the service interface model (WSDL specification) will provide a definition of the required and provided interfaces of every service \((s_1), (s_2), (s_3), (s_4),...,(s_7)\) that are aggregated in the travel planner. Those service descriptions provide a machine-readable description of the service functionality including:

- the set of operations \((o_1), (o_2), (o_3),...,(o_7)\) associated with each service,
the input and output messages of every operation \((o_1),(o_2), (o_3),...,(o_7)\),

- how the services can be called,

- what parameters those expect, and

- what data structures those return.

Next, we have to consider the actual service composition model.

**Service Composition Model** This model is required in order to specify abstract processes with the unbounded invocation to external services (e.g., web services that implement the abstract service process plan corresponding to the service composition); which may be individual or composite services. In this model we are mainly interested on representing:

- the main service components,

- their binding structure – including the interconnections and interdependencies among them and

- their workflow patterns as our composition model needs to be able to model sequence of elements such as loops, conditional branching, or concurrent blocks to model the control flow. In our travel planner scenario such elements are for example the conditional branching between services \((t_5)\) and \((t_6)\), or the concurrent block of \((t_1), (t_2), (t_3)\) corresponding

Regarding the functional specification of the service composition model, including the interfaces and composition relations of atomic and composite services, this is expressed by existing specification languages for services and their compositions, including BPMN [3] and BPEL [4]. Those composition models are used in combination, to provide an abstract process plan of the composition (BPMN 2.0) and a description of composition execution workflow (BPEL) of the services realising the abstract process plan.
**Provenance Data Model** The *provenance data model* of composed services describes the provenance data that may be recorded and stored by executing that service. We have expressed provenance data in the form of a conceptual ontology-based model, namely ServiceProv Ontology, as an extension of PROV-O [14], where its concepts map to a set of provenance templates. Inspired by an approach by Danger and Curcin [33] that introduce a higher-level abstraction of the provenance graph data called provenance template. This construct specifies the basic pattern of provenance data that a software tool may record in the provenance repository, e.g. an edit operation on an eligibility criterion of a study. Provenance templates use graph syntax similar to provenance graphs, however their artifact, process, and agent nodes refer not to concrete past instances but to domain concepts taken from some model. Thus, in a model-based software environment, templates describe predefined system actions whose concrete executions are recorded in provenance graphs. Further graphical constructs are introduced to model a subgraph occurring multiple times in sequential or parallel repetition patterns [33].

We use the notion of the *template* to introduce a higher level abstraction of the provenance graph data by parameterizing the concepts of ServiceProv ontology as part of parameterized model elements, namely *provenance templates*. Provenance templates provide a representation of the overall data provenance for the composite service execution cycle, as multiple provenance templates binding to the specific composite service system and infrastructure the service designers would like to model, will form provenance model instances complying to a *(service) provenance meta-model* (as stated in Figure 1.2). We provide the modelling notation for the provenance data model by extending PROV-DM concepts and PROV-O ontology to ServiceProv Ontology; where ServiceProv Ontology contains concepts (and their relationships) that describe provenance for SOC systems. Those concepts are then used as a basis in order to feed a provenance infrastructure model that expresses the provenance recording behaviour and activity of the composition.
Our contribution to provenance templates for the needs of our thesis is extending Danger’s and Curcin’s templates definition to describe SOA/SOC domain concepts as defined in the ServiceProv ontology, our proposed extension for service compositions. In our definition of provenance template 1) provenance nodes are annotated with the corresponding ServiceProv concepts, describing at a higher-level of abstraction the SOA/SOC provenance domain, and 2) provenance activities represent workflows or service compositions (choreographies or orchestrations). Therefore, a provenance template can be considered as an abstraction of the workflow specification and the provenance related to this. Our provenance templates are built using PROV-DM graph syntax, in comparison to Danger’s and Curcin’s approach which is using a syntax derived from the Open Provenance Model (OPM) standard.

**Provenance Infrastructure Model**  Provenance data concepts of ServiceProv Ontology are the fundamental elements used in order to express the provenance data captured for a SOC system. Yet, additional models are required to express the provenance recording and storage behaviour of the composition across the SOC execution cycle. We introduce the modelling notation for expressing the provenance infrastructure recording activity for SOC systems as a set of stereotyped activities in UML. We use UML tagged properties as part of the stereotyped activities to represent the corresponding provenance data model instances, templated parameters, we would like to record as part of a particular service composition design.

**Provenance Facets**  In order to define provenance awareness for service based systems in this thesis we also identify a number of (semantic) elemental provenance question categories, so that provenance awareness can be specified as the categories of questions a system can answer. Provenance questions have been categorised by following a faceted classification approach and identifying a set of provenance facets. We then contribute a set of formal models for the provenance questions. We first abstract from these questions, defining a number of elemental semantic provenance-question categories, organized using a set of (provenance
facets) [156], and then we propose their formalization as \textit{provenance-question-type-patterns}.

**Provenance Question Type Patterns.** These patterns form a set of formal models using the SPARQL query language for expressing the different kinds of provenance questions, classified using the set of identified provenance facets. Those formal models, namely \textit{Provenance Question Type Patterns}, are provided in a catalogue form. We use the query patterns catalogue as a basis to analyse a system’s provenance awareness, by checking at design time whether the provenance data – expressed in compliance to the \textit{provenance data model} – generated by the provenance infrastructure simulated behaviour, is captured either by one or a composition of provenance question type patterns in our catalogue.

**Model Integration - Modelling Environment for Provenance Awareness**

The required models we introduced have a number of dependencies on each other as we integrate those in order to form a specification and analysis framework for provenance awareness of service composition and SOC systems. In summary, the main building blocks of this framework, as presented in Figure 5.1, cover:

a) provenance models to specify the provenance data structure of service related processes and the corresponding provenance infrastructure required to capture this data, b) formal models for provenance questions (provenance question type patterns) exhibiting a number of semantic provenance question categories (provenance facets) to represent the provenance awareness requirements of the users, c) an automated modelling and analysis environment to explore provenance awareness support of composite service designs.

Integrating those building blocks into a framework enables service designers to specify provenance related properties of a system, the infrastructure needed for supporting these properties and then analyse whether the provenance infrastructure system’s design ensures that the necessary provenance data collection is effectively supported by the system’s infrastructure. In practice, modelling provenance awareness support for composite service designs requires modelling
the structure of the data collected considering both the system’s architecture and provenance infrastructure as a whole and how the behaviour of the provenance infrastructure enables the provenance data collection for answering different kinds of provenance questions. The provenance data model, *ServiceProv Ontology*, is used for capturing the structure of provenance data that would be collected, relevant to the execution of a service composition. The provenance infrastructure model is used to specify the recording/storage provenance infrastructure activity. Then we take an approach in combining those two types of models to enable analysis and verification at design-time of whether the modelled provenance recording activity is sufficient for the system to guarantee certain levels of provenance awareness provision and support. This works by using a set of rules to perform the matching between the provenance question type patterns (requirements) and sample provenance data graph structures (instantiated schemas) generated as part of the system’s simulated provenance infrastructure recording behaviour. As a second step, the framework enables an analysis of whether this is the correct data for answering a set of provenance questions (requirements).

To this end, the model integration provides a complete modelling environment solution for provenance awareness of service compositions. We took an approach in building a prototype implementation analysis tool (PROVa), that simulates the provenance infrastructure behaviour and applies checks at design-time of whether the provenance data captured by the provenance infrastructure, maps to one or a composition of provenance question type patterns in our catalogue. We extensively discuss the proposed architecture for our tool implementation, the roles and model bindings in Section 5.4.

### 5.3 Research Methodology

The methodology for approaching our research problem is structured into five phases:

1. The *Data Collection phase*: in this phase, we identified the fundamental
provenance concepts for services/their compositions and the SOC execution cycle (e.g., service execution, service discovery, service selection, orchestration) that need to be modelled in order to allow analysis for provenance awareness. In this phase, we also analysed provenance-related requirements based on a set of use case scenarios (asking particular provenance questions) and classified them using a set of provenance facets, according to the kinds of provenance data that are required to be recorded in order to answer the different types of questions.

2. The Modelling phase: in this phase, we defined the required set of models to specify provenance for service compositions (e.g., provenance data model, provenance infrastructure model, service interface model, service composition model) and the provenance awareness requirements (e.g., provenance question type patterns for the different provenance facets).

3. The Model Synthesis (Integration) and Analysis phase: in this phase, we integrated the corresponding set of models in order to provide a modelling environment solution for specifying and analysing provenance awareness of service compositions. We focused on formalising the dependencies and interconnections between the different models in order to build our analysis environment and provide tool support for this. Formalising the provenance questions constituted the basis of our analysis framework for applying checks against provenance awareness specifications of composite service systems and their infrastructure design. This phase also encountered the introduction of analytical metrics for provenance awareness. We integrated those metrics as part of our framework solution to enable metric validation by comparing the modelled (predicted) provenance awareness against the simulated/modelled provenance data measurements.

4. The Implementation phase: in this phase, we built a proof of concept for the framework proposed in this thesis by providing a prototype tool implementation, called PROVa, of the proposed architecture as discussed in Section 5.4.
5. The *Evaluation phase*: in this phase, we evaluated the contributions of our thesis and the proposed overall framework based on a set of hypotheses as discussed in detail in Chapter 11. Our evaluation methodology plan is comprised of a set of hypotheses, comprising a set of claims made with regards to our research questions (as those were discussed in Section 4.2), along with the type of validation we have introduced in order to prove our hypotheses. For the evaluation of the main contributions of this thesis, as those are listed in Section 5.5, we methodologically apply our proposed approach and framework to two different use case studies including a standard use case scenario taken from the service composition literature and an industrial use case scenario. The results of the use case studies are extensively discussed in Chapter 11.

### 5.4 Proposed Architecture

Having introduced our research methodology, in this section we propose an architecture to address our research questions, embodying the building blocks of our approach discussed in Section 5.2. Figure 5.1† presents the high-level overview of the core components of our approach and architecture for the proposed specification and analysis framework for provenance awareness of composite service systems; a complete research agenda of this has been published in [157]. Then, Figure 5.2 exhibits how the main building blocks of our proposed framework associate to the corresponding set of roles involved in the architecture and design process lifecycle; mainly by whom and how those are created, managed and used.

The main building blocks of this architecture cover:

- a set of service models to describe interfaces of atomic services (using WSDL models) along with the composition flow and sequence of service invocations within the composition (using a BPEL model),

†this is a simplified version of the diagrammatic version presented in Figure 1.2 excluding the consideration of contribution tags and metamodel/model levels of abstraction
Figure 5.2: Provenance Awareness Framework: Architecture and Roles

b. provenance models to capture the provenance data structure of service related processes (ServiceProv) and the corresponding infrastructure to collect this data (Provenance Infrastructure Template Model),

c. a binding model to bind the Provenance Infrastructure Template Model to specific atomic service and service composition descriptions (WSDL/BPEL files) (Template Binding Specification),

d. formal models of provenance questions (Provenance Question Type Patterns), exhibiting a number of provenance facets, to represent the provenance awareness requirements of the users,

e. an automated modelling and analysis environment to explore provenance awareness support of composite-service systems. The environment will enable designers involved in the system’s design workflow lifecycle to specify provenance related properties of composite service systems, the infrastructure needed for supporting these, and then to validate whether the prove-
nance infrastructure design would be sufficient to capture the desired data for particular services that are dynamically bound and composed. As a second step designers could use the modelling environment to analyse whether this is the correct provenance information for answering a set of provenance questions, exhibiting provenance awareness requirements of the users.

Formally specifying and analysing provenance awareness support of composite service systems and service compositions is a complex process that requires modelling and analysis effort from the different roles (Figure 5.2), involved in the system’s design workflow lifecycle (Figure 5.3). Thus, the models introduced as part of the architecture of our framework are aligned to roles such as:

- application designer,
- infrastructure designer,
- QoS (Provenance) analyst, and
- domain expert.

In particular the application designer is in charge of the Service Interface and Service Composition models that provide concepts to specify the functional properties of their applications, considering the operations offered by the specific applications and the control flow/sequence of individual services within the composition respectively. Those models are developed by using existing standard web service languages for describing services and service compositions including WSDL and BPEL/BPMN.

The application designer will then also need to incorporate provenance related aspects into the system’s design. Those correspond to the provenance data structure, represented by the ServiceProv ontological model, that needs to be captured in order to answer a set of provenance questions. Provenance questions are handled:
1. at a first step by the domain expert that is familiar with the business domain of different service-based systems and is able to analyse the users’ preferences and express more systematically their needs into questions; allowing the domain expert to clearly communicate users’ provenance awareness requirements to application designers. This interaction is beneficial for both role parties as there is a central figure with a clear understanding of the business domain and client requirements being able to express provenance questions in a fast-paced rhythm (the domain expert), that may be reusable across different applications handling the same or similar business domains, and another (the application designer) focusing on developing the system (service) design taking care of all the technical details required for realizing the provenance awareness requirements of the users.

2. at a second step the provenance questions are handled by a QoS (provenance) analyst that translates those into a machine-processable format. The format used is provenance question type patterns expressed in SPARQL [74], which are realised as SPARQL queries. In this way, QoS provenance specialists enable the automated analysis of the system’s provenance awareness support. They also help the application designers to interpret the results of this analysis into meaningful conclusions of whether the system’s provenance awareness design is sufficient to cover the users’ provenance needs. This separation of roles is valuable as analysts can focus on i) the assertions that form the body of the SPARQL queries to formally express provenance questions, ii) identifying the level of provenance awareness support by performing a set of analysis checks over a provenance aware application design. A full expert’s analysis about each provenance aspect at a finer level of granularity forms the base for a useful feedback interaction with application designers. This feedback interaction enables the application designers to take action in revising or re-engineering the existing application design in order to support additional provenance requirements.

In practice, by useful feedback we refer to the feedback that QoS provenance
specialists provide to the application designers as a result of interpreting the findings of the system’s automated analysis with regards to provenance awareness support (Provenance Data Graph Analysis activity, as shown in Figure 10.1) into meaningful conclusions of whether the system’s provenance awareness design is sufficient to cover the users’ provenance awareness requirements. As the provenance questions are expressed as SPARQL queries by the provenance analysts, the feedback takes the form of a query graph analysis report incorporating the query execution results \(^2\) (Query Graph Analysis Results artefact, as shown in Figure 10.1) exhibiting the provenance information that would (or would not) be generated by a given provenance infrastructure design. The feedback can also take the form of consulting by the QoS provenance specialists towards the application and infrastructure designers. The provenance specialists can consult about additional provenance infrastructure required to be provisioned in order satisfy new provenance awareness requirements. This feedback interaction has been evidenced in the industrial case study we have conducted with Singular Logic where we have acted as the provenance analysts that performed analysis over the provenance information captured by the ORBI application. With the analysis we managed to propose enhancements on additional provenance information that could be captured needed for answering particular provenance questions of the system users. To identify those provenance questions we coordinated with the marketing and strategists’ departments (domain experts).

In addition, the analysis feedback is useful for application designers that model system qualities at an early stage of the system’s design. It allows them to take decisions early enough about redesigning their provenance infrastructure support, by changing their model and running further simulations that allow them to validate whether the provenance awareness

\(^2\)examples of the queries are provided for the two use cases studies in Appendix C, while our experimentation analysis query results conducted for the travel planner and ORBI use case studies can be found online at https://sourceforge.net/projects/prova-tool/files/exeQ.log/download
requirements specified will be met. This early analysis feedback at design time have two main benefits: 1) it cuts down on engineering time and implementation costs, 2) it enables application designers to take design decisions considering the trade-offs between fully supporting provenance awareness requirements of users and the provenance infrastructure cost. This kind of feedback was evidenced to be useful for the cloud application department responsible for the ORBI application design in our industrial case study; i) it informed application designers about the provenance not captured, clarifying any limitations of the current provenance infrastructure they would like to implement in a future release version of ORBI application, ii) it indicated the infrastructure requirements they should plan for and allowed them to decide on the associated trade-offs in terms of reprioritizing the implementation of other features for ORBI.

As we will see in the implementation/evaluation phase in Chapter 11 the application designers can test/validate the SPARQL queries, written by the QoS provenance specialists by using the parser type checker functionality. The parser type checker enables them to apply syntactic and semantic checks on the SPARQL queries against a given application design as discussed in Section 11.3.4. Although, the QoS provenance analysts’ main contribution is on translating provenance questions into SPARQL and on consulting in the interpretation of the validation check results, they are the ones that acquire the expertise of ServiceProv model and what query assertions need to be defined in order to form the validation query checks corresponding to a set of specified provenance awareness requirements. Therefore, they can contribute on the validation process by manually checking whether the query assertions written by them are complying with the schema of the ServiceProv ontology, its relationships and concepts and identify syntactic or semantic errors. Yet, the parser type checker is there to provide automation on the validation process. It is a tool coded by the provenance experts which can benefit the application designers/developers to execute tests in order to validate new or redesigned applications on their own under the
right guidance and consulting by the provenance analysts.

As already mentioned, collecting the provenance required can only be guaranteed by the provision of sufficient provenance infrastructure mechanisms. Modelling the provenance infrastructure of a composite service system can be complicated and is therefore not a task an application designer would want to undertake on its own. At the same time, the same infrastructure may be relevant to different applications, so once the provenance infrastructure behaviour for an application system has been modelled, designers would like to be able to reuse it for different composite service applications. In order to enable this, provenance infrastructure specification needs to be made independent of specific applications. In our framework, this is achieved by introducing the *Provenance Infrastructure Model*.

The Provenance Infrastructure Model enables the specification of the provenance infrastructure behaviour – in the form of stereotyped provenance record-
ing activities with a tagged value containing the list of provenance data to be recorded. This is designed independently of specific applications but considers the different phases of the SOC execution cycle where provenance activities need to be set up. The provenance infrastructure behavioural patterns introduced can then be applied similarly to any composite service system where we require to design provenance infrastructure for, in order to enable analysis of the system’s provenance awareness support at design time.

This separation of models in our architectural design allows us to distinguish the roles of the infrastructure designer, who defines how the infrastructure behaves, and the application designer, that defines how a specific application makes use of such an infrastructure. This is accomplished by introducing a Template Binding Model that makes the essential connections about specific services and workflows (application designer) to the information about how provenance infrastructure behaves (infrastructure designer).

In addition, it is important to mention that the provenance awareness system’s design and analysis workflow, as presented in Figure 5.3, is structured in three different phases:

- the Modelling phase, where the application and infrastructure designers are focusing on the specification and modelling of the provenance data, services and provenance infrastructure components,

- the Model Integration and Analysis phase, where all the models and specifications worked out during the modelling phase are integrated under the same environment to enable simulation of the supported provenance and analysis over that against a set of provenance awareness requirements. At this phase the QoS (Provenance) analyst is joining up his effort – on the formally expressed provenance questions (requirements) – in order allow analysis checks by reasoning over the simulated provenance data graph results.
Finally, the domain expert is working at a separate phase, called the **Requirements phase**, in order to track down additional requirements and business use cases about provenance awareness that would need to be turned into analytical metrics allowing provenance awareness validation of the application and application infrastructure design.

Figure 5.3 provides a high level description of our framework’s architecture according to the provenance awareness design and analysis workflow cycle, where the foundational modelling artefacts are used as input in the modelling and analysis environment for provenance awareness. In Chapter 10 we provide an implementation of our framework realising the high-level architecture proposed in Figure 5.3, namely PROVa tool. In particular, Figures 10.1 and 10.11 show an activity/component and deployment diagrams for PROVa toolset respectively. Provenance Data Model in Figures 5.3 and 5.2 refers to the ServiceProv ontology. ServiceProv modelling notation is used to specify the provenance information we would like to capture as part of an instantiation of a provenance infrastructure activity in the Template Provenance Infrastructure Model (metamodel level). The result of the Provenance Awareness Graph Generation, as shown in Figures 5.3, 10.1 and 10.11, is an instantiated provenance data graph (Generated Provenance Data Graph) that is expressed using the ServiceProv notation. It does not refer to the ServiceProv Ontology used at the modelling phase of our design and analysis but represents an instantiation of this.

Next, we summarise the research contributions made as part of our proposed architecture.

### 5.5 Research Contributions

In the previous section we discussed our proposed architecture, incorporating a set of formal models, analysis methods and tools that address the set of research questions as presented in Section 5.2. In this section, we summarise the contributions made as part of our overall architecture solution grouping those based on
the associated roles of the architecture, design and analysis workflow process that would most benefit from each contribution. Our overall contribution is a formal specification and analysis framework of provenance awareness for SOC systems and compositions of services. Within that our proposed framework contributes and benefits each role group with the following:

- **Application Designer Contributions Grouping**
  - a Provenance Data Model (ServiceProv Ontology) for composite services; that is, a data model and schema that allows capturing the provenance data relevant to the execution cycle of composite services and SOC systems in particular

- **Infrastructure Designer Contributions Grouping**
  - a template provenance infrastructure metamodel which provides the required modelling notation for designing the infrastructure activity required to collect provenance for composite service systems
  - a template binding specification that enables binding specific services and business processes (orchestrations) under development with the required provenance infrastructure configuration design

- **Domain Expert Contributions Grouping**
  - a set of realistic questions about the execution lifecycle of service-oriented systems and service compositions, in order to express/determine provenance awareness requirements of the system users and other stakeholders

- **QoS (Provenance) Analyst Contributions Grouping**
  - a faceted classification of provenance questions, introducing a set of provenance facets, so that provenance awareness can be specified as the set of provenance questions a system can answer. We form this classification by analysing the kinds of provenance data that need to
be recorded, collected and stored to answer realistic questions. Those facets then work as a basis of building/expressing more complicated provenance awareness requirements though multifaceted questions.

- a set of provenance question type patterns to formally express the different kinds of provenance awareness requirements (questions) of the users by exhibiting the identified provenance facets; those patterns are then used as a basis for forming provenance awareness analytical metrics so that provenance awareness can be measured as the kinds of provenance questions answerable by a service-oriented (computing) system against the number of facets (elemental provenance question categories).

- a provenance awareness modelling and analysis environment and method to analyse whether a particular provenance infrastructure design will provide sufficient provenance information to answer a set of provenance questions (requirements),

- a provenance generation analysis algorithm that simulates the system’s provenance infrastructure behaviour by generating the corresponding sample provenance data graphs that would be captured by a specific implementation

- a set of rules to perform/apply provenance awareness analysis checks at design-time of whether the provenance recorded by the system, represented by the generated sample provenance data graphs, is captured either by one or a composition of provenance patterns in our catalogue.

**Summary:** In this chapter we introduced our proposed architecture and summarised the research contributions as part of this. We next present the first part of the research contributions of this thesis, introducing a set of formal models required for the specification and analysis for provenance awareness of service compositions.
Part I

Formal Models for the Specification and Analysis of Provenance Awareness
Chapter 6

Provenance Facets

In the previous chapter we presented our approach, architecture and overall thesis goal towards a formal specification and analysis framework for provenance awareness of service compositions. In this chapter, we present the first contribution of our thesis addressing research question RQ 1, which is about identifying a way to express provenance awareness as a non-functional requirement (NFR). In particular, we focus on its first subquestion RQ 1.1 addressing the identification of a set of questions based on which we can define provenance awareness requirements for SOC systems. We also develop a faceted classification for the different kinds of questions by identifying a set of semantic provenance question categories using a set of provenance facets in Section 6.2; the faceted classification addresses subquestion RQ 1.2 and builds the foundations for analysing provenance awareness requirements for different aspects of a SOC system’s design. The results of this contribution have been previously published in [156].

Contents

6.1 Provenance Questions .................................. 136
6.1.1 Travel Planner Use Case Scenario ................. 136
6.1.2 Questions for the Travel Planner Scenario .......... 138
6.2 Faceted Classification of Provenance Questions .......... 144
6.2.1 Provenance Facets for Provenance Awareness ...... 145
6.2.2 Multi-Facet Provenance Questions ................. 150
6.1 Provenance Questions

Our first step is to identify the types of provenance questions based on which we can define provenance awareness requirements for compositions of services and SOC systems. As part of this first step of our approach we are using a standard use case scenario of the service composition literature [40], namely a travel planner composition scenario as previously introduced in Chapter 4. In this chapter, we briefly reintroduce the basics of our scenario considering some additional aspects that are significant for identifying provenance question categories and provenance facets for representing provenance awareness requirements of a composite service design. Those include environmental factors (e.g., execution environment, network and physical resources), composition type aspects (e.g., orchestration versus choreography) and the different kinds of users that may express different kinds of provenance questions corresponding to their different provenance needs and therefore provenance awareness requirements.

6.1.1 Travel Planner Use Case Scenario

Figure 4.1 depicts our service composition scenario for booking multiple travel arrangements. Starting from a high-level description of this, as shown in the upper part of Figure 4.1, there are a number of aspects we need to consider with regards to modelling a service composition scenario such as 1) user requirements in the form of questions, 2) execution environment details (e.g., network latency, availability of physical resources, workload), 3) the structure and workflow/dataflow of the composite service in association to the atomic services this is composed of.

At a second step, the composition is represented as a statechart, where a state corresponds to an abstract task in which a named service operation is invoked,
with given input and output messages (as shown in the middle plane of Figure 4.1). In a given instantiation, each task would be performed by a particular service discovered at run-time. The tasks as those have been described before in Chapter 4 are to book a flight \((t_1)\), book a hotel \((t_2)\), book an attraction to visit \((t_3)\), calculate the driving time from the hotel to the attraction \((t_4)\), rent a bike to travel between the hotel and attraction \((t_5)\), rent a car to travel between the hotel and attraction \((t_6)\), and take payment for the travel package by credit card \((t_7)\). The composition is expressed in terms of sequential transitions, AND-states to express concurrency, and multiple transitions from a state to represent conditional branching.

The users of the system may have requirements with regards to non-functional and other qualities that are related to the system’s (composition’s) environment such as performance, availability of resources, network latency, reliability, security and provenance awareness. With regards to provenance awareness, the user wants to ask questions related to the data provenance of the travel planner results, such as why a particular flight provider was chosen instead of another, why a bike was booked instead of a car, which the third party authorised the card during the online payment.

To answer particular provenance questions – questions about the system’s provenance awareness – we would have to record the values of data such as input and output messages, identification information about services, non-functional property values, the routes followed in the dynamic composition flow, the activities related to the SOC execution cycle and the agents involved in those activities. Different questions require different kinds and levels of detail, and it depends on what the user will or is likely to require as to what provenance data should be captured.

Next, we follow a systematic approach to creating an exhaustive list of the questions different users may have about the core aspects of SOC systems and service composition scenarios with regards to provenance awareness, by using...
the travel planner composition as a use case study. These questions may be posed by two types of users: 1) end users of the system that pose high-level questions as part of a user experience activity and trustworthiness or 2) technical users such as service designers and QoS (provenance) analysts that would like to question the system’s provenance awareness/accountability design with regards to quality, NFPs trade-offs, data processing and other aspects of the SOC system’s execution cycle. The questions of the technical users are in fact representative of the high-level questions of the end-users; those are posed as a result of the need for formalising the provenance needs (provenance awareness requirements) of the users and check those against the provenance awareness design of a composite service system.

6.1.2 Questions for the Travel Planner Scenario

In this section we provide a comprehensive list – this is complete with regards to the core concepts of SOC systems and service compositions as presented in Tables 6.1 and 6.2 and discussed in Section 11.3.1 that were identified after a thorough literature review of the fundamental pieces of work in the SOA, service (compositions) and the SOC paradigm research space – of questions representing user requirements related to different provenance aspects the end-users or service designers would like to get answers to. We build this list of questions in the context of the example travel planner scenario.

1. Queries relevant to the service composition workflow and specification. Those questions are about the interaction between atomic services within the composition, their invocation through the use of service interfaces describing a set of required and provided operations, and the data that flow between those services. More generic examples of those questions may contain the following: Which activities (particular services, operations) were invoked during the service composition execution? What were the parameters passed in input and output messages associated to particular operations? What are the dependencies between the input and output parameters
Table 6.1: SOC and SOA Evaluation Criteria Checklist (I)

<table>
<thead>
<tr>
<th>SOA and SOC Concepts</th>
<th>Facet No</th>
<th>ServiceProv Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Basic/Composite) Service</td>
<td>1</td>
<td>serviceprov:Service</td>
</tr>
<tr>
<td>Service Description</td>
<td>1, 9</td>
<td>serviceprov:ServiceDescription, serviceprov:ServiceCollection</td>
</tr>
<tr>
<td>Server</td>
<td>1, 8</td>
<td>serviceprov:ServiceExecutionServer, serviceprov:Server</td>
</tr>
<tr>
<td>Repository</td>
<td>1</td>
<td>serviceprov:Registry</td>
</tr>
<tr>
<td>Message(s)</td>
<td>2</td>
<td>serviceprov:Message</td>
</tr>
<tr>
<td>Message Semantics</td>
<td>2, 9</td>
<td>serviceprov:ServiceDescriptionItem, serviceprov:Parameter</td>
</tr>
<tr>
<td>Operation(s)/Behaviour/Capability</td>
<td>2, 9</td>
<td>serviceprov:ServiceDescriptionItem, serviceprov:ServiceDescriptionDiscoveredItem</td>
</tr>
<tr>
<td>Service Interface</td>
<td>1, 9</td>
<td>serviceprov:ServiceDescription Published Item</td>
</tr>
<tr>
<td>Execution</td>
<td>5, 9</td>
<td>serviceprov:ServiceExecution, serviceprov:OrchestratorExecution, serviceprov:AtomicServiceExecution</td>
</tr>
<tr>
<td>Publication</td>
<td>1</td>
<td>serviceprov:ServiceInterfacePublishing</td>
</tr>
<tr>
<td>Service Client</td>
<td>8</td>
<td>serviceprov:Broker</td>
</tr>
<tr>
<td>Service Provider</td>
<td>1, 8</td>
<td>serviceprov:ServiceProvider</td>
</tr>
<tr>
<td>Service Publisher</td>
<td>1, 8</td>
<td>serviceprov:Broker</td>
</tr>
<tr>
<td>Service Registry</td>
<td>1, 8</td>
<td>serviceprov:Registry</td>
</tr>
<tr>
<td>Service Discovery</td>
<td>1</td>
<td>serviceprov:Discovery</td>
</tr>
<tr>
<td>Service Broker/Discovery Agency</td>
<td>1, 8</td>
<td>serviceprov:Broker</td>
</tr>
<tr>
<td>Service Selection</td>
<td>5, 9</td>
<td>serviceprov:Selection</td>
</tr>
<tr>
<td>Service Selection Model</td>
<td>5, 9</td>
<td>serviceprov:SelectedServiceDescription</td>
</tr>
</tbody>
</table>
Table 6.2: SOC and SOA Evaluation Criteria Checklist (II)

<table>
<thead>
<tr>
<th>SOA/SOC Concepts</th>
<th>Facet No</th>
<th>ServiceProv Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Aggregation/Coordination</td>
<td>5, 9</td>
<td>serviceprov:Planning, serviceprov:Selection</td>
</tr>
<tr>
<td>Service Aggregator</td>
<td>8</td>
<td>serviceprov:Aggregator</td>
</tr>
<tr>
<td>Service Invocation</td>
<td>2, 5</td>
<td>serviceprov:AtomicServiceExecution</td>
</tr>
<tr>
<td>Orchestration(Composition)</td>
<td>2, 5</td>
<td>serviceprov:OrchestratorExecution</td>
</tr>
<tr>
<td>Orchestration Model</td>
<td>2, 5</td>
<td>serviceprov:WorkPlan</td>
</tr>
<tr>
<td>Orchestrator</td>
<td>8</td>
<td>serviceprov:Orchestrator</td>
</tr>
<tr>
<td>Choreography(Composition)</td>
<td>2</td>
<td>serviceprov:Orchestration missing the Orchestrator management aspect</td>
</tr>
<tr>
<td>QoS Model/Conformance</td>
<td>7</td>
<td>serviceprov:QoSRequirements, serviceprov:QoSContract</td>
</tr>
<tr>
<td>NFPs</td>
<td>6, 7</td>
<td>serviceprov:NFP</td>
</tr>
<tr>
<td>Reliable Messaging</td>
<td>6, 7</td>
<td>serviceprov:Reliability as a specialization of serviceprov:NFP</td>
</tr>
<tr>
<td>Security Model</td>
<td>7</td>
<td>serviceprov:Security as a specialization of serviceprov:NFP or</td>
</tr>
<tr>
<td>Resource</td>
<td>3</td>
<td>serviceprov:Resource and serviceprov:ResourceCollection</td>
</tr>
</tbody>
</table>

passed during service invocation?

With regards to the travel planner scenario a list of example questions may contain the following:

- What was the input information (e.g., user preferences, data parameters passed from the previously executed services in the workflow) for the flight booking and the attraction search services?
- What parameters were passed as input from the flight booking to the hotel booking service?
- What were the input messages passed for the driving time calculation service?
- What were the output messages of the driving time calculation service?
− Were the hotel and flight booked individually or as an offer package?
− What routes were not followed/skipped in the composite service execution (e.g., the task for credit card authentication)?
− Which service invoked the calculation of the driving time service?

2. Queries about the service discovery and service selection processes of the SOC execution cycle. Those questions refer to the process of brokering/discovering different services published as part of a repository/registry and selecting the service candidates that can satisfy the required functional and non-functional requirements for a service composition.

− What were the flight booking services of low-cost companies that were discovered?
− What was the reason that the particular service provider for the hotel or flight booking was selected (e.g., user preferences, availability, cost)?
− Which hotel booking provider was selected for the travel planner composition?

3. Queries related to the physical deployment and resources’ availability (data centers, CPUs, disks, memory, network).

− What was the availability of the flight booking service?
− How was the travel planner service performing during execution?
− What resources were allocated for the execution of the atomic services (e.g., flight booking, hotel booking atomic services) and the composite service (e.g., travel planner composite service)?

4. Queries related to execution time and timestamps of specific events and activities throughout the service composition execution cycle.

− At which timestamp the flight booking service sent back a failure execution event notification?
– When was the bike rental service invoked?
– When was the car rental service discovered?
– At which timestamp was there a pick in the network latency of the travel planner composite service?
– What were the timestamps when the flight booking service was unavailable?
– At which timestamp the authentication of the credit card was completed?
– When did the travel planner execution started (or ended)?

5. Queries related to routes of the composition not followed/not executed.

– Are there branches in the composition service workflow that have not been explored (e.g., the bike rental service providers)?
– What are the two different routes of making a travel planner arrangement either with the credit card or PayPal service?

6. Queries related to keeping a record of past history data.

– What was the non-availability of the flight booking service the last two days?
– How many service invocations have failed in the last month for the travel planner service?
– What was the failure rate of service $t_4$ for the driving time calculation the last two months?

7. Queries related to user preferences.

– What were the user preferences for the hotel booking service (e.g., cost, location, hotel rating)
– Were there any user constraints for making the selection of the car/bike service provider?
8. Queries related to NFPs such as performance, reliability, availability, trust, security.

- What was the average response time for booking the flight or the hotel booking service?
- What was the average response time for booking the complete travel plan service package?
- What was the average response time for calculating the distance from the attraction place to the hotel?
- What was the performance of the travel planner composite service?
- What was the network latency for making the credit card authorization?
- What was the reputation of the car rental service booked?
- Were all the service flight providers discovered available during the service selection?
- What service level agreements (SLAs) between services t1 and t2 have been violated?
- What was the form of factor (e.g., knowledge factor is something the user knows such as a password or PIN, ownership factor is something the user owns such as an ID card, or security token) for the credit card authentication?
- What were the security claims passed to the bank system through the authorization third party to confirm acceptance of the credit card information?

9. Queries related to actors/agents accountable of 1) associated activities incorporated in the SOC execution cycle such as the service discovery, service selection, orchestration 1, 2) legal matters and confidentiality of data: third

---

1 at this point we may find similarities with questions about agents/actors related to the service discovery and service selection activities
parties managing SLAS, other trustworthy authoritative sources that manage personal data exchanged during the execution of the system.

- What was the service provider of the flight or hotel booking services?
- What were the trusted/untrusted agents involved in passing through the system card number/personal cardholder information?
- Was the authentication of the credit card made by a trusted third party?
- What types of credit cards were accepted by the trusted third party?
- Who was the service owner?
- Which third party manages the SLAs between the services that are part of the service composition execution?
- Which third party takes control of the monitoring for QoS (e.g., capture response time, availability, performance for individual services and for the composite service of the travel planner)?
- Which third party published a new travel package offer at the timestamp the user’s offer was booked?

6.2 Faceted Classification of Provenance Questions

Our second step is to examine which are the semantic categories of questions requiring different kinds of provenance data to be recorded for SOC systems. Those categories will allow one to classify provenance questions based on the provenance information that needs to be captured, therefore making more concrete and organised the different requirements users may have about the system’s provenance awareness. In addition identifying a set of semantic provenance question categories will allow service designers and potential clients to compare the provenance support of a system across different dimensions of provenance. This is an important step that enables us to get one step closer to our goal to specify provenance
awareness as an NFP, by identifying the extent to which a composite service can answer different kinds of provenance questions regarding past executions.

For each different service or composition, the provenance questions will be different. Therefore, a general model of provenance-awareness as an NFP cannot be based on specific questions, but generalised semantic categories of questions and the provenance knowledge those require. Different kinds of data, of different sizes and occurring with different frequencies, are required to answer different provenance questions. It is the capture of these kinds of data that can preserve the system’s accountability and may affect the other NFPs of a service. Therefore, a category of provenance questions should be defined by the kinds of data it requires to be captured and stored.

Many realistic provenance questions require a combination of kinds of data. Therefore, instead of dividing all questions into distinct categories, we define a number of provenance facets, each corresponding to some kind of captured data, and then show how a given provenance question exhibits a combination of facets, meaning that it requires those facets’ corresponding data to be captured. The facets act as orthogonal axes for the space of provenance questions.

In this section, we first define nine provenance facets for provenance questions, and then discuss questions exhibiting a combination of facets. We illustrate them using example questions from the motivating travel planner use scenario as presented in the previous section.

6.2.1 Provenance Facets for Provenance Awareness

We identify nine facets for categorising provenance questions below, with accompanying example questions that could be asked of our motivating composite service. Each facet is defined intuitively by the kinds of data that are required to be captured and stored during a (composite) service’s execution. This is mainly a definition of the core provenance data concepts required for answering a set
of questions within each category, therefore it does not include the dependencies between data within a provenance graph, which are also relevant for answering these questions. In the chapter that follows we formalise the questions exhibiting the proposed facets as a set of provenance question type patterns. Provenance-question-type-patterns are a formal abstraction of provenance graphs, containing only those elements required to answer particular types of questions.

1. **Service and provider identity** *Requires:* DNS hostnames or IP addresses of the service providers; service names or URIs of selected services; IP addresses of the broker.

*Example questions:*

1.1 Who was the service provider of the flight booking service?

1.2 What service broker discovered the flight booking service?

1.3 What was the repository/registry location for the service description of the discovered attraction search service?

2. **Data flow** *Requires:* Input and output messages exchanged between services, parameters passed in message exchanges, an identifier for every service (service name or URI), which messages relate to every service.

*Example questions:*

2.1 What were the input parameters passed to the driving time calculation service (Min)?

2.2 What was the output of the driving time calculation service (Mout)?

2.3 Were any data inserted as changes to the initial user preferences for the flight booking service (e.g., 1st class instead of a 2nd class seat, a ticket for under-25-year olds)?
3. **Resources & physical deployment** Requires: Information on availability of different types of resources (e.g., during atomic/composite service execution), the IP address as an identifier for every resource.

*Example questions:*

3.1 What resources (memory, hard disks, CPUs) were used for the attraction search service concurrently to the flight and hotel booking services?

3.1 What resources were available during the execution of the composite service?

4. **Time** Requires: Information on composite (or individual) service execution timestamps: timestamps for start/completion, for service invocation/discovery, an identifier for the service (service name or URI).

*Example questions:*

4.1 When did the composite service travel planner execution start/end?

4.2 When was the car rental invoked/discovered?

4.3 How long did it take to complete the authentication process of the credit card?

5. **Routes not followed** Requires: information on which branches of the workflow have been taken, on the actual composition execution plan and on additional branches that have not been followed including related services, their policies and their provided interfaces. It is important to clarify that this facet requires more information than simply the data flow (facet 2) or design information (facet 9): Recording only the data flow, may never record a potential route if it is never followed by any service execution. Conversely, recording only the design information will provide information about all potential routes, but will not tell us which ones have actually been followed or not followed in a particular service execution.
Example questions:

5.1 Are there branches in the composite service that have not been explored (e.g., for the bike rental service)?

5.2 What are the alternative routes for authenticating and confirming the travel package booking by using Paypal service instead of simply sending the credit card information?

6. Past history Requires: Information to be stored for more than one past invocations of individual and composite services; different questions will differ according to the specific information to be stored (as covered by other facets) and the number of past invocations/the length of time for which to store the information.

Example questions:

6.1 How often has the service of the chosen flight booking service been accessed/invoked the last 2 days?

6.2 What was the failure rate in the time frame the travel planner service is running the last 8 months?

6.3 What was the success/failure rate in the time frame the travel planner service is running the last 8 months?

7. NFPs and QoS Requires: Different provenance data to be recorded for every particular NFP of every service that takes part in the composition. Information on SLAs between the different services e.g., \((t_4)\) and \((t_5)\).

Example questions for individual services:

7.1 What was the response time for \((t_4)\) ?

7.2 What was the processing time for \((t_5)\) ?
7.3 What was the network latency for \((t_7)\)?

7.4 What was the execution cost for the hotel booking service \((t_3)\)?

7.5 What was the reputation for service \((t_6)\)?

7.6 Was all credit-card information encrypted securely throughout processing?

*Example questions for composite services:*

7.7 What was the average response time /processing time /overall execution time for the travel planner service?

7.8 What was the total execution time for the travel planner service?

7.9 What was the execution cost for the travel planner service?

8. **Actors** *Requires:* Information about different actors that own /manage /monitor services, an identifier for them (e.g., IP address), their association with the services or the data that they manage.

*Example questions:*

8.1 Who managed (e.g., particular bank) the authentication of the credit card number?

8.2 Was the authentication of the credit card made by a trusted third party?

8.3 Which third party takes control of the monitoring for QoS (e.g., response time, availability)?

9. **Design information** *Requires:* Service interfaces (required and provided); dependencies in the composition of services; information on protocols or message format used between services.
Example questions:

9.1 Which task invokes the service instantiating \( t_4 \)?

9.2 Which services preceded \( t_7 \)?

9.3 Did all the messages exchanged between the different services of the composite service use the same format /protocol?

6.2.2 Multi-Facet Provenance Questions

Many real questions will exhibit multiple of the facets defined in the previous section. In this section, we present some examples illustrated using our travel planner use case scenario.

- Example questions: Was the reputation the reason that the flight booking service was selected? Was a prior violation of the SLA for reliability the reason that the attraction search service of service provider Expedia was not selected?

Facets exhibited: Facets 2, 6 and 7.

Provenance data required: Values passed directly between services about service’s reputation (facet 2) during service invocation; past history data of user’s feedback about the service (facet 6); SLAs of \( t_7 \) with other services (facet 7); an identifier for the service such as the service name or a URI (facet 1).

- Example questions: At what time did the flight booking service fail to be completed or invoked? At which timestamp was there a peak in the network latency for the travel planner service? At which timestamp was service \( t_2 \) not available?

Facets exhibited: Facets 4, 6 and 7.
Provenance data required: Information on failures of completion /invocation (facet 6) or latency (facet 7) for atomic services executing as part of the service composition, timestamps for those critical events (facet 4); information on link-up state /availability (facet 7) of the service and timestamps for those states (facet 4).

In this section, we have introduced a faceted classification of provenance questions for service-oriented systems considering atomic and composite services. We have defined facets of provenance questions according to the different data required to be recorded and stored in order to answer those questions. Realistic questions may exhibit a composition of multiple facets. In every case, we have to consider the provisions we have to make in designing a provenance-aware system, acquiring the necessary recording and storing mechanisms for capturing the provenance data required. Next, we discuss those provisions and we try to analyse the requirements for provenance collection and storage and the impact on other NFPs.

6.2.3 Genericity of Facets and Impact on other NFPs

In the previous section, we presented a provenance facet classification scheme by defining a set of provenance facets. The distinction between facets has an impact on other NFPs. For example, the question “Which flight booking services were available at the time of performing the service discovery at broker L?” requires data of several facets, such as the availability of the service (facet 7), the timestamp of discovery (facet 4), the service name or URI to identify the particular broker service location (facet 1). As questions require more and more kinds of data, the more that provenance collection mechanisms need to record and store, and so the higher the impact on storage demands and performance of the service.

Provenance data need to be recorded and stored in order for future provenance questions to be answered. In Table 6.3 we show a ranking of the impact of the
different facets. The ranking here will remain intuitive rather than experimentally verified as this is not a part we extensively researched as part of the contributions of this thesis and remains a proposal for future work.

In this ranking the past history data facet requires storage of provenance data for potentially long time intervals, possibly the lifetime of the service, so could require any amount of storage. Similarly, monitoring software required to assess NFPs can capture the operation of services at a fine detail. Therefore, both of these facets are said to have “very high” impact. Messages between services can be arbitrarily large, even if they only occur at well-defined points, so the impact is “high”. Timestamps, choices between routes, and the identities of actors in the system are all kinds of information that may be present frequently during a service’s execution but are each small data to record. The resources used by services will not normally change during execution, meaning recording this has a lower impact. Finally, some data, such as interface specifications, will be present after execution without recording, though copies may need to be stored to answer provenance questions in the future as past history data.

Our ultimate aim in this was to formally specify provenance-awareness of SOC systems as an NFP so that the facets the service may support are defined in a generic way. Genericity will enable future work on identifying trade-offs of provenance awareness with other NFPs, such as performance and storage, more easily. In order to accomplish this, we specify a set of provenance question type patterns for the provenance that would need to be recorded in Chapter 8. Composite services are of particular interest, because the properties of a composition is a function of the properties of its parts, and this applies to provenance-awareness as much as to other NFPs. The composition’s design clearly relates to the form of provenance that will be recorded, and so to the provenance question type patterns used for specifying provenance-awareness.

Provenance question type patterns constitute a formal abstraction of provenance graphs, containing only those elements required to answer particular types
<table>
<thead>
<tr>
<th>Facets</th>
<th>Provenance collection required</th>
<th>Performance and Storage impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Past history data</td>
<td>Recording mechanisms to capture failure/success rates or keeping logs of information for long time intervals/the life of the service</td>
<td>very high</td>
</tr>
<tr>
<td>NFPs and QoS</td>
<td>Recording mechanisms for different NFPs for atomic and composite services (e.g., performance/availability)</td>
<td>very high</td>
</tr>
<tr>
<td>Data flow</td>
<td>Recording mechanisms to continuously capture the parameters and data passed in the input and output messages (during message exchanges) between different services of the composition plan execution</td>
<td>high</td>
</tr>
<tr>
<td>Time</td>
<td>Service and message timestamps annotations, use of clocks to record timestamps for service discovery, service execution, service invocation and critical events</td>
<td>medium</td>
</tr>
<tr>
<td>Service and provider identity</td>
<td>Recording mechanisms to capture information on the identification of service providers and services (IPaddress and URIs)</td>
<td>medium</td>
</tr>
<tr>
<td>Routes not followed</td>
<td>Recording mechanisms to capture branches of the workflow being taken during the composition execution plan</td>
<td>medium</td>
</tr>
<tr>
<td>Actors</td>
<td>Recording mechanisms to capture information on the identification of third parties and their association with services they manage</td>
<td>medium</td>
</tr>
</tbody>
</table>
Table 6.4: Facets Ranking according to Provenance Recording Impact (II)

<table>
<thead>
<tr>
<th>Resources &amp; physical deployment</th>
<th>Recording mechanisms to capture the availability and usage of resources for the services during the composition execution plan</th>
<th>low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design information</td>
<td>No need for provenance recording mechanisms</td>
<td>storage of application data, no performance impact</td>
</tr>
</tbody>
</table>

of questions. We aim to analyse whether provenance graphs for composite service specifications contain instances of individual or a composition of provenance question type patterns. This analysis would help designers to identify the types of questions that can be answered by a given service or application design.

Summary: In this chapter we presented a way to express provenance awareness as a non-functional requirement (NFR) focusing on the identification of questions we could use to define provenance awareness requirements for SOC systems. We also developed a faceted classification of the different kinds of questions using a set of provenance facets for expressing the provenance data requirements those questions could cover. Evaluation of the provenance facets with regards to the evaluation criteria of completeness, adequacy and realism is extensively discussed in Chapter 11. Before formalising our faceted questions into a set of provenance question type patterns, it is essential that we discuss the extended version of PROV-DM, into ServiceProv ontology, which will provide the provenance concepts, the syntax and semantics required for the provenance questions’ formalisation. Next, we present our research results towards a provenance data model, namely ServiceProv ontology, for SOC systems and compositions of services.
Chapter 7

A Provenance Data Model - ServiceProv

In this chapter, we present an extension of PROV ontology model (PROV-O) [14], namely ServiceProv ontology. ServiceProv ontology provides a domain specific language for representing the provenance data of services and service compositions that would need to be recorded for answering question exhibiting our provenance facets. This work has been previously published in [158].

Contents

7.1 Context and Motivation ................................. 155
7.2 A Provenance Data Model for SOC Systems ............. 158
  7.2.1 PROV-O Background .............................. 159
  7.2.2 ServiceProv Ontology by Example ................. 160
7.3 The ServiceProv Ontology ............................. 168
7.4 Extensibility vs Completeness .......................... 176

7.1 Context and Motivation

In the previous chapter we identified a set of provenance questions to express provenance awareness requirements of users for SOC systems and compositions of
services. We also proposed a set of semantically cohesive set of provenance question categories, as part of our provenance faceted classification scheme. Facets are a key component of our proposed approach, enabling the classification of the different kinds of questions – on the basis of the provenance data required to be captured and stored for answering those – so that the requirements of the users can be expressed in a generic and methodological way by combining facets as needed. Capturing provenance information is crucial for analysing provenance awareness of SOC system designs and service compositions; therefore it is important to provide the modelling notation and basic concepts for representing the provenance data that could be captured. Our proposed framework simulates provenance data graphs which are expressed by using a common graph syntax and schema derived from an extension of the PROV-O [14]. The extended model, namely ServiceProv ontology, introduces provenance concepts and relationships that map to the overall service composition model/design.

In our proposed framework specifications about the provenance data records of a SOC system are supported by an ontological (meta)model which enables reasoning and allows meaningful querying over provenance data instances – being an instantiation of a provenance data model (schema). This analysis is performed by executing a set of parameterized queries implemented through PROVa – as we will extensively discuss in Chapter 10 – where the queries derive from realising a set of provenance question type patterns that formalise questions exhibiting the provenance facets, as presented in Chapter 8. Provenance question type patterns are a formal abstraction of provenance data graphs, containing only those elements required to answer particular types of questions. However, before introducing this formalisation it is essential that we discuss the extended version of PROV-DM which will provide the data concepts, the syntax and semantics allowing us to formalise our faceted questions into a set of provenance question type patterns.

We introduce a conceptual data model for service provenance and compositions of services, (ServiceProv Ontology), which also provides a higher-level abstraction
ServiceProv ontology introduces the core domain concepts for the provenance of service-oriented (computing) systems; it is a provenance schema that:

1. allows capturing the provenance data relevant to the execution of a service-based system (such as provenance of atomic, orchestrated and choreographed services’ execution, provenance of service discovery and selection, provenance of service description publishing and registries, and resource and QoS provenance) and their relationships and,

2. allows designers to query and reason over provenance instances of particular composite service designs.

Such a data model for service provenance and compositions of services has seen very little research attention so far, with previous research in the area focusing primarily on the recording/storing and infrastructure mechanisms for provenance [101, 125, 149, 150]. Being an important element of our framework, as this was described in Chapter 4, we contribute the first schema that captures the structure of provenance data for service related processes (such as the service discovery, the service execution, service selection) and their properties (such as non-functional properties and resource availability). However, it is the provenance representation related to the aggregation and orchestration and choreography (so the composition of services) that gives an added-value and novelty to the proposition of this schema. Representing the structure of provenance data for service compositions is really challenging as it requires not only coping with the provenance data collection of each atomic service’s use but also with how each of them contributes in providing a connected picture of the composition’s processing history.

Existing work on representing provenance data [15, 16, 58] has mainly focused on workflow systems, but those models are not able to represent composition as in SOA(s) and SOC systems, where a collection of network available services
can be automatically discovered and integrated into applications. We built our provenance data model as an extension of the PROV-O ontology [14].

In the remainder of this chapter, we first give a short introduction to provenance and the PROV-Data Model (PROV-DM) [13]. Based on the composite service scenario (travel planner as presented in Section 6.1.1), in Section 7.2.2 we incrementally introduce the concepts of the provenance data model for service-oriented computing systems. We have expressed those concepts in OWL [73] as an extension of PROV-O ontology [14], details of which are given in Section 7.3. We also discuss the qualities and nature of our ontological model with regards to extensibility versus completeness in Section 7.4.

7.2 A Provenance Data Model for SOC Systems

In Chapter 5, we have presented an overall framework and research agenda towards a design and analysis framework for provenance awareness of service compositions. One of the main building blocks of our framework is a provenance data model for service-oriented (computing) systems; that is, a provenance schema that allows capturing the data provenance relevant to the execution of a SOC system and service compositions.

We present our proposed extension of the PROV-O ontology [14], containing all the concepts required for capturing provenance of service-based systems and service compositions. We have derived this ontology incrementally, by working through provenance questions exemplifying different provenance question type categories (as published in [156]) and modelling the information required for answering these questions.

We will first outline the foundations of both provenance in general and the PROV-DM [57] in particular.
7.2.1 PROV-O Background

“Provenance covers the information about entities, activities, or people involved in the process that produced a data item or thing” [13]. It can be used to understand how data was collected, to determine ownership and rights over the data or to verify that the process and steps used to obtain the data comply with given requirements. We can, thus, consider provenance to represent the ‘origin’ of a digital object [57].

PROV is the W3C’s “specification to express provenance records, which contains descriptions of the entities and activities involved in producing and delivering or otherwise influencing a given object” [57]. The PROV specification [13] covers different types of information that may be captured in provenance records, namely:

Activities represent processes that have occurred over a period of time and act upon entities.

Entities are digital, physical or conceptual things with some fixed values, that existed. Activities generated new entities and used existing entities. One entity may have been derived from another. Entities can be structured collections of entities.

Agents denote something that was responsible for an activity having taken place.

PROV also allows us to express the role played by an entity or agent in an activity, the time at which an entity was generated or used by an activity, the plan that was followed by an activity in execution and much more.

PROV-O ontology [14] is at the W3C “Recommendation” maturity level, forming an OWL2 [73] ontology allowing the mapping of the PROV-DM to RDF [91]. PROV-O contains a set of classes, properties, and restrictions allowing the representation of provenance information generated in different systems executing under different contexts. It can be specialised in order to create new classes
and properties to model provenance information in different (application) do-
 mains [14]. In this thesis we extend PROV-O’s concepts, introducing concepts
for representing provenance of service-based systems including service-oriented
(computing) systems. We are particularly interested in the concepts expressing
provenance for service compositions.

7.2.2 ServiceProv Ontology by Example

In this section, we discuss a number of provenance questions for the travel
planner scenario. Those questions have been chosen to cover the provenance
facets we have previously identified in Chapter 6. For each question, we provide
an example of provenance data for a run of the travel planner scenario. This allows
us to incrementally introduce and motivate the concepts required for capturing
provenance of service-oriented (computing) systems and service compositions.
Information about the complete ontology is provided in Section 7.3.

7.2.2.1 Provenance of Atomic Services

Before we discuss the provenance associated with service compositions, we in-
 troduce the basic notions required for capturing provenance of atomic services.
These correspond to the data needed for answering the following kinds of ques-
tions:

Q1.1: Which server executed the flight booking service? To be able to
answer this question, we need to capture data about the execution of the flight
booking service. We introduce the concepts :ServiceExecution\(^1\) (which are a sub-
type of prov: Activity), to capture the actual execution applicable to a specific
service, and the concept of :Service (a sub-type of prov: Entity) that represents
the service as a whole. Then :ServiceExecution links back to its respective :Ser-
vice using property :executionOf. The server executing the service can then be
captured as a prov:Agent that prov:wasAssociatedWith the service execution. To

\(^1\)We also introduce the concept of :AtomicServiceExecution as a sub-type of :ServiceExecution
to capture atomic services in particular within a composition.
distinguish servers from other agents, we introduce a new concept :Server, which is-a prov:SoftwareAgent. Note in particular that we consider servers to be pieces of running software rather than just machines: web services are typically executed by an application server infrastructure, which is what we capture with this concept. Consequently, a server can be identified by an IP address (using a :hasIPaddress property) and a port (using a :hasPort property). To distinguish this use of the server from other uses of the same server, we annotate the fact that it was playing the role of :ServiceExecutionServer. Figure 7.1 shows an example of the use of these concepts.

Q1.2: What were the parameters of the input/output messages for the flight booking service? We can model input and output messages as entities that were, respectively, used (prov:used) and generated (prov:wasGeneratedBy) by a :ServiceExecution. To capture the structure of web-service messages, we introduce the concept :Message, which is-a prov:Collection of :Parameters. In Figure 7.1 we can see, for example, that the service was provided with an input value of “Paris” for its destination parameter.

7.2.2.2 Provenance of Service Composition

When a composed service, like the travel planner service, is executed, there is an orchestrator that manages the execution of the overall work plan, triggering execution of the individual services as required [4]. All communication between services is moderated via the orchestrator (we are not currently considering tech-
niques that separate control and data flow in workflow execution e.g., [42]). To capture provenance of orchestrated composite services, we thus need to capture data about the execution of this orchestrator and the message flow between it and the individual services.

Q2.1: What orchestrator executed the flight and hotel booking services of the travel planner? We model the execution of an orchestrator as an :OrchestratorExecution, which is-a prov:ServiceExecution. The :Server that prov:wasAssociatedWith an :OrchestratorExecution then is the service orchestrator for the composite service. We express this by annotating a role of :Orchestrator to the association relationship prov:wasAssociatedWith. Figure 7.2\(^2\) shows an example of provenance data for (part of) an orchestrated service run for the travel planner.

Q2.2: How were the input messages of the flight and hotel booking services derived during the travel planner execution? As stated above, the orchestrator manages execution of the individual services and moderates all communication. We can model this by saying that all input messages of atomic services were generated by the :OrchestratorExecution and all output messages were used by it. In addition, we record prov:wasDerivedFrom links between output messages and input messages of services to indicate that the orchestrator passes on the output message of one service as an input message of another service. Figure 7.2 shows this for some of the services invoked as part of the travel planner.

\(^2\)This figure like all following figures uses the same meaning of symbols as introduced in Figure 7.1.
So far, we have only discussed service composition by orchestration. Services can also be composed by choreography. In terms of provenance data, this equals to services directly invoking each other, with the root service execution standing in for the entire composite service where required. This can be modelled by directly linking :ServiceExecution(s) via :Message(s) and appropriate usage and generation properties, as shown in Figure 7.3.

7.2.2.3 Provenance of Service Discovery and Selection

An important aspect of SOA is the idea of dynamic service discovery; that is, the ability to find, at runtime, a service that implements a given service specification. To realise this, service providers register services with a central registry service [41]. This registry can then be used to discover services based on a service request. Understanding which particular services were available during a run of a service composition and which of those were selected is an important aspect of service provenance. Figure 7.4 shows an example of provenance data from a run of the travel planner.

Information about services is held in a registry in the form of service descriptions. These can take a number of formats, such as WSDL [107], OWL-S [98] or RDF [91]. To keep our provenance data model open to different existing and future formats, we introduce the concept of a :ServiceDescription, as shown in Figure 7.5, which is-a prov:Collection, whose members are entities that identify a specific service description (e.g., a WSDL file) by its URI. To model those members we introduce the :ServiceDescriptionItem concept and reuse the existing prov:hadMember property relationship.

Q3.1: Which flight booking services were available at the time of booking? Questions like this one ask for the service descriptions that were dis-
covered. To capture this, we introduce :ServiceDiscovery, which is-a prov:Activity. :ServiceDiscovery uses a :ServiceRequest (a subtype of prov:Entity) describing the service required [2] and generates a collection (:Service Collection) of service descriptions of the services discovered as shown in Figure 7.4.

Q3.2: Which were the service providers of the discovered services? Service providers are the servers which register a service with the registry. We model this through a prov:Activity subtype :ServiceInterfacePublishing, which generates a :ServiceDescriptionPublishedItem. This is modelled as a specialisation of the :ServiceDescriptionItem concept which represents a service description in general. Figure 7.5 shows an example that covers those terms for the flight-booking service.

Q3.3: Which registry/ies was/were involved in the discovery of the flight booking services? A registry is a :Server that prov:hadRole :Registry in a qualified attribution with a :ServiceDescriptionDiscoveredItem. This is also a specialization of the more general :ServiceDescriptionItem term and derives from the corresponding :ServiceDescriptionPublishedItem. Figure 7.5 shows this for a flight booking service description that was published and discovered as part of
Q3.4: Which broker was used in the service discovery of the flight booking service? This is a :Server that prov:wasAssociatedWith a :ServiceDiscovery and prov:hadRole a :ServiceBroker. We show an example of this in Figure 7.4 for the discovery of the hotel booking service.

Q3.5: Which flight booking service was selected to be invoked? To answer this question we model both the :ServiceSelection (a type of prov:Activity) and the :Server that prov:wasAssociatedWith this (playing the role of :Aggregator). :ServiceSelection prov:generated the :SelectedServiceDescription, which prov:wasDerivedFrom a :ServiceDescription (a type of prov:Entity) previously discovered. Note that this is different from capturing which services where actually executed. Modelling service selection explicitly provides the opportunity to model situations where services are selected initially, but then are not executed (for example, because they became unavailable in the meantime).

Q3.6: What were the reasons the specific flight booking service was selected? Selection uses :QoSRequirements (described in e.g., WSMO [141] or OWL-Q [89]) to choose between otherwise functionally equivalent services. Modelling these as an additional input to :ServiceSelection, together with capturing QoS properties as part of the :ServiceDescription provides rationale for the service selection.
To tie in service selection and discovery with service execution, we need to model an additional prov:Activity :Planning, which is executed before orchestration and determines the actual plan to be executed. In service-oriented (computing) systems planning corresponds to the procedure where a composer/aggregator does the mapping from a user’s request in order to form an abstract process plan of the user’s requirements considering only the functional aspects (having no consideration for QoS requirements) [153]. In our provenance data model :Planning prov:generated a :ServiceRequest using an :AbstractProcessPlan to drive the services’ discovery. This is different from the :ServiceSelection activity, where the QoS requirements are taken into consideration. In this way :Planning enables the binding between the discovered and selected services in order to generate a :WorkPlan to be used by the :OrchestratorExecution. This, together with how it connects to the orchestration flow and the other activities can be seen in Figure 7.4.

7.2.2.4 QoS Provenance

Quality of Service (QoS) is an important aspect of service-oriented (computing) systems. There may be questions about the QoS actually delivered by a service. Thus, capturing QoS information as part of the service provenance is useful.

Q4.1: What was the response time of the hotel booking service?

With QoS properties such as response time that are applicable directly to a single execution of a specific service, we capture an individual of an appropriate sub-type of :NFP directly attached to the :ServiceExecution via a :hadNFP property. The left-hand side of Figure 7.6 shows an example of such provenance data. Those qualities are either generated by monitoring mechanisms that keep track of the system’s/service’s non-functional qualities and performance metrics or refer to QoS descriptions/specifications of the atomic and composite services created by the service owners/providers and distributed by the service brokers/aggregators. Performance of those values is not something that we were looking to address.
Q4.2: What was the reliability of the hotel booking service? With aggregate properties like reliability, which are applicable to a number of executions of a specific service, we capture an individual of an appropriate sub-type of :NFP attached to a :Service entity, which represents the service as a whole, perhaps focused on a particular period of time. Individual :ServiceExecution(s) link back to their respective :Service using property :executionOf. The right-hand side of Figure 7.6 shows an example. The :Service concept can also be used to record or analyse other properties that must be aggregated over the (partial) history of service executions.

Q4.3: What was the QoS offered by the flight booking service? This question is not about the QoS actually observed during an execution of the flight booking service, but about the QoS it promised its users (and on the basis of which it may have been selected). This can be captured as part of the service description, being just another :ServiceDescriptionItem that is a member of the appropriate :ServiceDescription of a :Service.

In order to answer questions about performance etc., we may require information about time stamps of start and completion of a service execution or about service invocation. These can already be captured by standard PROV-O notions such as prov:startedAtTime, prov:endedAtTime, or prov:atTime. For example in Figure 7.7 we use these time-related concepts to represent the time of start and completion of the flight booking service execution or the time the input message for this service was generated.
7.2.2.5 Resource Provenance

Closely related to the provenance of QoS is information about the resources that were available to a service execution.

Q5.1: What resources were available for the execution of the flight booking service? We represent available resources by saying that :ServiceExecution :usedResource a :ResourceCollection. The members of a :ResourceCollection are a number of :Resource(s). We reuse the resource computing concepts of the SEALS ComputingResourceOntology [25] to represent specific resource types such as SEALS: CPU, SEALS: Memory etc.

7.3 The ServiceProv Ontology

In the previous section, we have discussed a number of potential provenance questions one may wish to ask of a service-oriented (computing) system and what provenance concepts would be required to answer these questions. We have
defined an ontology for service provenance and compositions of services as an extension of PROV-O [14] that precisely defines all of these concepts. While we have used the empty namespace for our concepts in the examples given above, the actual namespace IRI is https://sourceforge.net/projects/serviceprov/files/serviceprov# with the prefix serviceprov. The ontology can be obtained on-line at https://sourceforge.net/projects/serviceprov/files/ServiceProvOntology/; a detailed definition of each concept and property is provided in Appendix 14. Next, we present a more systematic summary of the concepts introduced in the ServiceProv ontology.

Activities Extension

We extend the prov:Activity class with a number of subtype concepts in Tables 7.1 and 7.2, inheriting properties such as prov:used, prov:wasGeneratedBy, prov:wasAssociatedWith. We are particularly interested in conceptualizing service related types of activities such as ServiceExecution, OrchestratorExecution, ServiceInterfacePublishing, ServicePlanning, ServiceSelection etc., in order to represent the provenance data required to answer provenance questions about them.

Entities Extension

In our proposed provenance data model we extend the prov:Entity class with a number of concepts as those are defined in Tables 7.4 and 7.5. An entity may form a prov:Collection of other entities that provides a structure of some constituent, called members. Therefore, we say that a prov:Entity prov:hadMember some other prov:Entity. In our provenance data model we introduce a number of concepts as types of prov:Collection to represent the terms of Message, Service Collection or Resource Collection.

Agents Extension

In our proposed provenance model we extend the class prov:Agent with the sub-class concept of Server as described in Table 7.3. Agent forms a subtype of
prov:SoftwareAgent, bearing some form of responsibility for running software, that may play a number of roles in a qualified association with a type of prov:Activity. We define those roles, such as those of the ServiceProvider or the ServiceBroker, as shown in Table 7.6.

**Roles Extention**

We extend the prov:Role class with the following types, shown in Table 7.6.
<table>
<thead>
<tr>
<th>Class Definition</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>serviceprov:ServiceExecution</strong></td>
<td>prov:used</td>
</tr>
<tr>
<td><strong>serviceprov:ServiceExecution</strong></td>
<td>prov:wasGeneratedBy</td>
</tr>
<tr>
<td><strong>serviceprov:ServiceExecution</strong></td>
<td>prov:startedAtTime</td>
</tr>
<tr>
<td><strong>serviceprov:ServiceExecution</strong></td>
<td>prov:endedAtTime</td>
</tr>
<tr>
<td><strong>serviceprov:ServiceExecution</strong></td>
<td>prov:wasAssociatedWith</td>
</tr>
<tr>
<td><strong>serviceprov:AtomicServiceExecution</strong></td>
<td></td>
</tr>
<tr>
<td><strong>serviceprov:AtomicServiceExecution</strong></td>
<td></td>
</tr>
<tr>
<td><strong>serviceprov:OrchestratorExecution</strong></td>
<td></td>
</tr>
<tr>
<td><strong>serviceprov:OrchestratorExecution</strong></td>
<td></td>
</tr>
<tr>
<td><strong>serviceprov:ServiceInterfacePublishing</strong></td>
<td>prov:used</td>
</tr>
<tr>
<td><strong>serviceprov:ServiceInterfacePublishing</strong></td>
<td>prov:wasGeneratedBy</td>
</tr>
<tr>
<td><strong>serviceprov:ServiceInterfacePublishing</strong></td>
<td>prov:startedAtTime</td>
</tr>
<tr>
<td><strong>serviceprov:ServiceInterfacePublishing</strong></td>
<td>prov:endedAtTime</td>
</tr>
<tr>
<td><strong>serviceprov:ServiceInterfacePublishing</strong></td>
<td>prov:wasAssociatedWith</td>
</tr>
<tr>
<td><strong>serviceprov:ServiceDiscovery</strong></td>
<td>prov:used</td>
</tr>
<tr>
<td><strong>serviceprov:ServiceDiscovery</strong></td>
<td>prov:wasGeneratedBy</td>
</tr>
<tr>
<td><strong>serviceprov:ServiceDiscovery</strong></td>
<td>prov:startedAtTime</td>
</tr>
<tr>
<td><strong>serviceprov:ServiceDiscovery</strong></td>
<td>prov:endedAtTime</td>
</tr>
<tr>
<td><strong>serviceprov:ServiceDiscovery</strong></td>
<td>prov:wasAssociatedWith</td>
</tr>
<tr>
<td><strong>serviceprov:Planning</strong></td>
<td>prov:used</td>
</tr>
<tr>
<td><strong>serviceprov:Planning</strong></td>
<td>prov:wasGeneratedBy</td>
</tr>
<tr>
<td><strong>serviceprov:Planning</strong></td>
<td>prov:startedAtTime</td>
</tr>
<tr>
<td><strong>serviceprov:Planning</strong></td>
<td>prov:endedAtTime</td>
</tr>
<tr>
<td><strong>serviceprov:Planning</strong></td>
<td>prov:wasAssociatedWith</td>
</tr>
</tbody>
</table>

Table 7.1: Service Provenance Model Activity Type Concepts (I)

- **serviceprov:ServiceExecution**: is a `prov:Activity` that captures the actual execution of a service.
- **serviceprov:AtomicServiceExecution**: is a `serviceprov:ServiceExecution` that captures the actual execution of an atomic service.
- **serviceprov:OrchestratorExecution**: is a `serviceprov:ServiceExecution` that captures the actual execution of an orchestrator. It represents the execution of a central process which takes control over the involved web services in a business process and coordinates the exchange of input and output messages of each atomic service execution in this process. A `serviceprov:OrchestratorExecution` `prov:used` some `serviceprov:WorkPlan` or some other output `serviceprov:Message`. Then an input `serviceprov:Message` `prov:wasGeneratedBy` the `serviceprov:OrchestratorExecution`.
- **serviceprov:ServiceInterfacePublishing**: is a `prov:Activity` that represents service provider’s publishing of `serviceprov:ServiceDescriptionPublishedItem`s (including their functional and non-functional description or related policies) to a `serviceprov:Server` that plays the role of the `serviceprov:Registry`.
- **serviceprov:ServiceDiscovery**: is a `prov:Activity` that represents the discovery of a collection (serviceprov:ServiceCollection) of service descriptions (serviceprov:ServiceDescription) that meet certain functional criteria with the goal to find an appropriate web service to satisfy the requester’s needs. (serviceprov:ServiceDiscovery `prov:used` serviceprov:ServiceRequest). A discovery service facilitates the process of performing discovery. It is a logical role performed by a `serviceprov:Server` that plays the role of the `serviceprov:ServiceBroker`.
- **serviceprov:Planning**: is a `prov:Activity` that represents the actual plan to be executed. In service-based systems it corresponds to the procedure where a composer/aggregator performs the mapping between a user request and a set of user requirements to form an abstract process plan considering mainly the functional aspects of services.
### Table 7.2: Service Provenance Model Activity Type Concepts (II)

<table>
<thead>
<tr>
<th>Class Definition</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>In this way the <code>serviceprov:Planning</code> <code>prov:generated</code> a <code>serviceprov:ServiceRequest</code> based on the use of the <code>serviceprov:AbstractProcessPlan</code> to enable the service’s discovery. This is different from the <code>serviceprov:ServiceSelection</code> activity, where the QoS requirements are taken into consideration. <code>serviceprov:Planning</code> enables the mapping of the discovered and selected services in order to generate a <code>serviceprov:WorkPlan</code> that will be <code>prov:used</code> by the <code>serviceprov:OrchestratorExecution</code>.</td>
<td><code>prov:used</code>&lt;br&gt;<code>prov:wasGeneratedBy</code>&lt;br&gt;<code>prov:startedAtTime</code>&lt;br&gt;<code>prov:endedAtTime</code>&lt;br&gt;<code>prov:wasAssociatedWith</code></td>
</tr>
<tr>
<td><code>serviceprov:ServiceSelection</code>: is-a <code>prov:Activity</code> that represents the selection of functionally equivalent service candidates, discovered based on the user’s <code>serviceprov:QoS</code> requirements. <code>serviceprov:ServiceSelection</code> <code>prov:used</code> <code>serviceprov:ServiceCollection</code> and <code>prov:generated</code> a <code>serviceprov:SelectedServiceDescription</code>.</td>
<td><code>prov:used</code>&lt;br&gt;<code>prov:wasGeneratedBy</code>&lt;br&gt;<code>prov:startedAtTime</code>&lt;br&gt;<code>prov:endedAtTime</code>&lt;br&gt;<code>prov:wasAssociatedWith</code></td>
</tr>
</tbody>
</table>

### Table 7.3: Service Provenance Model Agent Type Concepts

<table>
<thead>
<tr>
<th>Class Definition</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>serviceprov:Server</code>: is-a <code>prov:SoftwareAgent</code> that we consider as a piece of running software rather than just a machine: web services are typically executed by an application-server; which is what we actually capture with this term. It may be associated with the execution of a number of service related activities such as: <code>serviceprov:ServiceExecution</code>, <code>serviceprov:OrchestratorExecution</code>, <code>serviceprov:ServiceSelection</code>, <code>serviceprov:ServiceDiscovery</code>, <code>serviceprov:ServicePlanning</code> as those are defined in Tables 7.1 and 7.2. For each of them <code>serviceprov:Server</code> played different types of roles as those are defined in Table 7.6. <code>serviceprov:Server</code> can be identified by its data properties <code>serviceprov:hadIPaddress</code> and <code>serviceprov:hadIPPort</code> while it inherits the properties of <code>serviceprov:SoftwareAgent</code>.</td>
<td><code>prov:wasAssociatedWith</code>&lt;br&gt;<code>serviceprov:hadIPaddress</code>&lt;br&gt;<code>serviceprov:hadIPPort</code>&lt;br&gt;<code>prov:hadRole</code></td>
</tr>
</tbody>
</table>
### Table 7.4: Service Provenance Data Model Entity Type Concepts (I)

<table>
<thead>
<tr>
<th>Class Definition</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>serviceprov:Message: is-a prov:Collection of service-</td>
<td>prov:wasGeneratedBy prov:used prov:wasDerivedFrom</td>
</tr>
<tr>
<td>prov:Parameter(s) used to model the input and output</td>
<td>prov:hadMember</td>
</tr>
<tr>
<td>messages as entities that were, respectively, used and</td>
<td></td>
</tr>
<tr>
<td>generated by a serviceprov:ServiceExecution.</td>
<td></td>
</tr>
<tr>
<td>serviceprov:Parameter: is-a type of prov:Entity that</td>
<td>prov:hadMember</td>
</tr>
<tr>
<td>represents the input or output data parameters as</td>
<td>prov:value</td>
</tr>
<tr>
<td>members of serviceprov:Message that were prov:used or</td>
<td></td>
</tr>
<tr>
<td>prov:generatedBy the serviceprov:ServiceExecution.</td>
<td></td>
</tr>
<tr>
<td>serviceprov:ServiceCollection: is-a prov:Collection</td>
<td>prov:wasGeneratedBy prov:used</td>
</tr>
<tr>
<td>of service descriptions that prov:wasGeneratedBy the</td>
<td>prov:hadMember</td>
</tr>
<tr>
<td>serviceprov:ServiceDiscovery.</td>
<td></td>
</tr>
<tr>
<td>serviceprov:ServiceDescription: is-a type of prov:</td>
<td>prov:wasGeneratedBy prov:used</td>
</tr>
<tr>
<td>Collection that represents information about service’s</td>
<td>prov:hadMember</td>
</tr>
<tr>
<td>functional and non-functional properties; it can take</td>
<td></td>
</tr>
<tr>
<td>a number of formats such as WSDL [107] or OWL-S [98].</td>
<td></td>
</tr>
<tr>
<td>We model those members with the serviceprov:Service</td>
<td></td>
</tr>
<tr>
<td>DescriptionItem concept. serviceprov:ServiceDescription forms a member of the</td>
<td></td>
</tr>
<tr>
<td>serviceprov:ServiceCollection which prov:wasGenerat-</td>
<td></td>
</tr>
<tr>
<td>edBy a serviceprov:ServiceDiscovery activity.</td>
<td></td>
</tr>
<tr>
<td>serviceprov:ServiceDescriptionItem: is-a prov:Entity</td>
<td>prov:wasGeneratedBy prov:used prov:value</td>
</tr>
<tr>
<td>member of serviceprov:ServiceDescription collection.</td>
<td></td>
</tr>
<tr>
<td>It represents possible members of a service description</td>
<td></td>
</tr>
<tr>
<td>such as a URI or a WSDL functional description. QoS</td>
<td></td>
</tr>
<tr>
<td>offered by the services in the form of QoS policies</td>
<td></td>
</tr>
<tr>
<td>(e.g., WS-Security Policy) are also captured by the</td>
<td></td>
</tr>
<tr>
<td>serviceprov:ServiceDescriptionItem concept.</td>
<td></td>
</tr>
<tr>
<td>serviceprov:ServiceDescriptionPublishedItem:</td>
<td>prov:wasGeneratedBy prov:used prov:specializa-</td>
</tr>
<tr>
<td>is-a prov:specializationOf the servicepro-</td>
<td>tionOf</td>
</tr>
<tr>
<td>v:ServiceDescriptionItem entity. It represents possi-</td>
<td></td>
</tr>
<tr>
<td>ble service description items published by a providers</td>
<td></td>
</tr>
<tr>
<td>to web registries</td>
<td></td>
</tr>
<tr>
<td>serviceprov:ServiceDescriptionDiscoveredItem:</td>
<td>prov:wasGeneratedBy prov:used prov:specializa-</td>
</tr>
<tr>
<td>is-a prov:specializationOf the servicepro-</td>
<td>tionOf</td>
</tr>
<tr>
<td>v:ServiceDescriptionItem entity. It represents possi-</td>
<td></td>
</tr>
<tr>
<td>ble service description items discovered by the servi-</td>
<td></td>
</tr>
<tr>
<td>ceprov:ServiceDiscovery.</td>
<td></td>
</tr>
<tr>
<td>serviceprov:Service: is-a prov:Entity that represents</td>
<td>prov:value</td>
</tr>
<tr>
<td>a particular service provided over a given period in</td>
<td>prov:generatedAtTime prov:invalidatedAtTime</td>
</tr>
<tr>
<td>the past where the period of time is denoted by the</td>
<td>serviceprov:executionOf serviceprov:hadNFP</td>
</tr>
<tr>
<td>prov:generatedAtTime and prov:invalidatedAtTime prop-</td>
<td>serviceprov:hadDescription serviceprov:serviceName</td>
</tr>
<tr>
<td>erties. This may link to a number of past servicepro-</td>
<td></td>
</tr>
<tr>
<td>v:ServiceExecutions using serviceprov:executionOf</td>
<td></td>
</tr>
<tr>
<td>property, or associate to NFP properties through</td>
<td></td>
</tr>
<tr>
<td>Class Definition</td>
<td>Properties</td>
</tr>
<tr>
<td>---------------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>serviceprov:ServiceRequest: is-a prov:Entity that represents the description of</td>
<td></td>
</tr>
<tr>
<td>the functional requirements request prov:generatedBy the serviceprov:ServicePlanning</td>
<td></td>
</tr>
<tr>
<td>activity. It was prov:used by the serviceprov:ServiceDiscovery to generate a</td>
<td></td>
</tr>
<tr>
<td>collection of service descriptions to satisfy those requirements.</td>
<td></td>
</tr>
<tr>
<td>serviceprov:QoSRequirements: is-a prov:Entity representing the desirable quality</td>
<td></td>
</tr>
<tr>
<td>of service characteristics for a service to be discovered.</td>
<td></td>
</tr>
<tr>
<td>serviceprov:SelectedServiceDescription: is-a prov:Collection that represents the</td>
<td></td>
</tr>
<tr>
<td>selected service descriptions that were generated by the serviceprov:ServiceSelection</td>
<td></td>
</tr>
<tr>
<td>activity based on a number of QoSRequirements.</td>
<td></td>
</tr>
<tr>
<td>serviceprov:WorkPlan: is-a prov:Plan that represents the actual plan to be</td>
<td></td>
</tr>
<tr>
<td>executed. It is an executable business process description (e.g., WS-BPEL</td>
<td></td>
</tr>
<tr>
<td>description) of services considering their functional and non-functional</td>
<td></td>
</tr>
<tr>
<td>properties.</td>
<td></td>
</tr>
<tr>
<td>serviceprov:AbstractProcessPlan: is-a prov:Plan that represents an abstract</td>
<td></td>
</tr>
<tr>
<td>process plan (e.g., BPMN description) that specifies the control flow of</td>
<td></td>
</tr>
<tr>
<td>services considering only the functional requirements of them. This was</td>
<td></td>
</tr>
<tr>
<td>prov:used by the serviceprov:ServicePlanning to generate a</td>
<td></td>
</tr>
<tr>
<td>serviceprov:ServiceRequest.</td>
<td></td>
</tr>
<tr>
<td>serviceprov:NFP: represents a non-functional property value of a</td>
<td></td>
</tr>
<tr>
<td>serviceprov:ServiceExecution such as :Performance (:Throughput, :ResponseTime,</td>
<td></td>
</tr>
<tr>
<td>serviceprov:Cost, :Security, or of a serviceprov:Service, for a number of its</td>
<td></td>
</tr>
<tr>
<td>executions in the past, such as :Reliability (:MMTF,:MTBF), :Dependability</td>
<td></td>
</tr>
<tr>
<td>(:Availability, :Robustness, :Accuracy, :ReliableMessaging) and :Trust.</td>
<td></td>
</tr>
<tr>
<td>serviceprov:ResourceCollection: is-a prov:Collection of the available</td>
<td></td>
</tr>
<tr>
<td>serviceprov:Resources prov:used during serviceprov:ServiceExecution.</td>
<td></td>
</tr>
<tr>
<td>serviceprov:Resource: represents a member of the serviceprov:ResourceCollection</td>
<td></td>
</tr>
<tr>
<td>made available for a Service-Execution. It contains a number of subtypes such as</td>
<td></td>
</tr>
<tr>
<td>CPU, Storage (Memory and Disk), NetworkAdapter, OperatingSystem, concepts being</td>
<td></td>
</tr>
<tr>
<td>reused as part of the SEALS Computing Resource Ontology.</td>
<td></td>
</tr>
</tbody>
</table>
Table 7.6: Service Provenance Data Model Role Type Concepts

<table>
<thead>
<tr>
<th>Class Definition</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>serviceprov:ServiceExecutionServer:</strong> is-a prov:Role that captures the role</td>
<td>serviceprov:wasRoleIn</td>
</tr>
<tr>
<td>played by a serviceprov:Server in a qualified association with the</td>
<td></td>
</tr>
<tr>
<td>serviceprov:ServiceExecution of a service.</td>
<td></td>
</tr>
<tr>
<td><strong>serviceprov:ServiceProvider:</strong> is-a prov:Role that captures the role played</td>
<td>serviceprov:wasRoleIn</td>
</tr>
<tr>
<td>by a serviceprov:Server in a qualified association with the</td>
<td></td>
</tr>
<tr>
<td>serviceprov:ServiceInterfacePublishing of a serviceprov:ServiceDescriptionPublishedItem through which a</td>
<td></td>
</tr>
<tr>
<td>service is published and made discoverable.</td>
<td></td>
</tr>
<tr>
<td><strong>serviceprov:Orchestrator:</strong> is-a prov:Role that captures the role played by</td>
<td>serviceprov:wasRoleIn</td>
</tr>
<tr>
<td>a serviceprov:Server in a qualified association with the serviceprov:Orchestrator</td>
<td></td>
</tr>
<tr>
<td>Execution.</td>
<td></td>
</tr>
<tr>
<td><strong>serviceprov:Aggregator:</strong> is-a prov:Role that captures the role played by a</td>
<td>prov:wasRoleIn</td>
</tr>
<tr>
<td>serviceprov:Server in a qualified association with both the</td>
<td></td>
</tr>
<tr>
<td>serviceprov:ServiceSelection and the serviceprov:Planning types of prov:Activities. It discovers and groups services that are provided by other service providers into a distinct value added service. Service aggregators may become service providers by publishing the composite service descriptions they create.</td>
<td></td>
</tr>
<tr>
<td><strong>serviceprov:Registry:</strong> is-a type of prov:Role that represents the role played</td>
<td>prov:wasRoleIn</td>
</tr>
<tr>
<td>by a serviceprov:Server in a qualified attribution with a serviceprov:ServiceDescriptionDiscoveredItem sent in a message by the registry in response to a query for that description. It maintains a number of service descriptions which were made available to the registry through the serviceprov:ServiceInterfacePublishing activity.</td>
<td></td>
</tr>
<tr>
<td><strong>serviceprov:ServiceBroker:</strong> is-a type of prov:Role that represents the role</td>
<td>prov:wasRoleIn</td>
</tr>
<tr>
<td>played by a serviceprov:Server in a qualified association with the</td>
<td></td>
</tr>
<tr>
<td>serviceprov:ServiceDiscovery process. It prov:used a serviceprov:ServiceRequest and prov:generated a serviceprov:ServiceCollection of service descriptions.</td>
<td></td>
</tr>
</tbody>
</table>
7.4 Extensibility vs Completeness

The proposed ontology can be used to capture data required for answering the kinds of questions from which we have derived it. This, of course, raises a potential attack: Have we indeed covered all relevant questions? In our derivation here, we have discussed questions from each of the provenance faceted categories identified in Chapter 6, except for the categories on routes not followed and design information. Omitting those two categories does not cause any problems, as they do not require any additional information to be collected at runtime. Instead, they rely on the existence of design-time information and an ability to correlate this with runtime data. Having both pieces of this information – design information and the actual data flow captured during service execution – is sufficient in order to calculate the delta of all the specified paths/routes that could be taken at design time versus those that were actually taken during the service (composition) execution. This delta allows us to answer questions about the routes/branches not taken during the service composition execution.

By building this conceptual provenance model our intention is not to claim completeness of the ontology in the coverage of any provenance questions and related concepts. Although we have targeted into adequately covering the core concepts and principles of SOC and SOA state of the art and updated literature, ontologies are built in order to form a formal representation of the structure of data for a particular domain of interest, aiming for extensibility and reuse of other existing ontologies and not being used as prescriptive models. Therefore, our proposed ServiceProv ontology is naturally extensible, similarly to its starting point PROV-O [14], while it re-uses concepts from the SEALS [25] ontology for the provenance resources representation. In this way our ontological model enables composite service designers 1) to extend ServiceProv ontology in order to capture provenance of service related properties that may emerge with the evolution of service-oriented (computing) system’s infrastructure and architecture, 2) to extend ServiceProv ontology in order to cover provenance requirements in terms of the specialized infrastructure the composite service design they are interested in.
modelling may have.

**Summary:** In this chapter we presented the provenance data concepts and semantics of ServiceProv ontology – an extension of PROV-O ontology that enables the representation of provenance data records for SOC systems using a common graph syntax (RDF [91], OWL [73]). Evaluation of the proposed provenance data model with regards the criteria of conceptual adequacy/expressivity, reasoning and realism is presented in detail in Chapter 11. Next, we formalise the questions exhibiting our provenance facets into a set of provenance question type patterns considering the concepts introduced in ServiceProv ontology. ServiceProv concepts express the provenance information required to be recorded for answering any set of service related provenance questions.
Chapter 8

Provenance Question Type Patterns

In the previous chapter we presented a provenance data model to enable the representation of provenance data for SOC systems and compositions of services – required to be recorded for answering provenance questions – in a graph form. In this chapter, we identify a set of provenance question type patterns that allows us to formally express provenance questions (provenance awareness requirements) exhibiting a number of facets as proposed in Chapter 6. Expressing those requirements in a generic way as question type patterns allows us to build a set of reusable metrics for analysis and reasoning over provenance data instances; those metrics can then be applied in different application contexts to verify provenance awareness support of composite service systems. We have derived those patterns through the exploitation of a number of provenance questions/queries for our composition use case scenario (travel planner), by identifying the assertions that need to be made with regards to the provenance data records required to answer potential provenance questions for the travel planner’s execution.

Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>Context and Motivation</td>
<td>179</td>
</tr>
<tr>
<td>8.2</td>
<td>Provenance Graphs and Templates</td>
<td>181</td>
</tr>
<tr>
<td>8.3</td>
<td>Provenance Question Type Patterns</td>
<td>185</td>
</tr>
</tbody>
</table>
8.1 Context and Motivation

Modelling both the provenance data records and provenance awareness requirements of a SOC system is essential for analysing the system’s provenance awareness at system’s design time. We have expressed requirements about provenance awareness of a SOC system as a set of provenance questions/queries about the provenance data that would be generated, derived and processed at system’s execution time. To analyse whether the system is able to answer particular provenance questions, we would also need to model the recorded and stored provenance data such as the message contents and service identifiers, as discussed in Chapter 7. Different questions require different kinds and levels of detail, and it depends on what the user will or is likely to require as to what provenance data records should be modelled. In Chapter 6 we have examined and proposed nine provenance facets, where each facet has been defined intuitively by the kinds of data that are required to be captured and stored during a composition’s execution.

To enable the formalisation of the proposed facets we introduce a number of provenance question type patterns for each of the different types of questions exhibiting those facets, on the basis of the provenance data that is required to be captured. Expressing the assertions (about elements and concepts) that are
required to be made for answering different types of questions in a generic way as provenance question type patterns, allows 1) QoS analysts to built a reusable solution for provenance awareness analysis and 2) software/service developers to use those patterns as benchmarks in order to measure and assess provenance awareness as an NFP. We then form a catalogue of solution provenance question type patterns, expressed in a graph form, that could be applied to different SOC systems and application contexts to analyse their provenance awareness support.

In practice this analysis is about verifying whether provenance as part of composite service specifications represented as a set of provenance data graphs – forming model instances of ServiceProv ontology – can be captured by an individual or a composition of our provenance question type patterns expressed as graph patterns. Provenance question type patterns represented as graph-like constructs express conceptually and semantically the data structures required to be captured for answering a set of provenance questions in a graph form. Then using a standard query notation, such as SPARQL query language, is about the assertions we need to make to realise the former as part of a provenance awareness analysis environment (and toolset). Therefore, SPARQL queries will enable the analysis that allows QoS analysts, service and infrastructure designers to identify the types of questions that can be answered by the given system design by checking against the composite service system’s provenance properties’ specification.

Our solution for formally expressing those patterns was to turn the higher-level abstraction of provenance data graph(s) into provenance template(s). The template construct then specifies the basic pattern of provenance data that a composite service or SOC system may record. Templates are parameterizable model elements, therefore they can provide an abstracted view of the provenance data domain concepts and structures for SOC systems that enable us to build and analyse provenance properties (including the provenance data structure captured and infrastructure) independently of specific applications. In particular, in Chapter 8 we have introduced and discussed the domain concepts and relationships of our ontology-based service provenance data model, namely ServiceProv.
Exhibiting the practical use of how these concepts link, interwork and provide a meaningful semantics for the provenance of a composite service travel planner scenario, we have also identified a number of domain concepts and their relations, repeated several times within a provenance graph structure (repetition patterns containing fragments as shown in Figure 8.2).

Next, we give a full definition of provenance templates and how those are expressed using a graph and a query notation. We are going to use the terms patterns and templates interchangeably; yet it is important to clarify that, patterns are a tool more useful in the context of analysis and benchmarking closer to the understanding of developers or QoS analysts that want to apply analytical metrics in order to measure provenance awareness, while templates are more useful within the context of data modelers, architects and provenance data infrastructure designers allowing the parameterization of the data concepts we need to capture for a set of provenance awareness requirements, independently of specific applications.

8.2 Provenance Graphs and Templates

In the previous chapter, we described a domain-specific extension of PROV-DM for SOC systems and compositions of services, namely ServiceProv ontology.

PROV-DM has a modular design and is structured according to six components covering various facets of provenance:

- component 1: entities and activities, and the time at which they were created, used, or ended;
- component 2: derivations of entities from others;
- component 3: agents bearing responsibility for entities that were generated and activities that happened;
- component 4: bundles, a mechanism to support provenance of provenance;

...
• component 5: properties to link entities that refer to the same thing;

• component 6: collections forming a logical structure for their members.

A provenance graph can be derived from the three basic concepts as described in PROV-DM, namely entity, activity and agent, and their causal relationships to describe provenance information. In particular, in a provenance graph instance, based on PROV-DM, activities, entities, and agents are represented as nodes, while the causal relationships between them are mapped as directed edges. For example, if an entity E was generated by an activity A, two provenance nodes E and A are defined, and an edge annotated by the “prov:wasGeneratedBy” relationship connects E to A. Similarly, if an entity E was consumed by an activity A, an edge annotated by the “prov:used” relationship connects A to E. Both activity and entity nodes may have multiple incoming and outgoing edges. When tracking the provenance information of an entity, the provenance node associated with the entity is used as the origin for provenance graph exploration.

A provenance template of an entity is an outline of the provenance graph structure for the entity, without the instance information about the actual entity and its associated activities. We express provenance templates with a graph syntax similar to this of provenance graphs, namely using PROV-O and ServiceProv ontology, with ServiceProv forming our proposed extension for SOC systems and compositions of services as discussed in Chapter 7. Therefore, each provenance node in a provenance template is annotated with its corresponding ServiceProv concept, describing at a higher-level of abstraction conceptually the service composition and SOC provenance domain, but does not contain details of the actual data or service instances. Provenance graphs generated by executing an activity A with different inputs, constraints and parameters will all be instances of the same provenance template of A. In the case of activities being modelled as workflows or service compositions (and orchestrations) in the SOC space, a provenance template can be considered as an abstraction of the workflow specification and the provenance related to this.
Figure 8.1 illustrates a provenance template for the travel planner composition that is an implementation of the workflow introduced in Figure 4.1. Ellipse nodes represent entities, rectangle nodes represent activities while polygon nodes represent agents associated with a particular activity for the generation of a specific entity. The label for a node is the semantic domain concept that it represents (e.g., Service Execution (activity), Server (agent), Message Input notated as Min (entity) etc.). Edges from entities to activities are “was Generated By” relationships, edges from activities to entities are “used” relationships, edges from entities to entities are “was Derived From” relationships and edges from activities to agents are “was Associated With” relationships - we retain the past tense form of the PROV-DM relationship names for consistency, even though the template refers to a possible future occurrence we would like to record.

From a design/modelling perspective a provenance template can be considered as a logic construct that specifies a graph pattern of provenance data, required to answer particular question types, facilitating in this way analysis at design time; it does so by modelling the provenance data recording requirements corresponding to the different provenance questions exhibiting facets as those are proposed in Chapter 6. The provenance template is an abstract representation of the provenance graph structure that would be captured, therefore its entity, activity, and agent nodes refer not to concrete past instances but to instances of a design-time model simulating and predicting the expected provenance; in fact, templates describe predefined system actions whose concrete executions are recorded in provenance graphs.

As part of our analysis framework we aim to analyse whether specific provenance instances of composite service specifications can be mapped against a particular or a composition of our provenance templates. For an instance of the service provenance model – ServiceProv ontology representing the provenance snippets for a service composition execution, questions can be made about the provenance data that may be recorded and stored in the execution cycle of the service composition. Those questions initially expressed in a natural language
Figure 8.1: Provenance Templated Model for the Travel Planner Use Case Scenario
by users can be turned into a more formal expression becoming typed questions, adhering to a provenance data model (schema) that provides the core concepts and constructs of a provenance and the assertions that need to be made in order to query certain provenance information. Then provenance question types are realised into provenance queries that use provenance information as query conditions to retrieve answers for particular data entities, agents or activities and their specialised concepts as those were introduced with regards to the service-oriented computing execution cycle.

Next, we present a set of questions, illustrated through our motivating travel planner scenario. On the basis of such questions/queries, exhibiting particular facet categories, we propose our formal models for provenance questions, namely a set of provenance question type patterns. We first define those patterns as part of abstract provenance templates and then we realise those as query patterns using a semantic query language notation.

The query language notation used to express our provenance query patterns is SPARQL [74]. SPARQL is the W3C recommendation towards a standard query language for the Semantic Web, focusing on querying RDF [91] graphs. SPARQL can be used to query an RDF Schema or OWL [73] model to filter out individuals with specific characteristics [74]. We use SPARQL to reason over service provenance model instances of the travel planner use case scenario. We execute those queries in the SPARQL query engine of Protege platform [84] or through Jena API [24]. The result of such queries corresponds to the way in which the provenance query patterns and the assertions made, match the corresponding provenance data graph snippets describing the provenance captured by particular service compositions.

8.3 Provenance Question Type Patterns

In this section we introduce the set of provenance question type patterns expressed both in an abstract graph form – provenance template – and then for-
malised in an SPARQL query form. We introduce each pattern by providing query examples of the travel planner scenario, exhibiting the proposed provenance facets of Chapter 6.

Patterns contain variables, and in the case of provenance question type patterns, those variables correspond to the key provenance data concepts required for satisfying a set of provenance awareness requirements. To express provenance awareness requirements we start with asking high-level provenance questions expressed in natural language. Then those questions are expressed formally becoming typed questions, adhering to a provenance data model (schema), where a provenance data model allows one to formally express the core concepts and constructs of a provenance question and the assertions that need to be made in order to query certain provenance information. The results of querying specific provenance information allow one to measure provenance awareness of a system. Therefore, the provenance question type becomes a pattern that validates support of specific provenance awareness requirements with regards to the system’s provenance-aware design.

A provenance pattern query searches for specific data elements using provenance information as the information filter criteria (provenance assertions). A formal definition of this is given below:

**Definition:** A provenance pattern query is a 3-tuple \(<TE, MA, ME>\) where

- TE (Target Entity) specifies the data type of the query target data entity (or entities if there are multiple query targets). The data type is usually expressed by using the domain concept(s) annotated to the target entity(ies). For example, TE can be “Message” if we are looking for a specific input/output message of a service in the composition workflow.
- MA (Matching/Mapping Activity) specifies a set of key-value pairs, where each key indicates a historical activity that should appear in the provenance
graph of the target entities, and the corresponding value denotes the query conditions on the activity.

- ME (Matching/Mapping Entity) specifies a set of key-value pairs, where each key indicates an entity that should appear in the provenance graph of the target data entity. Such an entity can be an input entity or an intermediate result, or the target entity itself. The value associated with the key denotes the query conditions on the entity. The query conditions in this definition specify searching criteria on entities and activities that are required to appear in the provenance graph records of the target entity. Query conditions on an activity A are mainly focusing on the parameter settings of A, as well as the controlling agents of A. Query conditions on an entity E are mainly focusing on the properties and values of E. Since entities and activities are annotated by domain concepts in our service provenance model, these query conditions can be expressed using our semantic domain model, namely ServiceProv Ontology.

The query condition can also be expressed by another provenance pattern query (nested query pattern). In this way, we can aggregate provenance query patterns to formally express the multi-faceted provenance questions as those were described in Section 6.2.

MA and ME basically contain two kinds of criteria:

The first kind specifies what entities, activities and agents should appear in the transitive provenance of the target entity, thus defines “reachability criteria” for the provenance graph, i.e., there should exist paths in the provenance graph between the target entity and the entities/activities mentioned in the pattern query. The reachability criteria can be analysed in a provenance template graph. Although the way we structure the reachability criteria may seem very similar to the assertions in a standard SPARQL query, the main difference here is that we can actually parameterize what we need to capture in order to satisfy the set of provenance requirements we are interested in, and formally express the data
types required in order to provide provenance awareness support for provenance questions crosscutting different dimensions (facets).

The second kind of criteria specifies the conditions that values or properties of individual entities and activities in the provenance graph should satisfy. For example, the query condition that the discovered flight booking service is “EasyJet” belongs to this kind of criteria. We call this kind of criteria “property criteria” and use it to look for specific instances of “entity” and “activity”.

The first step for identifying a provenance query pattern is to identify the structure of the provenance templates for the target data entity or entities. In order to formulate those templates, we need to identify what query conditions – reachability criteria – should be satisfied, where the reachability criteria can be represented by a group of paths that should appear in the provenance template graph.

Figure 8.2 represents a UML class diagram describing the main relationships between the different modelling elements (artefacts) that form part of the provenance templates introduced in Section 8.3. We use a UML class diagram notation
in order to make concrete the relationships between the different modelling elements defined at the different layers of modelling (e.g., metamodel, model layer). In particular, a template is an aggregation of one or multiple repetition patterns. We identify three types of provenance (graph) patterns repeated in the abstract service provenance templates: parallel, sequential and past history data. Those patterns may be independent of each other, yet they form an aggregation of fragments linked sequentially or in parallel as part of the same provenance template. Therefore, the instance data defined could be present in a parallel, sequential, or concurrent time interval pattern. In addition, patterns have a cardinality that defines the possible number of pattern instances that may appear within a provenance template. At instance level, patterns are realised as provenance data graphs using the ServiceProv ontological notation. Fragments form on their own an aggregation of fragment elements as members that correspond to the ServiceProv ontological concepts at instance level. Therefore, the ontology graph notation is only meaningful on a realization of a provenance template, at instance level.

Within the abstract templates the repeated graph patterns, visualize the provenance of service related processes (e.g., service discovery, service selection), that may occur sequentially, in parallel or in repeated time intervals. Those patterns may have a number of fragments, where each fragment is a collection of fragment elements that correspond to ServiceProv concepts. The process of defining the content of a fragment depends on which of these types of data your fragment contains and would return as part of a query result.

Next, we introduce a set of provenance question type patterns formalising the set of faceted provenance questions we defined in Chapter 6 by exhibiting those through the travel planner scenario. We follow a methodology where:

- we introduce sections that correspond to each of the individual provenance facets previously identified that we use as classifiers \(^1\) for the provenance question (query patterns),

\(^1\)A classifier (facet) corresponds to particular types of information being recorded about provenance and so particular sets of provenance questions being answerable.
• we express a set of example provenance questions in natural language using the travel planner scenario for each of the classifiers,

• we turn the provenance questions into provenance question types,

• we represent the provenance data required to be captured for a particular provenance question type as a provenance template (graph pattern),

• we realise each provenance template as a provenance query pattern expressed in SPARQL.

8.3.1 Service and Provider Identity Patterns (Facet 1)

The first set of provenance question type patterns are providing a solution for expressing requirements about provenance where the data required to satisfy those requirements are about the identity of the service provider and the service itself, exhibiting Facet 1 as described in Section 6.2.1; those include provenance information such as the DNS hostname of the server, the IP address and port of the providers; service names or URIs of the candidate/selected services; IP address of the service broker that was involved in the discovery of the provided services.

Provenance Questions Inspired by a set of provenance example questions for the travel planner scenario, we abstract from those in order to express a parameterized form of the questions, and identify the assertions/query conditions required in order to answer questions/queries such as the following:

Example questions for the travel planner scenario:

1.1 Who was the service provider of the discovered flight booking service?

1.2 Which server executed the hotel booking service?

1.3 Which repository/registry was involved in the discovery of the flight booking service?
1.4 Which service broker was used in the service selection of the car rental service?

1.5 What was the repository/registry location of the service description of the discovered service attraction search service?

Next, we 1) express each question falling in the service and provider identity patterns category in a parameterized/abstract form (provenance question type), 2) formulate an abstract version of the provenance graph – provenance template – containing all the required provenance data and paths (reachability criteria) that would successfully answer the corresponding questions and should therefore appear in the provenance templated graph, and 3) we formalize this provenance template into a provenance question/query (type) pattern form using RDF/OWL triple notation.

**Provenance Question Type 1.1**: Who was the Service Provider Pi of the discovered service Si?

**Provenance Template 1.1** In order to answer this question we need to capture information about the IP address i and Port i of the Server i, which played the Service Provider Pi role in association to the Service Interface Publishing Ai activity, of the corresponding Service Description i of the discovered Service Si. The difference between the way we express the templates here in comparison to the facets described in Chapter 6 is that all these concepts are parameterized (e.g., Pi, Si). These set of entities, activities and agents that are a necessary part of the provenance graph template are namely the reachability criteria towards the target entities IP address and port which comprise the identity information for the service provider. The set of paths in the provenance graph between the target entities and the entities/activities/agents forming part of the required provenance information to be recorded are analysed in the provenance template graph of Figure 8.3. It is often the case that one pattern in a template graph may be

\[^2\text{Note: In the provenance pattern descriptions that follow we are going to refer to provenance question type patterns expressed in SPARQL as provenance query patterns for short.}\]
repeated several times in parallel or sequentially, e.g., different service providers publishing services in parallel during the discovery activity. We represent those as parallel/sequential fragments in a graph and we denote those by dotted line boxes in the provenance template labelled with the pair $<\text{cardinality}, \text{repetition type}>$, where the cardinality defines the possible number of instances.

**Provenance Query Pattern 1.1** The provenance template in Figure 8.3 is a graph representation matching the query conditions/assertions forming part of the provenance query pattern search for identifying the provider identity information. We formalize this query pattern as a set of triple assertions using SPARQL query language as shown in Listing 8.1, designed to retrieve the identifiers (e.g., IP address, port, provider name) for the providers that published the descriptions of the discovered services. Pointing from the future to the past, the query starts with identifying the triples related to our search query parameters ($\text{?provider}$), ($\text{?ip}$) and ($\text{?port}$) (lines 1-6), and it continues with identifying the agents ($\text{?provider}$) with the service provider role ($\text{serviceprov:ServiceProvider}$) that were associated with publishing a set of service descriptions ($\text{?serviceDescriptionPublishedItem}$) (lines 7-16).
SELECT ?service ?provider ?ipAddress ?port
WHERE {
    ?provider serviceprov:hadPort ?port.
    ?provider serviceprov:hadIPaddress ?ipAddress.
    ?association prov:hadRole ?role.
    ?role rdf:type serviceprov:ServiceProvider.
    ?serviceInterfacePublishing rdf:type serviceprov:
        ServiceInterfacePublishing.
    ?serviceInterfacePublishing prov:qualifiedAssociation
        ?association.
    ?serviceDescriptionPublishedItem prov:wasGeneratedBy
        ?serviceInterfacePublishing.
    ?serviceDescriptionPublishedItem rdf:type serviceprov:
        ServiceDescriptionPublishedItem.
    ?serviceDescriptionDiscoveredItem prov:wasDerivedFrom
        ?serviceDescriptionPublishedItem.
    ?serviceDescriptionDiscoveredItem rdf:type serviceprov:
        ServiceDescriptionDiscoveredItem.
    ?service serviceprov:hadDescription
        ?servdescription.
    ?servdescription rdf:type serviceprov:ServiceDescription.
    ?servCollection prov:hadMember ?serviceDescription.
    ?servCollection prov:wasGeneratedBy ?discovery.
    ?discovery prov:used ?servRequest.
    ?servRequest rdf:type serviceprov:ServiceRequest.
    ?servRequest prov:wasGeneratedBy ?planning.
    ?serviceDescription prov:hadMember
        ?serviceDescriptionDiscoveredItem.
    ?workplan prov:wasGeneratedBy ?planning.
    ?orchestratorExecution prov:used ?workplan.
    ?orchestratorExecution rdf:type
        serviceprov:OrchestratorExecution.
    ?orchestratorExecution prov:value ?orchestratorExecutionID.
}

Listing 8.1: Service Provider Identity Provenance Query Pattern

It then narrows down the results by making assertions about the discovered service description items ("?serviceDescriptionDiscoveredItem"), derived from ("?serviceDescriptionPublishedItem(s)") being members of descriptions ("?serviceDescription") that correspond to a service ("?service") (lines 18-23). Service descriptions are identified as members of a collection of services ("?servCollection"), where the service collections are being generated during service discovery ("?discovery") based on a particular service request ("?servRequest") that was generated by a planning activity ("?planning"), generated on the basis of a composition workflow specification ("?workplan") (lines 25-36). Workplan was used by an orchestrator execution
(\texttt{?orchestratorExecution}) which managed the atomic services involved in the service composition. We then narrow down our results to the particular run we are interested by using as a property criterion its unique orchestration execution identifier (\texttt{?orchestratorExecutionID}) (lines 37-41).

**Provenance Question Type 1.2:** What was the service identity of the atomic service $S_i$ that was executed as part of orchestration execution $i$?

**Provenance Template 1.2** In order to answer this question we need to capture information such as the $\text{serviceName}_i$ of atomic service $\text{Service } S_i$, that corresponds to a particular atomic execution $\text{ServiceExecution}_i$ of this. These set of entities and activities are a necessary part of the provenance graph template towards the target entity $\text{service name}$, where service name comprises the service execution identity information. The set of paths in the provenance graph between the target entity and the entities/activities/agents forming part of the required provenance information to be recorded are analysed in the provenance template graph of Figure 8.4. The pattern of the service execution identity could be repeated several times in parallel e.g., corresponding to the execution of the different atomic services that may be executed in parallel during the composition (orchestration) execution. We represent this behaviour as a parallel fragment in provenance template of Figure 8.4, labelled with the pair \texttt{<cardinality, repetition type>}, denoting the possibility of $n$ number of service execution instances existing and running in parallel.

**Provenance Query Pattern 1.2** The provenance template in Figure 8.4 is a graph representation matching the query conditions/assertions forming part of the provenance query pattern search for identifying the service execution identity information. We formalize this query pattern as a set of triple assertions using SPARQL query language as shown in Listing 8.2, designed to retrieve the service identity information (e.g., service, service name) for the atomic services that executed as part of the orchestration execution. Pointing from the future to the past, the query starts with identifying the triples related to our search query.
parameters (?service) and (?serviceName) (lines 1-6), and it continues with identifying the atomic service execution instance (?serviceExecution) associated with the particular discovered service that formed part of the composition execution (?orchestratorExecution) (lines 7-23). The query then narrows down the results by making assertions about the input message (?inMessage), used by the particular execution of the (?service) we are interested, where the input message was generated by the (?orchestratorExecution) with unique orchestration execution identifier (?orchestratorExecutionID) (lines 24-27).

Provenance Question Type 1.3: Who was the Service Execution Server i of service Si?

Provenance Template 1.3 In order to answer this question we need to capture information such as the IP address i and Port i of the Server i, which played the Service Execution Server i role in association to the Atomic Service Execution Ai activity that corresponds to a particular atomic execution of Service Si. These set of entities and activities are a necessary part of the provenance graph template towards the identity of the targeted agent Service Execution Server i. The complete set of paths in the provenance graph forming part of the required provenance information to be recorded, are analysed in the provenance template.
graph of Figure 8.5. The pattern of the service execution server identity could be repeated several times in parallel e.g., corresponding to the execution of the different atomic services that may be executed in parallel by different servers during the composition (orchestration) execution. We represent this behaviour as a parallel fragment in provenance template of Figure 8.5, labelled with the pair <cardinality, repetition type>, denoting the possibility of n number of service execution instances being executed by different servers in parallel.

**Provenance Query Pattern 1.3** The provenance template in Figure 8.5 is a graph representation matching the query conditions/ assertions forming part of the provenance query pattern search for identifying the service execution server identity information. We formalize this query pattern as a set of triple assertions using SPARQL query language as shown in Listing 8.3, designed to retrieve the service execution server identity information (e.g., IP address, port). Pointing from the future to the past, the query starts with identifying the triples related to our search query parameters (?ipAddress) and (?port) (lines 1-4), and it continues with identifying the atomic service execution instance (?serviceExecution) of
Figure 8.5: Service Execution Server Identity Provenance Template

(\texttt{?service}) that was associated with the (\texttt{?server}) (agent) that played the (\texttt{Service-ExecutionServer}) role in the particular association (lines 5-16). The particular execution of (\texttt{?service}) formed part of an (\texttt{?orchestratorExecution}). We narrow down the results to this particular orchestration execution by making assertions about the input message (\texttt{?inMessage}), used by the particular execution of the (\texttt{?service}) we are interested; where the input message was generated by the (\texttt{?orchestratorExecution}) with unique orchestration execution identifier (\texttt{?orchestratorExecutionID}) (lines 17-24).

**Provenance Question Type 1.4:** Who was the Registry \texttt{i} involved in the discovery of service \texttt{Si}?  

**Provenance Template 1.4** In order to answer this question we need to capture information such as the IP address \texttt{i} and Port \texttt{i} of the Server \texttt{i}, which played the Registry \texttt{i} role in an attribution relationship to the ServiceDescriptionDiscovered-Item \texttt{i} entity that is member of the particular ServiceDescription \texttt{i} corresponding
to Service Si. These set of entities and activities are a necessary part of the provenance graph template towards the identity of the targeted agent Registry i. The complete set of paths in the provenance graph forming part of the required provenance information to be recorded are analysed in the provenance template graph of Figure 8.6. The pattern of the registry identity could be repeated several times in parallel e.g., corresponding to the multiple discovered service description items of the different atomic services being discovered in parallel. We represent this behaviour as a parallel fragment in provenance template of Figure 8.6, labelled with the pair <cardinality, repetition type>, denoting the possibility of n number of service description discovered item being attributed to different registries in parallel.

Provenance Query Pattern 1.4 The provenance template in Figure 8.6 is a graph representation matching the query conditions/assertions forming part of the provenance query pattern search for identifying the registry identity information. We formalize this query pattern as a set of triple assertions using SPARQL query language as shown in Listing 8.4, designed to retrieve the registry identity information (e.g., IP address, port). Pointing from the future

<table>
<thead>
<tr>
<th>Listing 8.3: Service Execution Server Identity Provenance Query Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT ?server ?ipAddress ?port</td>
</tr>
<tr>
<td>WHERE{</td>
</tr>
<tr>
<td>?server serviceprov:hadPort ?port .</td>
</tr>
<tr>
<td>?server serviceprov:hadIPAddress ?ipAddress .</td>
</tr>
<tr>
<td>?association prov:hadRole ?role .</td>
</tr>
<tr>
<td>?servExecution prov:wasAssociatedWith ?server .</td>
</tr>
<tr>
<td>?association prov:hadRole ?role .</td>
</tr>
<tr>
<td>?servExecution prov:wasAssociatedWith ?server .</td>
</tr>
<tr>
<td>?servExecution serviceprov:serviceName ?serviceName .</td>
</tr>
<tr>
<td>?servExecution prov:used ?inMessage .</td>
</tr>
<tr>
<td>?inMessage rdf:type serviceprov:MessageIn .</td>
</tr>
<tr>
<td>FILTER ( regex (str (?orchestratorExecutionID), &quot;id&quot;) )</td>
</tr>
<tr>
<td>})</td>
</tr>
</tbody>
</table>
to the past, the query starts with identifying the triples related to our search query parameters (\(?ipAddress\)) and (\(?port\)) (lines 1-5), and it continues with identifying the service description discovered item (\(?servDescriptionDiscoveredItem\)) of the (\(?service\)) that was associated with the (serviceprov:Server) (agent) that played the (Registry) role in the particular association (lines 6-19). The particular (\(?servDescriptionDiscoveredItem\)) formed a member of (\(?servDescription\)) that was part of (\(?servCollection\)) being generated during service (\(?discovery\)) based on a particular service request (\(?servRequest\)); the (\(?servRequest\)) was generated by a planning activity (\(?planning\)), generated on the basis of a composition workflow specification (\(?workplan\)) (lines 20-29). Workplan was used by an orchestrator execution (\(?orchestratorExecution\)) which managed the atomic services involved in the service composition. We then narrow down our results to the particular run we are interested by using as a property criterion its unique orchestration execution identifier (\(?orchestratorExecutionID\)) (lines 30-35).
Provenance Question Type 1.5: Who was the Service Broker $i$ involved in the discovery of (collection of) service(s) $S_i$?

Provenance Template 1.5  In order to answer this question we need to capture information such as the IP address $i$ and Port $i$ of the Server $i$, which played the ServiceBroker $i$ role in association to the ServiceDiscovery $i$ activity that generated a collection of ServiceDescription(s) being part of Service(s) $S_i$. These set of entities and activities are a necessary part of the provenance graph template towards the identity of the targeted agent ServiceBroker $i$. The complete set of paths in the provenance graph forming part of the required provenance information to be recorded are analysed in the provenance template graph of Figure 8.7. The pattern of the service broker identity could be repeated several times in parallel e.g., corresponding to the discovery of multiple descriptions discovered for
different atomic services that may be executed in parallel during the composition (orchestration) execution. We represent this behaviour as a parallel fragment in provenance template of Figure 8.7, labelled with the pair <cardinality, repetition type>, denoting the possibility of n number of services (descriptions) being discovered in parallel.

**Provenance Query Pattern 1.5** The provenance template in Figure 8.7 is a graph representation matching the query conditions/ assertions forming part of the provenance query pattern search for identifying the broker identity information. We formalize this query pattern as a set of triple assertions using SPARQL query language as shown in Listing 8.5, designed to retrieve the broker identity information (e.g., IP address, port). Pointing from the future to the past, the query starts with identifying the triples related to our search query parameters (?ipAddress) and (?port) for the broker identity (lines 1-5), and it continues
with identifying the discovery instance (?discovery) that was associated with the (?server) (agent) that played the (ServiceBroker) role in the particular association)(lines 6-9). The particular (?discovery) generated a (?servCollection) based on a particular service request (?servRequest) that was generated by a planning activity (?planning), generated on the basis of a composition workflow specification (?workplan) (lines 10-24). Workplan was used by an orchestrator execution (?orchestratorExecution) which managed the atomic services involved in the service composition. We then narrow down our results to the particular run we are interested by using as a property criterion its unique orchestration execution identifier (?orchestratorExecutionID) (lines 26-31).

**Provenance Question Type 1.6:** Who was the Aggregator involved in the service selection of service Si?
Figure 8.8: Aggregator Identity Provenance Template
In order to answer this question we need to capture information such as the IP address and Port of the Server, which played the Aggregator role in association to the ServiceSelection activity that generated the set of SelectedServiceDescription(s) to be used by the Planning activity that generates the composition WorkPlan. These set of entities and activities are a necessary part of the provenance graph template towards the identity of the targeted agent Aggregator, playing a dual role in association to the service selection and planning activities of the SOC execution cycle. The complete set of paths in the provenance graph forming part of the required provenance information to be recorded are analysed in the provenance template graph of Figure 8.8. The pattern of the aggregator identity incorporating the service selection and planning activities could be repeated several times in parallel e.g., corresponding to the service selection and planning of different services discovered for generating an execution work plan that satisfies a set of QoS and functional requirements.

We represent this behaviour as a parallel fragment in provenance template of Figure 8.8, labelled with the pair <cardinality, repetition type>, denoting the possibility of n number selected services being used in parallel for identifying the optimal work plan during the planning activity.

The provenance template in Figure 8.8 is a graph representation matching the query conditions/assertions forming part of the provenance query pattern search for identifying the aggregator identity information. We formalize this query pattern as a set of triple assertions using SPARQL query language as shown in Listing 8.6, designed to retrieve the aggregator identity information (e.g., IP address, port). Pointing from the future to the past, the query starts with identifying the triples related to our search query parameters (?ipAddress) and (?port) for the aggregator identity (lines 1-5), it continues with identifying the service selection instance (?servSelection), that was associated with the (?server) (agent) that played the (Aggregator) role in the particular association)(lines 6-14). The service selection instance (?servSelection), used the (?servCollection) that was previously generated based on a particu-
Listing 8.6: Aggregator Identity Provenance Query Pattern

In association to the service selection and planning activities, where the aggregator plays its dual role, we might be interested in querying the identity of the actual services selected during those activities. Therefore, an additional pattern
to our list would be the **Selected Service Identity Template** as shown below 8.9.

**Provenance Question Type 1.7:** Which service $S_i$ was selected to be invoked during the **Service Selection $i$** and **Planning $i$** activity?

**Provenance Template 1.7**  In order to answer this question we need to capture information such as the **serviceName $i$**, **selectedServiceDescription $i$** of the selected service **Service $S_i$**, that corresponds to the result of the service selection **Service-Selection $i$** activity. These set of entities and activities are a necessary part of the provenance graph template towards the target entity **serviceName**, where serviceName comprises the selected service identity information. The set of paths in the provenance graph between the target entity and the entities/activities/agents forming part of the required provenance information to be recorded are analysed in the provenance template graph of Figure 8.9. The pattern of the selected service identity could be repeated several times in parallel e.g., corresponding
to service descriptions of the services that were generated by the service selection and used by the planning activities until an optimal work plan solution was found. We represent this behaviour as a parallel fragment in provenance template of Figure 8.9, labelled with the pair <cardinality, repetition type>, denoting the possibility of n number of selected services instances generated/used in parallel.

**Provenance Query Pattern 1.7** The provenance template in Figure 8.9 is a graph representation matching the query conditions/assertions forming part of the provenance query pattern search for identifying the selected service(s) identity information. We formalize this query pattern as a set of triple assertions using SPARQL query language as shown in Listing 8.7, designed to retrieve the selected service identity information (e.g., selected service description, service name). Pointing from the future to the past, the query starts with identifying the triples related to our search query parameters (?serviceName) and (?service) (lines 1-4), and it continues with identifying the service description (?servdescription) of the service (?service) that was discovered during the service discovery activity and

```sparql
SELECT ?serviceName ?service
WHERE {
  ?service serviceprov:serviceName ?serviceName .
  ?servdescription rdf:type serviceprov:ServiceDescription.
  ?servdescription prov:hadMember ?servdescriptionDiscoveredItem.
  ?servCollection prov:hadMember ?servdescription.
  ?servdescriptionDiscoveredItem rdf:type serviceprov:ServiceDescriptionDiscoveredItem.
  ?servdescriptionDiscoveredItem prov:value ?value.
  ?servSelection prov:used ?servCollection.
  ?selectedServiceDescription prov:wasGeneratedBy ?servSelection.
  ?selectedServiceDescription rdf:type serviceprov:SelectedServiceDescription.
  ?planning prov:used ?selectedServiceDescription.
  ?workplan prov:wasGeneratedBy ?planning.
  ?orchestratorExecution prov:used ?workplan.
  ?servprov:OrchestratorExecution prov:value ? orchestratorExecutionID.
}
```

Listing 8.7: Selected Service Identity Provenance Query Pattern
formed part of the service collection (?servCollection) that was used by the service selection (?servSelection) activity (lines 5-13). The particular execution of (?servSelection) activity generated a (?selectedServiceDescription) based on the collection of discovered service (descriptions) that was then used by the (?planning) activity in order to construct the composition (?workplan) (lines 14-19). Workplan was used by an orchestrator execution (?orchestratorExecution) which managed the atomic services involved in the service composition. We then narrow down our results to the particular run we are interested by using as a property criterion its unique orchestration execution identifier (?orchestratorExecutionID) (lines 20-23).

8.3.2 Data Flow Patterns (Facet 2)

The second set of provenance question type patterns are providing a solution for expressing requirements about provenance where the data required to satisfy those requirements are about the data flow of the composition workflow, exhibiting Facet 2 as described in Section 6.2.1; those include provenance information such as the input and output messages used/derived or generated by the atomic and orchestration service execution and exchanged between services, parameters passed in message exchanges, which messages relate to every service.

Provenance Questions  Inspired by a set of provenance example questions for the travel planner scenario, we abstract from those in order to express a parameterized form of the questions, and identify the assertions/query conditions required in order to answer questions/queries such as the following:

Example questions for the travel planner scenario:

2.1 What were the input parameters passed to the driving time calculation service (Min)?

2.2 What was the output of the driving time calculation service (Mout)?
Next, we express each question falling in the data flow patterns category in a template and query form notation.

**Provenance Question Type 2.1:** What were the parameters of the input messages (Parameter Input $i$) for the service execution of service $S_i$?

**Provenance Template 2.1** In order to answer this question we need to capture information such as the Message Input $i$ and Parameters Input $i$ of atomic service Service $S_i$, that corresponds to a particular atomic execution ServiceExecution $i$ of this. These set of entities and activities are a necessary part of the provenance graph template towards the target entity parameter Input, forming the input parameter information used as part of the input message as part of a XML/SOAP message structure of an atomic service. Also, service description information is required as it encapsulated the corresponding service identification details as part of the WSDL specification. The set of paths in the provenance graph between the target entity and the entities/activities/agents forming part of the required provenance information to be recorded are analysed in the provenance template graph of Figure 8.10. The pattern of the input message parameters could be repeated sequentially as the orchestrator uses a set of input messages (and their parameters) e.g., corresponding to the execution of the different atomic services that may be executed sequentially during the composition (orchestration) execution. We represent this behaviour as a sequential fragment in provenance template of Figure 8.10, labelled with the pair $<$cardinality, sequential type$>$, denoting the possibility of $n$ number of input messages (and their parameters) used by different service execution instances executing sequentially.

**Provenance Query Pattern 2.1** The provenance template in Figure 8.10 is a graph representation matching the query conditions/assertions forming part of the provenance query pattern search for identifying the parameter input information forming part of the input message of an execution of service $S_i$. We formalize this query pattern as a set of triple assertions using SPARQL query language as shown in Listing 8.8, designed to retrieve the parameter input in-
Figure 8.10: Input Message Parameters Provenance Template

Listing 8.8: Input Message Parameters Provenance Query Pattern

formation (e.g., parameterInput, MessageInput) for the atomic services that executed as part of the orchestration execution. Pointing from the future to the past, the query starts with identifying the triples related to our search query parameters (?parameterIn) and (?serviceName) (lines 1-6), and it continues with identifying the atomic service execution instance (?serviceExecution) associated with the particular discovered service that formed part of the composition execution (?orchestratorExecution) (lines 7-13). It then narrows down the results by making assertions about the input message (?inMessage), used by the particular execution of the (?service) we are interested, where the input message was gener-
ated by the (?orchestratorExecution) with unique orchestration execution identifier (?orchestratorExecutionID) (lines 14-17).

Provenance Question Type 2.2: What were the parameters of the output messages (parameterOutput i) for service execution of service Si?

Provenance Template 2.2 In order to answer this question we need to capture information such as the Message Output i and parameter Output i of atomic service Service Si, that corresponds to a particular atomic execution ServiceExecution i of this. These set of entities and activities are a necessary part of the provenance graph template towards the target entity parameter Output, forming the output parameter information generated as part of the output message of an atomic service execution and then used by the orchestrator. The set of paths in the provenance graph between the target entity and the entities/activities/agents forming part of the required provenance information to be recorded are analysed in the provenance template graph of Figure 8.11. The pattern of the output message parameters could be repeated sequentially as the orchestrator uses a set of output messages (and their parameters) e.g., corresponding to the execution of the different atomic services that may be executed sequentially during the composition (orchestration) execution. We represent this behaviour as a sequential fragment in provenance template of Figure 8.11, labelled with the pair <cardinality, sequential type>, denoting the possibility of n number of output messages (and their parameters) used by different service execution instances executing sequentially.

Provenance Query Pattern 2.2 The provenance template in Figure 8.11 is a graph representation matching the query conditions/assertions forming part of the provenance query pattern search for identifying the parameter output information forming part of the output message of an execution of service Si. We formalize this query pattern as a set of triple assertions using SPARQL query
language as shown in Listing 8.9, designed to retrieve the parameter output information (e.g., parameterOutput, MessageOutput) for the atomic services that executed as part of the orchestration execution. Pointing from the future to the past, the query starts with identifying the triples related to our search query parameters (?parameterOut) and (?serviceName) (lines 1-6), and it continues with identifying the atomic service execution instance (?serviceExecution) associated with the particular discovered service that formed part of the composition execution (?orchestratorExecution) (lines 7-11). It then narrows down the results by making assertions about the output message (?outMessage), generated by the particular execution of the (?service) we are interested, and used by the (?orchestratorExecution) with unique orchestration execution identifier (?orchestratorExecutionID) (lines 12-19).

8.3.3 Resource/Physical Deployment Patterns (Facet 3)

The third set of provenance question type patterns are providing a solution for expressing requirements about provenance where the data required to satisfy those requirements are information about the resources and physical deployment of atomic and composite services of the service composition execution workflow,
exhibiting Facet 3 as described in Section 6.2.1; those include provenance information such as the availability of different types of resources (e.g., CPU, memory, hard disk, operating system etc.) during the execution of atomic or composite services.

Provenance Questions Inspired by a set of provenance example questions for the travel planner scenario, we abstract from those in order to express a parameterised form of the questions, and identify the assertions/query conditions required in order to answer queries such as the following:

*Example questions for the travel planner scenario:*

3.1 What resources (memory, hard disks, CPUs) were used for the attraction search service concurrently to the flight and hotel booking services?

3.2 What resources were available during the execution of the composite service?

Next, we express each question falling in the resources and physical deployment patterns category in a template and query form notation.

**Provenance Question Type 3.1:** What resources (Resource(s) $i$) were available during the execution of service $S_i$?
Provenance Template 3.1 In order to answer this question we need to capture information such as the ResourceCollection i of Resource(s) i that were used during the ServiceExecution i of Service Si. These set of entities and activities are a necessary part of the provenance graph template towards the target entity resource(s) i, forming the available member resources of the resource collection used during the execution of Service Si. The set of paths in the provenance graph between the target entity and the entities/activities/agents forming part of the required provenance information to be recorded are analysed in the provenance template graph of Figure 8.12.

The pattern of the services executing could be repeated sequentially, corresponding to the execution of the different atomic services during the composition (orchestration) execution, while a number of resources may be available to be used by a particular service execution in parallel. We represent this behaviour as a parallel fragment in provenance template of Figure 8.12, labelled with the pair <cardinality, parallel type>, denoting the possibility of n number of resource members used in parallel by different service execution instances executing sequentially.
Provenance Query Pattern 3.1 The provenance template in Figure 8.12 is a graph representation matching the query conditions/assertions forming part of the provenance query pattern search for identifying the resources that were available and used for the execution of service Si. We formalise this query pattern as a set of triple assertions using SPARQL query language as shown in Listing 8.10, designed to retrieve the information about the collection of resources that were used for the atomic services that executed as part of the orchestration execution. Pointing from the future to the past, the query starts with identifying the triples related to our search query parameters (?serviceName) and (?resource) (line 1), and it continues with identifying the collection of resource members that were used by the execution instance (?serviceExecution) associated with the particular (?service) that formed part of the composition execution (?orchestratorExecution) (lines 3-11). It then narrows down the results by making assertions about the (?orchestratorExecution) with unique orchestration execution identifier (?orchestratorExecutionID) (lines 13-16).

Our provenance query pattern for resource availability can be specialised to match the assertions about specific resource types such as CPU, memory, operating system etc. In order to express those assertions we reuse the resource computing concepts of the SEALS Computing Resource Ontology [25] to represent specific resources as presented in 7.8.
For example Listing 8.11 is designed to retrieve the information about the collection of resources of CPU type. Pointing from the future to the past, the query starts with identifying the triples related to our search query parameters (?serviceName) and (?resource) (line 1), and it continues with identifying the collection of resource members of type (seals:CPU along with their (seals:numberOfCores) value that were used by the execution instance (?serviceExecution) associated with the particular (?service) - that formed part of the composition execution (?orchestratorExecution) (lines 4-15). It then narrows down the results by making assertions about the (?orchestratorExecution) with unique orchestration execution identifier (?orchestratorExecutionID) (lines 17-21).

Similarly, Listing 8.12 is designed to retrieve the information about the collection of resources of Memory type (seals:Storage, seals:hasStorageSize), while Listing 8.13 is designed to retrieve the information about the collection of resources of Operating System type (seals:OperatingSystem, seals:osName).

8.3.4 Time Patterns (Facet 4)

The fourth set of provenance question type patterns are providing a solution for expressing requirements about provenance where the data required to satisfy
those requirements are information on composite (or individual) service execution timestamps: timestamps for start /completion, for service invocation /discovery, an identifier for the service (service name or URI).

Provenance Questions Inspired by a set of provenance example questions for the travel planner scenario, we abstract from those in order to express a parameterized form of the questions, and identify the assertions/query conditions required in order to answer queries such as the following:

Example questions for the travel planner scenario:

4.1 When did the composite service travel planner execution start/end?

4.2 When was the execution of the car rental service was invoked?

Next, we express each question falling in the time patterns category in a template and query form notation.

Provenance Question Type 4.1: When did the execution (serviceExecution $i$) of service $S_i$ start/end?
SELECT ?serviceName ?resource ?value
WHERE
{
?resourceCollection prov:hadMember ?resource.
?resource seals:osName ?value.
?servexecution prov:used ?inMessage.
?inMessage rdf:type serviceprov:MessageIn.
?orchestratorExecution prov:value ?orchestratorExecutionID.
FILTER ( regex (str(?orchestratorExecutionID), "orchestratorExecutionID", "i"))
}

Listing 8.13: Operating System Resource Availability Provenance Query Pattern

Provenance Question Type 4.2: When was (serviceExecution i) of service Si was invoked?

Provenance Templates 4.1/4.2 In order to answer these questions we need to capture information such as the start ststart i and end stend i time of the ServiceExecution i of Service Si or the generation timestamp i of the input message used to invoke ServiceExecution i of Service Si respectively. These set of entities and activities are a necessary part of the provenance graph template towards the target entity timestamp i, ststart i, stend i, forming the timestamps about the invocation and start/end execution time of Service Si. The set of paths in the provenance graph between the target entity and the entities/activities/agents forming part of the required provenance information to be recorded are analysed in the provenance template graph of Figure 8.13.

The provenance template in Figure 8.13 is a graph representation matching the query conditions/ assertions forming part of the provenance query pattern search for identifying the timestamps for the invocation and start/end of execution of service Si. We formalise this query pattern as a set of triple assertions using SPARQL query language as shown in Listing 8.14 and Listing 8.15, designed to
retrieve the information about the timestamp values of invocation/execution for the atomic services that executed as part of the orchestration execution. The time patterns can be generalized to be applied to 1) various activities associated with the SOC execution cycle such as the service discovery or service selection, 2) the generation/derivation of entities (such as messages or service descriptions and QoS contracts), 3) the provenance time snippets of particular events being part of the provenance history workflow.

**Provenance Query Pattern 4.1** Pointing from the future to the past in Listing 8.14 the query starts with identifying the triples related to our search query parameters (?serviceName) and (?ststart, ?stend) (line 1), and it continues with identifying identity of the execution instance (?serviceExecution) associated with the particular (?service) that formed part of the composition execution (?orchestratorExecution) (lines 2-9) we are interested. It then narrows down the results by making assertions about the (?orchestratorExecution) with unique orchestration execution identifier (?orchestratorExecutionID) (lines 10-16).

**Provenance Query Pattern 4.2** On the other hand, in Listing 8.15 we are interested in the timestamp of invocation of a particular atomic service execution, as expressed in provenance question type 4.2. The query starts with identify-
SELECT ?serviceName ?ststart ?stend
WHERE
{
?servexecution serviceprov:serviceName ?serviceName.
?servexecution prov:used ?inMessage.
?inMessage prov:wasGeneratedBy ?orchestratorExecution.
?orchestratorExecution prov:value ?orchestratorExecutionID.
FILTER (regex (str (?orchestratorExecutionID), "orchestratorExecutionID", "i"))
}

Listing 8.14: Service Execution Timestamps Provenance Query Pattern

SELECT ?serviceName ?timestamp
WHERE
{
?servexecution serviceprov:serviceName ?serviceName.
?servexecution prov:used ?inMessage.
?inMessage prov:wasGeneratedBy ?orchestratorExecution.
?orchestratorExecution prov:value ?orchestratorExecutionID.
FILTER (regex (str (?orchestratorExecutionID), "orchestratorExecutionID", "i"))
}

Listing 8.15: Service Invocation Timestamps Provenance Query Pattern

The triples related to our search query parameters (?serviceName) and invocation timestamp (?timestamp) (line 1), and it continues with identifying the identity of the execution instance (?serviceExecution) that was invoked associated with the particular (?service) that formed part of the composition execution (?orchestratorExecution) (lines 2-10) we are interested. It then narrows down the results by making assertions about the (?orchestratorExecution) with unique orchestration execution identifier (?orchestratorExecutionID) (lines 11-14).

8.3.5 Routes (not) followed Patterns (Facet 5)

The fifth set of provenance question type patterns are providing a solution for expressing requirements about provenance where the data required to satisfy those requirements are information on which branches of the workflow have
been taken, on the actual composition execution plan and on additional branches that have not been followed including the relevant services, their policies and their provided interfaces. It is important to clarify that this facet requires more information than simply the data flow (facet 2) or design information (facet 9): Recording data flow only may never record a potential route if it is never followed by any service execution. Conversely, capturing the history of design information only will provide information about all potential routes, but will not tell us which ones have actually been followed or not followed in a particular service execution. Having both pieces of information captured – design information and the actual data flow during service execution – can be used in order to identify the branches of the workflow that have not been taken by calculating the delta of all the specified branches at design time minus the branches that were actually taken during the service (composition) execution.

**Provenance Questions** Inspired by a set of provenance example questions for the travel planner scenario, we abstract from those in order to express a parameterized form of the questions, and identify the assertions/query conditions required in order to answer queries such as the following:

*Example questions for the travel planner scenario:*

5.1 Are there branches in the composite service that have not been explored (e.g., for the bike rental service)?

5.2 What are the alternative routes for authenticating and confirming the travel package booking by using Paypal service instead of simply sending the credit card information?

We can express each question falling in the “routes not followed” patterns category in a parameterized/abstract form and formulate an abstract version of the provenance graph – provenance template – containing all the required provenance data and paths that would successfully answer the corresponding questions by reusing and combining templated elements described in the service and provider
identity and data flow provenance templates. Therefore the provenance graph template of this category is an aggregation of previously defined pattern categories that can be composed as required. The corresponding SPARQL query patterns, formalising the provenance questions for the “routes not followed category”, can then be formed through combining triple assertions falling under the individual SPARQL query patterns (e.g., service provider and identity provenance query patterns) used for constructing the aggregated “routes not followed” pattern.

8.3.6 Past History Patterns (Facet 6)

The sixth set of provenance question type patterns are providing a solution for expressing requirements about provenance where the data required to satisfy those requirements are information to be stored for more than one past invocations of individual and composite services; the requirements on the number of invocations and the length of time for which infrastructure designers may need to store the information may vary. Again in this category the information captured is an aggregation of information already presented in other pattern categories such as the time pattern category (e.g., the time interval for which we want to keep logs of provenance data, timestamps of invocations, timestamps for the number of times the service failed to be completed etc.), the service and provider identity pattern category (e.g., the identifier of the service such as URI, or service name) or an aggregation of those (e.g., the total number of invocations for the maximum expected time frame - 8 months).

Provenance Questions  Inspired by a set of provenance example questions for the travel planner scenario, we abstract from those in order to express a parameterized form of the questions, and identify the assertions/query conditions required in order to answer queries such as the following:

Example questions for the travel planner scenario:
6.1 How often has the service of the chosen flight booking service been accessed/invoked the last 2 days?

6.2 What was the failure rate in the time frame the travel planner service is running the last 8 months?

6.3 What was the success/failure rate in the time frame the travel planner service is running the last 8 months?

Similarly to the “routes not followed pattern” category we can express each question falling in the “past history” patterns category in a parameterized/abstract form and formulate an abstract version of the provenance graph – provenance template – containing all the required provenance data and paths that would successfully answer the corresponding questions by reusing and combining templates described in the service and provider identity, timestamps, data flow patterns and NFPs/QoS patterns that follow in the next section. Therefore this category is an aggregation of other elemental defined pattern categories and SPARQL queries formalising the provenance questions for this category are a combination of triple assertions falling under the corresponding patterns used for the aggregation.

8.3.7 NFPs and QoS Patterns (Facet 7)

The seventh set of provenance question type patterns are providing a solution for expressing requirements about provenance where the data required to satisfy those requirements are information for every particular NFP of every service that takes part in the composition, information on SLAs between the different services and QoS contracts describing the QoS offered or required by each service.

**Provenance Questions**  Inspired by a set of provenance example questions for the travel planner scenario, we abstract from those in order to express a parameterized form of the questions, and identify the assertions/query conditions required in order to answer queries such as the following:
Example questions for the travel planner scenario:

7.1 What was the response time for the flight booking service?
7.2 What was the processing time for the hotel booking service?
7.3 What was the network latency for the payment service?
7.4 What was the execution cost for the hotel booking service?
7.5 What was the reputation for the car rental service?
7.6 Was all credit-card information encrypted securely throughout processing?
7.7 What was the QoS offered (QoS service description) for the flight booking service?
7.8 What were the QoS requirements for service selection procedure of the flight booking service?

Example questions for composite services:

7.9 What was the average response time /processing time /overall execution time for the travel planner service?
7.10 What was the total execution time for the travel planner service?
7.11 What was the execution cost for the travel planner service?

Next, we express each question falling in the NFPs and QoS patterns category in template and query form notation.

Provenance Question Type 7.1: What was the NFPvalue i for the execution of service Si?
Provenance Template 7.1 In order to answer this question we need to capture information such as the non-functional properties (NFP(s) i) and their values (NFPvalues(s) i) for particular executions of services (ServiceExecution i) of Service Si, or for a particular Service Si lifecycle respectively. These set of entities and activities are a necessary part of the provenance graph template towards the target entities NFP i, NFPvalue i, forming the non-functional property (values) about particular services Service Si and the services’ execution instances. The set of paths in the provenance graph between the target entity and the entities/activities/agents forming part of the required provenance information to be recorded are analysed in the provenance template graph of Figure 8.14. It is often the case that a service execution instance or a service itself may have a number of associated non-functional properties that we may be interested in capturing in parallel, therefore we represent those as parallel fragments in the template graph of Figure 8.14, labelled with the pair <cardinality, parallel type>, where the cardinality defined the possible number of NFP instances and their values respectively.
Provenance Query Pattern 7.1  The provenance template in Figure 8.14 is a graph representation matching the query conditions/assertions forming part of the provenance query pattern search for identifying the non-functional properties and their values associated to particular atomic service executions of the services that formed part of the orchestration execution or that were associated with particular services for a greater period of time and/or the service’s lifecycle. Our provenance query pattern for non-functional properties can actually be realised if we specialise this to match the assertions about specific non-functional property types such as performance, response and execution time, availability, reliability, security, reputation, cost and penalty etc. In order to express those assertions, we extend the NFP concept of ServiceProv ontology to represent specific non-functional properties of atomic or/and composite services. In particular, we formalise the NFPs and QoS query pattern as a set of triple assertions using SPARQL query language for the different types of NFPs as shown in Listings (8.16 –8.23), designed to retrieve the information about the NFP values of the atomic services that executed as part of the orchestration execution.

For example, in Listing 8.16 pointing from the future to the past the query starts with identifying the triples related to our search query parameters such as

```sql
SELECT ?service ?serviceName ?value
WHERE {
  ?service serviceprov:hadNFP ?availability.
  ?servprov serviceName ?serviceName.
  FILTER (regex (str(?orchestratorExecutionID), “orchestratorExecutionID”, “i”))
  ?servprov:wasGeneratedBy ?orchestratorExecution.
  ?orchestratorExecution prov:value ?orchestratorExecutionID.
  FILTER ( regex (str(?orchestratorExecutionID), “orchestratorExecutionID”, “i”))
}
```

**Listing 8.16:** Service Availability Provenance Query Pattern
SELECT ?service ?serviceName ?value
WHERE {
  ?reliability rdf:type serviceprov:ReliableMessaging.
  ?service serviceprov:hadNFP ?reliability.
  ?service serviceprov:serviceName ?serviceName.
  ?servexecution prov:used ?inMessage.
  ?inMessage prov:wasGeneratedBy ?orchestratorExecution.
  ?orchestratorExecution prov:value ?orchestratorExecutionID.
  FILTER (regex (str(?orchestratorExecutionID), "orchestratorExecutionID", "i"))
}

Listing 8.17: Service Reliability Provenance Query Pattern

the service identity (?service, ?serviceName) and the corresponding NFP(value) we are interested to capture (?value) (line 1), and it continues with identifying the corresponding type of NFP and identity of the execution instance (?serviceExecution) associated with the particular (?service) that formed part of the composition execution (?orchestratorExecution) (lines 2-14) we are interested. Since, availability of a service is a non-functional property for which we would like to explore its value for a fixed time interval or the lifecycle of a service, we narrow down our results by making assertions about the generation/invalidation timestamps for the particular service (lines 12-13). The query assertions continue by narrowing down the results to the (?orchestratorExecution) with unique orchestration execution identifier (?orchestratorExecutionID) (lines 16-23).

Similarly, Listings (8.17- 8.23) are designed to retrieve NFP information of a service (execution) such as reliability, security and reputation, cost and penalty, execution and response time.

Provenance Question Type 7.2: What were the QoSvalue(s) i of the QoS requirement(s) i offered for the selected service Si?
SELECT ?service ?serviceName ?value
WHERE
{
?service serviceprov:hadNFP ?security.
?service serviceprov:serviceName ?serviceName.
?servexecution prov:used ?inMessage.
?inMessage prov:wasGeneratedBy ?orchestratorExecution.
?orchestratorExecution prov:value ?orchestratorExecutionID.
FILTER ( regex ( str (?orchestratorExecutionID ), " orchestratorExecutionID ", "i") )
}

Listing 8.18: Service Security Provenance Query Pattern

Provenance Question Type 7.3: What was the QoSDescription i for the discovered service Si?

Provenance Template 7.2/7.3 In order to answer this question we need to capture information such as the QoSDescription(s) i, QoSRequirement(s) i offered and their values QoSvalue(s) i with regards to particular executions ServiceExecutions(s) i of service(s) Service Si. These set of entities and activities are necessary part of the provenance graph template towards the target entity QoSDescription i, QoSvalue i, forming the QoS contract (descriptions) and QoS requirement (values) offered as part of the discovered services that are used as candidates for the service selection process of the service composition. The set of paths in the provenance graph between the target entity and the entities/activities/agents forming part of the required provenance information to be recorded are analysed in the provenance template graph of Figure 8.15. It is often the case that a service execution instance or a service itself may have a number of associated QoS descriptions or requirements that we may be interested in capturing in parallel, therefore we represent those as parallel fragments in the template graph of Figure 8.15, labelled with the pair <cardinality, parallel type>, where the cardinality defined the possible number of QoS descriptions/requirements and their values respectively. We
WHERE {
  ?service serviceprov:hadNFP ?reputation.
  ?service serviceprov:serviceName ?serviceName.
  ?servexecution prov:used ?inMessage.
  ?inMessage prov:wasGeneratedBy ?orchestratorExecution.
  FILTER (regex (str (?orchestratorExecutionID), " orchestratorExecutionID ", "i"))
  ?servexecution prov:wasAssociatedWith ?server.
  ?association prov:hadRole ?role.
  ?server serviceprov:hadPort ?port.
  ?server serviceprov:hadIPAddress ?ip.
}

Listing 8.19: Service Reputation Provenance Query Pattern

formalise this query pattern as a set of triple assertions using SPARQL query language as shown in Listings 8.24 and 8.25, designed to retrieve the information about the QoS offered and QoS requirements for the atomic services that were discovered and selected to be part of the orchestration execution.

Provenance Query Patterns 7.2/7.3  Pointing from the future to the past in Listing 8.24 the query starts with identifying the triples related to our search query parameters (?QoSDescription) and (?QoSvalue) (line 1), and it continues with identifying the service selection activity instance (?servSelection) that used the particular (?QoSdescription) in order to generate the (?selectedServiceDescription) used by the planning activity (?planning) (lines 2-8). It then narrows down the results by making assertions about the (?orchestratorExecution) with unique orchestration execution identifier (?orchestratorExecutionID) that used a (?workplan) generated by the planning activity (lines 9-18). Similarly, in Listing 8.25 the query makes assertions in order to identify the values of QoSdescrip-
WHERE
{
?service serviceprov:hadNFP ?cost.
?servexec serviceprov:serviceName ?serviceName.
?servexec prov:used ?inMessage.
?inMessage prov:wasGeneratedBy ?orchestratorExecution.
?orchestratorExecution prov:value ?orchestratorExecutionID.
FILTER ( regex (str(?orchestratorExecutionID), "orchestratorExecutionID", "i"))
?servexec prov:wasAssociatedWith ?server.
?server serviceprov:hadPort ?port.
?server serviceprov:hadIPAddress ?ip.
}

Listing 8.20: Service Cost Provenance Query Pattern

ations or QoScontracts (?servDescriptionItem) associated with the collection of the discovered services for the particular orchestration execution we are interested.

Actors Patterns (Facet 8)

The eighth set of provenance question type patterns are providing a solution for expressing requirements about provenance where the data required to satisfy those requirements are information about different actors that own /manage /monitor services, an identifier for them (e.g., IP address), their association to services or the data that they manage.

Provenance Questions   Inspired by a set of provenance example questions for the travel planner scenario, we abstract from those in order to express a parameterized form of the questions, and identify the assertions/query conditions required in order to answer queries such as the following:
WHERE
{
?service serviceprov:hadNFP ?penalty.
?service serviceprov:serviceName ?serviceName.
?servexecution prov:used ?inMessage.
?inMessage prov:wasGeneratedBy ?orchestratorExecution.
?orchestratorExecution prov:value ?orchestratorExecutionID.
FILTER ( regex (str (?orchestratorExecutionID), "orchestratorExecutionID", "i"))
?servexecution prov:wasAssociatedWith ?server.
?association prov:hadRole ?role.
?server serviceprov:hadPort ?port.
?server serviceprov:hadIPAddress ?ip.
}

Listing 8.21: Service Penalty Provenance Query Pattern

Example questions for the travel planner scenario:

8.1 Who managed (e.g., particular bank) the authentication of the credit card number?

8.2 Was the authentication of the credit card made by a trusted third party?

8.3 Which third party takes control of the monitoring for QoS (e.g., response time, availability)?

We can express each question falling in “actors” patterns category in a parameterized/abstract form and formulate an abstract version of the provenance graph – provenance template – containing all the required provenance data and paths that would successfully answer the corresponding questions by reusing and combining templates described in the service and provider identity patterns. Therefore the provenance graph template of this category is a specialisation of the service and provider identity patterns category. The SPARQL queries formalising the corre-
Listing 8.22: Service Execution Time Provenance Query Pattern

Corresponding provenance questions of the actors patterns category are expressed as a set of service and provider identity triple assertions specialised for identifying different actors with a specific role.

8.3.8 Design Information Patterns (Facet 9)

The ninth set of provenance question type patterns are providing a solution for expressing requirements about provenance where the data required to satisfy those requirements are information about dependencies in the composition of services such as the abstract process plan or execution work plan describing the composition workflow; information on protocols or particular message format used between services.

Provenance Questions Inspired by a set of provenance example questions for the travel planner scenario, we abstract from those in order to express a
WHERE
{
?service serviceprov:serviceName ?serviceName.
?servexecution prov:used ?inMessage.
?inMessage prov:wasGeneratedBy ?orchestratorExecution.
?orchestratorExecutionExecution prov:value ?orchestratorExecutionID.
FILTER ( regex ( str(?orchestratorExecutionID),
"orchestratorExecutionID", "i") )
?servexecution prov:wasAssociatedWith ?server.
?server serviceprov:hadPort ?port.
?server serviceprov:hadIPAddress ?ip.
}

Listing 8.23: Service Response Time Provenance Query Pattern

parameterized form of the questions, and identify the assertions/query conditions required in order to answer queries such as the following:

Example questions for the travel planner scenario:

9.1 Which services preceded the payment service in the composition (abstract process) plan?

9.2 Did all the messages exchanged between the different services of the composite service use the same format /protocol?

We can express each question falling in “design information” patterns category in a parameterized/abstract form and formulate an abstract version of the provenance graph – provenance template – containing all the required provenance data and paths that would successfully answer the corresponding questions by capturing information about the abstract process work plan of the composition design.
SELECT ?QoSDescription ?QoSvalue
WHERE {
3 ?servSelection prov:used ?QoSDescription .
4 ?QoSDescription prov:value ?QoSvalue .
6 ?selectedServiceDescription prov:wasGeneratedBy ?servSelection .
7 ?selectedServiceDescription rdf:type serviceprov:SelectedServiceDescription .
8 ?workplan prov:wasGeneratedBy ?selectedServiceDescription .
10 ?orchestratorExecution prov:used ?workplan .
FILTER ( regex ( str (? orchestratorExecutionID ), " orchestratorExecutionID ", "i" ))
}

Listing 8.24: QoS Requirements Provenance Query Pattern

Usually, design information is described by a combination of BPMN (describing the abstract process plan), BPEL (describing the actual orchestration execution work plan) and WSDL (describing the services/message types and protocols used) descriptions. This information is provided by design so no extra provenance recording mechanism is required to capture this provenance information.

With regards to the paths not taken, as mentioned in the routes (not) followed patterns recording data flow only may never record a potential route if it is never followed by any service execution. Yet, capturing the history of design information only (such as BPMN and BPEL specifications) will provide information about all potential paths, but will not tell us which ones have actually been followed or not followed in a particular service execution. Therefore, in our design information templates we capture information about the WSDL descriptions (atomic services) BPMN and BPEL (composite services/service composition) specifications but this is only one piece of information we need in order to capture provenance of the all the paths that could be taken/not taken in the service composition workflow. At a second step we simulate all possible paths taken as defined within the BPEL code following the BPEL workflow execution. Those paths also incorporate information about the invocation of atomic services when the type of service composition execution is an orchestration. The path simulation
SELECT ?serviceName ?service ?servdescriptionItem ?value
WHERE {
  ?service serviceprov:serviceName ?serviceName.
  ?servdescription serviceprov:hadDescription ?servdescription.
  ?servdescription rdf:type serviceprov:ServiceDescription.
  ?servdescriptionItem prov:hadMember ?servdescriptionItem.
  ?servdescriptionItem rdf:type serviceprov:ServiceDescriptionDiscoveredItem.
  ?servdescriptionItem prov:value ?value.
  ?servCollection prov:hadMember ?servdescription.
  ?servCollection prov:wasGeneratedBy ?discovery.
  ?discovery prov:used ?servRequest.
  ?servRequest rdf:type serviceprov:ServiceRequest.
  ?workplan prov:wasGeneratedBy ?planning.
  ?orchestratorExecution prov:used ?workplan.
  ?orchestratorExecution prov:value ?orchestratorExecutionID.
  FILTER ( regex ( str(?orchestratorExecutionID), "^\w+$") )
}

Listing 8.25: QoS (Descriptions) Offered Provenance Query Pattern

is realised as part of the PROVa implementation of the Template Binding and ProvInfra Model Processing activities that are later in Chapter 10. Having those two pieces of information can then be used in order to identify the paths that have never been taken by calculating the delta of all the specified paths at design time minus the paths that were actually taken during the service (composition) execution.

In order to answer these questions we need to store though information such as the abstract process plan value BPMN Specification i, the execution work plan BPEL Specification i and the service specifications WSDL description i of Service Si respectively. These set of entities and activities are a necessary part of the provenance graph template towards the target design entity information. The set of paths in the provenance graph between the target entity and the entities/activities/agents forming part of the required provenance information to be recorded are analysed in the provenance template graph of Figure 8.16.

Summary: In this chapter we identified a set of provenance question type patterns that allows us to formally express provenance questions – representing prove-
Figure 8.15: QoS Requirements Offered Template

Provenance requirements of the users – exhibiting a number of facets as proposed in Chapter 6. Evaluation of the proposed provenance question type patterns with regards the criteria of well-formedness, expressiveness and realism is presented in detail in Chapter 11. Next, we present our research results on a behavioural model and a meta-model language for provenance infrastructure. A metamodel notation for provenance infrastructure would allow capturing the different sets of provenance data required to satisfy provenance awareness requirements for a composite service system throughout the SOC execution cycle.
Figure 8.16: Design Information Template
Chapter 9

A Behavioural Model for Provenance Infrastructure

In Chapter 8 we identified a set of provenance question type patterns for enabling analysis of provenance awareness support of a system. Those allow us to formally express provenance questions (provenance awareness requirements) in both a query and a graph form, exhibiting the proposed provenance facets as presented in Chapter 6; the provenance question type patterns were formed on the basis of the assertions that need to be made about the respective data required for answering potential questions about provenance of a SOC system.

For a full analysis of the provenance awareness support of a SOC system, we need to provide a description of the infrastructure design as well. In this chapter, our purpose is to propose a metamodel (language notation) that enables the specification and design of the infrastructure required for capturing provenance data for SOC systems. This allows modelling the structures of the provenance data that could be collected in the different phases of the SOC execution cycle, along with the provenance recording infrastructure activity being accountable for generating this data.

Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1</td>
<td>Context and Motivation</td>
<td>239</td>
</tr>
<tr>
<td>9.2</td>
<td>Background</td>
<td>241</td>
</tr>
</tbody>
</table>
9.1 Context and Motivation

Collecting the provenance required for satisfying the users’ provenance awareness requirements can only be guaranteed by the provision of sufficient provenance infrastructure mechanisms and architecture for capturing the provenance information consistently. Therefore, designing provenance awareness support for service-oriented computing (SOC) systems requires not only specifying and modeling the provenance data structures needed that enables us to make assertions and answer provenance questions of users, but also the infrastructure activity for recording and storing this data, the behavior of the provenance infrastructure activity which is consistent with the SOC execution cycle and how this infrastructure binds to a particular composite-service design.

Considering those various aspects required for developing consistently a provenance infrastructure design for a composite service system, we realise that this
could be a rather complicated task for an application designer to develop on his/her own. The application designer would need to develop applications satisfying an aggregated set of functional and non-functional requirements, as specified by a set of atomic and composite service specifications.

In order to take away the full focus and relax additional complications required to be managed by one role (person/application designer) with regards to composite service provenance infrastructure design and development, we have taken a solution towards formally specifying provenance infrastructure independently of specific services and applications. The real benefit here is that we can now allocate the provenance design and development work of a system to different people who can focus each time on one aspect of the overall design. Also, this approach enables decoupling any dependencies from the actual services’ design and allows reuse of the infrastructure models and specifications in different contexts.

In particular, taking a route towards formal modelling of the provenance infrastructure independently of the service’s design time gives us the ability to:

- *model* a particular provenance infrastructure design, its activity and behaviour and *simulate* the output result generated by the provenance infrastructure recording activity at design time,

- *bind* the given provenance infrastructure design to a particular composite-service system specification and *analyse* whether this leads to sufficient recording activity that would capture the data required to satisfy a set of provenance awareness requirements; a discussion about the simulation and analysis of the system’s provenance recording activity at the appropriate phases of the SOC execution cycle by generating sample provenance data graph snippets is presented in Section 9.6.

- *revise* and apply changes in the provenance infrastructure design independently and beforehand the system’s actual implementation and services’ realisation.
In summary, in this chapter we contribute:

1. a template provenance infrastructure model, which provides the required notation for modelling the provenance recording infrastructure activity for composite service systems, taking into consideration the SOC principles and fundamental concepts of the SOC execution cycle,

2. behavioural rules and repetition patterns modelling and exhibiting the provenance infrastructure recording behaviour as the system executes throughout the SOC execution cycle,

3. a template binding specification that enables binding the provenance infrastructure design to concrete service and service composition (orchestration) specifications.

4. a provenance awareness analysis algorithm that analyses instantiated provenance infrastructure models and generate the corresponding provenance data graphs. The algorithm interprets a set of behavioural rules and patterns describing the provenance infrastructure activity of a system design and the output result comprises a simulation of the provenance infrastructure recording utilities as part of the system’s design.

9.2 Background

One of the main building blocks of our framework that is described in this chapter is a template provenance infrastructure model for SOC systems which is built independently of particular service designs. In this section, we outline the foundations of themes and parametrized templates that allow us to built a modular provenance infrastructure design and then bind this to specific applications/services.

Describing non-functional requirements such as tracing or provenance requirements has an impact on multiple operations of a composite service system design
and across the different phases of the execution cycle of a service-oriented computing (SOC) system. The nature of such type of requirements makes a necessity they support scattering across those multiple operations and steps of the SOC execution cycle. Therefore, the realisation of a single operation of an atomic service within a service composition may also need to provide support for such crosscutting requirements/behavioural aspects tangled up with the core responsibilities of that operation. However, scattering and tangling makes a system’s design and therefore its realisation and development difficult to understand, extend and reuse.

9.2.1 Themes/UML

Though the design is an important activity, its benefits are reduced when crosscutting requirements are present. Theme/UML (Unified Modeling Language) [8] proposes a solution to such problems by providing a design for behaviour that crosscuts other behaviour in the design, allowing the separation of crosscutting requirements at the design phase, which leads into providing more flexibility and independence at the realisation and implementation phase of those designs.

A theme design model removes structural mismatch (e.g., modelling artefacts with crosscutting concerns that need to be bound into existing templated models) with decomposition capabilities (that separate/make independent modelling of those concerns but also enable integration of crosscutting artefacts as part of existing templated models) that support the separation of the design for each requirement into different design models called themes (meaning different templates therefore separate instantiations). A crosscutting theme is a design model that specifies the design of a crosscutting requirement independently from any design it may potentially crosscut and how that design may be reused wherever it may be required. Crosscutting themes utilise templates, which are parametrized elements (similar to the ones found in C++) proposed in the Unified Modeling Language (UML) [67].
Theme/UML supports separate design models as independent views called *design themes* denoted with a theme stereotype on a UML package. Within the model, design themes may specify crosscutting behaviour to be composed with other design themes. Crosscut elements are not referred to directly by name but are reasoned about within the design through the use of parameterized elements. A composition specification mechanism is provided to bind those parameters to “real elements” that will be crosscut. The composition specification is achieved by Theme/UML’s composition relationship that allows the designer to identify those parts of the theme designs that are related to each other and that therefore should be composed.

UML defines a template as a parametrized model element that cannot be used directly in a design model [67]. Instead, it may be used as the basis to generate other model elements using a binding dependency relationship. A binding relationship defines arguments to replace each of the template parameters of the template model element. UML orders template parameters in a dotted box on the template class. Within the dotted box, template parameters may be grouped by class and ordering is important to support composition specification.

In our proposed framework we adopt a similar approach in order to define crosscutting provenance infrastructure requirements/behavioural aspects at design level tangled up with the core operations/functionality of atomic and/or composite service specifications and service compositions (orchestrations) as described in Section 9.3.

Next we discuss in detail the fundamental building blocks of UML used to support the implementation of themes mechanism, including templates and parameterized template elements, UML profiles and stereotypes.
9.2.2 UML Profiles and Stereotypes

**Profiles** The Profiles package in UML contains mechanisms that allow metaclasses from existing metamodels to be extended to adapt them for different purposes. This includes the ability to tailor the UML metamodel for different platforms or domains (e.g., business process modelling).

The Profile mechanism has been specifically defined for providing a lightweight extension mechanism to the UML standard. In UML 1.1, stereotypes and tagged values were used as string-based extensions that could be attached to UML model elements in a flexible way. In subsequent revisions of UML, the notion of a Profile was defined in order to provide more structure and precision to the definition of Stereotypes and Tagged values. The UML2.0 infrastructure and superstructure specifications [67] have carried this further, by defining it as a specific metamodeling technique, and this is the notation we currently follow as part of our framework. Stereotypes are specific metaclasses, tagged values are standard meta attributes, and profiles are specific kinds of meta packages.

Within the requirements for profile semantics it is important to note that a profile should be able to specialise the semantics of standard UML metamodel elements (like in our case we require to specialise activity metamodel element as a provenance activity). The vast majority of UML CASE (Computer Assisted Software Engineering tools should be able to implement Profiles. The design of UML profiles should therefore not constrain these tools to have an internal implementation based on a meta-metamodel/metamodel architecture. In our implementation, we use Papyrus UML [53] for the provenance activity Profile realisation.

**Profiles Extensibility** The profiles mechanism is not a first-class extension mechanism (i.e., it does not allow for modifying existing metamodels). Rather, the intention of profiles is to give a straightforward mechanism for adapting an existing metamodel with constructs that are specific to a particular domain, plat-
form, or method. Each such adaptation is grouped in a profile. It is not possible to take away any of the constraints that apply to a metamodel such as UML using a profile, but it is possible to add new constraints that are specific to the profile. The only other restrictions are those inherent in the profiles mechanism; there is nothing else that is intended to limit the way in which a metamodel is customised.

There are several reasons why you may want to customise a metamodel:

- Give a terminology that is adapted to a particular platform or domain. For example, in our model case, we provide a terminology for a provenance infrastructure activity metamodel.

- Give a syntax for constructs that do not have a notation (such as in the case of actions). For instance, in our thesis, we provide a syntax for provenance infrastructure actions/activities and their behaviour by extending the base metaclass action of UML.

- Add information that can be used when transforming a model to another model or code (such as defining mapping rules between a model and Java code). For example, in our model case, we use the provRecording (stereotype) and provConcepts labels as part of a UML profile to translate provenance data modelled elements to RDF triples representing the generated provenance data graphs.

**Stereotypes** A stereotype defines how an existing metaclass may be extended, and enables the use of platform or domain specific terminology or notation in place of, or in addition to, the ones used for the extended metaclass [67].

Just like a class, a stereotype may have properties, which may be referred to as tag definitions. When a stereotype is applied to a model element, the values of the properties may be referred to as tagged values. A stereotype can change the graphical appearance of the extended model element by using attached icons. When this association is not null, it references the location of the icon content to be displayed within diagrams presenting the extended model elements [67].

245
A stereotype is a limited kind of metaclass that cannot be used by itself but must always be used in conjunction with one of the metaclasses it extends. Each stereotype may extend one or more classes through extensions as part of a profile [67].

Similarly, a class may be extended by one or more stereotypes. An instance “S” of Stereotype is a kind of (meta) class. Relating it to a metaclass “C” from the reference metamodell (typically UML) using an Extension (which is a specific kind of association), signifies that model elements of type C can be extended by an instance of “S” (such as in Figure 9.1). At the model level (such as in Figure 9.2) instances of “S” are related to “C” model elements (instances of “C”) by links (occurrences of the association/extension from “S” to “C”).

Any model element from the reference metamodell (any UML model element) can be extended by a stereotype. For example in UML, States, Transitions, Activities, Use cases, Components, Attributes, Dependencies, etc. can all be extended with stereotypes. For the needs of our framework, we will use the stereotype (meta)class mechanism to extend metaclass “Action/Activity” with the stereotype “ProvRecording” (as shown in Figure 9.6) as part of a profile.

**Stereotype Notation and Presentation**

A Stereotype uses the same notation as a Class, with the addition that the keyword “stereotype” is shown before or above the name of the Class. When a stereotype is applied to a model element (an instance of a stereotype is linked to an instance of a metaclass), the name of the stereotype is shown within a pair of
guillemets above or before the name of the model element. If multiple stereotypes are applied, the names of the applied stereotypes are shown as a comma-separated list with a pair of guillemets [67].

In UML 2.0 [67], a tagged value can only be represented as an attribute defined on a stereotype. Therefore, a model element must be extended by a stereotype in order to be extended by tagged values. There are different ways to represent the values of a stereotyped element. In the modelling notation, we are going to introduce a compartment and a comment symbol is used.

In the case where a compartment or comment symbol is used, the stereotype name may be shown in guillemets before the name string in addition to being included in the compartment or comment. The values are displayed as name-value pairs:

For the requirements of our framework we use this latter notation. In particular, we introduce stereotyped provRecording activities with a multi-valued property provConcepts that carries a value string with a comma-separated list of the provenance data concepts.
When a stereotype includes the definition of an icon, this icon can be graphically attached to the model elements extended by the stereotype. Every model element that has a graphical presentation can have an attached icon. As part of our framework, we introduce our own prov icon that we graphically attach to the boxes representing the model elements extended by the provRecording stereotype.

UML Templates

The UML Templates package specifies how foundational UML model elements can be parametrized using template parameters. The package introduces mechanisms for defining templates, template parameters, and bound elements [67].

Parameterable Element A parameterable element is an element that can be exposed as a formal template parameter for a template, or specified as an actual parameter in a binding of a template.

In the second case the ParameterableElement can be referenced by a TemplateParameterSubstitution when used as an actual parameter in a binding of a template. In an element bound to the template, any use of the template parameter will be substituted by the use of the actual parameter.

Templateable Element or Template A templateable element is an element that can optionally be defined as a template and bound to other templates.

TemplateableElement may contain a template signature that specifies the formal template parameters and is often referred simply as a template since it mainly forms a template specification. TemplateableElement may contain bindings to templates that describe how the templateable element is constructed by replacing the formal template parameters with actual parameters and is often referred to as a bound element.
Such a template is a parameterized element that can be used to generate other model elements using TemplateBinding relationships (see Template Binding para. in 9.2.2). The template parameters form part of the template signature (see Template Signature para. in 9.2.2) that specifies the formal parameters that will be substituted by actual parameters in a binding. A template parameter (see Template Parameter para. in 9.2.2) is defined in the namespace of the template, but the template parameter represents a model element that is defined in the context of the binding.

The semantics and well-formedness rules for the bound element must be evaluated as if the bindings were expanded with the substitutions of actual elements for formal parameters. The semantics of a binding relationship is equivalent to the model elements that would result from copying the contents of the template into the bound element, replacing any elements exposed as a template parameter with the corresponding element(s) specified as actual parameters in this binding.

A bound element may have multiple bindings, possibly to the same template. In addition, the bound element may contain elements other than the bindings. The specific details of how the expansions of multiple bindings, and any other elements owned by the bound element, are combined together to fully specify the bound element are found in the subclasses of TemplateableElement. The general principle is that one evaluates the bindings in isolation to produce intermediate results (one for each binding), which are then merged to produce the final result. It is the way the merging is done that is specific to each kind may contain both a template signature and bindings. Thus a template able element may be both a template and a bound element.

**Templateable Element Notation and Presentation** If a TemplateableElement has template parameters, a small dashed rectangle is superimposed on the symbol for the template able element, typically on the upper right-hand corner of the notation. The dashed rectangle contains the list of the formal template parameters. The parameter list must not be empty, although it might be suppressed
A bound element has the same graphical notation as other elements of that kind. Each binding is shown using the notation described under TemplateBinding. An alternative presentation for the bindings for a bound element is to include the binding information within the notation for the bound element. Typically the name compartment would be extended to contain a string with the following syntax as shown in Listing 9.2.

**Template Binding** A template binding represents a relationship between a templateable element and a template. A template binding specifies the substitutions of actual parameters for the formal parameters of the template.

TemplateBinding is a directed relationship from a bound templateable element to the template signature of the target template. It mainly owns a set of template parameter substitutions. A template parameter substitution specifies the set of actual parameters to be substituted for a formal template parameter within the context of a template binding.

The presence of a TemplateBinding relationship implies the same semantics as if the contents of the template owning the target template signature were copied into the bound element, substituting any elements exposed as formal template parameters by the corresponding elements specified as actual parameters in this
binding. If no actual parameter is specified in this binding for a formal parameter, then the default element for that formal template parameter (if specified) is used.

Template Binding Notation and Presentation A TemplateBinding is shown as a dashed arrow with the tail on the bound element and the arrowhead on the template and the keyword bind. The binding information is generally displayed as a comma-separated list of template parameter substitutions as shown in Listing 9.3.

Template Parameter A template parameter exposes a parameterable element as a formal template parameter of a template. This parameterable element is meaningful only within the template, or other templates that may have access to its internals (e.g., if the template supports specialisation).

A TemplateParameter may own the exposed ParameterableElement in situations where that element is only referenced from within the template. Each exposed element constrains the elements that may be substituted as actual parameters in a binding. A TemplateParameter may reference a ParameterableElement as the default for this formal parameter in any binding that does not provide an explicit substitution or it may own this default ParameterableElement in situations where the exposed ParameterableElement is not owned by the TemplateParameter.

Template Parameter Notation The general notation for a template parameter is a string displayed within the template parameter list for the template as shown in Listing 9.4.
Listing 9.4: Template Parameter Notation

Figure 9.3: Template Class and Bound Class [67]

Template Signature  A template signature bundles the set of formal template parameters for the associated templated element. It is owned by a TemplateableElement and has one or more TemplateParameters that define the signature for binding this template - specify the elements that may be substituted in a binding of the template.

Bound Template Example  The example shows a class template (named FArray) with two formal template parameters. The first formal template parameter (named T) is an unconstrained class template parameter. The second formal template parameter (named k) is an integer expression template parameter that has a default of 10. There is also a bound class (named AddressList) that substitutes the Address for T and 3 for k.
9.2.3 Specifying Services and Service Compositions

As part of our specification and modelling framework solution for provenance awareness, we also need to specify service and service composition specifications. Those specifications contain the element (values) about specific services and composite service applications we would like to bind to the provenance infrastructure design. This is enabled through the template binding mechanisms introduced as part of our framework that we discuss in Section 9.2.2.

WSDL  Web Services Description Language (WSDL) [107] for defining services has been introduced in detail in Chapter 2. WSDL specification is a service description that describes:

- the operations that can be performed by a particular service,
- the SOAP bindings including input/output messages and parts of messages,
- the service name (targeted Namespace - URI) and type,
- information about the location of the server under which the service is executed and maintained including the PortTypeName and IPaddress.

An example WSDL diagram for the the travel planner case scenario created using the Eclipse WSDL Editor [152] is shown in Figure 9.4.

BPEL  Business Process Execution Language for Web Services (BPEL4WS) [4] is a language used for the definition and execution of business processes using Web services. BPEL enables the top-down realisation of Service Oriented Architecture (SOA) through composition, orchestration, and coordination of atomic (Web)
services. It provides a relatively easy and straightforward way to compose several atomic services, which involves the integration of the underlying applications and their functionalities, into new composite services called business processes. It forms an XML-based language that supports the Web services technology stack, including SOAP, WSDL, UDDI, WS-Reliable Messaging, WS-Addressing, WS-Coordination, and WS-Transaction and a rich vocabulary for the description of business processes.

Web services can be combined in two ways: Orchestration or Choreography. An extended discussion on the definition of those two types is provided in Chapter 2. From the perspective of composing Web services to execute business processes, the orchestration is a more flexible paradigm and we choose to use an example of this to showcase the BPEL specification of a standard example composite service in Section 9.5.2.

A BPEL process specifies the order\(^1\) in which participating atomic (Web) services should be invoked, either sequentially or in parallel. With BPEL, you can express conditional behaviours, construct loops, declare variables, copy and assign values, define compensation, termination and fault handlers. By combining all these constructs, you can define complex business processes in an algorithmic manner.

A BPEL process consists of steps; each step is called an “activity”. BPEL supports basic as well as structured activities. Basic activities represent basic constructs and are used for common tasks, such as the following:

- Invoking other Web services, using `<invoke>`

- Waiting for the client to invoke the business process by sending a message, using `<receive>` (receiving a request)

---

\(^1\)BPEL does not necessarily specify the exact order of the service invocations. Activities in flows can happen in any order as long as some constraints are met, and you could have asynchronous event-driven compositions where the composed services set the tune.
• Generating a response for synchronous operations, using <reply>

• Manipulating data variables, using <assign>

• Indicating faults and exceptions, using <throw>

• Waiting for some time, using <wait>

• Terminating the entire process, using <terminate>

We can then combine these and other basic activities to define complex algorithms that specify exactly the steps of business processes. To combine basic activities, BPEL supports several structured activities such as:

• Sequence <sequence>, which allows the definition of a set of activities that will be invoked in an ordered sequence

• Flow <flow> for defining a set of activities that will be invoked in parallel

• Case-switch construct <switch> for implementing branches

• While <while> for defining loops

• The ability to select one of the several alternative paths, using <pick>

The structure of a BPEL specification starts with an empty BPEL process outline that presents the basic structure of each BPEL process definition document e.g., <process name="BusinessTravelProcess" ... > as shown in Listing 17.2 line 3. In a typical scenario, the BPEL business process receives a request. To fulfil it, the process invokes the involved Web services and then responds to the original caller. Because the BPEL process communicates with other Web services, it relies heavily on the WSDL description of the Web services invoked by the composite Web service. Therefore, before starting to write the BPEL process definition, one needs to become familiar with the Web services invoked from the business process. These services are mainly the partner Web services bound to partnerLinks (as shown in Listing 17.2 line 18).
Partner link types represent the interaction between a BPEL process and the involved parties, which include the Web services the BPEL process invokes and the client that invokes the BPEL process. Partner link types are defined in the WSDL within a special namespace: `http://schemas.xmlsoap.org/ws/2003/05/partner-link/`. Each partner link type can have one or two roles and for each role, one needs to specify the portType it uses. For synchronous operations, there is a single role for each partner link type because the operation is only invoked in a single direction. For asynchronous callback operations, two roles need to be specified. The first role describes the invocation of the operation by the client. The second role describes the invocation of a callback operation.

BPEL specification also declares variables as shown in Listing 17.2 line 29. Variables in BPEL processes are used to store, reformat, and transform messages. One usually needs a variable for every message e.g., input message in Listing 17.2 line 31 and output message in listing 17.2 line 36, sent to and received from the partners. For each variable you have to specify the type e.g., WSDL message type, XML Schema simple type, or XML Schema element.

The process main body specifies the order in which the partner Web services are invoked and the orchestration logic, as shown in Listing 17.2 line 44. Usually, it starts with a `<sequence>` that allows you to define several activities that will be performed sequentially. Within the sequence, you first specify the input message that starts the business process. This is done with the `<receive>` construct, which waits for the matching message. Within the `<receive>` construct, you do not specify the message directly. Rather what is specified is the partner link, the port type, the operation name, and optionally the variable that holds the received message for consequent operations.

Finally, as long as a business process is defined in BPEL, this essentially forms a definition of a new Web service that is a composite of existing services. The interface of the new BPEL composite Web service uses a set of port types through which it provides operations like any other Web service. To invoke a business pro-
An example of the BPEL specification in a diagrammatic version for the travel planner scenario created with Eclipse BPEL Designer [45] plugin is shown in Figure 9.5.

Figure 9.5: BPEL Diagram for the Travel Planner Use Case
9.3 Provenance Infrastructure Design

In Chapter 5 we presented the main building architectural blocks for the proposed provenance awareness specification and analysis framework. Among those we discussed:

- a Provenance Infrastructure Template Model providing the required modelling notation to model the provenance data recording activity and behaviour for the provenance infrastructure design.

- a Template Binding Specification used to bind the provenance infrastructure design, reflected in the Provenance Infrastructure Template Model, to specific atomic service and service composition descriptions expressed as WSDL and BPEL/BPMN specifications respectively.

As we have already mentioned in our motivation and context section, designing the provenance infrastructure of a service-oriented computing system can become quite a complex task for a service/application designer on its own to deliver; it requires taking into consideration all the principles, basic constructs, concepts and processes forming part of the SOC execution cycle and how those fit together with the provenance infrastructure design and reflect on the provenance infrastructure behavior. It, therefore, becomes essential that specialised people with the infrastructure designer role would work along with application or service designers in order to model the provenance infrastructure required for a service-oriented computing system.

In addition, an infrastructure design solution may be reused in different contexts, finding fit for different composite-service systems or applications. Therefore, once the provenance infrastructure behaviour for a system has been modelled, designers may apply the same model design into different composite service systems or applications. In order to enable this, the provenance infrastructure model design needs to be made independent of specific applications or services. In our framework this is achieved by introducing 1) the Provenance Infrastructure
The *Template Model* as a separate model from specific service specifications and 2) the *Template Binding Specification*, a binding mechanism between the two (as shown in Figure 5.2).

The *Provenance Infrastructure Model*, is mainly a behavioural model that enables the specification of the provenance infrastructure required in order to issue the relevant provenance recording calls that take place as a service request is routed throughout the different phases of the SOC execution cycle. Those calls are expressed as *stereotyped* provenance recording activities of a UML activity diagram with a tagged value containing the list of provenance data to be recorded as part of each recording activity (as shown in Figure 9.6).

The provenance data tagged value content of the designed activities is independent of specific services and service composition specifications but considers the different concepts and artefacts introduced in the different phases of the SOC execution cycle where provenance activities need to be set up as described in detail in Section 9.4.
The provenance infrastructure model forms a templateable element (template) and part of this is a template signature. Template signature bundles the set of formal template parameters owned by the model, corresponding to SOC concepts and artefacts, that could be substituted by actual parameters in a template binding with specific services and their compositions.

Instantiating such model at the level of each activity can then be used as a basis in order to simulate the provenance data recording result of the provenance infrastructure design of a system and analyse/qualify the system’s provenance awareness support at system’s design time. In order for a provenance infrastructure model to take into account service-specific, composition-specific or invocation-specific provenance activities, we rely on building different infrastructure models for instantiating different services and their corresponding provenance activities/behaviour.

The functionality (operations) offered by the specific applications and the control flow/sequence of individual services within the composition are introduced by the Service Interface and Service Composition Model respectively, while the connection between them and the infrastructure design is realized with the Template Binding Specification, that specifies the essential connections among the provenance properties (artifacts) introduced in the infrastructure model and the corresponding services and their composition workflow. This separation of models in our architectural design allows us to clearly distinguish now the roles of the infrastructure designer, who defines how the infrastructure behaves, and the application designer, that defines how a specific application makes use of such an infrastructure.

In particular the application designer is mainly in charge of the Service Interface and Service Composition models that provide concepts to specify the functional properties of their applications, considering the operations offered by the specific applications and the control flow/sequence of individual services within the composition respectively. The application designer will then also need to incor-
porate provenance related aspects into the system’s design. Those correspond to the provenance data structures, represented by the ServiceProv model, that need to be captured in order to answer a set of provenance questions. Provenance questions are handled at a first step by the domain expert that is familiar with the business domain of different service-based systems and is able to analyse the users’ preferences and express more systematically their needs (questions) and at a second step by a QoS (provenance) analyst that translates the users’ questions into a machine-processable format such as provenance query patterns expressed in SPARQL [74].

In this way provenance specialists enable the automated analysis of the system’s provenance awareness support and also help the application designers to interpret the results of this analysis into meaningful conclusions of whether the system’s provenance awareness design is sufficient to cover the provenance users’ requirements. We achieve this by implementing a generic analysis algorithm that simulates the systems provenance infrastructure behaviour by generating the corresponding provenance-data graphs that would be captured by a particular provenance infrastructure in a queryable form as described in Section 9.6.1.

9.4 Provenance throughout the SOC Lifecycle

Before we present our provenance infrastructure modelling approach in detail, we need to discuss the main activities introduced by the SOC paradigm and identify the key points where provenance data needs to be recorded. We express the provenance infrastructure modelling requirements at the different SOC phases based on provenance questions for the travel planner scenario. The questions have been chosen to cover facets we have previously identified in Chapter 6, while a detailed description of the structure of this data can be found in Chapter 7.

The SOC phases [118] identified where we may need to set up provenance collection activity are:
1. **Service Interface Publishing:** where service providers announce the new services ready for consumption, by publishing their service description into a central registry [120]. Identifying the services that were available during a run of a SOC system gives the opportunity to collect data for answering questions like which were the providers of the available services during their service discovery or which services were actually available? Those questions require capturing provenance data such as the **IP address** and **Port** of the provider or the **Service Description Published Item(s)**.

2. **Service Registry Activity:** where a third-party service advertises descriptions of the available services, making the services discoverable to other providers or brokers serving as clients [120]. In this phase we could capture data about the **Registry’s identity** that answers questions like which registries were involved in the discovery of the flight booking service(s).

3. **Service Brokering & Discovery:** where a service broker serves as an intermediary that is interposed between service requesters and providers to enable discovery of services that satisfy the requesters’ functional/non-functional requirements [120]. Here we are interested to answer questions such as what was the service request of the travel planner execution or which broker was used in the service discovery of the flight booking service, which require collection of provenance data such as **Service Request**, **IP address** and **Port** of the broker involved in the discovery and the actual **Service Description(s)** discovered.

4. **Service Planning:** where a planner i) computes an abstract process plan that is then used as a service request plan for the discovery of services by the service broker, ii) computes the final composition work plan for execution [119]. The provenance that can be captured at this stage answers questions like the following: what was the **Abstract Process Plan** that enabled discovery of services for the travel planner or which planner computed the actual composition **Work Plan**.
5. **Service Selection:** where a service selector selects the most suitable services satisfying the requester’s functional and QoS properties requested [119]. The selector receives a set of descriptions discovered through the service broker and then makes the selection by checking those against a set of QoS specifications. In this phase, we are interested to answer questions like which flight booking service was selected to be invoked, what were the reasons the specific service was selected or which was the identity of the service selector. Those questions require capturing provenance data such as Selected ServiceDescription(s), QoS requirements descriptions and IPAddress/Port of the Service Selector.

6. **Orchestration Execution:** where a central orchestrator manages the execution of the overall work plan, invoking atomic services as required [119]. Provenance questions identified at this point are related to the orchestrator and the message flow between the orchestrator and the individual services, such as which Orchestrator executed the flight booking service or how were the Input/Output Messages of the flight booking service derived during the travel-planner execution.

7. **Atomic Service Execution:** where an atomic service hosted by a web service container is invoked to be executed by the orchestrator [120]. Capturing provenance of atomic services is used to answer questions like which server executed the flight booking service, what were the input/output messages of this execution, which was the start/end service execution time, or what was the performance/resource availability for the service execution. Answering those queries requires collecting data such as identity information for the Service Executor, Input/Output Message(s) and their Parameter(s), Execution Timestamps, NFPs and Resources available which were associated with the particular execution.

This provenance SOC life cycle provides some insight on what provenance recording activity needs to be set up during the execution of a composite ser-
vice summarised in Table 9.1. Next, we specify the way in which the provenance infrastructure activity and its behaviour should be modelled in order to ensure the respective provenance data collection.

9.5 Modelling Provenance Infrastructure

Having identified the key points where provenance needs to be recorded throughout the service execution life cycle, in this section, we introduce our behavioural model for representing provenance awareness recording activity of composite service systems and applications. We then show how generic (*ServiceProv* ontology) provenance concepts associated with the provenance recording activities of our behavioural model and concepts specific to the application under development fit together in Section 9.6.3.

9.5.1 Provenance Infrastructure Template Model

Our proposed model for describing the provenance infrastructure behaviour of composite service systems is formalized as a UML activity diagram, that represents how a service request is routed through the execution phases of a composite service system as a whole (e.g., service interface publishing and registry activity, service discovery, service selection, planning, and orchestration/atomic service execution), and specifies:

1. the steps where provenance recording activity takes place,
2. the type/structure of provenance data that could be captured at each step,
3. the data flow connections between provenance data that could be captured in different provenance recording steps, and
4. the template signature of the template infrastructure model which specifies the set of formal template parameters that could be substituted by actual parameters in a template binding with specific services and SOC systems.
The UML activity diagram presented in this section forms an example model rather than a canonical model to represent the structure of a composite service system. The key contribution of our behavioural model lies in introducing special model elements to express the provenance recording activities and the type/structure of provenance data captured. Those elements essentially maintain their generalisability aspect which makes them applicable for specifying the provenance infrastructure behaviour of different service-based systems. Thus, we introduce ProvRecording Activity, a stereotyped [67] provenance recording activity, as shown in Figure 9.7, extending the UML activity model element to adapt for the modelling of provenance infrastructure recording purposes. This is rendered as an activity with its name in bold enclosed in guillemets and indicated by a specific Prov icon.

Specific properties – tagged properties [67]– are introduced for the stereotyped activities in order to represent the list of provenance data to be captured in each Prov Recording Activity through the different phases of the SOC execution cycle. In particular, we introduce the tagged property provConcepts that carries a String tagged value, where the tagged value equals to the list of provenance data concepts we would like to be captured under a particular provenance recording activity. The tagged value of provConcepts mainly represents a list of concept types as those defined in the ServiceProv ontology model [158]. In the actual implementation the provConcepts, value is defined as a set of ServiceProv triples as we will discuss in Chapter 10 that are passed as a string list value.

In our model, we also represent the data flow between ProvRecording activities which contains the identifiers for provConcepts, such as subtypes of ServiceProv agents, activities and entities used by each activity in constructing its own PROV subgraph. We describe the data flow by introducing unique ProvIDs for any elements listed as part of the tagged value that we would like to pass between the different activities. The provenance data flow is critical in order to provide a connected picture of the systems processing history by linking together the provenance captured in each ProvRecording Activity. It is the means to ensure a
Figure 9.7: Provenance Recording throughout SOC Execution Cycle

queryable provenance data structure of the SOC execution cycle that allows cross-
system linking and the systems’ processes (involving service brokering, discovery,
selection, orchestration systems/agents) by using URI identifiers.
With regards to expressing the control flow of the activity of the service execution life-cycle phases identified in Section 9.4 our model uses a set of standard UML model elements following the UML Superstructure Specification v2.2 [67]. The list of model elements contains:

- **Action** elements used to specify the main actions of the SOC execution life cycle activity,

- **Expansion Region** elements used to surround the processes/actions to be executed multiple times following either a parallel or iterative concurrency type. *Parallel* reflects that the process/action in the expansion region can occur at the same time (e.g., service publishing, discovery, selection), whereas *Iterative* specifies that execution must occur sequentially (e.g., atomic service execution),

- **Partition** elements used to separate actions throughout the SOC life cycle which are performed by different classifiers. An activity partition is a kind of activity group for identifying actions that have some characteristic in common e.g., in our provenance infrastructure model we identify as classifiers agents playing the roles of the service requester, service executor, orchestrator, service planner, service selector, service broker, registry and service provider as shown in Figure 9.7,

- **Datastore** stereotype (introduced by UML) used to represent an object which stores objects persistently. In our provenance infrastructure model, we introduce a special *Registry Datastore* to represent a central repository for service descriptions including Quality and SLAs contracts.

- UML notation also allows us to express the **ObjectFlow** connections between the SOC actions and **ProvRecording** activities respectively, realized through **Input and Output Pins**, and the **Initial/Final Nodes** of the activity model.

Figure 9.7 presents an example model for a specific infrastructure configuration of a composite service system. This is formalized as a UML activity diagram,
Table 9.1: Provenance Data Recording Steps throughout Service Execution Lifecycle

<table>
<thead>
<tr>
<th>Provenance Recording Steps</th>
<th>Provenance Data Recorded</th>
</tr>
</thead>
<tbody>
<tr>
<td>step 1: Service Publishing</td>
<td>Server(ServiceProvider), :IPaddress, :Port, :ServiceInterfacePublishing, :ServiceDescriptionPublishedItem</td>
</tr>
<tr>
<td>step 2: Service Brokering</td>
<td>ServiceRequest, :Planning, :Server(Planner), :IPaddress, :Port</td>
</tr>
<tr>
<td>step 3: Access AbstractPlan</td>
<td>AbstractProcessPlan</td>
</tr>
<tr>
<td>step 4: Registry Activity</td>
<td>Server(Registry), :IPaddress, :Port, :ServiceDescriptionDiscoveredItem</td>
</tr>
<tr>
<td>step 5: Service Discovery</td>
<td>ServiceDiscovery, :ServiceCollection, :ServiceDescription, :serviceDescriptionDiscoveredItem, :Server(ServiceBroker), :IPaddress, :Port</td>
</tr>
<tr>
<td>step 6: QoS Repository</td>
<td>QoSrequirements</td>
</tr>
<tr>
<td>step 7: Service Selection</td>
<td>Server(ServiceSelector), :ServiceSelection, :IPaddress, :Port, :SelectedServiceDescription</td>
</tr>
<tr>
<td>step 8: Planning</td>
<td>SelectedServiceDescription, :Planning, :Server(Planner), :IPaddress, :Port, :WorkPlan</td>
</tr>
<tr>
<td>step 9: Orchestration</td>
<td>Server(Orchestrator), :IPaddress, :Port, :OrchestratorExecution, :OrchestratorExecutionID, :Message(s), (prov:used/prov: wasGeneratedBy/prov:DerivedFrom), :start, :end (orchestration execution timestamps)</td>
</tr>
</tbody>
</table>

using a set of standard UML model elements [67] to express the activity (control/data flow) of the service execution cycle phases identified in Section 9.4. Our model also introduces the key points (provenance recording steps as summarized in Table 9.1) where ProvRecording activity could take place along with the provenance data (provConcepts) that could be recorded in each activity and the provenance data flow connections between those activities (ProvRecording Activities) where the data flow takes the form of provenance data having unique ProvIDs flowing from one provenance recording activity to another. It is important to note that this model includes the maximal provenance recording configuration for a SOC system, but specific applications will make specific choices of what provenance to record at any of these steps.

Before and independently of any service request sent out by a requester, one or more service providers would publish descriptions of the services they offer. This is the first step (step1: Prov ServicePublishing) where we could set up capturing provenance information such as identification information about the service provider and the descriptions published. Prov ServicePublishing activity is surrounded by a (parallel) expansion region that denotes that this activity could be executed multiple times in parallel reflecting that the process/action in the
expansion region can occur at the same time. Then, we assume that the flow of our activity model starts when a requester initiates a service request (Send Service Request action) which is received by a broker that acts on its behalf. As we are interested in handling more complex requests, the broker first invokes a planner (Invoke Planner action) that is able to compute a composite service plan based on the received request. In order to achieve this, the planner uses a set of abstract process plans kept in a repository (Access Abstract Plan action). Those actions are associated with additional provenance recording steps, namely step2: Prov ServiceBrokering and step3: Prov AccessAbstractProcessPlan. The broker receives the updated service request and then discovers (Discover Services action) a set of service descriptions items by accessing a central (Registry). Data about the identification of the registry’s identity is captured in step4: Prov RegistryActivity, while additional provenance about the service discovery process in step5: Prov ServiceDiscovery. The service descriptions discovered are received by the service selector that takes the (Select Services action) based on a set of QoS specifications (Access QoS Requirements action). Provenance for these actions are covered by the step6: Prov QoSRepository and step7: Prov ServiceSelection activities respectively. The service descriptions selected are sent to a planner that computes the actual work plan for execution (Produce Concrete WorkPlan action). Provenance recording activity with regards to the selected service descriptions and the planner could take place in step8: Prov Planning. Finally, an orchestrator receives the computed work plan and starts the invocation of atomic services (Orchestrate WorkPlan action). Recording activity for capturing orchestration execution provenance is modelled by step9: Prov Orchestration, while recording with regards to the provenance of atomic service execution is set up in step10: Prov Service Execution.

The Template Infrastructure Model contains a template signature with a set of template parameters that could be bound to actual values. This is shown as a dashed rectangle on the upper right-hand corner of the model as shown in Figure 9.7 and the full list of template parameters contained in the signature are
listed in below in pseudocode.

- Template Signature of Template Provenance Infrastructure Model

  - Template Parameters for Atomic Services
    * templateParameter :Service
    * templateParameter :ServiceName
    * templateParameter :MessageIn
    * templateParameter :MessageOut
    * templateParameter :Parameter
    * templateParameter :IPaddressValue
    * templateParameter :PortValue
    * templateParameter :ServiceDescription
    * templateParameter :ServiceDescriptionDiscoveredItem
    * templateParameter :ServiceDescriptionPublishedItem

  - Template Parameters for Compositions (Business Processes)
    * templateParameter :OrchestratorExecution
    * templateParameter :OrchestrationExecutionID
    * templateParameter :IPaddressOrchestrator
    * templateParameter :PortOrchestrator
    * templateParameter :WorkPlanValue
    * templateParameter :AbstractProcessPlanValue

The infrastructure can then be bound to specific SOC systems and services by introducing a Template Binding Specification that makes the essential connections by mapping actual values of service (WSDL) and business process (BPEL) models to template parameters of the provenance infrastructure model. We describe how these parameters are bound to specific service models through the Template Binding Model in the next section.
9.5.2 Template Binding

Part of the template provenance infrastructure architecture design is a template binding specification (model). This owns a set of template parameter substitutions and binds parameters of specific services and composite business processes under development with the required infrastructure configuration.

In the previous section we showed the template parameters forming part of the template signature of the provenance infrastructure model. Here, we first discuss how the set of template parameters representing provConcepts values as part of the provRecording Activities map to model elements of WSDL and BPEL/BPMN descriptions. Second, we look into how the template parameters get substituted by specific service and service composition (orchestration) information and information about the sequence of services within the composition (orchestration) workflow.

9.5.3 Template Binding Specification

In order to be able to analyse provenance awareness support of specific services and SOC systems we need to bind a set of actual service parameters to the template signature parameters of the provenance infrastructure model. We thus introduce a Template Binding Specification that specifies the essential substitutions of the template parameters with actual parameter values of services (WSDL) and business process (BPEL) models. The list of bindings specified contain bindings of service specific information such as identification for the atomic services and their input/output messages (e.g., ServiceName, Port and IPAddress, MessageIn, MessageOut, Parameter) that are bound through the WSDL service models and orchestration/business process specific information such as identification information for the orchestration execution and the orchestrator (e.g., orchestratorExecutionID, Orchestrator, PortOrchestrator, IPAddressOrchestrator).

Examples of the WSDL and BPEL specification structures and syntax based on the travel planner scenario are provided in Listings 17.1 and 17.2 respectively, while the WSDL and BPEL diagrams for the travel planner use case are shown.
The BPEL specification forms a service composition (orchestration) specification that specifies the bindings to specific implementations of atomic services, invoking them sequentially or parallely through a set of request/response messages and then specifies the final response to the original caller. BPMN is more about the abstract sequence of services. For the purposes of showcasing and implementing our Template Binding Specification Model, we decide to use BPEL for the execution and definition of the orchestrations (service compositions).

The substituted template parameters in the template infrastructure model are then used in order to update the Prov Recording Activities with application specific information. This will give way to analysis with regards to the provenance awareness support of a particular system design under development.

In particular, mappings in the Provenance Infrastructure Model mainly take place with regards to template parameters of the template signature corresponding to:

- provConcepts of the **Prov ServiceExecution** Activity, where we need to map/bind those to service description information for each atomic service invoked as part of the service composition specification; we substitute those concepts with real values coming out from the instantiated WSDL descriptions

- provConcepts of the **Prov ServiceDiscovery**, **Prov ServicePublishing**, **Prov Registry** and **Prov ServiceSelection** Activities – where we need to map/bind those to service description information for each atomic service discovered, published and selected as part of the service discovery, service interface publishing and service selection processes of the SOC execution cycle respectively; we substitute those concepts with real values coming out from the instantiated WSDL descriptions.
- provConcepts of the **Prov Orchestration** Activity, where we need to map/bind those to information about the orchestration execution and service composition invocation; we substitute those concepts with real values coming out from the instantiated BPEL description. Part of the orchestrator specification is also a WSDL description that describes the basic identity and location information for the composite service.

A complete list of the template bindings of the Template Binding Specification is shown in Listing 9.5. This is a high level description of the mappings, expressed as a pseudocode specification, that are required to be implemented in order to bind template parameters (provenance concepts located at the left-hand side of the arrow) to actual values corresponding to the schema attributes of the WSDL and BPEL specifications (shown at the right hand side of the arrow). The arrow symbolises the binding/mapping movement from a template parameter to the corresponding service and service composition related elements of the service interface and composition specifications.

For example for the **Service** template parameter being part of the template bindings to WSDL specifications of atomic services we would need to bind to it an actual value found in the WSDL specification under the modelling element `wsdl:service name`. Similarly for the rest of the template parameters (e.g., **ServiceName**, **MessageIn**, **MessageOut**, **Parameter**, **PartValue**, **IPaddressValue** etc.) corresponding to the rest of the bindings of a WSDL specification we would bind the actual values found in the WSDL specification at the respective modelling elements located at the left-hand side of the arrow e.g., `wsdl:port binding`, `wsdl:input message`, `wsdl:output message` etc. The bindings are implemented iteratively for each and every WSDL specification corresponding to the rest of the atomic services being part of the service composition. At a second step, we implement the bindings for the template parameters that correspond to the BPEL business process specification. For example the OrchestrationExecution and WorkPlanValue template parameters would be bound with actual values found under the `bpel:process name` modelling element of the BPEL specification.
9.6 Provenance Generation Analysis Algorithm

In the previous section, we proposed a behavioural model that enables designers to specify their detailed design of provenance infrastructure for composite service systems. In this section, we show how such a specification can be analysed. In particular, we introduce an algorithm that extracts a description of the structure of all provenance graphs that could be generated by an activity diagram with provenance-recording activities. This will eventually form the basis for checking whether a given set of provenance questions could possibly be answered by a
given design. Next, we discuss the structure of the proposed algorithm.

9.6.1 Provenance Awareness Generation Algorithm Steps

Modelling the provenance infrastructure behaviour of a SOC system independently of specific applications enables us to simulate the result of its provenance recording activity and analyse whether the designed infrastructure if bound to a composite service system will lead to actual recording/storage of the corresponding provenance data required to satisfy a set of provenance awareness requirements. This is accomplished by a generation analysis algorithm that simulates the system’s execution and generates provenance data graphs at the appropriate places.

The generation/analysis algorithm we propose simulates the system’s provenance infrastructure behaviour and generates the corresponding provenance data graphs. In particular, it analyses the Prov Recording Activities and provenance data flow links between those activities specified in a given provenance infrastructure behavioural model and applies a set of analysis rules that generate the respective provenance data graphs. Those provenance graphs then form the essence for analysing whether the infrastructure provided is sufficient to answer a set of provenance questions.

In more detail the algorithm takes as input a UML - based specification of provenance infrastructure behaviours and follows the structure of the control and data flow of the UML activity diagram using a depth-first traversal. Each time the next node is a Prov Recording Activity the algorithm applies our proposed rules for generating the corresponding provenance data graphs as shown in Listing 9.6. Before introducing the generation provenance graph rules we first produce a global storage for the provenance graph by introducing a global variable to hold the resulting provenance graph (line 2). Then on every node showing to the current Activity we are looking at (line 5), where the activity has a stereotype Prov Recording Activity, our algorithm takes the following steps:
**step1**: Get the tagged value `provConcepts` of the *Prov Recording Activity* (line 7).

**step2**: Generate instances for each of the provenance data concepts (that form part of the `provConcepts` tagged value) to be recorded in this activity (line 11).

**step3**: Generate a globally unique ID (ProvID) for all `provConcept` instances and add the labelled individuals to the subgraph (line 14).

**step4**: Generate connections between elements by checking the concept type of each individual and creating the corresponding links between `provConcept` instances. The rules for generating those links (expressed as case statements in lines 18-45), follow the formation of possible triple statements as those were modelled in *ServiceProv*. The triples contain nodes (`provConcept` individuals with a unique ID) which are of some type of the ServiceProv’s concepts linked with property relationships defined in PROV-O and ServiceProv ontology.

**step5**: Add provenance triples to the subgraph.

**step6**: Store subgraph to the provenance graph storage (line 48).

**step7**: Make connections to previously generated elements by checking the type for each of the individuals of the *incoming provenance data flow* (special dataflow that conveys connections between different prov activities) and add the provenance data flow links to the main graph storage (lines 51-69).

### 9.6.2 Provenance Awareness Infrastructure Analysis Rules

Our rules will only generate the possible concepts/links for concepts defined in our specification. In case that no provConcepts are tagged in the *Prov Recording Activities* or no ProvIDs are defined in the incoming data flow of these activities, our graph result may form an incomplete graph with missing connections between sub-graphs. Then, we can do further analysis on the provenance graph...
results by checking whether the provenance data and links between this data in
the graph generated is sufficient to answer different kinds of provenance ques-
tions. It is important also to note that in cases of parallel (e.g., service pub-
lishing) or iterative patterns (e.g., service execution) in our specification model,
the associated Prov Recording Activity will be called a number of times based
on the number of parallel/iterative action occurrences in the expansion region.
Therefore, multiple provenance subgraphs would be generated in parallel/itera-
tively where we can identify their concepts by their generated unique ProvIDs.
A complete version of the algorithm with the full list of rules can be found on-
line at https://sourceforge.net/projects/provenanceinfrastructure/files/ProvIn-
fraGenAlgorithm.txt/.

```java
// a global store for the provenance graph produced so far.
PRE { var provGraph = new ProvGraph }

// variable name for the currently looking activity
TRANSFORM a : Activity
    WHERE a.hasStereotype("ProvRecording")
    var concepts : String = a.getTaggedValue("provConcepts");
    var subgraph = new ProvGraph;

// generate instances for all concepts mentioned
foreach (String sConcept in concepts.split(",")) {
    subgraph.add (generateFreshID() + "a" + sConcept); }

// generate connections between elements just created
foreach (Individual i in subgraph.individuals) {
    switch (i.type) {
        case ServiceRequest:
            foreach (Individual iSel in subgraph.selectAllOfType("Planning"))
                subgraph.add (i.ID + " prov:wasGeneratedBy " + iSel.ID);
        case ServiceCollection:
            foreach (Individual iSel in subgraph.selectAllOfType("ServiceDiscovery"))
                subgraph.add (i.ID + " prov:wasGeneratedBy " + iSel.ID);

      case ServiceCollection:
            foreach (Individual iSel in subgraph.selectAllOfType
```

277
Listing 9.6: Provenance Awareness Generation Analysis Algorithm

```java
("ServiceDescription")
{subgraph.add (i.ID + "prov:hadMember" + iSel.ID)}
case ServiceDescription:
  foreach (Individual iSel in subgraph.selectAllOfType
("ServiceDescriptionDiscoveredItem"))
  {subgraph.add (i.ID + "prov:hadMember" + iSel.ID)};

```

```java
case WorkPlan:
  foreach (Individual iSel in subgraph.selectAllOfType
("Planning"))
  {subgraph.add (i.ID + "prov:wasGeneratedBy" + iSel.ID)};
```

```java
case OrchestratorExecution:
  foreach (Individual iSel in subgraph.selectAllOfType
("Server"))
  {subgraph.add (i.ID + "prov:wasAssociatedWith" + iSel.ID)};
```

```java
// ... other rules for other combinations of elements }
```

```java
// copy over to global graph
provGraph.addAll (subgraph);
```

```java
// connect to previously generated elements
foreach (DataFlow df in a.incomingFlows) {
  switch (df.type) {
    case ServiceRequest:
      foreach (Individual iSel in subgraph.selectAllOfType
("ServiceDiscovery"))
      {provGraph.add (df.ID + "prov:used" + iSel.id)};
    case ServiceCollection:
      foreach (Individual iSel in subgraph.selectAllOfType
("ServiceSelection"))
      {provGraph.add (iSel.id + "prov:used" + df.ID)};
    case SelectedServiceDescription:
      foreach (Individual iSel in subgraph.selectAllOfType
("Planning"))
      {provGraph.add (iSel.id + "prov:used" + df.ID)};
    case Workplan:
      foreach (Individual iSel in subgraph.selectAllOfType
("OrchestrationExecution"))
      {provGraph.add (iSel.id + "prov:used" + df.ID)};
    // ... other rules for other kinds of connection }
  }
```

```java
```
```
9.6.3 Exhibiting Proposed Approach by Example

In order to show the practical use of our model and how the generation analysis algorithm/rules would apply in practice, in this section we discuss the results of running our algorithm over a specific example specification for the travel planner scenario (Figure 9.8). The specification\(^2\) describes an explicit set of **Prov Recording Activities** for capturing provenance data at the following SOC execution cycle phases: **Service Brokering** (provRecording step2), **Service Discovery** (provRecording step5), **QoS Repositories and ServiceSelection** (provRecording steps 6 and 7), **Planning** (provRecording step8) and **Orchestration** (provRecording step9).

The algorithm starts by the initial node and follows the control/data flow of the UML specification. When a **Prov Recording Activity** is found, it triggers the corresponding list of rules as defined in listing 9.6. It then applies the case statements that match the **tagged value provConcepts** and the **incoming provenance data-flow** specified. In our example specification a travel planner service request is initiated and received by a broker where the algorithm identifies the first **Prov Recording Activity** (provRecording step2). At this step it iterates over the list of concepts of the **tagged value provConcepts** (**ServiceRequest**), and generates instances and unique identifiers for them. It then checks their type and applies the corresponding case statements (line 19) to create connections with other **prov instances**. The new triples are added to a subgraph which is then copied to a global store for the provenance graph to be produced. The algorithm keeps traversing the control/data flow of the given specification until it identifies the next **Prov Recording Activity** (provRecording step5). Similarly, it applies the corresponding rules for generating a subgraph based on the tagged properties of the current activity but it also checks additional rules about the incoming provenance data flow (e.g., **ServiceRequest.ProvID**) to connect the current subgraph with previously generated elements. The rules apply similarly for the rest of the **Prov Recording**

---

\(^2\)Figure 9.7 presents an example model for a specific infrastructure configuration of a composite service system, formalized as a UML activity diagram, that includes the maximal provenance recording configuration for a SOC system. However, specific applications like the travel planner of Figure 9.8 will make specific choices of what provenance to record at any of these steps.
Activities in the specification. The result of those actions would be an instantiated provenance data schema as shown in Figure 9.9, following the ServiceProv triple notation.

Having generated the corresponding provenance data graphs we can now use these in order to do an analysis with regards to the kinds of provenance questions we would like our system to be able to answer. For example, the sample provenance generated allows us to answer questions about the discovery/selection/planning phases, such as what were the discovered/selected flight booking services of the travel planner, what were the QoS requirements offered by those services, what was the concrete work plan produced etc. Yet, the produced graph
does not contain provenance data that will answer questions about the identity of servers involved in the service discovery/selection phases. In this way, the algorithm results enable provenance analysts to explore which questions can or cannot be answered by the current provenance infrastructure design. This further allows infrastructure designers to understand if they have provided sufficient provenance recording infrastructure mechanisms for their provenance needs.

**Summary:** In this chapter we proposed a metamodel language notation that enables the specification and design of provenance infrastructure behaviour for SOC systems and a provenance awareness generation analysis algorithm simulating the results of the infrastructure design apriori to a system’s implementation, at design-time. Evaluation of the proposed Template Infrastructure Model with regards the criteria of expressivity of concepts and behaviour of provenance infrastructure technical aspects for SOA and SOC, reusability and realism is presented in detail in Chapter 11. Next, we present our tool implementation for the provenance awareness modelling and analysis framework of atomic services and service compositions. This includes a full implementation series providing modelling utilities for the template provenance infrastructure, the provenance data structures and the provenance query patterns. It also contains an implementation of the generation analysis algorithm and the provision of the execution environment to allow simulations with regards to the provenance awareness design of specific services and SOC systems.
Part II

Provenance Awareness Modelling and Analysis Prototype

Environment and Evaluation
Chapter 10

Framework Implementation

In this chapter we discuss the implementation part of our overall framework getting into the details of the basic constructs and artefacts comprising our tool prototype.

Contents

10.1 Context and Motivation ............................... 283
10.2 PROVa Tool Implementation ......................... 285
10.2.1 PROVa Tool Design ................................. 287
10.2.2 PROVa Toolset Deployment ......................... 297

10.1 Context and Motivation

In the previous chapters, our thesis has addressed the design and development of the required models, specifications and analysis methods to cope with provenance for SOC systems. In particular, in Chapter 5 we presented our approach and proposed architecture towards a formal specification and analysis framework of provenance awareness for SOC systems and compositions of services.

As part of this framework we first identified a number of provenance question categories, so that provenance awareness of a system can be specified as the ability of system to answer questions spread across the different provenance
facets, while we also proposed a set of provenance question type patterns – expressed as SPARQL queries – to formally express the different kinds of provenance awareness requirements exhibiting the faceted questions. Secondly, we extended PROV-O [14], building (ServiceProv) for describing provenance data concepts of composite service-based designs; that forms both a conceptual model for SOC provenance as well as a provenance schema that allows describing provenance data structures that can be captured through the SOC execution cycle. Thirdly, we proposed a template provenance infrastructure model and metamodel language notation that allows us to specify the provenance infrastructure activity required in order to collect certain provenance information for SOC systems. This goes hand in hand with a template binding specification that enables binding specific services and business processes (orchestration) under development with the provenance infrastructure design. Fourthly, we addressed the issue of provenance awareness analysis support for composite service designs. Along those lines, we presented a generation analysis algorithm that simulates the systems provenance infrastructure behaviour by generating the corresponding provenance data graphs. The generated graph result can then be queried in order to identify whether the provenance that would be captured by the particular design is sufficient to satisfy the provenance awareness requirements of the users.

Finally, we realise our framework, by developing a prototype modelling and analysis tool (PROVa). PROVa enables:

1. modelling the provenance-related properties of composite service systems and the infrastructure needed for supporting these,

2. simulating the provenance infrastructure recording activity, and

3. analysing whether the provenance infrastructure design would be sufficient to capture the required data for satisfying the provenance awareness requirements for particular services that are dynamically bound and composed.

Next, we present the design details for the prototype implementation of our approach.
10.2 PROVa Tool Implementation

Tool support is crucial for the specification and analysis of provenance awareness for SOC systems and for evaluating the proposal of our framework. The key aspect with regards to the implementation of our proposed framework was focusing on designing a prototype that will allow integration of the different types of models and specifications, produced and consumed during the model-driven development and design provenance awareness life cycle, in a modular way. Those artefacts are produced and consumed by different actors associated with a particular role (e.g., application designer, provenance infrastructure designer, QoS and provenance analyst, domain expert as shown in Figure 5.2). Modularity enables joining up the artefacts required for analysis at design-time in a constructive way towards building up a complete environment in terms of modelling and analysis utilities. In addition, it gives the associated actors the flexibility and independence to focus on their special role tasks and responsibilities as part of the provenance awareness design life cycle.

The need to leverage modularity as a feature of our framework realisation led to the development of an integrated toolset based on the Eclipse platform that:

1. allows to specify and express provenance awareness requirements of users,
2. incorporates a set of modelling and design utilities for the provenance awareness infrastructure of SOC systems,
3. simulates the generated result of the provenance awareness design, and
4. supports analysis checks at design-time of whether the provenance data generated by the provenance infrastructure design can satisfy provenance awareness requirements of users.

Next, we are going to explore in detail the set of components, activities and artefacts that form part of our tool implementation. In particular we are going to answer questions about how those artifacts are produced and consumed as part of
Figure 10.1: PROVa - Components, activities, artifacts and tools
the execution of the different toolset utilities/processing activities involved, how those are exchanged between the different components, how the major components get deployed as part of the integrated toolset and which roles are involved in this process.

10.2.1 PROVa Tool Design

PROVa Core Components  The implementation toolset PROVa is composed of three major components as shown in Figure 10.1:

- A Template Binding Parser which takes as input and processes the Template Provenance Infrastructure Model – in order to identify the parametrized elements of its template signature – and the WSDL/BPEL specifications, and then makes the necessary bindings between actual values of the service models and the template parameters based on the Template Binding Specification. In this way, the provenance concepts of the stereotyped provenance recording activities of the provenance infrastructure model are instantiated by and bound to service specific information. In particular, the BPEL specification is used as an input to populate the provenance concepts of the Orchestration provenance recording activity while the WSDL specification is used to feed and instantiate the parameterized elements associated with the Service Execution, Service Discovery, Service Selection and Service Interface Publishing stereotyped activities.

Template Provenance Infrastructure Model, used as an input artefact to the pre-processing activity of the Provenance Infrastructure Model, can be considered as a separate implementation (modelling) component. Within this we have built a metamodel language notation by creating the Provenance Infrastructure Profile as shown in Figure 9.6, by using the Papyrus UML modelling environment [53] an eclipse-based plugin [100] for UML2 [67]. The profile basically extends the base metaclass action\(^1\) of

\(^1\)Note: Action metaclass has been extended here instead of Activity metaclass due to limitations in the Papyrus UML tool to provide profile extension for the full set of UML modelling elements including activities.
UML to provide the corresponding syntax for provenance infrastructure activities. We, therefore, can use the extended notation composed of the `provRecording stereotype` and the `provConcepts label`, being part of the UML Provenance Infrastructure Profile, in order to represent provenance data structures associated with the behaviour of a provenance recording activity. Papyrus provides a set of different modelling views including a textual, a diagrammatic representation and a model explorer view (see Figures 10.2 and 10.3).

- **A Provenance Awareness Generator** that takes as input the Template Provenance Infrastructure Model instantiated with service-specific information corresponding to the WSDL and BPEL specifications and generates the corresponding provenance data graph snippets. This provenance infrastructure model owns a set of provenance recording stereotyped activities incorporating a set of tagged properties – meta-attributes – that represent the provenance data structures. In particular we introduce the tagged property `provConcepts` that carries a String `tagged value`, where the tagged value equals to the list of provenance data concepts we would like to be captured under a particular provenance recording activity – essentially as a set of ServiceProv assertions expressed in an RDF/OWL triple notation as shown in Figure 10.3. Then the sets of abstract ServiceProv data structures are analysed through our proposed generation analysis algorithm (as described in Chapter 9) that applies analysis checks and simulates the corresponding provenance data graph instance result conforming to the ServiceProv ontology schema. For the implementation of our analysis algorithm, we have built a set of behavioural analysis rules that we express in java code as switch cases.

The Provenance Awareness Generator is therefore basically focusing on generating and simulating the actual provenance data graph snippets of a service (composition) design model by constructing provenance sub-graphs and their connections per a rule analysis basis. The result of this gener-
Figure 10.2: Papyrus Model Explorer
ation can then be used as input on the Provenance Data Graph Analyzer component, which uses a set of analytical queries to reason over the generated provenance snippets and derives conclusions about the provenance awareness support of the particular provenance design.

- A Provenance Data Graph Analyzer, which takes as input the ServiceProv ontology, the instantiated provenance data graph generated based on the designed provenance infrastructure containing service-specific information, and a set of provenance question type patterns expressed as SPARQL queries. The Provenance Data Graph Analyzer first makes the necessary syntax consistency checks of the SPARQL queries with regards to the ServiceProv provenance data schema and the generated provenance data graph result of the system design under analysis. It then executes\(^2\) the set of SPARQL queries over the generated provenance data graph to analyse whether those queries can be answered/not answered by the particular provenance recording infrastructure design. In this way, our tool implementation provides the necessary mechanisms to analyse and verify provenance awareness support of a SOC system at system’s design time.

\(^2\)SPARQL queries are executed by making a Jena API call
PROVa Activities  Each of the major components of PROVa listed above is associated with a set of activities as shown in Figure 10.1. Those include:

- The **Template Binding Parsing** activity – takes as input the BPEL/WSDL specifications defined by the application designer, the Template Binding Specification and the Template Signature (set of template parametrized elements) forming part of the Template Infrastructure Model as those were modelled by the infrastructure designer and gives, as a result, the binding replacements to be applied in the Provenance Infrastructure model.

- The **ProvInfra Model (Pre)Processing** activity – reads and processes the model elements of the initial version of the Template Provenance Infrastructure Model and applies the binding replacements to produce an instantiated version of the model with service-specific information.

- The **Provenance Awareness Generation** activity – takes as input the instantiated provenance infrastructure model and the ServiceProv model and by calling the Jena libraries (Jena API) it translated the model into a set of RDF triples representing the provenance data graph result.

- The **Provenance Data Graph Analysis** activity – analyses the provenance data graph results of the Provenance Awareness Generation activity in an effort to identify whether the set of provenance query patterns, expressing provenance awareness user requirements, can be supported by the provenance infrastructure design. This is done by running the set of SPARQL queries over the provenance data graph simulated result and identifying which of the queries can/can not be answered.

- The **SPARQL Execution** activity – is associated with the execution of the provenance query patterns expressed in SPARQL.

- The **SPARQL Formalisation** activity – is related to the formalisation of provenance questions defined by the domain experts, representing the provenance awareness requirements of users, into a set of formalised queries
(provenance query patterns). This is done with the support of the ServiceProv ontology that provides the essential conceptualization to express provenance awareness requirements for SOC systems.

**PROVa Artefacts** The main artefacts involved as part of the execution of the main activities and components involved in our prototypical implementation of the modelling and analysis framework for provenance awareness of SOC systems include:

- **The ProvInfrastructure Model** (or more formally the Template Provenance Infrastructure Model) – is the model representation of the provenance infrastructure design modelled by the provenance infrastructure designer, incorporating the core provenance recording activities and the infrastructure behaviour. An example of the Provenance Infrastructure Model as implemented in the Papyrus Modelling Environment is shown in Figure 10.4.

- **WSDL specification(s)** – is the description of a service about the operations that can be performed by a particular service, the SOAP bindings including input/output messages and parts of messages, the service name (targeted Namespace - URI) and type, information about the location of the server under which the service is executed and maintained including the PortTypeName and IPaddress. It is used to specify basically atomic services (as shown in Figure 10.8) but composite services as well (as shown in Figure 10.9). In the case of composite services, the WSDL specification mainly contains the information of the composite service interface and is used in combination with a BPEL or BPMN specification to invoke and execute a set of standard Web service interfaces (as shown Figure 10.10). WSDL specifications are basically created by service or application designers.

- **BPEL or BPMN specification** – a BPEL specification forms a service composition (orchestration) specification that specifies the bindings to specific implementations of atomic services, invoking them sequentially or par-
Figure 10.4: Provenance Infrastructure Model - Diagrammatic View
Figure 10.5: Provenance Infrastructure Model - Diagrammatic View Break Down Part 1

Figure 10.6: Provenance Infrastructure Model - Diagrammatic View Break Down Part 2
**Figure 10.7:** Provenance Infrastructure Model - Diagrammatic View Break Down Part 3

**Figure 10.8:** WSDL specification for atomic services

**Figure 10.9:** WSDL Specification for Orchestration
Figure 10.10: BPEL Specification for Orchestration

allelly through a set of request/response messages and then specifies the final response to the original caller. BPMN is more about the abstract sequence of services. A BPMN specification is the model that describes the business logic but can also be used by software architects and service designers to provide a description of the artefacts executed by a business process engine. WSDL description of the service that exposes that orchestration

For the purposes of our prototype implementation we use BPEL for the execution and definition of any orchestrations (service compositions) mod-
elled by service designers.

- **Template Binding Specification** – specifies the bindings between the provenance infrastructure design concepts, reflected in the Provenance Infrastructure Template Model, and information of specific atomic services and their compositions as part of their WSDL and BPEL or BPMN specifications respectively.

- **Processed Provenance Infrastructure Model** – is the processed version of the Template Provenance Infrastructure Model after having been instantiated and bound to service and orchestration specific information.

- **ServiceProv Ontology Model** – is a PROV-O extension for the conceptualization of provenance for service compositions and SOC systems.

- **Generated Provenance Data Graph** – is the provenance data graph generated by the Provenance Awareness Generator based on the provenance infrastructure design reflected in the Processed Provenance Infrastructure Model.

- **Provenance Questions** – are the questions formed by users and gathered as requirements about the provenance awareness of composite service system design.

- **Provenance Query Patterns** – are the formalisation of the provenance questions expressed in SPARQL.

- **Query Analysis Results** – are the query results after the execution of the provenance query patterns over the Generated Provenance Data Graph.

### 10.2.2 PROVa Toolset Deployment

In this section we provide details about the deployment of the core components of PROVa and the execution environment used in practice for the realisation of the associated artefacts being produced, requested and consumed by the core
components and subsumed activities in our toolset prototype implementation. A
diagrammatic view of PROV\textsubscript{a} deployment is illustrated in Figure 10.11.

WSDL specification artifacts are designed and executed using the eclipse based
WSDL Editor [152] while BPMN and BPEL artifacts are created using the BPMN
Designer - Modeler Diagram Editor [44] for Eclipse [49]. The execution envi-
ronment used for both components is Eclipse Mars – Version Mars.2 Release
(4.5.2) [48].

The Provenance Infrastructure Model artefact is designed by using the eclipse-
based plugin Papyrus Modelling Environment [100]. Papyrus was executed under

All the models designs along with the SPARQL query patterns are then de-
ployed along with the core components of PROV\textsubscript{a} (namely the Template Binding
Parser, the Provenance Awareness Generator and the Provenance Data Graph
Analyzer) as shown in Figure 10.11, in Eclipse Java EE IDE for Web Developers
Version: Juno Service Release 2 [46].

The reason for using different versions of Eclipse for supporting our prototype
implementation, is that the three basic components introduced as part of our
toolset require Eclipse plugins for supporting modelling utilities such as those
of Papyrus UML, BPEL/BPMN and WSDL Editors that can be integrated and
executed at specific versions of the Eclipse platform.

The prototype PROV\textsubscript{a} tool implementation for the proposed provenance aware-
ness modelling environment and analysis framework can be found online under
PROV\textsubscript{a} Tool project https://sourceforge.net/projects/prova-tool/.\textsuperscript{3}

\textbf{Summary:} In this chapter we introduced our prototype modelling and analysis
environment, namely PROV\textsubscript{a}s. This forms an integrated set of tools built in or-

\textsuperscript{3}The code can be found at https://sourceforge.net/p/prova-tool/code/HEAD/tree/
and checked out using https://svn.code.sf.net/p/prova-tool/code/
Figure 10.11: PROVa - Deployment Diagram
der to evaluate the modelling and analysis concepts and methodologies proposed in this thesis within the scope of providing a formal specification and analysis framework for provenance awareness of service compositions. Evaluation of the proposed PROVa specification and analysis environment with regards to the criteria of genericity/reusability, computability and realism is presented in detail in Chapter 11. Next, we discuss the evaluation methodology we have followed with regards to our thesis hypotheses and how PROVa has been used for validating the proposed framework in combination to two use case studies.
Chapter 11

Evaluation

In the previous chapter we presented our prototype tool implementation, PROVa, realising our specification and analysis framework proposal for provenance awareness of SOC systems and compositions of services. Our prototype has been a fundamental piece for the evaluation of our framework’s proposed architecture and specification and analysis methodology through an implementation to achieve two use cases (a standard example use case scenario and an industrial case study).

In this chapter we first introduce our hypothetical statements, aiming to answer the research questions discussed extensively in Chapter 4 and the evaluation methodology for testing those hypotheses. We then present our supporting evidence through the implementation of the introduced use case studies and a set of criteria that validate fundamental qualities of our proposed modelling and analysis approach and methodology.

Contents

11.1 Hypotheses ................................................................. 302
11.2 Use Case Studies ......................................................... 313
11.2.1 Travel Planner Case Study ........................................ 314
11.2.2 Industrial Case Study ............................................... 314
11.3 Evaluation Results ....................................................... 316
11.3.1 SOC and SOA Criteria Checklist Evaluation ................. 317
11.1 Hypotheses

In the introductory chapter we presented our overall hypothesis for this thesis where we state that: “It is possible to specify and analyse provenance awareness systematically as a non-functional property (NFP) for SOC systems and compositions of services at design-time”. We claimed that this is possible based on our thesis’ contributions as those were presented in Section 1.6 and realised as part of our prototype tool implementation for the proposed specification and analysis provenance awareness framework.

In this section we present the evaluation results of our thesis along with the evaluation methodology followed. We evaluate the proposed framework of this thesis constructively by building a set of individual secondary hypotheses corresponding to the main research questions (and sub-questions) we have addressed in this thesis. We therefore validate the key contributions of this thesis, comprising our subjects of evaluation as presented below. Each of the subjects of evaluation are associated with one or a combination of testable hypotheses for which we use different types of validation that are appropriate for the type of research results we are evaluating. We therefore obtain supporting evidence of different forms that we then combine to defend our overall hypothesis.

1.1 Subject of Evaluation: Provenance Questions Categories using a Faceted Classification Approach and a set of identified Provenance Facets

Hypothesis 1.1: With regards to RQ 1 and RQ1.1 (as presented in Section 4.2) we make the following hypothesis: Using provenance facets, we can
express provenance awareness as a non-functional requirement for SOC systems and service compositions in order to be able to analyse a system’s accountability support – with regards to when and where, which data was processed to generate new data, which new data derived through the SOC execution cycle. Hypothesis 1.1 is evaluated in a constructive way by addressing three different criteria with regards to the ability to express provenance awareness requirements, namely: completeness, adequacy, and realism which further refine Hypothesis 1.1 into the following sub-statements:

- **Hypothesis 1.1.1** The provenance facets are complete with regards to covering key aspects of service-oriented computing (SOC) and service-oriented architecture (SOA) concepts and principles. We define a set of facets as complete if those can provide complete conceptual coverage for expressing provenance awareness requirements of fundamental aspects coming from the wider literature of service oriented computing (SOC) [119] (including the processes of service discovery, service selection, interface publishing, composition/aggregation), and the extended service-oriented architecture (SOA) [120], and the service-oriented computing (SOC) pyramid as shown in Figure 2.1.

**Evaluation Methodology:** For the evaluation of whether the proposed facets exhibit completeness with regards to covering key aspects of the SOC systems (paradigm) and the SOA architecture, we evaluate those by using our standard example scenario (travel planner composer) taken from the service compositions’ literature. However, completeness of our facets identifies to the completeness of the SOC/SOA concepts, and then the use case Travel Planner scenario is used to show evidence that nothing outside those identified facets has occurred. We accomplish this by:

1. listing the core SOC and SOA concepts and principles to build a criteria checklist against which we can validate our faceted classification scheme for completeness
2. using our standard use case scenario and the provenance awareness
requirements we have systematically expressed for this – exhibiting the proposed facets – to compare and validate concepts’ completeness against our criteria checklist. In this way we are able to showcase complete coverage of the concepts expressed as part of the provenance awareness requirements for our case study (to the extend that this allows us) and verify that those cover the criteria checklist’s core concepts.

- **Hypothesis 1.1.2** We can express an adequate set of questions, organised across a set of provenance facets using a faceted classification approach, that enables service and/or data quality designers to express provenance awareness requirements for service-oriented computing (SOC) systems and service compositions. A classifier (facet) corresponds to particular types of information being recorded about provenance and so particular sets of provenance questions being answerable. Adequacy refers to whether we have chosen a sufficient and correct mechanism that allows to easily express provenance awareness requirements represented as provenance questions spanning across different semantic categories and gives us the flexibility to systematically derive more complex questions that correspond to the expression of composite provenance awareness requirements; those complex questions are expressed as multi-faceted questions.

**Evaluation Methodology:** For the evaluation of the provenance facets’ adequacy our methodology was to use a standard example (use case travel planner scenario as presented in Section 6.1.1) taken from the service composition literature to showcase an adequate \(^1\) list of provenance questions posed by users as provenance awareness requirements for this scenario and identify whether the provenance data that is required to be captured for answering those questions are possible to be expressed and organized through/across the proposed provenance facets. We claim that a single standard example, the travel planner, is sufficient to support our adequacy hypothesis.

---

\(^1\)by adequate we refer to conceptual coverage of the principle concepts and processes introduced by SOC systems and SOA architecture)
for three reasons: 1) the travel planner is one of the standard examples that has been used in the service composition and service-oriented computing literature, therefore covers all aspects of SOA principles and the SOC execution cycle throughout which we are interested to capture provenance information for. Standard examples are proven work on technology solutions that improve design and establish quality, and that makes the comparison between different solutions easier. As our objective is to improve the design of service compositions with regards to provenance awareness for SOAs and SOC solutions, using a common standard example for SOAs will provide the best possible platform upon which we can build a framework that extends qualities of an already improved solution. 2) To our knowledge this is the first attempt of capturing provenance information throughout the complete SOC execution cycle and we therefore claim that a standard example widely accepted and studied by the SOA/SOC research community enables us to explore and capture the full range of provenance awareness requirements we are trying to model and validate, 3) existing industrial case studies would not present the full set of requirements we are trying to capture and are not necessarily implemented according to the full standards presented by the SOC/SOA architecture and principles. In practice industrial implementations of different systems may be customized according to the business needs, therefore do not always exhibit in full the implementation of the SOC pyramid - Extended Service-Oriented Architecture [120]. Therefore, case studies in an industrial context do not always form the best platform for full exploration of the provenance awareness requirements needed to be identified at this stage of our research. Yet, an industrial case study has been used for validation purposes of our framework as part of this thesis (ORBI industrial case study), therefore we have showcased that our approach is applicable in the industrial context.

We can express an adequate set of questions, organised across a set of provenance facets using a faceted classification approach, that enables service and/or data quality designers to express provenance awareness requirements
for service-oriented computing (SOC) systems and service compositions. A classifier (facet) corresponds to particular types of information being recorded about provenance and so particular sets of provenance questions being answerable. Adequacy refers to whether we have chosen a sufficient and correct mechanism that allows to easily express provenance awareness requirements represented as provenance questions spanning across different semantic categories and gives us the flexibility to systematically derive more complex questions that correspond to the expression of composite provenance awareness requirements; those complex questions are expressed as multi-faceted questions.

In particular, we achieve this by:

1. systematically expressing example questions (provenance awareness requirements) exhibiting one/or a combination of our provenance facets and,

2. validating that critical aspects and processes of the SOC systems execution cycle and SOA architecture that a provenance-aware system design should become accountable for, can be easily expressed through the proposed facets or their composition (aggregation).

- **Hypothesis 1.1.3:** The provenance questions exhibiting the provenance facets are realistic enough being able to represent provenance awareness requirements of real-world systems.

**Evaluation Methodology:** Realism of the provenance question categories and provenance awareness requirements is evaluated by methodically applying the same approach to systematically express provenance awareness requirements for an industrial case scenario. The subject of study is a real-world composite-service, namely ORBI ² a web-based service system for providing different kinds of financial, time and reporting management services for self-employed professionals.

²ORBI has been one of the services provided as a case study through our industrial research partner Singular Logic participating in the FP7 ITN RELATE
We achieve this evaluation by:

1. systematically expressing provenance awareness requirements of ORBI’s system users and other stakeholders in terms of provenance questions

2. showcasing that a set of realistic requirements for provenance awareness is covered by our elemental faceted classification which enables us to validate that all semantic provenance questions categories, organized using a set of provenance facets, can be instantiated in the context of the case study of a real-world system.

1.2 Subject of Evaluation: Provenance Question Type Patterns

**Hypothesis 1.2:** With regards to RQ 1.2 and RQ 1.3 (as presented in Section 4.2) we make the following hypotheses: We can formally express provenance questions (provenance awareness requirements), classified using a faceted classification scheme, as a set of provenance question type patterns; the latter are realised as a set of provenance query patterns in a standard query language. Those query patterns could then be used as analytical metrics for analysing provenance awareness aspects for compositions of services. Hypothesis 1.2 is constructively evaluated by addressing three different criteria with regards to the provenance question (query) type patterns, namely: well-formedness, expressiveness, and realism which further refine Hypothesis 1.2 into the following sub-statements:

- **Hypothesis 1.2.1:** The set of provenance question type patterns introduced are well-formed. The provenance question type patterns are realised as query patterns expressed in SPARQL query language. Each provenance query pattern is formed as a set of assertions in the form of RDF/OWL triples as part of a SPARQL SELECT query. We define a query pattern as well-formed if it is constructed as a) sequence of syntactically correct assertions in the form of RDF/OWL triples and is consistent with the semantic rules of the ServiceProv ontological/conceptual model which is our proposed extension of PROV W3C standard; ServiceProv ontology introduces a set of
concepts (classes/properties and their relationships) to capture provenance for service compositions.

**Evaluation Methodology:** We evaluate the well-formedness of our provenance question (query) type patterns by building and implementing a set of algorithms that apply:

1. syntax checks of the provenance question (query) type patterns against RDF/OWL syntax
2. semantic checks with regards to making use of the ServiceProv and PROV concepts and constructing a meaningful sequence of triple assertions conforming to PROV-O and ServiceProv ontological concepts and relationships.

- **Hypothesis 1.2.2:** SPARQL is a sufficiently expressive language that allows quality designers to capture all relevant kinds of provenance questions exhibiting the full list of introduced provenance facets.

**Evaluation Methodology:** Expressiveness of the identified query patterns is validated by rigorous derivation of expressing patterns for the full list of questions identified in our faceted/multi-faceted classification through a standard composition use case scenario (travel planner scenario).

- **Hypothesis 1.2.3:** The provenance question type patterns exhibiting the faceted categories are realistic enough being able to formally express provenance awareness requirements of real-world systems.

**Evaluation Methodology:** Realism of the provenance questions type (query) patterns proposed is evaluated by applying our patterns in an industrial case scenario in order to formally express provenance awareness requirements of a real-world system.

**1.3 Subject of Evaluation:** ServiceProv Ontology (provenance data model)

**Hypothesis 1.3:** With regards to RQ 2 (RQ 2.1.1 and RQ 2.1.2) (as presented in Section 4.2) we make the following hypotheses: We can formally express the
structure of provenance data for service-oriented systems and compositions of services exhibiting the different aspects of SOA and the SOC execution cycle as an ontological model. Hypothesis 1.3 is constructively evaluated by addressing three different criteria with regards to the provenance data model (ServiceProv ontology), namely: **conceptual adequacy/expressivity of concepts, reasoning, and realism** which further refine Hypothesis 1.3 into the following sub-statements:

- **Hypothesis 1.3.1**: ServiceProv model is conceptually adequate to faithfully represent provenance information for service-oriented computing systems and their execution cycle including atomic services and their compositions as defined in the SOC [119] and SOA [120] literature. We define as conceptual adequacy/expressivity of concepts, the model’s ability to faithfully represent provenance information for service-oriented computing systems incorporating the aspects of the extended SOA and the SOC pyramid as presented in Figure 2.1 (including the aspects of service discovery, selection, composition and service publishing/brokering).

**Evaluation Methodology**: We evaluate the conceptual adequacy/expressivity of concepts of ServiceProv by using a standard example (travel planner example) from the SOA and SOC literature. In particular we systematically introduce instances using ServiceProv ontological model to capture the provenance information (creating an ontological provenance data schema) against the complete set of provenance awareness requirements for the travel planner use case scenario.

- **Hypothesis 1.3.2**: ServiceProv is applicable for reasoning over provenance instances in order to answer specific provenance questions.

**Evaluation Methodology**: We evaluate the practical use of our ontology-based model, ServiceProv, in two steps:

1. by illustrating a number of provenance queries (representing provenance awareness requirements) for a standard use case scenario used in the service-oriented and composition literature (travel planner scenario)
2. by extracting provenance information query results used to answer the
provenance questions (requirements of users) through reasoning over
instances of an instantiated ServiceProv model.

- **Hypothesis 1.3.3**: ServiceProv is realistic enough with regards to pro-
viding the right concepts to represent provenance data information for a
real-world composite-service system or service composition.

**Evaluation Methodology**: Realism of the provenance data model is eval-
uated though an industrial case study by checking whether ServiceProv con-
tains a set of adequate concepts required to model realistic provenance data
of actual real-world service-based systems and compositions of services.

**1.4 Subject of Evaluation**: Template Provenance Infrastructure Model

**Hypothesis 1.4**: With regards to RQ 2.2 (as presented in Section 4.2) we make
the following hypotheses: We can model the provenance infrastructure activity
and provenance infrastructure behaviour for service-oriented computing systems
and compositions of services exhibiting the different aspects of SOA and the SOC
execution cycle independently of specific services and applications. Hypothesis
1.4 is constructively evaluated by addressing three different criteria with regards
to the Template Infrastructure Model: expressivity of concepts and behaviour of
provenance infrastructure technical aspects for SOA and SOC, reusability, and
realism which further refine Hypothesis 1.4 into the following sub-statements:

- **Hypothesis 1.4.1**: We can introduce a modelling notation for specifying
provenance recording infrastructure activity and behaviour for com-
positions of services which acquires conceptual expressivity and adequacy
with regards to the technical aspects of the SOC execution execution cy-
cle (e.g., the ability to express parallel, iterative and sequential patterns of
provenance recording behaviour with regards to provenance captured dur-
ing service discovery, service selection, service execution, service publishing
phases).
Evaluation Methodology: Conceptual expressivity and adequacy of the modelling notation introduced through our proposed provenance infrastructure behavioural model has been validated through showcasing the modelling requirements of a standard realistic use case scenario used in the service-oriented and compositions of service literature.

- **Hypothesis 1.4.2**: The template provenance infrastructure behavioural model and template binding specification are reusable for the formalisation of other provenance infrastructure models. We claim that the provenance infrastructure specification is reusable if this is independent of specific services and applications and can be used to bind to specifications of different service-oriented system designs.

Evaluation Methodology: In order to make provenance infrastructure models independent of specific applications we have introduced a template model solution. This acquires a template signature which specifies the set of formal template parameters that could be substituted by actual parameters in a template binding with specific services and SOC systems. Reusability of our template infrastructure model and the template binding specification is showcased through two use cases: by introducing template parameters for a literature standard web-service composite service scenario (travel planner) and an industrial case scenario (ORBI). In our case studies, the bindings of actual services and orchestrations have been described in WSDL and BPEL specification languages respectively.

- **Hypothesis 1.4.3**: The Template Provenance Infrastructure Model can represent the provenance infrastructure activity and behaviour of real-world service-based system designs.

Evaluation Methodology: Realism of the provenance infrastructure models and bindings are validated by systematically applying the modelling framework introduced in order to model the provenance infrastructure behaviour of an industrial case study system (ORBI).
1.5 **Subject of Evaluation:** PROVa specification and analysis environment.

**Hypothesis 1.5:** With regards to RQ 3 (as presented in Section 4.2) we make the following hypotheses: We can build a method and supporting framework that will enable the specification and analysis of service-oriented systems and compositions of services at system’s design time. This framework will further incorporate and enable the execution of analytical metrics for measuring the provenance awareness design of a system against a set of provenance awareness requirements. Hypothesis 1.5 is constructively evaluated by addressing three different criteria with regards to the PROVa specification and analysis environment: **genericity/reusability, computability,** and **realism** which further refine Hypothesis 1.5 into the following sub-statements:

- **Hypothesis 1.5.1:** Our specification and analysis framework solution is generic enough to be used for the specification and simulation/analysis of the provenance awareness properties of different types of service-oriented systems.

**Evaluation Methodology:** Genericy of our framework is evaluated by showcasing its use and applicability to two different use case studies. Reusability can be claimed since we exhibit our approach both in an industrial case study scenario and a standard example composition scenario (travel planner scenario).

- **Hypothesis 1.5.2:** Provenance awareness is computable as an analytical metric through a specification and analysis framework implementation.

**Evaluation Methodology** The computability of provenance awareness as an analytical metric is evaluated by providing a prototype tool implementation as a proof of concept for our modelling and analysis methods introduced as part of our provenance awareness specification and analysis framework. Our prototype incorporates 1) an implementation of our provenance awareness generation analysis algorithm and rules for simulating/predicting the system’s provenance awareness recording behaviour, 2) an implementation
of the provenance query patterns (expressing formally a set of provenance awareness requirements), and 3) an environment that provides automated validation/analysis checks with regards to which of the requirements can be supported by the specified provenance infrastructure design.

- **Hypothesis 1.5.3**: Our method and framework for modelling and analysis of provenance awareness can be used to predict and analyse provenance awareness of real-world system designs.

**Evaluation Methodology** The realism of our approach is evaluated by applying the modelling and analysis methods incorporated in our tool implementation on the system design of a real-world system (ORBI) through an industrial case study.

### 11.2 Use Case Studies

In the previous section we presented in detail our research hypotheses. In order to validate those hypotheses we have used two use cases studies:

1. a standard example taken from the SOC and SOA literature, namely a Travel Planner Composer,

2. an industrial case study using a real-world web-based composite service, namely ORBI.

Our case studies enables us to conclude in positive results that validate our hypotheses that we present and discuss in detail in Section 11.3.2 for the Travel Planner Composer case study, and in Section 11.3.3 for the industrial case study. Also, our experimentation results showcasing the realization of our two use case studies through PROVa toolset is presented in Section 11.3.5 presenting the context of our use cases.

We use both of our case studies to provide supporting evidence that enables us to defend our set of hypotheses as those were described in Section 11.1. Then, in Section 11.3.5 we showcase how the assumptions we have made in our hypotheses,
about the specification and analysis of provenance properties and requirements as exhibited in our case studies, are realised as part of our prototype tool implementation.

11.2.1 Travel Planner Case Study

Our travel planner case study has been presented in detail in Section 6.1.1\(^3\). This is a standard example scenario taken from the Service-Oriented Architecture (SOA) and service compositions literature that has been previously used to exhibit modelling of functional and non-functional properties for service-based systems and service composition workflow models. In our thesis this forms our second form of validation providing supporting evidence with regards to validating the following hypotheses: the provenance facets’ conceptual completeness (Hypothesis 1.1.1) and adequacy (Hypothesis 1.1.2), the expressiveness of the provenance question type (query) patterns (Hypothesis 1.2.2), the conceptual adequacy and applicability of ServiceProv ontological model for reasoning over provenance instances for compositions of services and its conceptual expressivity (Hypothesis 1.3.1 and 1.3.2), the adequacy and reusability of the Template Provenance Infrastructure Model (Hypothesis 1.4.1 and Hypothesis 1.4.2). Details with regards to the evaluated hypotheses and the evaluation results are showcased in Section 11.3.2.

11.2.2 Industrial Case Study

In this section we present our industrial case study that we use in order to explore real-world provenance. In particular, we present a worked example of capturing and querying provenance data from ORBI service-based system [129]; that is a SaaS solution\(^4\). It consists a three-tier architecture, realising the data,

\(^3\)This is a backward reference to the already introduced standard use case scenario of the travel planner; for reasons of avoiding repetition please refer back to Section 6.1.1 for a full description of the details of the travel planner scenario

\(^4\)SaaS is a software distribution model in which a third-party provider hosts applications and makes them available to customers over the Internet. SaaS is one of the three main categories of cloud computing alongside infrastructure as a service (IaaS) and platform as a service (Paas)
business logic and presentation layer specially designed to meet the needs of self-employed professionals that is currently available in the Greek market by Singular Logic; Singular Logic has been one of the industrial partners in RELATE FP7 project supporting this research. In order to explore this real-world case, we rely on a set of provenance questions (provenance awareness requirements) that we formally express in SPARQL.

In this study ServiceProv is used to represent the structure of the provenance data captured relevant to the ORBI execution. Beyond representing the structure of data provenance, it also allows us to reason over the collected information from the perspective of answering provenance questions valuable for the users of the ORBI application specific functionality. The results of this case study as presented in Section 11.3.3 have been used to provide supporting evidence with regards to our hypotheses concerning realism; those hypotheses include validating realism of the provenance facets (Hypothesis 1.1.3), validating realism of the provenance question type patterns (Hypothesis 1.2.3) and at last validating realism and reusability of the Template Provenance Infrastructure Model (Hypotheses 1.4.2 and 1.4.3).

Context of the Case Study

Our industrial case scenario contains a web-based service, ORBI [129], offered as a SaaS solution by Singular Logic [129]. ORBI provides a set of functionalities that enables management of financial transactions (e.g., create order/sales/purchases, make payment, issue invoice) together with contact and project management tasks (e.g., reporting, appointment organisation) designed to satisfy the needs of self-employed professionals. These functionalities form individual services, called orblets, associated with some specific service capability. This fragmentation provides the opportunity for combining functionalities according to
the specific user’s needs to form service aggregations or else service compositions.
The control flow of such an aggregation is presented in Figure 11.1 as a UML
activity diagram where an action corresponds to the specification of a specific
service task (orblet).

The tasks are to request a contact service ($t_1$), arrange (search/create/update)
an appointment ($t_2$), make a calendar request ($t_3$), request a report (journal/project) ($t_4$), create an order (sales/purchase/item service/collection) ($t_5$), make a
payment ($t_6$), and issue an invoice ($t_7$). A weather and RSS service also form
part of the ORBI aggregation, yet we do not show this as part of the particular
control flow here.

Having presented the context of both our case studies the second part of this
chapter presents and discusses the evaluation results of our thesis in detail.

11.3 Evaluation Results

Having presented the set of hypotheses for evaluating the research contributions
of this thesis and our evaluation methodology for these, next we are going to
present how we have tested our hypotheses in practice along with the assembled
supporting evidence and providing an explanation with regards to our evaluation
results.

Our evaluation methodology as already described contains different types of
validation including:

- an evaluation criteria checklist of the core SOA/SOC concepts
- a use case study using a standard example scenario from the SOA and SOC
  literature - namely the Travel Planner Use Case Study
- an industrial case study using a real-world composite service system - ORBI
• a parser and type checking implementation that applies syntactic and semantic checks

• a prototype tool implementation - namely PROVα - that realises our two use case studies

Next, we present in detail the evaluated hypotheses and evaluation results showcasing those through the use of the different types of validation that we have listed above, starting with the SOC and SOA evaluation criteria checklist.

11.3.1 SOC and SOA Criteria Checklist Evaluation

One of our tools for our evaluation methodology was to build an evaluation criteria checklist of the core SOA/SOC concepts and service principles.

Evaluated Hypotheses

We use our evaluation criteria checklist, against which we validate the different types of modelling artefacts as part of our proposed specification and analysis framework with regards to:

• Hypothesis 1.1.1 - completeness of provenance facets

• Hypothesis 1.3.1 - conceptual adequacy/expressivity of ServiceProv model

Evaluation Results

In this section we list the core concepts of service-oriented architecture (SOA) and service-oriented computing (SOC) paradigm considering the service and service composition foundations and principles as those were described in Section 2.1 and referred from the wider service-oriented computing (SOC) [119] literature (considering the processes of service discovery, service selection, service interface publishing, composition and aggregation), the extended service-oriented architecture (SOA) [120], the service-oriented computing (SOC) pyramid as shown in Figure 2.1 and the state of the art and grand challenges in services’ research [119] as presented in Table 11.1.
Table 11.1: State of the Art and Challenges in Services Research [119]

<table>
<thead>
<tr>
<th>State of the Art</th>
<th>Grand Challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Service Foun-</strong></td>
<td><strong>Enterprise Service Bus:</strong> Open Standards-based message backbone, Implementation/deployment/management, Set of Infrastructure Capabilities implemented by middleware technology, Implementation backbone for SOA (application as services)</td>
</tr>
<tr>
<td><strong>dations</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Service Com-</strong></td>
<td>Orchestration: Service Interaction at message level, Perspective and control of single endpoint, Executable business process</td>
</tr>
<tr>
<td><strong>position</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Service Man-</strong></td>
<td>Web Services Distributed Management (WSDM), Management Using Web Services (MUWS), Management of Web Services (MOWS)</td>
</tr>
<tr>
<td><strong>agement</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Service De-</strong></td>
<td>Post existing components using wrapper, Component-Based Development, Object Oriented Analysis and Design, Do not address key elements: services, composition, components realising services, Only address part of the requirements</td>
</tr>
<tr>
<td><strong>sign and De-</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Development</strong></td>
<td></td>
</tr>
<tr>
<td><strong>(Service En-</strong></td>
<td></td>
</tr>
<tr>
<td><strong>gineering)</strong></td>
<td></td>
</tr>
</tbody>
</table>

318
An extended description and definitions of the core SOA/SOC concepts and service principles of our evaluation criteria checklist as presented in Tables 11.2 and 11.3 can be found in Appendix 13.1. In Table 11.2 we make use of those core concepts in order to validate our different conceptual elements and types of modelling artefacts as part of our proposed framework for provenance awareness specification and analysis of service compositions. We do so by comparing whether for each core concept we have identified in fundamental pieces of literature for SOA and SOC literature (column 1 in Tables 11.2 and 11.3), 1) all the core concepts are subsumed/covered by one or a number of the proposed provenance facets (column 2 in Tables 11.2 and 11.3) and 2) there is an equivalent concept defined within the ServiceProv ontology that describes the core aspects of the SOA/SOC domain in order to provide an adequate vocabulary and provenance data schema structure for expressing provenance properties and provenance requirements of service-oriented computing systems (column 3 in Tables 11.2 and 11.3). In this way we are able to validate in particular 1) completeness of our proposed provenance facets (evaluating Hypothesis 1.1.1) and 2) conceptual adequacy/expressivity of the ServiceProv ontological model (evaluating Hypothesis 1.3.1).

**Evaluating Hypothesis 1.1.1** Completeness of our proposed faceted classification scheme is validated at a first step by checking whether each and every one of the core SOC and SOA concepts being part of our criteria checklist is fully covered through the provenance facets as shown in Tables 6.3, as one of the key provenance concepts for service-oriented computing systems and compositions of services. We claim over a complete list of core concepts identified as the criteria of our checklist after having conducted an extended literature review over fundamental pieces of research in the SOC and SOA space. Therefore, in the second column of Tables 11.2 and 11.3 we map the core SOC and SOA concepts to the corresponding provenance facets where those concepts are required to be used for expressing provenance awareness requirements. At a second step, completeness of the proposed facets is validated by using our standard use case scenario, as
Table 11.2: SOC and SOA Evaluation Criteria Checklist (I)

<table>
<thead>
<tr>
<th>SOA and SOC Concepts</th>
<th>Facet No</th>
<th>ServiceProv Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Basic/Composite) Service</td>
<td>1</td>
<td>serviceprov:Service</td>
</tr>
<tr>
<td>Service Description</td>
<td>1, 9</td>
<td>serviceprov:ServiceDescription, serviceprov:ServiceCollection</td>
</tr>
<tr>
<td>Server</td>
<td>1, 8</td>
<td>serviceprov:ServiceExecutionServer, serviceprov:Server</td>
</tr>
<tr>
<td>Repository</td>
<td>1</td>
<td>serviceprov:Registry</td>
</tr>
<tr>
<td>Message(s)</td>
<td>2</td>
<td>serviceprov:Message</td>
</tr>
<tr>
<td>Message Semantics</td>
<td>2, 9</td>
<td>serviceprov:ServiceDescriptionItem, serviceprov:Parameter</td>
</tr>
<tr>
<td>Operation(s)/Behaviour/</td>
<td>2, 9</td>
<td>serviceprov:ServiceDescriptionItem, serviceprov:ServiceDescriptionDiscoveredItem</td>
</tr>
<tr>
<td>Capability</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Service Interface</td>
<td>1, 9</td>
<td>serviceprov:ServiceDescription Published Item</td>
</tr>
<tr>
<td>Execution</td>
<td>5, 9</td>
<td>serviceprov:ServiceExecution, serviceprov:OrchestratorExecution, serviceprov:AtomicServiceExecution</td>
</tr>
<tr>
<td>Publication</td>
<td>1</td>
<td>serviceprov:ServiceInterfacePublishing</td>
</tr>
<tr>
<td>Service Client</td>
<td>8</td>
<td>serviceprov:Broker</td>
</tr>
<tr>
<td>Service Provider</td>
<td>1, 8</td>
<td>serviceprov:ServiceProvider</td>
</tr>
<tr>
<td>Service Publisher</td>
<td>1, 8</td>
<td>serviceprov:Broker</td>
</tr>
<tr>
<td>Service Registry</td>
<td>1, 8</td>
<td>serviceprov:Registry</td>
</tr>
<tr>
<td>Service Discovery</td>
<td>1</td>
<td>serviceprov:Discovery</td>
</tr>
<tr>
<td>Service Broker/Discovery</td>
<td>1, 8</td>
<td>serviceprov:Broker</td>
</tr>
<tr>
<td>Agency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Service Selection</td>
<td>5, 9</td>
<td>serviceprov:Selection</td>
</tr>
<tr>
<td>Service Selection Model</td>
<td>5, 9</td>
<td>serviceprov:SelectedServiceDescription</td>
</tr>
</tbody>
</table>
Table 11.3: SOC and SOA Evaluation Criteria Checklist (II)

<table>
<thead>
<tr>
<th>SOA/SOC Concepts</th>
<th>Facet No</th>
<th>ServiceProv Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Aggregation/Coordination</td>
<td>5, 9</td>
<td>serviceprov:Planning, serviceprov:Selection</td>
</tr>
<tr>
<td>Service Aggregator</td>
<td>8</td>
<td>serviceprov:Aggregator</td>
</tr>
<tr>
<td>Service Invocation</td>
<td>2, 5</td>
<td>serviceprov:AtomicServiceExecution</td>
</tr>
<tr>
<td>Orchestration(Composition)</td>
<td>2, 5</td>
<td>serviceprov:OrchestratorExecution</td>
</tr>
<tr>
<td>Orchestration Model</td>
<td>2, 5</td>
<td>serviceprov:WorkPlan</td>
</tr>
<tr>
<td>Orchestrator</td>
<td>8</td>
<td>serviceprov:Orchestrator</td>
</tr>
<tr>
<td>Choreography(Composition)</td>
<td>2</td>
<td>serviceprov:Orchestration missing the Orchestrator management aspect</td>
</tr>
<tr>
<td>QoS Model/Conformance</td>
<td>7</td>
<td>serviceprov:QoSRequirements, serviceprov:QoSContract</td>
</tr>
<tr>
<td>NFPs</td>
<td>6, 7</td>
<td>serviceprov:NFP</td>
</tr>
<tr>
<td>Reliable Messaging</td>
<td>6, 7</td>
<td>serviceprov:Reliability as a specialization of serviceprov:NFP</td>
</tr>
<tr>
<td>Security Model</td>
<td>7</td>
<td>serviceprov:Security as a specialization of serviceprov:NFP or</td>
</tr>
<tr>
<td>Resource</td>
<td>3</td>
<td>serviceprov:Resource and serviceprov:ResourceCollection</td>
</tr>
</tbody>
</table>

described in Section 11.2.1 to systematically express provenance awareness requirements for this scenario making use of those core SOC and SOA concepts being part of our criteria checklist. This systematic work of expressing provenance questions for our travel planner scenario has been already presented in detail in Section 6.1.2. With regards to that a single standard example (ie the travel planner) is sufficient to support your hypothesis.

**Evaluating Hypothesis 1.3.1** Conceptual adequacy of the ServiceProv ontological model (evaluating Hypothesis 1.3.1) is validated by checking whether the ServiceProv ontological model expresses all the core concepts of the Service-oriented architecture (SOA) and the Service-oriented Computing (SOC) criteria checklist. In order to showcase ServiceProv’s conceptual adequacy, we map the core SOC and SOA concepts to the equivalent ServiceProv ontological concepts introduced in the ServiceProv ontology as shown in the third column of Tables 11.2
and 11.3.

**Note:** An extensive list of non-functional specialized properties (e.g., Performance, Availability, Execution Time, Response Time, Cost and Penalty, Reputation) and resource types (e.g., CPU, Memory, Operating System) that are expressed both as part of our provenance facets and our ServiceProv ontological model are not explicitly presented in our criteria checklist as those concepts are considered to be subsumed by their supertype concepts – NFP and Resource.

### 11.3.2 Travel Planner Case Study Evaluation Results

Next we present the results of evaluation for our travel planner case study with regards to the following evaluated hypotheses.

**Evaluated Hypotheses**

Through our travel planner case study we evaluate: Hypothesis 1.1.1, 1.1.2, 1.2.2, 1.3.1, 1.3.2, 1.4.1 and 1.4.2.

**Evaluation Results**

**Evaluating Hypothesis 1.1.1**  With regards to Hypothesis 1.1.1 we use the travel planner use case to validate the conceptual completeness of the proposed facets. We accomplish this by systematically expressing provenance awareness requirements for the travel planner composer example scenario - that is a standard example used as a model problem in the service compositions’ literature. In particular, we have exhibited that the provenance facets as presented in Chapter 6 are complete with regards to covering key aspects of the SOA and SOC extended architecture and compositions of services reflecting our criteria checklist as shown in Tables 11.2 and 11.3. Our effort to systematically express provenance awareness requirements for the travel planner scenario has been analytically presented in Section 6.1.2 first as a set of non-categorized queries about provenance aspects for composite service designs incorporating our full list of SOC/SOA principles and core concepts. At a second step we categorise and express those in a more
structured fashion across the provenance facets introduced in Section 6.2.1 where we exhibit in practice that those provide conceptual coverage with regards to the provenance data concepts required to be captured for answering provenance questions about the travel planner scenario.

**Evaluating Hypothesis 1.1.2** With respect to Hypothesis 1.1.2 we use the travel planner use case to validate the adequacy of the proposed provenance facets by identifying whether we can adequately express a set of provenance questions organized using one or a combination of our facets. The meaning of adequacy mainly here refers to whether we have chosen a sufficient and correct mechanism that 1) allows to easily express provenance awareness requirements represented as provenance questions spanning across different semantic categories, and 2) provides the flexibility to systematically derive more complex questions that correspond to the expression of composite provenance awareness requirements. We have illustrated this ability of our proposed facets by expressing composite provenance questions spanning across multi-faceted categories for the travel planner scenario in Section 6.2.2.

**Evaluating Hypothesis 1.2.2** In Hypothesis 1.2.2 the travel planner case study is used to validate the expressiveness of the identified provenance question type patterns by expressing and realizing the full list of the provenance questions (requirements) for the travel planner scenario as a set of SPARQL query patterns. For example, in order to answer the provenance question “What was the output message for the flight booking service execution?”, we translate this to the SPARQL query as shown in Listing 11.1 and explained in detail in 11.3.2. The full list of SPARQL queries expressing the provenance awareness requirements for the travel planner use case is exhibited in Section 15.1 of the Appendix and those queries correspond to a realisation of the formal provenance question type patterns introduced in Chapter 8. The responsibility for writing those queries is granted to the QoS provenance specialist as this was defined in the architecture diagram in Figure 5.3.
SELECT ?serviceName ?outMessage
WHERE {?
  parameterout p1:value ?outputValue.
  ?outMessage p1:hadMember ?parameterout.
  ?outMessage p1:wasGeneratedBy ?servexecution.
  ?service serviceprov:serviceName ?serviceName.
  FILTER (regex (str (?serviceName), "easyJetflight", "i"))
  ?servexecution p1:used ?inMessage.
  ?inMessage p1:wasGeneratedBy
  orchestratorExecution.
  ?orchestratorExecution. p1:value
  ?orchestratorExecutionID.
  FILTER (regex(str(?orchestratorExecutionID),
  "travelPlannerNWX1UX", "i"))
}

Listing 11.1: Querying Data Flow’s Provenance

Evaluating Hypothesis 1.3.1 and 1.3.2 We further make use of the travel planner case study in order to validate that 1) the ServiceProv ontological model is conceptually adequate to faithfully represent provenance information for service-oriented computing systems incorporating the aspects of extended SOA and the SOC pyramid as those were presented in Table 11.2 – (Hypothesis 1.3.1), and 2) ServiceProv ontological model is applicable for reasoning over provenance instances in order to answer specific provenance questions – (Hypothesis 1.3.2).

The hypotheses discussed above correspond to two potential attacks on our ontological model:

1. The concepts provided do not adequately reflect the reality and service-oriented computing.

2. This is not a complete model with regards to its applicability for reasoning over relevant provenance questions.

For the evaluation of Hypothesis 1.3.1 we identify how the proposed concepts map to standard concepts of the service-oriented computing literature exhibiting those mappings through the travel planner use case. Then for the evaluation of Hypothesis 1.3.2, we discuss a number of provenance queries in order to show how our model can be used to extract provenance data instances in practice, reasoning over an instantiated provenance data schema (ServiceProv ontology model), being useful for answering different kinds of provenance questions.
Mapping ServiceProv Ontology Concepts to SOA and SOC standards through an Example

The proposed ontology-based model (ServiceProv ontology) concepts represent the provenance of service-oriented (computing) systems. Below, we provide additional support for this claim to the criteria checklist already presented in Section 11.3.1 by referring to standard literature aspects of service-oriented computing and relating those to the concepts in ServiceProv ontology and to a fully instantiated version of our proposed ontology for our TravelPlanner scenario; this can be found at [https://sourceforge.net/projects/serviceprov/files/TravelPlannerCaseStudyFullInstantiatedModel/](https://sourceforge.net/projects/serviceprov/files/TravelPlannerCaseStudyFullInstantiatedModel/).

Basic or atomic :Service(s) (e.g., serviceprov:flightServiceEasyJet, serviceprov:flightServiceBritishAirways) are a pair of a service interface and service implementation defined by a :ServiceDescription. The service interface is published (e.g., serviceprov:servDescriptionPublishedItemEasyJet) through a :ServiceInterfacePublishing activity as part of the service description and gives an abstract definition of the service’s allowed operations and its message exchange flow. It defines the service identity (e.g., serviceprov:ServiceName “easyJet flight”) and its invocation logistics, giving access to the service by other applications. When an application invokes one of the operations of the service interface, the service is executed (:ServiceExecution, e.g., serviceprov:flightBookingeasyJetExecution) [120].

The service implementation is hosted by a service provider (:ServiceProvider, e.g., serviceprov:role1) that defines and publishes (:ServiceInterfacePublishing, e.g., serviceprov:servInterfacePublishingEasyJet) the service description (:ServiceDescriptionPublishedItem, e.g., serviceprov:servDescriptionEasyJetFlight, serviceprov:servDescriptionPublishedItemEasyJet) to a registry (:Registry, e.g., serviceprov:flightsRegistry) making the description discoverable (:ServiceDescriptionDiscoveredItem, e.g., serviceprov:servDescriptionDiscoveredItemEasyJet) [120]. The service gets discovered (:ServiceDiscovery, e.g., serviceprov:discoveryFlights) from a service discovery agency (:Broker, e.g., serviceprov:flightsBroker) that acts as an intermediate between the service requester and the provider. Service composition en-
compasses the necessary roles and functionality for the aggregation of multiple services into a single composite service. Service aggregators (Aggregator, e.g., serviceprov:flightsAggregator) accomplish this task by selecting (ServiceSelection, e.g., serviceprov:selectionFlights) and grouping services that are provided by other service providers into a distinct value added service [119]. Service aggregators can act as service providers by publishing the service descriptions of the composite services they create (SelectedService Description, e.g., serviceprov:servSelectedServiceDescriptionItemEasyJet). They develop specifications (Planning, AbstractProcessPlan, e.g., serviceprov:processPlanTravelPlanner) with the goal to control the execution of the composite services (Orchestrator, OrchestratorExecution, e.g., serviceprov:role2 and serviceprov:travelPlannerExecution) [119]. Orchestration describes how services can interact with each other at the message level (Message, e.g., serviceprov:flightBookingeasyJetMessageIn), including the business logic and execution order (WorkPlan, e.g., serviceprov:workplanTravelPlanner) of the interactions. Successful service compositions need to be aware of QoS policies and requirements (QoSrequirements, e.g., we could express those as constraints over NFPs such as the following NFP instance serviceprov:availability easyJetFlight) [118]. A consistent monitoring and management infrastructure is essential for production-quality aggregated services [119]. We capture related concepts in our provenance model such as non-functional property (NFP e.g. serviceprov:availabilityeasyJetFlight) and resource (Resource or ResourceCollection e.g. SEALSComputingResource:CPU, SEALSComputingResource:Memory). Our ontology does not contain concepts that do not map to at least one of the fundamental steps of service processing as described above.

**Reasoning over ServiceProv Ontology Instances**

In order to show the practical use of our ontology-based model and validate its conceptual adequacy and applicability for reasoning over provenance instances of service-oriented systems, in this section we illustrate a number of provenance queries for our motivating travel planner example. This shows how extracting different kinds of provenance information represented in our
schema can be used to answer various provenance questions. To make our queries more precise, we will use the SPARQL query language for RDF [74]. We test our queries over ontology instances defined in Protege-OWL [84] using its incorporated implementation for SPARQL query engine [122] (based on a mechanism that wraps a Jena [24] model). A Protege file of the travel planner instances queried for the example questions exhibited in this section can be found at https://sourceforge.net/projects/serviceprov/files/serviceprovTravelPlannerExample/. Also, a fully instantiated version of ServiceProv instances for the travel planner use case study has been developed that can be found at https://sourceforge.net/projects/serviceprov/files/TravelPlannerCaseStudyFullInstantiatedModel/. This instantiated model has been used for validation of ServiceProv syntax and semantics as this is discussed in detail in section 11.3.4.

Given an instantiated provenance data schema of the travel planner, provenance questions can be asked by its users — for example “Why did the travel planner book a high-cost flight instead of a low-cost flight with easyJet?”.

In order to answer such a question the designers will first need to answer a couple of other questions such as “What were the service providers of the flight booking services that were discovered?”, by querying over the provenance instances of the travel planner. This question corresponds to the query shown in Listing 11.5 (exhibiting the service and provider identity facet from [156]), which is designed to retrieve the identifiers (e.g., IP address, port, provider name) for the providers that published the descriptions of the discovered flight booking services. Pointing from the future (provenance data query results) to the past, the query starts with identifying the triples related to our search query parameters (provider), (ip) and (port) (lines 1-3), and it continues with identifying the agents (provider) with the service provider role (serviceprov:ServiceProvider) that were associated with publishing a set of service descriptions (servdescription PublishedItem) (lines 4-11). It then narrows down the results to the discovered service description items derived (servdescriptionDiscoveredItem), being members of de-
SELECT ?serviceName ?provider ?ip ?port
WHERE {
?provider serviceprov:hadPort ?port.
?provider serviceprov:hadIPAddress ?ip.
?servInterfacePublishing a
serviceprov:ServiceInterfacePublishing.
?association pl:hadRole ?role.
?role a serviceprov:ServiceProvider.
?servdescriptionPublishedItem pl:wasGeneratedBy ?servInterfacePublishing.
?servdescriptionPublishedItem pl:wasDerivedFrom ?servdescriptionPublishedItem.
?servdescription pl:hadMember ?servdescriptionDiscoveredItem.
?service serviceprov:hadDescription.
?service serviceprov:serviceName ?serviceName.
FILTER ( regex (str(?serviceName), "flight", "i") )
?servCollection pl:hadMember ?servdescription.
?servCollection pl:wasGeneratedBy ?discovery.
?discovery pl:used ?servRequest.
?servRequest pl:wasGeneratedBy ?planning.
?workplan pl:wasGeneratedBy ?planning.
?orchestratorExecution pl:used ?workplan.
?orchestratorExecution pl:value
?orchestratorExecutionID.
FILTER ( regex (str(?orchestratorExecutionID), "travelPlannerNWX1UX", "i") )
}

Listing 11.2: Querying Service Provider’s Identity Provenance

scriptions (?servdescription) of services with a (?servicename) that corresponds to flight booking services (lines 12-18). Service descriptions are identified as members of a collection of services (?servCollection) for booking flights, with service collections being generated during service discovery (?discovery) based on a particular service request (?servRequest) that was generated by a planning activity (?planning), generated on the basis of a travel planner workflow specification (?workplan) (lines 19-23). Workplan was used by an orchestrator execution (?orchestratorExecution), the travel planner execution in our example, which manages the atomic services involved in the composition such as the flight booking service. We narrow down our results to the particular run we are interested in using its unique execution identifier (?orchestratorExecutionID) (lines 24-29).

If in the list of our query results there is a match for the easyJet provider, as shown in Table 11.7 then we may want to know whether the reason for not booking a flight with easyJet was the non-availability of tickets for the particular dates requested by the user.
In order to answer such a question we would need to query the provenance of the output message parameters for the flight booking service. The provenance question “What were the parameters of the output message for the flight booking service execution of easyJet service?”, will be translated to the SPARQL query shown in Listing 11.6 (exhibiting the Data flow facet from [156]). Again starting from the selected query data results and looking back to its provenance, the query first identifies the output parameters (?parameterout) of the output messages (?outputMessage) that were generated by the different executions (?execution) of the flight booking services (?service) discovered in the previous query (lines 2-3). Then it narrows down the service execution results to the one with the easyJet flight (?serviceName) (lines 5-7) that corresponds to the particular travel planner run we are interested based again on its unique execution identifier (?orchestratorExecutionID) (line (9-10). Yet, in order to make the connection between the output message of the flight booking service execution and the particular travel planner run, we need first to query the provenance of the input message (?inMessage) generated by the travel planner execution (?orchestratorExecution) that triggered the particular flight booking execution (?servexecution) (lines 8-8).

In the case in the list of our query results easyJet service provider is nowhere to be found, we may need to identify whether the particular service was not available (referring to the downtime of a service) after the service discovery took place. Therefore, the provenance question to ask will be: “What was the availability of the easyJet flight booking service after the service discovery?”. In order to answer this question we need to query over the QoS and NFPs (non-functional

### Table 11.4: Query Results for Service Providers

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Service Provider</th>
<th>IP address</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>easyJet flight</td>
<td>easyJet</td>
<td>70.32.02.90</td>
<td>80</td>
</tr>
<tr>
<td>British flight</td>
<td>British Airways</td>
<td>64.31.02.81</td>
<td>80</td>
</tr>
<tr>
<td>Lufthansa flight</td>
<td>Lufthansa</td>
<td>82.32.03.73</td>
<td>80</td>
</tr>
</tbody>
</table>
properties) provenance for the easyJet service, and over the availability property in particular its uptime or downtime values within a certain time frame. Such query shown in Listing 11.4 (exhibiting both the NFPs and Time facets from [156]), starts by identifying the uptime or downtime values (?value) for the availability (?availability) of a (?service) within a particular time frame (?ststart, ?stend) that the service was executing (?servexecution) (lines 3-7). It then narrows down the results to the service corresponding to the easyJet flight service and its executions for the particular travel planner run (?orchestratorExecution) identified by its unique id (lines 8-16). Yet, in order to find the required availability for the particular service, we need to filter this to the overlapping time frame that both the service and the orchestrator executed (lines 18-21). The query results for the service availability value will be of boolean data type, returning a false value whenever the service is “down” and a true value whenever the service stays “up”.

**Evaluating Hypothesis 1.4.1 and 1.4.2** With regards to Hypotheses 1.4.1 we use the travel planner use case to validate the conceptual expressivity and adequacy of the modelling notation for specifying provenance recording infrastructure activity and behaviour for compositions of services and the technical aspects of the SOC execution cycle as those were presented Chapter 9 and exhibited through showcasing the modelling requirements of a realistic standard use case scenario of the service composition literature as presented in Figure 9.6.3.
SELECT ?value
WHERE {?availability serviceprov:NFPvalue ?value.
?service serviceprov:hadNFP ?availability.
?service p1:invalidatedAtTime ?stend.
?service serviceprov:serviceName ?serviceName.
FILTER ( regex (str (?serviceName), "easyJetflight", "i"))
?execution p1:used ?inMessage.
?inMessage p1:wasGeneratedBy ?orchestratorExecution.
?orchestratorExecution p1:value ?orchestratorExecutionID.
FILTER ( regex (str(?orchestratorExecutionID), "travelPlannerNWX1UX", "i") )
?orchestratorExecution p1:startedAtTime ?otstart.
FILTER (?ststart <= ?otend )
FILTER (?stend >= ?otstart )}

Listing 11.4: Querying NFP’s Provenance (Availability)

We then realise this through our prototype tool implementation discussed later in Section 11.3.5.

With regards to Hypotheses 1.4.2 we use the travel planner scenario to showcase the reusability of our template infrastructure model (as shown in Figure 10.4) and proposed template binding specification (as described in Listing 9.5) for various composite service designs including ORBI industrial case in Section 11.2.2. The results of the industrial case study are presented in 11.3.3) exhibiting that our proposed modelling notation and approach for designing provenance infrastructure for a composite service system can be independent of specific applications as showcased in Section 9.5.2.

Next, we showcase the results of our industrial case study and how this is used to validate our hypotheses concerning realism.

11.3.3 Industrial Case Study (ORBI) Evaluation Results

We have already presented the context of our industrial case study in section 11.2.2. Here we are going to exhibit how using a real-world composite service system can evaluate/exhibit certain aspects of our hypotheses.
Evaluated Hypotheses

We use an industrial case study using a real-world composite service system to evaluate:

- Hypothesis 1.1.3 - realism for the proposed provenance facets
- Hypothesis 1.2.3 - realism for the proposed provenance question type patterns
- Hypothesis 1.3.3 - realism with regards to core concepts of the ServiceProv model representing provenance data for compositions of services
- Hypothesis 1.4.2 and Hypothesis 1.4.3 - reusability and realism of the Template Provenance Infrastructure Model

Evaluation Results

An Exploration and Exploitation of Real-world Provenance

The objective of our study is the exploration of real-world provenance through an industrial case. In order to accomplish this, we record the provenance data required for covering a set of provenance needs (questions) about ORBI’s processing history as presented in Table 11.5. The data captured is then used to construct an application-specific provenance data model by instantiating concepts of ServiceProv ontology with real provenance data – ORBI individuals. This enables us to check whether the captured data represent sufficient information to answer provenance questions for our system by reasoning over ORBI individuals expressed as OWL/RDF triples.

The steps that we performed in the industrial case study can be revised more concretely to the following:

i Identify potential provenance data requirements: We draw out the control flow of the use case application scenario (ORBI) and we identify the provenance data we would like to capture in order to answer different kinds of
provenance questions (requirements) exhibiting facets of Table 14.1 that are
tailored to the aggregated ORBI’s functionality.

ii *Formally express questions into SPARQL:* As long as we identify interesting
questions for our scenario, we formally express those as SPARQL queries.
The data required for forming those SPARQL queries should drive the data
capture. The responsibility for writing those queries is granted to the QoS
provenance specialist as this was defined in the architecture diagram in
Figure 5.3.

iii *Capturing/Recording real-world provenance:* We record data about the sys-
tem’s history processing (provenance data) by tracing the system of our
study, namely ORBI, during its execution. Those records include the kind
of data required for answering different types of provenance questions as
those were presented in Table 11.5, exhibiting a set of previously identified
facets.

iv *Instantiate ServiceProv:* We use the provenance data captured to popu-
late our ontological-based model (ServiceProv) with real-world provenance
data instances, namely ORBI individuals. This step will in practice give
as a result an instantiated schema that represents ORBI’s provenance data
structure through the use of the corresponding ServiceProv concepts and
the links/relationships between different kinds of provenance data repre-
sented by ServiceProv’s main concepts. Those links include association
relationships with different kinds of agents (e.g., Execution Server, Ser-
vice Broker, Orchestrator, Service Provider), generation relationships with
different kinds of activities (e.g., ServiceExecution, Orchestration) and us-
age/derivation relationships with different kind of entities (e.g., Message(s),
Resource(s), NFP(s)) where the relationships are denoted through Service-
Prov’s object/datatype properties.

iv *Execute and test SPARQL queries:* We test the set of SPARQL expressions
by querying/reasoning over ORBI’s provenance data instances – defined in

333
ServiceProv Ontology – through Protege tool and its incorporated implementation of SPARQL query engine (based on a mechanism that wraps a Jena model).

v Analyse query results: Finally, we analyse our data query results in order to check: 1) whether this is sufficient information to answer our queries (ORBI requirements) by checking how many of these questions can/can not be answered, 2) whether we have sufficiently explored and exploited provenance data that exhibit the different facets, 3) the accuracy of the answered questions. We evaluate our two last points by using as a metric the number of successfully answered questions/queries that exhibit each facet.

Next, we present how we have applied those steps for our case study.

Provenance Requirements for ORBI In this section we provide motivation for the need to introduce provenance awareness support for ORBI by presenting a set of provenance questions (requirements) along with the provenance data that would need to be captured in order to answer those in Table 11.5. The questions have been chosen to cover facets we have previously identified in Chapter 6 by abstracting and conceptualising the provenance information required for answering questions exhibiting those categories as summarised in Table 14.1.

Users may have questions such as what was the service executor of the payment service or what was the make order service execution ID (exhibiting the Service and Provider Identity facet), where provenance data such as the IP address and port number of the payment execution server or the service execution ID of the make order service need to be captured respectively. Similarly, the rest of the questions in Table 11.5 cover the set of provenance requirements for ORBI application related to its data flow (e.g., what was the input/output parameters of the payment service), related to resources’ availability (e.g., what were the available resources such as CPU usage, memory consumption) for ORBI service
Table 11.5: Provenance Data Recording Requirements for ORBI

<table>
<thead>
<tr>
<th>Facets</th>
<th>Provenance Questions</th>
<th>Provenance Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service &amp; Provider Identity</td>
<td>What was the service executor of the payment /issue invoice/make order services? What was the make order service execution ID?</td>
<td>IPaddress/Port of ORBI/VIVA service executor, ServiceExecution ID</td>
</tr>
<tr>
<td>Data Flow</td>
<td>What was the service request for the create order service execution? What were the input/output parameters for the payment service?</td>
<td>ServiceRequest Message, Payment Input/Output Messages (Parameters)</td>
</tr>
<tr>
<td>Resources</td>
<td>What were the available resources (CPU usage, memory consumption, operating system) for ORBI service execution?</td>
<td>CPU/ memory resource values, type of operating system</td>
</tr>
<tr>
<td>NFPs &amp; QoS</td>
<td>What was the response time, latency for the ORBI service execution? Where there any particular SLA agreements that needed to be satisfied for the payment service execution?</td>
<td>Response time, latency values SLA requirements with regards to the payment gateway</td>
</tr>
<tr>
<td>Past History Data</td>
<td>What was the availability of ORBI service for the timeframe between 5/05/14 to 12/05/14?</td>
<td>Uptime/downtime values</td>
</tr>
<tr>
<td>Timestamps</td>
<td>When did the make order service execution start/end?</td>
<td>service execution start/end timestamps</td>
</tr>
<tr>
<td>Actors</td>
<td>What was the service provider of the payment service? What was the broker of the payment service?</td>
<td>IPaddress/Port of payment service provider and payment gateway</td>
</tr>
</tbody>
</table>

execution, to non-functional properties (NFPs) (e.g., what was the response time of the ORBI execution service), to past history data where capturing and storing provenance information for long time intervals or more for the whole service life cycle is needed (e.g., what was the availability of the ORBI execution for the period between 5/05/12 and 12/05/14), to timestamps for different service processes/entities started/generated during ORBI execution (e.g., when did the make order execution start/end, and to actors that may manage or affect ORBI’s execution (e.g., what was the broker for the payment service).

Those questions are a representative set of provenance questions for ORBI system, along with the kind of provenance data that are required to be captured for answering them. The complete set of provenance questions (requirements) with regards to ORBI functionality that we used in order to explore and exploit real provenance for this case study can be found online at http://sourceforge.net/projects/serviceprov/files/ORBI/ORBI_queryExamples.txt. Those ques-
tions are realistic questions we have gathered as part of a short interviews process conducted in Singular Logic with developers, IT operations and infrastructure department, and business and marketing analysts and strategists responsible for ORBI application.

Our effort to express systematically provenance awareness requirements for a real world system validates Hypothesis 1.1.3 while it also showcases that a set of realistic requirements is covered by the proposed elemental faceted classification scheme as those can be instantiated in the context of a real-world case study. The full list of provenance questions is listed in Appendix 15.2.

**Exhibiting Real World Provenance with ORBI** In Table 11.5 we identified the provenance requirements users may have with regards to the ORBI service execution. Those requirements take the form of questions that exhibit a set of provenance facets based on the kind of provenance data that need to be captured in order to answer those questions. The methodology we follow is that we first translate those users’ questions into a machine-processable format such as provenance query patterns expressed in SPARQL and we then use those queries to reason/query over the instantiated ORBI provenance data schema. The instantiated data schema represents the provenance data structure of an ORBI execution and is build by creating ORBI individuals using the ServiceProv’s basic concepts and properties; exhibiting the provenance data structures for ORBI real-world system allows us to evaluate Hypothesis 1.3.3 with regards to the realism of the ServiceProv ontological provenance data model. We therefore validate that the ServiceProv ontology is conceptually adequate to model realistic provenance data of an actual real-world composite service system – ORBI.

In summary the methodology followed in our industrial case study practically enable us to draw meaningful conclusions about

- the provenance awareness of our case study system by looking into what questions can/can not be answered based on real provenance data representation of ORBI
whether capturing different kinds of provenance data is realistic in such a structure and

whether this data is sufficient to cover the corresponding provenance awareness requirements (provenance question answering) exhibiting the previously identified provenance facets.

Expressing Provenance Requirements with SPARQL  In this section we illustrate a set of provenance query examples for our case study – expressed in SPARQL – along with a detailed description of their structure and the data query results. Those show how extracting different kinds of provenance information for ORBI system, represented using ServiceProv concepts and properties, can be used to answer provenance questions for ORBI. An OWL file of the ORBI provenance instances captured and queried for the example questions described here, along with the complete list of queries tested in our case study can be found at http://sourceforge.net/projects/serviceprov/files/ORBI/serviceprovORBI.owl.

The first example query, shown in Listing 11.5, is a formal expression of the provenance question “which server executed the create order service”, exhibiting the service and provider identity facet. Answering this question requires recording information such as the IPaddress and port number of the execution server of the create order service. With this we can create instances for the respective ServiceProv concepts such as the :IPaddress, :Port, :Server etc., and then reason over these. The execution query results are shown in Table 11.6.

If we take a concrete look into the example query data results back to its provenance, the query first identifies the triples related to our search query parameters (?server), (?ip) and (?port) (lines 1- 3), and it continues with identifying the agents (?server) with the service executor role (serviceprov:ServiceExecutionServer) that were associated with a set of service executions (?servExecution) (lines 4-8). It then narrows down the results to the service executions of services with
Listing 11.5: Querying Service & Provider Identity Provenance

<table>
<thead>
<tr>
<th>Service</th>
<th>Server</th>
<th>IP</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateOrder</td>
<td>server3</td>
<td>79.131.33.143</td>
<td>443</td>
</tr>
</tbody>
</table>

Our second example query, shown in Listing 11.6, is a formal expression of the provenance question “what were the input parameters for the payment service for a particular ORBI execution”, exhibiting the data flow facet. Answering this question requires recording information such as the input parameter values of the input messages for the payment service (VIVApayment service). With this, we can create instances for the respective ServiceProv concepts such as :Message,
SELECT ?parameterIn ?inputValue
WHERE {?parameterIn p1:value ?inputValue .
?inMessage p1:hadMember ?parameterIn .
?servexecution p1:used ?inMessage .
?service serviceprov:serviceName ?serviceName .
FILTER (regex (str (?serviceName), "VIVApayment", "i"))
?inMessage p1:wasGeneratedBy ?orchExecution .
?orchExecution p1:value ?orchExecID . FILTER (regex(str(?orchExecID) "app.orbi.gr/client/dispatch.php", "i"))
"app.orbi.gr/client/dispatch.php", "i"))

Listing 11.6: Querying Data Flow’s Provenance

Table 11.7: Query Example Results (II)

<table>
<thead>
<tr>
<th>Input Parameter</th>
<th>Input Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequestAmount</td>
<td>11</td>
</tr>
<tr>
<td>MaxInstallments</td>
<td>0</td>
</tr>
<tr>
<td>RequestLanguage</td>
<td>el-GR</td>
</tr>
<tr>
<td>MerchantTransf</td>
<td>10457_F</td>
</tr>
</tbody>
</table>

If we again take a concrete look from the query data results back to its provenance, the query first identifies the parameters (?parameterIn) of the input messages (?inMessage) that were used by the (?execution(s)) of the payment (?service) (lines 2-3). Then it narrows down the service execution results to the one with the VIVApayment (?serviceName) (lines 5-7), that belongs to a particular ORBI run identified by a unique execution ID (?orchExecID) (line 9-10). Yet, once again in order to create a connected picture between the inMessage of the payment service execution and the particular ORBI run, we also need to query the provenance of the (?inMessage) generated by the ORBI execution (?orchExecution) that triggered the particular payment service execution (?servexecution) (line 8).

Our next example query, shown in Listing 11.7, is a formal expression of the provenance question “what was the execution time of the ORBI service execution”, exhibiting the NFPs and QoS facet. Answering this question requires recording information such as the execution time/nfps value of the particular ORBI exe-
SELECT ?orchExecution ?serviceName ?value
WHERE {?executionTime serviceprov:NFPvalue ?value.
?executionTime a serviceprov:ExecutionTime.
?service serviceprov:serviceName ?serviceName.
FILTER ( regex (str(?serviceName), "ORBI", "i"))
?orchExecution a serviceprov:OrchestrationExecution.
?association pl:hadRole ?role.
?role a serviceprov:Orchestrator.
?orchExecution pl:value ?orchestratorExecutionID.
FILTER (regex(str(?orchestratorExecutionID), "app.orbi.gr/client/dispatch.php", "i"))
}

Listing 11.7: Querying NFPs and QoS Provenance

Table 11.8: Query Example Results (III)

<table>
<thead>
<tr>
<th>orchExecution</th>
<th>serviceName</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORBIexecution</td>
<td>ORBI</td>
<td>0.02831sec</td>
</tr>
</tbody>
</table>

cution we are interested in. With this we can create instances for the respective ServiceProv concepts such as :NFP(s), :ExecutionTime, :OrchestrationExecution etc., and then reason over these in order to bring an answer to the corresponding provenance questions. The execution results of this query are shown in Table 11.8.

If we take a concrete look into the example query data results back to its provenance, the query first identifies the NFP (execution time) values (?value) with regards to orchestration executions (?orchExecution) (lines 1-5). Then it narrows down the orchestration execution results to the one with the ORBI (?serviceName) (lines 6-11), that corresponds to a particular ORBI run identified by a unique execution ID (?orchExecID) (lines (12 -13).

The complete set of tested query examples exhibiting the rest of the facets (provenance question categories) and ORBI provenance requirements similar to the ones presented in Table 11.5 can be found online at http://sourceforge.net/projects/serviceprov/files/ORBI/ORBI_queryExamples.txt and in Appendix 15.2. Those queries comprise the complete list of supporting evidence for validating Hypothesis 1.2.3.
Table 11.9: ORBI Case Study Analysis Results

<table>
<thead>
<tr>
<th>Facets</th>
<th>Provenance data collection</th>
<th>Number of Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service &amp; provider</td>
<td>Identification information for service executor/provider &amp; service (e.g., IP address, port number, service name, URI)</td>
<td>19</td>
</tr>
<tr>
<td>Identity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data flow</td>
<td>Data parameters passed of input/output messages</td>
<td>20</td>
</tr>
<tr>
<td>Resources</td>
<td>Availability/Usage of resources during execution</td>
<td>3</td>
</tr>
<tr>
<td>NFPS/ QoS</td>
<td>Recording info for different NFPS (e.g., RTT, response time)</td>
<td>3</td>
</tr>
<tr>
<td>Past history data</td>
<td>Data about availability (up/down time of services, failure rates)</td>
<td>2</td>
</tr>
<tr>
<td>Time</td>
<td>Timestamps for service/ORBI execution start/end</td>
<td>16</td>
</tr>
<tr>
<td>Actors</td>
<td>Identification information of third parties (broker, orchestrator)</td>
<td>3</td>
</tr>
</tbody>
</table>

Analysis Discussion  In this section we mainly discuss our analysis results of whether we have sufficiently explored and exploited provenance data that exhibit the different facets (provenance question categories). Facets are identified through representative provenance questions (provenance requirements) for ORBI’s service provenance. We measure/evaluate our analysis results by using as a metric the number of successfully answered questions/queries that exhibit each facet question, the results of which are shown in Table 11.9.

In practice our results have shown us that with the data tracing procedure we have managed to record data required to answer provenance questions that cover the complete list of facets we have previously identified (validating Hypothesis 1.1.3). This is done more extensively through instantiating our ServiceProv ontology with the provenance data captured for ORBI execution (validating Hypothesis 1.2.3) and then by reasoning over those OWL instances (ORBI individuals) using the set of SPARQL queries (validating Hypothesis 1.3.3). The number of the queries for which we managed to get the corresponding provenance results is a proof of both the soundness of how the provenance questions are expressed and the completeness of service provenance data requirements we managed to record.
and exploit for a real-world aggregated (composite) service-based system.

Taking a step back, we can argue that these results reflect the completeness of the current implementation according to our set of questions. The fact that these questions constitute a representative set from all the identified facets of service provenance allows us to defend the idea that our provenance data recording is suited for, at least, the key provenance data types, providing sufficient information for composite service systems to support robust provenance awareness mechanisms. Naturally, the service domain is not a closed world one, and there probably are a few provenance questions that would be more complex (requiring combination or aggregation of provenance information for answering multi-faceted questions). It may even be impossible to answer some questions with the current implementation in the future as the service-oriented (computing) architecture is extended or transitioned into an enhanced architecture with additional characteristics (e.g., service-based systems on top of a cloud infrastructure).

On the long term it would appear the only natural for the provenance data capture to be driven, not by a standard implementation, but by the provenance awareness requirements (e.g. expressed as SPARQL queries), which is the main goal of our specification and analysis framework of provenance awareness. In this perspective, it is interesting that we have managed to achieve a level of automation by providing a modelling and analysis environment to explore provenance awareness support of composite service systems. This allows designers and architects of a system to specify its provenance related properties, the infrastructure needed for supporting these, and then to verify whether the provenance infrastructure design would be sufficient to capture the desired data for particular services that are dynamically bound and composed. Such mechanism implies a strong constraint: the necessity for provenance queries to be known before the execution. Yet the main benefit there is to enable the service designer/architect/developer to check whether there are cases where a provenance question cannot be answered by the current provenance recording infrastructure. From a performance perspective, it also helps to reduce the provenance data collection footprint.
by considering only the necessary provenance data they would like to capture for a particular service design based on the provenance awareness requirements specification.

The core component and research contribution of our proposed framework supporting the features of automation and reusability in the process of specifying and analysing provenance awareness of composite service designs is the proposed Template Infrastructure Model and Template Binding Specification mechanism. As we have already mentioned this allows modelling and fine-tuning in the architecture decisions of the provenance infrastructure design independently of specific applications which allow us to reuse and apply a provenance infrastructure design for supporting a set of provenance awareness requirements to potential different service-based system designs. The Template Provenance Infrastructure Model and Binding Specification as applied for our industrial case study scenario and realised through our prototype tool implementation validates our hypothetical statements 1.4.2 and 1.4.3 about reusability and realism. We provide supporting evidence with regards to that in Section 11.3.5.

**Evaluation Summary**  We presented an industrial case study on provenance awareness of service-based systems (considering the aspects of service compositions). We have used ORBI system to exploit real-world provenance by tracing the system and recording different kinds of data based on a set of provenance question requirements, representing a set of provenance facets. We found that it is possible to capture representative provenance information for the complete set of questions and we have exploited the practical use of this information by querying and reasoning over an instantiated ORBI provenance data model (schema). Our focus on analysing and measuring the ability of a system to answer/not answer questions about the system’s history processing and execution – defined as provenance awareness – in an automated manner is realised through our prototype tool implementation as described in Section 11.3.5.
11.3.4 Parser and Type Checking Evaluation

As part of our evaluation methodology we have built a parser and type checking implementation that applies syntactic and semantic checks on SPARQL queries on ServiceProv models. Syntactic checks are not limited to testing well-formed triple assertions within a SPARQL query but extend to syntax checks of the classes and properties defined as part of the triple assertions to be consistent with the syntax and concepts defined in PROVo and ServiceProv ontology. They form syntactic checks in terms of checking the syntax of class property names of PROVo and ServiceProv ontology models. Those turn into semantic checks when we are looking into building the assertions with a valid (subject, predicate, object) structure according to PROVo and ServiceProv semantics. The parser and type checking implementation formed our first effort for developing a program executing provenance query patterns expressed in SPARQL – representing our provenance awareness requirements for a service-based system – over ServiceProv instantiated models. It mainly formed our first automated specification and analysis environment for provenance awareness of service-oriented computing systems and service compositions.

Evaluated Hypotheses

We have built the parser and type checking implementation to validate:

- Hypothesis 1.2.2 - with regards to the well-formedness of our provenance question (query) type patterns with regards to ServiceProv model

- Hypothesis 1.3.2 - the applicability of the extended ServiceProv ontology for modelling provenance properties of service-oriented computing systems, the reasoning and analysis qualities of the proposed provenance query patterns by building an instantiated model of the TravelPlanner scenario based on ServiceProv \(^5\) and showcasing our implementation’s parsing and type checking functionality over this.

\(^5\)https://sourceforge.net/projects/serviceprov/files/TravelPlannerCaseStudyFullInstantiatedModel/
Experimentation - Parser and Type Checking Functionality

Well-formedness of the provenance question type patterns is validated by testing whether the realised query patterns are expressed as a set of syntactically correct assertions in the form of RDF/OWL triples that are consistent with the semantic rules of the ServiceProv ontological/conceptual model. Therefore, our parser and type checker implementation performs a combination of syntax and semantic checks that are applied primarily to our provenance query patterns but also validate the well-formedness of our ServiceProv model.

The provenance query patterns are expressed in SPARQL query language where each pattern is formed as an SPARQL SELECT query (e.g. see example query 11.1). We evaluate the well-formedness of our provenance question (query) type patterns by building and implementing a set of algorithms that apply syntax and semantic checks incorporating the former Sesame framework [22], currently transformed into RDF4J [23].

Our algorithm’s syntax and semantic checks for the provenance queries against ServiceProv are summarised in the following steps:

1. Read all SPARQL queries from a folder container.
2. Parse RDF/OWL for the instantiated ServiceProv model for the Travel Planner case study to triples and add them to an OWL list (namely spolist) as triplets of subject, predicate, object values.
3. Then apply a set of syntax and semantic checks with regards to the validity of the queries.
3.1 For each triple query assertion if subject and object has a value then return false (syntax check).

---

6RDF4J a set of Java API for RDF: it provides a set of libraries that allow you to create, parse, write, store, query and reason with RDF data in a highly scalable manner.
7In RDF, each fact is called a statement. Each statement consists of three parts (for this reason, it is also often called a triple): The subject is the starting node of the statement, representing the resource that the fact is about; the predicate is the property that denotes the edge between two nodes; the object is the end node of the statement, representing the resource or literal that is the property value.
3.2 For each triple query assertion if the subject has no value and object has
value then return false (syntax check).

3.3 For each triple query assertion if the subject has value, the object is null and
the predicate is of rdf:type, search in the OWL list (list maintaining all the
triple assertions read when parsing the RDF/OWL instantiated ServiceProv
models) with the OWL triples.

3.3.1 If predicate found with name rdf:type AND subject of the triple is a Class
AND object of a triple has the same value with the object of the query
triple then the triple is valid (syntax and semantic check). If the triple is
not valid GOTO the next triple query.

3.3.2 For every triple assertion found in the OWL list, having a predicate that
matches the predicate assertions of the query triples list, then return as a
result that the triple assertion is valid (semantic check).

3.4 If predicate in the OWL list is of rdf:type AND object is a data type or
object type property:

3.4.1 Go to OWL list and check whether if for each triple assertion the subject
and object value is not null (syntax and semantic check).

3.4.2 Then check in the OWL list if you find an element in the list with the same
object (semantic check).

With the type checker algorithm we are able to define a query pattern as well-
formed by checking that this is composed of syntactically correct assertions in
the form of RDF/OWL triples and that this is consistent with the semantic rules
of the ServiceProv ontological/conceptual model by looking whether this is valid
with regards to the set of concepts (classes/properties and their relationships) for
capturing provenance for service compositions. As long as those query patterns
are syntactically and semantically valid, they could then be used as analytical
metrics for analysing provenance awareness aspects of service compositions as
discussed later in our experimentation results Section 11.3.4.
Figure 11.2: SPARQL Parser and Type Checker Implementation

The code for the parser and type checker implementation can be found online at https://sourceforge.net/p/parser-and-type-checker/code/\(^8\) while a snippet of the implementation using Eclipse Platform environment and how the validation results for the syntactic and semantic checks for one of the query runs looks like is shown in Figure 11.2.

Evaluation Results

In the previous section we have described the complete list of syntax and semantic checks applied by the parser/type checker execution. At a second step our parser/type checker implementation provides a provenance awareness analysis utility by executing the set of SPARQL queries against ServiceProv instantiated models, therefore it acts as a provenance awareness analyser where at this stage no information about the provenance infrastructure design is considered. Such execution returns a true or false boolean result for every valid query pattern that could be answered/not answered based on the current data structures of the instantiated model and the provenance query pattern.

\(^8\)The code can be checked out using https://svn.code.sf.net/p/parser-and-type-checker/code/
We have made an experiment with the parser/type checker implementation by using the Travel Planner case study. In practice we have created an instantiated provenance data model based on ServiceProv for the travel planner; this can be found at https://sourceforge.net/projects/serviceprov/files/TravelPlannerCaseStudyFullInstantiatedModel/ and then executed the set of query patterns expressed in SPARQL as listed in the Appendix 15.1. In more detail our evaluation methodology first used the implementation’s algorithm in order to apply the set of syntax and semantic checks at the provenance query patterns (provenance queries) against the provenance data structures of the travel planner instantiated model. Second the analyzer program executed each of those queries over the travel planner provenance data model instances to derive a true or false result based on whether those meet both the syntax/semantic validity and content requirement.

The results of this experiment are shown in detail in table 11.10. Using our parser/type checker implementation we have managed to showcase that all our tested queries, representative of the proposed provenance facets, and the number of question (query) type patterns acting as analytical metrics per facet, have been successfully validated with regards to the syntactic and semantic checks applied. The type checker has also been tested against invalid queries and in those cases the validator returns a FALSE statement as a result.

<table>
<thead>
<tr>
<th>Facet Category (No of Query Patterns)</th>
<th>Number of Queries Successfully Validated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service and Provider Identity(7)</td>
<td>7</td>
</tr>
<tr>
<td>Data flow (2)</td>
<td>2</td>
</tr>
<tr>
<td>Resources and Physical Deployment (2)</td>
<td>2</td>
</tr>
<tr>
<td>Time (1)</td>
<td>1</td>
</tr>
<tr>
<td>Past History Data (1)</td>
<td>1</td>
</tr>
<tr>
<td>NFPs and QoS (9)</td>
<td>9</td>
</tr>
<tr>
<td>Actors (5)</td>
<td>5</td>
</tr>
<tr>
<td>Design Information(2)</td>
<td>2</td>
</tr>
</tbody>
</table>
11.3.5 Prototype Toolset (PROVa) Evaluation

In the previous sections we described our two use case studies and their evaluation results. We use both of our case studies to provide supporting evidence with regards to defending a set of hypotheses as those were described in Section 11.3 while here we present how the issues raised by our hypotheses and exhibited in our case studies are realised as part of our prototype tool implementation. Our prototype constructively evaluates the main contribution of our thesis towards a specification and analysis framework for provenance awareness of service compositions.

As part of our evaluation methodology we have built a prototype tool implementation as a proof of concept to exhibit the genericity/reusability, computability and realism of our proposed modelling and analysis framework of provenance awareness for compositions of services as a whole (Hypothesis 1.5.1 and Hypothesis 1.5.2). Our prototype implementation further enables the validation of our proposed approach and definition of provenance awareness among other non-functional qualities as an analytical metric for composite service systems.

Evaluated Hypotheses

The key components, artefacts, architecture design and deployment of our prototype tool implementation, namely PROVa, have been extensively discussed in Chapter 10. The development of our tool implementation enables us to validate:

- Hypothesis 1.5.1 and 1.5.2 - genericity/reusability and realism of our proposed modelling and analysis framework of provenance awareness for compositions of services as a whole – considering reusability of the template provenance infrastructure metamodel, genericity of our provenance question (query) type patterns, provenance facets and ServiceProv ontological model, and realism of all those aspects with regards to SOC systems and compositions of services,

- Hypothesis 1.5.3 – the validation of our proposed approach and definition of
provenance awareness among other non-functional qualities as an analytical metric for composite service systems, and its computability as such through a specification and analysis framework implementation.

Genericity of our framework (Hypothesis 1.5.1) is validated by showcasing its use through the realisation of two use case studies in PROVa prototype tool implementation (the travel planner and the industrial (ORBI) case study). Realism (Hypothesis 1.5.3) is validated by showcasing the framework’s applicability for our industrial ORBI case study. Computability (Hypothesis 1.5.1) is showcased by using our developed toolset to support a provenance awareness analysis experiment over the provenance infrastructure design of the two different use case studies by simulating their provenance recording behaviour and reasoning over the provenance data snippets generated at system’s design time.

Next, we discuss how our two use case studies are realised through PROVa. Then results of our experimentation analysis for provenance awareness are presented in Section 11.3.5.

**Experimentation - Realisation of Use Cases through PROVa**

In this section we present the realisation of our two use case studies through PROVa toolset. This enables us to validate the full functionality of our framework for provenance awareness, including its modelling/specification and analysis utilities.

Our toolset environment is composed of different set of interfaces for the source code execution as shown in the UML class diagram for PROVa in Figure 11.3. The PROVa UML class diagram includes details on all the operations and attributes that belong to each class while a diagrammatic view of the components/activities and deployment for PROVa toolset is visualized in 10.1 and 10.11 respectively. Also, Figure 11.4 shows a PROVa execution UML sequence diagram that exhibits the operation calls and message interchange between a number of lifelines; lifelines here represent the PROVa interfaces and the classes realizing those interfaces.
High-level diagrams for the PROVa instantiated models of the Travel Planner and ORBI use case studies are shown in Figures 11.5 and 11.6, while the actual instantiated models can be found online incorporated as running examples of the PROVa source code at https://sourceforge.net/projects/prova-tool/.

Our PROVa toolset also incorporates the catalogue of query analysis patterns, expressed as SPARQL queries, an execution snippet of which is shown for PROVa in Figure 11.7.

**Travel Planner Case Study Realisation** With regards to the travel planner scenario as presented in Section 11.2.1 we used PROVa infrastructure modelling notation (Template Infrastructure Model) to model the provenance properties for the travel planner composer, namely ProvInframodel.uml as shown instantiated for the travel planner in Figure 11.5. A UML diagrammatic view of the generic provenance infrastructure metamodel we have realised for both of our case studies is exhibited in Figure 10.4. In addition, we have built
WSDL (Listing 17.1) and BPEL (Listing 17.2) specifications to describe the travel planner atomic services and orchestration logic respectively. The model artefacts are used by the core components of our prototype tool implementation as described in Figure 10.1 to bind the provenance infrastructure models to specific (atomic) service (composition) descriptions and generate the corresponding provenance data graph results (generatedProvenanceModel.ttl) that can be found online at https://sourceforge.net/projects/prova-tool/files/travel_planner_provenance_generated_model.txt.

The provenance data graph result expressed as a set of triples based on the PROV/ServiceProv schema language is then used as an input artefact by the PROVa Analyzer component that executes the list of provenance query patterns against the simulated provenance graph result. The experimentation results of this analysis for the travel planner is discussed in Section 11.3.5.
ORBI Industrial Case Study Realization  With regards to the industrial case scenario as presented in Section 11.2.2 we used PROVa infrastructure modelling notation to model the provenance properties for the ORBI orchestrator, namely ProvInframodel.uml as shown instiated for ORBI in Figure 11.6. In addition, we have built WSDL (Figure 17.1) and BPEL (Listing 17.3) specifications to describe the ORBI atomic services and orchestration logic respectively. Sim-
Similarly to the travel planner case study, those model artefacts are used by the core components of our prototype tool implementation to bind the provenance infrastructure models to specific atomic service and composition descriptions and generate the corresponding provenance data graph instance results (generated-ProvenanceModel.ttl) that can be found online at https://sourceforge.net/projects/prova-tool/files/orbi_provenance_generated_model.txt.
The provenance data graph result is then queried by the PROVa Analyzer component that executes the list of provenance query patterns against the generated/simulated provenance data graph. The experimentation results of this analysis for the industrial case scenario is discussed in Section 11.3.5.

Evaluation Results

In the previous section we have described how our two use case studies have been realised through PROVa toolset. At a second step, PROVa provides a full provenance awareness analysis functionality that allows analysis of the provenance infrastructure design support for particular services by executing SPARQL queries (representing the set of provenance awareness requirements) over the derived simulated provenance data graphs produced by such a provenance infrastructure design. The generated provenance data graphs for the travel planner case study can be found online at https://sourceforge.net/projects/prova-tool/files/travel_planner_provenance_generated_model.txt and for the ORBI case study at https://sourceforge.net/projects/prova-tool/files/orbi_provenance_generated_model.txt respectively, while the full list of the provenance query patterns executed over the provenance graphs generated for our experimentation using PROVa are listed in Section 16.1 of the Appendix.

Figure 11.7: PROVa Tool Prototype: Query Patterns
The fundamental difference of PROVa toolset compared to the parser/type checker implementation is that PROVa toolset provides a comprehensive implementation and modelling/analysis environment for provenance awareness as it first processes the templated provenance infrastructure model binding the template parameters to specific services, and then generates a simulated provenance data graph result in RDF/OWL triple format (through execution of TemplateParser.java). At a third step, it acts as a provenance awareness analyser executing query patterns over the generated provenance data graph snippets produced by the simulation. The query execution returns as answer the provenance data information that can be answered and therefore that can be captured by the particular provenance infrastructure design of the system under analysis.

Our experimentation analysis query results conducted for the travel planner and ORBI use case studies can be found online at https://sourceforge.net/projects/prova-tool/files/exeQ.log/download. A high-level overview of numbers of the results of our provenance awareness analysis experiment is shown in detail in Table 11.11. The results show the number of queries that were successfully tested for the Travel Planner and ORBI scenario (Table 11.11 - columns 2 and 3) against the full set of representative provenance question (query) type patterns per facet (Table 11.11 - column 1).

### Table 11.11: PROVa Query Experimentation Results for Use Case Studies

<table>
<thead>
<tr>
<th>Facet Category (No of Query Patterns)</th>
<th>Queries Successfully Tested - Travel Planner</th>
<th>Queries Successfully Tested - ORBI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service and Provider Identity (7)</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Data flow (2)</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Resources and Physical Deployment (4)</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Time (1)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Past History Data (1)</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>NFPs and QoS (9)</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Actors (5)</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Design Information (2)</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Practicality of Overall Approach - PROVa

With regards to the aspect of the practicality of the overall approach the different roles involved in the different stages of the provenance awareness design life cycle need to work on:

- identifying provenance awareness requirements of users in terms of provenance questions - those are collected by the domain experts through interviewing the business/system users (Requirements Phase)

- creating the WSDL and BPEL specifications that are consumed by the template binding functionality. Those are created by the application designer by using WSDL and BPEL editors to define the service signature, message sequence and orchestration workflow of the service composition execution (Modelling Phase)

- creating a provenance infrastructure templated model using Papyrus UML. This aspect requires some expertise on modelling and in particular in UML modelling background by the provenance infrastructure designers (Modelling Phase)

- turning the provenance questions into provenance queries expressed in SPARQL - this requires expertise in writing SPARQL assertions by the QoS (provenance) analysts (Model Integration and Analysis Phase)

PROVa has been tested as part of the industrial case study scenario with reference users the business analysts, application designers and infrastructure/network system designers of ORBI. In this case study application designers provided service and service composition descriptions and workflow diagrams, infrastructure designers provided information about ORBI’s infrastructure, NFPs and monitoring data, business analysts provided a list of provenance questions as a result of the requirements gathering process, while I acted as a provenance analyst turning the provenance questions into SPARQL queries and modelling the provenance infrastructure design of the overall ORBI application.
**Summary:** In this section we have presented the full list of our hypotheses with regards to the research questions tackled in this thesis. We have also presented in detail our evaluation methodology followed and the types of validation introduced in order to evaluate those hypotheses along with their evaluation results. Next, we present our conclusions and future research outlook and perspectives.
Part III

Conclusions
Chapter 12

Conclusions and Future Research

In this chapter we summarise the overall research aspects tackled in this thesis and the major contributions made towards fulfilling our research goals. We then conclude with our future perspectives and outlook towards extending our research on provenance awareness for service-based systems with regards to 1) how provenance awareness affects the design of other service non-functional properties (NFPs) such as reputation of services (looking into the provenance of feedback) and 2) what are the provenance awareness aspects that are important to explore, design and analyze for a cloud computing services architecture.

Contents

12.1 Thesis Summary ..................................................... 360
12.2 Assessment of Research Questions Against Research Contributions ..................................................... 362
12.3 Outlook and Future Perspectives ................................. 365

12.1 Thesis Summary

This thesis has addressed several aspects with regards to the specification and analysis of provenance for service-oriented (computing) systems and compositions of services by leveraging provenance awareness as an important non-functional property for service compositions that determines the provenance data infras-
structure capacity planning as a main design decision criterion for the level of accountability the services could offer with regards to a set of provenance awareness requirements.

It is important to understand that accountability of service compositions and the ability to answer questions about the history processing of service compositions – defined as provenance awareness – is not simply a quality that can be addressed by ad-hoc solutions for capturing provenance. However, we showed that provenance awareness support of a SOC system requires to be carefully designed by modelling both the structures of provenance data collected at the different phases of the SOC execution cycle (considering the inherent complexity that the aspects of service discovery, selection and orchestration incorporate) and the provenance infrastructure activity behaviour that generates the corresponding data provenance independently of specific applications. We also showed that the provenance data structures can be expressed in a queryable RDF data graph format over which we can reason to analyse the provenance awareness support of a system. The provenance awareness analysis requirements are formally modelled as a set of provenance query type patterns, expressed in SPARQL, which basically act as a set of analytical metrics for provenance awareness against the simulated provenance data measurements. In this thesis, we built a framework by providing generic concepts/models (Provenance Facets, Provenance Question Type Patterns, ServiceProv ontology, Template Infrastructure Model, Template Binding Specification), and tools (PROVa) that enable native support for provenance awareness (PROVa modelling and analysis environment) and allow service (composition) and infrastructure designers to validate whether the system provenance infrastructure design is sufficient to capture the provenance data required for satisfying a set of provenance awareness requirements. This model-driven engineering approach can further allow service designers to apply revisions and changes in the provenance infrastructure design apriori to the system’s actual implementation since a full provenance implemented solution can prove to be a rather costly business.

361
The first part of this thesis has dealt with all issues related to the formalisation of provenance awareness, describing formal models for the provenance data of service compositions and provenance faceted categories as well as template provenance data models for service compositions and the provenance infrastructure recording activity of SOC systems. The second part of this thesis has focused specifically on providing a modelling environment for analysing provenance awareness support and the evaluation of the proposed specification and analysis framework through the realisation and experimentation of two use case studies. To this end, we have developed and presented PROVa prototype tool implementation. PROVa forms a proof of concept for our proposed specification and analysis framework that enables validation of our framework’s overall modelling and analysis approach and utilities by experimenting through two different use case scenarios: the travel planner standard composition example and an industrial case study. The results of our experimentation showcase the ability of our framework to 1) model the provenance-related properties of service compositions and the infrastructure needed for supporting these, 2) verify whether the provenance infrastructure design would be sufficient to capture the required data for particular services that are dynamically bound and composed.

12.2 Assessment of Research Questions Against Research Contributions

In this section, we assess the proposed modelling and analysis methods in this thesis based on our research questions from Chapter 1.

Research Question - RQ 1 In order to express provenance awareness as a non-functional requirement for service-oriented (computing) systems and compositions of services to enable analysis of a system’s accountability support we have proposed a set of provenance question types to formally express provenance awareness requirements of users by exhibiting a number of identified semantic elemental provenance question categories. In particular, we have identified a
number of provenance facets in Chapter 6. We then contribute a set of formal models for the provenance questions in Chapter 8 using the SPARQL. Those formal models, namely provenance question type patterns are provided in a catalogue form. We use the provenance question type patterns as a basis to analyse a composite service system’s provenance awareness support. With regards to the realization of question type patterns those could alternatively be implemented as a set of inference rules using SPIN [85] rule language where we could introduce OPTIONALS into the queries. Optional matching provides a facility that enables queries to allow information to be added to the solution where the information is available without rejecting the solution in case part of the query pattern does not match; therefore if the optional part does not match it creates no bindings but does not eliminate the solution [74]. Optional parts of the graph pattern are specified syntactically with the OPTIONAL keyword applied to a graph pattern. In the context of our research the use of optional matches (OPTIONALS) would make our queries more resilient/tolerant of some elements of the provenance data graphs that a particular design may be missing but are not necessarily considered critical requirements of the provenance solution. The OPTIONAL facility would work in this context as natively basic RDF graph patterns allow applications to make queries where the entire query pattern must match for there to be a solution [74]. However, regular, complete structures cannot be assumed in all RDF graphs, and the same may hold for a provenance data graph.

We leave this implementation as future work for service and infrastructure designers for mainly two reasons: 1) at this point we wanted to exhibit completeness of the core provenance information structure that is required to be captured as part of the provenance question type patterns to answer the complete set of provenance awareness requirements for SOA/SOC-based compositions. Making all query assertions OPTIONAL will allow the provenance analyst to identify the information that could be captured as a result of the query execution, but won’t allow him to identify immediately whether the queries fail due to lack of completeness on the provenance information represented in the simulated provenance graph. Since the provenance information is formatted as ontology based
graphs and we predominately care about the completeness of assertions at schema level, any lack of connections between ontological nodes is critical to be identified. Those critical connections will enable the creation of a complete connected provenance picture in the form of a graph that analysts could query in order to get sufficient answers to their questions. 2) OPTIONALS introduce two main drawbacks, including low performance and scalability issues on the execution of SPARQL queries over large provenance data graphs (especially when those are implemented by using inference rules in a SPIN implementation); those limitations need to be considered by application and infrastructure designers and therefore OPTIONALS need to be coded in balance based on the requirements of the provenance infrastructure design.

**Research Question - RQ 2** In order to provide mechanisms for designing provenance awareness for service-based systems and compositions of services by considering the architectural aspects of the Service-Oriented Computing (SOC) paradigm and Service-oriented Architecture (SOA) we proposed 1) a provenance data model (ServiceProv ontology) in Chapter 7 for composite services; that is, a schema that allows to capture the provenance data structures relevant to the execution service-oriented systems and their compositions exhibiting the information that could be captured through the different phases of the SOC execution cycle (e.g., identifying patterns for service discovery, service selection, orchestration/choreography and planning), 2) a template provenance infrastructure metamodel in Chapter 9 which provides the required modelling notation for designing the provenance recording infrastructure activity for composite service systems, 3) a template-based binding specification in Chapter 9 that enables binding specific services and business processes (orchestrations) with the provenance infrastructure design.

**Research Question - RQ 3 and RQ 4** In order to provide a method and framework for analysing/measuring provenance awareness support of service-oriented (computing) systems and composite service designs at system’s design time against a set of provenance awareness requirements we have proposed a
provenance awareness modelling and analysis environment for compositions of services that analyse/validate whether a particular provenance infrastructure design will provide sufficient provenance information to answer a set of provenance questions (requirements). This incorporates a generation analysis algorithm that simulates the system’s provenance infrastructure behaviour by generating sample provenance graphs the infrastructure could capture. In particular, a generation analysis algorithm simulates the provenance recording life cycle of a system’s execution that the system could have at runtime by 1) analysing a design-time templated provenance data model bound in a service (composition) description and execution plan and 2) generating the provenance graph snippets of the provenance information that would be captured at execution time. To this end we have built a prototype tool implementation (PROVa) for provenance awareness, that applies provenance awareness analysis checks at design-time of whether the provenance data provided by the system (specified in the provenance data model snippets generated), is captured either by one or a composition of provenance question type patterns in our catalogue.

12.3 Outlook and Future Perspectives

The research addressed in this thesis is complete with regards to offering a framework that enables to specify and analyse the provenance awareness aspects, as presented in the service-oriented systems and service-oriented computing literature and compositions of services, apriori to a system’s implementation enabling designers to explore the impact of provenance on other service properties and take the right provenance infrastructure decisions by considering possible trade-offs. At the same time, it leaves enough room for further research with regards to analysing the trade-offs of provenance awareness with other NFPs such as the reputation or trustworthiness of services as well as additional provenance aspects that are required to be specified and analysed with regards to cloud-based services architecture. In the following sections, we provide an outlook on several aspects that remain open as future work.
Analysing Provenance Awareness Trade-Offs with other NFPs

This thesis discussed general requirements of provenance awareness for service-oriented (computing) systems realised through atomic and composite services.

As an ultimate goal that remains future perspective for this research we would like our provenance awareness specification and analysis framework to form a basis to support composite service designers in exploring the relations and accessing the impact of provenance awareness with and in relation to other NFPs. For example, we would like to enable service designers to analyse the provenance awareness for questions exhibiting different provenance facets in order to access their impact on performance and storage, due to the increased requirements for collecting and storing provenance data for a particular composite service specification. This could be accomplished by using an exploratory process where the composite service designer would apply performance and storage analysis checks by selectively enabling and disabling provenance data collection, corresponding to questions of the different provenance facets, for specific use case scenarios. According to the results of this analysis, the designer will have to decide if he is satisfied with both the provenance awareness and performance results, or else he should conduct a new performance analysis check by applying changes to the provenance-aware composite service specifications. By following recursively this methodology the designer will be able to balance the trade-offs of provenance awareness requirements with performance and storage requirements for his system.

A similar process can be followed by the designer to explore the impact on other NFPs as well. The results of this analysis will provide useful feedback to composite service designers about the trade-offs of provenance awareness with other NFPs, which they will need to consider for taking a final decision for their system’s overall design. The challenge here is that the proposed framework will be extended in such a way in order to form a basis for balancing the trade-offs
between provenance awareness and other NFPs.

**Provenance Awareness Aspects in Cloud-Based Services**

With the emergence of the Cloud Computing paradigm the functionality and capabilities of cloud-based applications are exposed by their service providers as on-demand available self-contained services. For service providers, the Cloud offers an opportunity to propose more sophisticated (and thus more valuable) services to a broader audience. Furthermore, the Cloud paradigm enables them to outsource a part of their own offer in order to increase their flexibility and cost-effectiveness. For the customer, the Cloud brings numerous advantages: digitalization of its infrastructure, so that a customer can simply get virtualized infrastructure as a service and access it remotely, being able to scale up or down its consumption based on his actual needs. The research community is already aware of the importance of provenance data collection and how this may be beneficial for the Cloud paradigm. For example, Abbadi et al. [1] focus on the problems of cloud logging and auditing for the different types of resources used in the cloud (e.g., physical, virtual, application), the building of self-managed services and on trust establishment for the cloud through the use of provenance information.

The complexity of those services increases, even more, when dynamically aggregating third party services. This composition implies the consideration of various providers’ QoS where the guarantees offered for each atomic service may differ or conflict with regards to the aggregated service’s QoS requirements. Therefore, service providers need to be accountable for their actions with respect to the actual QoS delivered against the expected one (described in the SLAs). As identified in [68], reputation management mechanisms and SLAs are tightly intertwined during the whole service lifecycle and can help in demonstrating the accountability of providers. Therefore, reputation is not a separate quality factor that requires only to be monitored (such as performance) but represents the way the different QoS properties of a service are perceived by its current and past users (what is the actual users’ feedback). We, therefore, looked into conceptual-
ising this idea by considering both objective (monitoring results) and subjective feedback (user evaluation). However the main problem here is not only the creation of feedback but for this to be processed by a reputation manager, feedback has to be authenticated to ensure its origin and identity and contextualised in order to assess its applicability, reliability and relevance. This can be summarised as the authenticity and credibility of the feedback information. These two pieces of information are obviously highly critical for the consideration of reputation in a massively distributed and entangled environment, such as the cloud-based architecture, as they are the ones that will condition the rating of services and ultimately their recommendation and selection. Yet, a proper formalisation of these mechanisms remains to be given.

In [155] we have proposed to explore the formalisation of feedback relying on the provenance of data. From our perspective, considering the provenance of data means considering the provenance of feedback. Hence we argue that the heart of authentication and credibility lies in the consideration of feedback origin, not simply by relying on some certificate or consolidated statistics, but by the consideration of the whole context of production of this data. In particular, we have proposed a rationalisation of the feedback consideration for cloud services by using provenance information to enable both the feedback’s authentication and credibility assessment. Those two tasks are highly critical in a cloud environment where everything is relying on loose coupling service interfaces between parties which are not always aware one of another.

We consider a holistic approach to the management of feedback, by considering both subjective and objective feedback. If monitoring information is relatively simple and fast to collect, it is often less considered by consumers than the evaluation done by other users (humans). As our approach considers both of them, it allows a reputation manager to provide consolidated information coming from both types of feedbacks, which in both cases considers their authentication and credibility. Besides, in the advantages of our approach we should add the fact that our proposed ontological-based design is 1) naturally extensible to include
further concepts that would give an added value to the evaluation of feedback credibility and authenticity and 2) is suitable for allowing reasoning/querying over provenance feedback instances of our ontology in order to facilitate answers for the different aspects. The added value on exhibiting the provenance of feedbacks by means of our ontological schema and querying over its instances, compared to capturing pure logging data, is that provenance gives a connected picture of the feedback’s history (in a graph form) by taking into account which activity (identity), when, where and under what circumstances the feedback was captured. As a result, this gives a strong mechanism for building trust by evaluating authentication and credibility.

In this perspective we know that our approach, even if offering the required basis to build on, will benefit by extending its horizons considering further research on the creation and proposal of credibility models. Then the question is how can we evaluate those by integrating those with our ontological feedback provenance approach or alternatively how we can feed them with provenance feedback information.

**Conclusion:** From our review of the literature as presented in Section 3.2 and to our current knowledge, no other concrete work has been proposed in the exploitation of provenance information for the management of feedback, making this research initiation a new trail to explore. Besides, as our approach can consider a large array of data, it may even be able to analyse the provenance data of anonymous feedback and extract additional information. According to its real interest, such feature could be the subject of future research.
Part IV

Appendixes
Chapter 13

Appendix A

Contents

13.1 SOA and SOC Glossary ........................................ 371

13.1 SOA and SOC Glossary

- **Aggregator**: The service aggregator performs a dual role. First, it acts as an application service provider as it offers a complete “service” solution by creating composite, higher-level services, which it provides to the service client; aggregators can accomplish this composition using specialised composition languages like BPEL and BPML. Second, the service aggregator acts as a service requester by requesting and reserving services from other service providers [120].

- **Business Process Model**: Business Process Model is an abstract process composition model expressed in a specialised composition language such as BPML [120].

- **Choreography**: Choreography is typically associated with the public (globally visible) message exchanges, rules of interaction, and agreements that occur between multiple business-process end points rather than a specific business process executed by a single party. Service choreogra-
phy is achieved via the Web Services Choreography Description Language (WS-CDL), which specifies the common observable behavior of all participants [118].

- **Execution**: The act of a service executing its operations in order to support a business task or process [120].

- **Message(s)**: Structured information exchanged between interacting services that are part of a business process [118].

- **Message Semantics**: Web services need to use meta-data to describe what other endpoints need to know to interact with them. Metadata describing a service typically contain descriptions of the interfaces of a service – the kinds of data entities expected and the names of the operations supported – such as the vendor identifier, the narrative description of the service, the Internet address for messages, the format of request and response messages. They may also contain choreographic descriptions of the order of interactions. Such descriptions may range from simple identifiers implying a mutually understood protocol to a complete description of the vocabularies, expected behaviours, and so on. Such meta-data give high-level semantic details regarding the structure and contents of the messages received and sent by Web services, message operations, concrete network protocols, and endpoint addresses used by Web services; it also describes abstractly the capabilities, requirements, and general characteristics of Web services and how they can interoperate with other services [120].

- **Non-Fuctional Properties (NFPs)**: We consider the non-functional properties to be constraints over the functionality of the service. We believe that a service the description is only complete once the non-functional aspects are also expressed [114].

- **Operation(s)/Behaviour/Capability**: Meta-data that give high-level semantic details regarding the structure and contents of the messages received and sent by Web services, message operations, concrete network
protocols, and endpoint addresses used by Web services; it also describes abstractly the capabilities, requirements, and general characteristics of Web services and how they can interoperate with other services [120].

- **Orchestration:** Orchestration describes how services interact at the message level, including the business logic and the execution order of interactions under control of a single end point. It is an executable business process that can result in a long-lived, transactional, multistep process model. With orchestration, one of the business parties involved in the process always controls the business process interactions. Orchestration is achieved via BPEL4WS and other XML-based process standard definition languages [118].

- **Orchestration Model:** The model followed by the orchestrator describing the execution plan of a service composition achieved via BPEL4WS and other XML-based process standard definition languages [118].

- **Orchestrator:** A single point that acts as the controller of an orchestration [120].

- **Publication:** The act of a service provider publishing the service descriptions of the services they provide/offer [118].

- **QoS Model:** Model that describes multiple qualities of service capabilities. As many network endpoints can implement the same service contract, the Enterprise Service Bus (ESB) should support service interactions with different values to the business. Several scenarios make it desirable for the client to select the best endpoint at run-time, rather than hard coding endpoints at build time. The Enterprise Service Bus (ESB) should, therefore, be capable of supporting various qualities of service. Clients can query a Web service, such as an organisational UDDI service, to discover the best instance with which to interact based on QoS properties. Ideally, these capabilities should be controlled by declarative policies associated with the services involved using a policy standard such as the
WS-PolicyFramework [120].

- **Reliable Messaging:** Refers to the NFP of reliable messaging and emerging standards such as WS-Reliable Messaging [120].

- **Repository:** A client discovers a service and retrieves the service description directly from the service, possibly through metadata exchange or from a repository such as UDDI [118].

- **Resource:** Resource refers to the hardware and software resources available for the execution of services and compositions of services. Resource selection for resource-intensive applications and resource allocation is a fundamental aspect for the execution of atomic services and successful service compositions [119].

- **Security Model:** Service compositions must be QoS-aware that is, understand and respect one another’s policies, performance levels, security requirements, service-level agreement (SLA) stipulations, and so on. A security model is required for a successful composition in order to enable the client to know if the services in the business process actually require WS-Security, what kind of security tokens they are capable of processing, and which ones they prefer [120].

- **Server:** Application servers are widely used to develop and deploy backend server logic. Application servers enable the separation of application (or business) logic and interface processing and also coordinate many resource connections. The most prominent features of application servers include secure transactional execution environment, load balancing, application-level clustering across multiple servers and failover management should one of these servers break down. In addition, application servers provide application connectivity and thus access to data and functions associated with EIS applications [120].

- **Service:** Software service (or simply service) are self-contained computational elements that support rapid, low-cost and easy composition of loosely
coupled distributed software applications [116].

- **Service Aggregation:** The act of an aggregator grouping services provided by other providers [118].

- **Service Broker/Discovery Agency:** Service brokers are trusted parties that force service providers to adhere to information practices that comply with privacy laws and regulations, or in the absence of such laws, industry best practices. In this way, broker-sanctioned service providers are guaranteed to offer services that are in compliance with local regulations and create a more trusted relationship with customers and partners. A service broker maintains an index of available service providers. The service broker is able to “add value” to its registry of application service providers by providing additional information about their services. This may include differences in the reliability, trustworthiness, the quality of the service, service level agreements, and possible compensation routes to name a few [120].

- **Service Client:** A service client (requestor) discovers a service (endpoint) and retrieves the service description directly from the service (metadata exchange) or from a registry or repository like UDDI [120].

- **Service Discovery:** The act of discovering a service through a service broker or by identifying a potential service description fit in a service registry/repository [118].

- **Service Description:** Service definition is the formation of a service offering, resulting in one or more service descriptions. Those descriptions are normally exposed externally to an organisation via advertisement with a service broker. Service brokers include the description(s) in a service catalogue that is searched in the provider discovery phase of the service request perspective. The resulting advertised service description can be considered a service offer [116].

- **Service Interface:** Web service interface and the WSDL standard defines the interfaces that enable a service-based application to communicate
directly with the application business logic containing information about the kinds of data entities expected and the names of the operations supported [120].

- **Service Invocation**: The act of a provider or service broker of invoking a service based on service description binding [118].

- **Service Provider**: In a typical service-based scenario, a provider hosts a network-accessible software module – an implementation of a given service – and defines a service description through which a service is published and is made discoverable [118].

- **Service Publisher**: The actor of publishing a service interface. A service registry (UDDI operator) being a specialised instance of a service broker could act as a service publisher. Under this configuration, the UDDI registry serves as a broker where the service providers publish the definitions of the services they offer using WSDL and where the service requestors find information about the services available [118].

- **Service Registry**: A client discovers a service and retrieves the service description directly from the service, possibly through metadata exchange or from a registry or repository such as UDDI [118].

- **Service Request(s)**: are messages formatted according to the Simple Object Access Protocol (SOAP). SOAP entails a light-weighted protocol allowing RPC-like calls over the Internet using a variety of transport protocols including HTTP, HTTP/S and SMTP. In principle, SOAP messages may be conveyed using any protocol as long as a binding is defined. The SOAP request is received by a run-time service (a SOAP listener) that accepts the SOAP message extracts the XML message body, transforms the XML message into a native protocol, and delegates the request to the actual business process within an enterprise [120].

- **Service Selection**: The selection of a service is usually done dynami-
cally on the basis of a set of policies, SLAs and QoS requirements conformance [119].

- **Service Selection Model:** A service selection model is the description of services selected dynamically to be aggregated where those services conform to a set of policies, SLAs and QoS requirements [119].
Chapter 14

Appendix B

Provenance Facets

Table 14.1: Provenance Facets

<table>
<thead>
<tr>
<th>Facets</th>
<th>Provenance data collection/recording required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service &amp; provider identity</td>
<td>Identification information of service providers &amp; services (e.g., IP address, port number, service name, URI)</td>
</tr>
<tr>
<td>Data flow</td>
<td>Data parameters passed in the input &amp; output messages exchanged between services</td>
</tr>
<tr>
<td>Resources</td>
<td>Availability of resources and resource usage data during execution</td>
</tr>
<tr>
<td>Time</td>
<td>Timestamps for service discovery, execution, invocation and critical events</td>
</tr>
<tr>
<td>Past history data</td>
<td>Logs of info for a long time interval or for the whole life of the service (e.g., failure rates for reliability)</td>
</tr>
<tr>
<td>NFPs and QoS</td>
<td>Logs for different NFPs (performance, SLAs) for atomic/composite services</td>
</tr>
<tr>
<td>Routes not followed</td>
<td>Capture branches taken during the composition execution plan</td>
</tr>
<tr>
<td>Actors</td>
<td>Identification information of third parties, their association with services they manage or interact with</td>
</tr>
<tr>
<td>Design information</td>
<td>No need for provenance recording</td>
</tr>
</tbody>
</table>

14.1 ServiceProv Ontology Documentation

Introduction

This section is a detailed description of the ServiceProv Ontology concepts and their properties. The ServiceProv ontology can be obtained on-line at https://sourceforge.net/projects/serviceprov/files/ServiceProvOntology/.
The actual namespace IRI used for all types defined in the ServiceProv Ontology is https://sourceforge.net/projects/serviceprov/files/serviceprov with the prefix “serviceprov”.

In this document we present the term definitions of the class concepts and their associated properties for our proposed ontology-based service provenance model.

Concepts and Properties

Activities

“An activity is something that occurs over a period of time and acts upon or with entities; it may include consuming, processing, transforming, modifying, relocating, using, or generating entities” [57]. We extend the prov:Activity class with a number of subtype concepts in Tables 14.2 and 14.3, with the latter inheriting properties such as prov:used, prov:wasGeneratedBy, prov:wasAssociatedWith. We are particularly interested in conceptualizing service related types of activities such as ServiceExecution, OrchestratorExecution, ServiceInterfacePublishing, ServicePlanning, ServiceSelection etc., in order to represent the provenance data required to answer provenance questions about them.

Entities

“An entity is a physical conceptual, or other kind of thing with some fixed aspects.” [57]. In our proposed provenance data model we extend the prov:Entity class with a number of concepts as those are defined in Tables 14.5 and 14.6. An entity may form a prov:Collection of other entities that provides a structure of some constituent, called members. Therefore, we say that a prov:Entity prov:hadMember some other prov:Entity. In our provenance data model we introduce a number of concepts as types of prov:Collection to represent the terms of Message, Service Collection or Resource Collection.
Agents

“An agent is something that bears some form of responsibility for an activity taking place, for the existence of an entity, or for another agent’s activity.” [57]. In our proposed provenance model we extend the class prov:Agent with the subclass concept of Server as described in Table 14.4. The latter forms a subtype of prov:SoftwareAgent, bearing some form of responsibility for running software, that may play a number of roles in a qualified association with a type of prov:Activity. We define those roles, such as those of the ServiceProvider or the ServiceBroker, as shown in Table 14.7.

Roles

“A role is the function of an entity or agent with respect to an activity, in the context of a usage, generation, invalidation, association, start, and end” [57]. We extend the prov:Role class with the following types, shown in Table 14.7.
Table 14.2: Service Provenance Model Activity Type Concepts (I)

<table>
<thead>
<tr>
<th>Class Definition</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>serviceprov:ServiceExecution</strong>: is-a prov:Activity that captures the actual execution of a service. <strong>serviceprov:ServiceExecution</strong> - it can be of type <strong>serviceprov: AtomicServiceExecution</strong> or <strong>serviceprov:OrchestratorExecution</strong>.</td>
<td><strong>prov: used</strong>&lt;br&gt;<strong>prov: wasGeneratedBy</strong>&lt;br&gt;<strong>prov: startedAtTime</strong>&lt;br&gt;<strong>prov: endedAtTime</strong>&lt;br&gt;<strong>prov: wasAssociatedWith</strong></td>
</tr>
<tr>
<td><strong>serviceprov: AtomicServiceExecution</strong>: is-a <strong>serviceprov:ServiceExecution</strong> that captures the actual execution of an atomic service.</td>
<td></td>
</tr>
<tr>
<td><strong>serviceprov:OrchestratorExecution</strong>: is-a <strong>serviceprov:ServiceExecution</strong> that captures the actual execution of an orchestrator. It represents the execution of a central process which takes control over the involved web services in a business process and coordinates the exchange of input and output messages of each atomic service execution in this process. A <strong>serviceprov:OrchestratorExecution</strong> <strong>prov: used</strong> some <strong>serviceprov: Workflow</strong> or some other output <strong>serviceprov:Message</strong>. Then an input <strong>serviceprov:Message</strong> <strong>prov: wasGeneratedBy</strong> the <strong>serviceprov:OrchestratorExecution</strong>.</td>
<td></td>
</tr>
<tr>
<td><strong>serviceprov:ServiceInterfacePublishing</strong>: is-a prov:Activity that represents service provider’s publishing of <strong>serviceprov:ServiceDescriptionPublishedItem(s)</strong> (including their functional and non-functional description or related policies) to a <strong>serviceprov:Server</strong> that plays the role of the <strong>serviceprov:Registry</strong>.</td>
<td><strong>prov: used</strong>&lt;br&gt;<strong>prov: wasGeneratedBy</strong>&lt;br&gt;<strong>prov: startedAtTime</strong>&lt;br&gt;<strong>prov: endedAtTime</strong>&lt;br&gt;<strong>prov: wasAssociatedWith</strong></td>
</tr>
<tr>
<td><strong>serviceprov:ServiceDiscovery</strong>: is-a prov:Activity that represents the discovery of a collection (<strong>serviceprov:ServiceCollection</strong>) of service descriptions (<strong>serviceprov:ServiceDescription</strong>) that meet certain functional criteria with the goal to find an appropriate web service to satisfy the requester’s needs. (<strong>serviceprov:ServiceDiscovery</strong> <strong>prov: used</strong> <strong>serviceprov:ServiceRequest</strong>). A discovery service facilitates the process of performing discovery. It is a logical role performed by a <strong>serviceprov:Server</strong> that plays the role of the <strong>serviceprov:ServiceBroker</strong>.</td>
<td><strong>prov: used</strong>&lt;br&gt;<strong>prov: wasGeneratedBy</strong>&lt;br&gt;<strong>prov: startedAtTime</strong>&lt;br&gt;<strong>prov: endedAtTime</strong>&lt;br&gt;<strong>prov: wasAssociatedWith</strong></td>
</tr>
<tr>
<td><strong>serviceprov:Planning</strong>: is-a prov:Activity that represents the actual plan to be executed. In service-based systems it corresponds to the procedure where a composer/aggregator performs the mapping between a user request and a set of user requirements to form an abstract process plan considering mainly the functional aspects of services.</td>
<td><strong>prov: used</strong>&lt;br&gt;<strong>prov: wasGeneratedBy</strong>&lt;br&gt;<strong>prov: startedAtTime</strong>&lt;br&gt;<strong>prov: endedAtTime</strong>&lt;br&gt;<strong>prov: wasAssociatedWith</strong></td>
</tr>
</tbody>
</table>
### Table 14.3: Service Provenance Model Activity Type Concepts (II)

<table>
<thead>
<tr>
<th>Class Definition</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>**serviceprov:**Planning prov:generated a **serviceprov:**ServiceRequest based on the use of the **serviceprov:**AbstractProcessPlan to enable the service’s discovery. This is different from the **serviceprov:**ServiceSelection activity, where the QoS requirements are taken into consideration. **serviceprov:**Planning enables the mapping of the discovered and selected services in order to generate a **serviceprov:**WorkPlan that will be prov:used by the **serviceprov:**OrchestratorExecution.</td>
<td>prov:used prov:wasGeneratedBy prov:startedAtTime prov:endedAtTime prov:wasAssociatedWith</td>
</tr>
<tr>
<td>**serviceprov:**ServiceSelection: is-a prov:Activity that represents the selection of functionally equivalent service candidates, discovered based on the user’s **serviceprov:**QoS requirements. **serviceprov:**ServiceSelection prov:used **serviceprov:**ServiceCollection and prov:generated a **serviceprov:**SelectedServiceDescription.</td>
<td>prov:used prov:wasGeneratedBy prov:startedAtTime prov:endedAtTime prov:wasAssociatedWith</td>
</tr>
</tbody>
</table>

### Table 14.4: Service Provenance Model Agent Type Concepts

<table>
<thead>
<tr>
<th>Class Definition</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>**serviceprov:**Server: is-a prov:SoftwareAgent that we consider as a piece of running software rather than just a machine: web services are typically executed by an application-server; which is what we actually capture with this term. It may be associated with the execution of a number of service related activities such as: **serviceprov:**ServiceExecution, **serviceprov:**OrchestratorExecution, **serviceprov:**ServiceSelection, **serviceprov:**ServiceDiscovery, **serviceprov:**ServicePlanning as those are defined in Tables 14.2 and 14.3. For each of them **serviceprov:**Server played different types of roles as those are defined in Table 14.7. **serviceprov:**Server can be identified by its data properties **serviceprov:**hadIPaddress and **serviceprov:**hadIPPort while it inherits the properties of **serviceprov:**SoftwareAgent.</td>
<td>prov:wasAssociatedWith **serviceprov:**hadIPaddress **serviceprov:**hadIPPort prov:hadRole</td>
</tr>
<tr>
<td>Class Definition</td>
<td>Properties</td>
</tr>
<tr>
<td>------------------</td>
<td>------------</td>
</tr>
<tr>
<td><code>serviceprov:Message</code>: is-a <code>prov:Collection</code> of <code>serviceprov:Parameter</code>(s) used to model the input and output messages as entities that were, respectively, used and generated by a <code>serviceprov:ServiceExecution</code>.</td>
<td><code>prov:wasGeneratedBy</code> <code>prov:used</code> <code>prov:hadDerivedFrom</code> <code>prov:hadMember</code></td>
</tr>
<tr>
<td><code>serviceprov:Parameter</code>: is-a type of <code>prov:Entity</code> that represents the input or output data parameters as members of <code>serviceprov:Message</code> that were <code>prov:used</code> or <code>prov:generatedBy</code> the <code>serviceprov:ServiceExecution</code>.</td>
<td><code>prov:hadMember</code> <code>prov:value</code></td>
</tr>
<tr>
<td><code>serviceprov:ServiceCollection</code>: is-a <code>prov:Collection</code> of service descriptions that <code>prov:wasGeneratedBy</code> the <code>serviceprov:ServiceExecution</code>.</td>
<td><code>prov:wasGeneratedBy</code> <code>prov:used</code> <code>prov:hadMember</code></td>
</tr>
<tr>
<td><code>serviceprov:ServiceDescription</code>: is-a type of <code>prov:Collection</code> that represents information about service’s functional and non-functional properties; it can take a number of formats such as WSDL [107] or OWL-S [98]. We model those members with the <code>serviceprov:ServiceDescriptionItem</code> concept. <code>serviceprov:ServiceDescription</code> forms a member of the <code>serviceprov:ServiceCollection</code> which <code>prov:wasGeneratedBy</code> a <code>serviceprov:ServiceDiscovery</code> activity.</td>
<td><code>prov:wasGeneratedBy</code> <code>prov:used</code> <code>prov:hadMember</code></td>
</tr>
<tr>
<td><code>serviceprov:ServiceDescriptionItem</code>: is-a <code>prov:Entity</code> member of <code>serviceprov:ServiceDescription</code> collection. It represents possible members of a service description such as a URI or a WSDL functional description. QoS offered by the services in the form of QoS policies (e.g., WS-Security Policy) are also captured by the <code>serviceprov:ServiceDescriptionItem</code> concept.</td>
<td><code>prov:wasGeneratedBy</code> <code>prov:used</code> <code>prov:value</code></td>
</tr>
<tr>
<td><code>serviceprov:ServiceDescriptionPublishedItem</code>: is-a <code>prov:specializationOf</code> the <code>serviceprov:ServiceDescriptionItem</code> entity. It represents possible service description items published by a providers to web registries</td>
<td><code>prov:wasGeneratedBy</code> <code>prov:used</code> <code>prov:value</code> <code>prov:specializationOf</code></td>
</tr>
<tr>
<td><code>serviceprov:ServiceDescriptionDiscoveredItem</code>: is-a <code>prov:specializationOf</code> the <code>serviceprov:ServiceDescriptionItem</code> entity. It represents possible service description items discovered by the <code>serviceprov:ServiceDiscovery</code>.</td>
<td><code>prov:wasGeneratedBy</code> <code>prov:used</code> <code>prov:value</code> <code>prov:specializationOf</code></td>
</tr>
<tr>
<td><code>serviceprov:Service</code>: is-a <code>prov:Entity</code> that represents a particular service provided over a given period in the past where the period of time is denoted by the <code>prov:generatedAtTime</code> and <code>prov:invalidatedAtTime</code> properties. This may link to a number of past <code>serviceprov:ServiceExecutions</code> using <code>serviceprov:executionOf</code> property, or associate to NFP properties through <code>serviceprov:hadNFP</code> for given period of past executions.</td>
<td><code>prov:value</code> <code>prov:generatedAtTime</code> <code>prov:invalidatedAtTime</code> <code>serviceprov:executionOf</code> <code>serviceprov:hadNFP</code> <code>serviceprov:hadDescription</code> <code>serviceprov:ServiceName</code></td>
</tr>
<tr>
<td>Class Definition</td>
<td>Properties</td>
</tr>
<tr>
<td>---------------------------------------------------------------------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td><strong>serviceprov:ServiceRequest</strong>: is-a prov:Entity that represents the description</td>
<td>prov:wasGeneratedBy</td>
</tr>
<tr>
<td>of the functional requirements request prov:generatedBy the serviceprov:ServicePlanning activity. It was prov:used by the serviceprov:ServiceDiscovery to generate a collection of service descriptions to satisfy those requirements.</td>
<td>prov:used prov:value</td>
</tr>
<tr>
<td><strong>serviceprov:QoSRequirements</strong>: is-a prov:Entity representing the desirable quality of service characteristics for a service to be discovered.</td>
<td>prov:used prov:value</td>
</tr>
<tr>
<td><strong>serviceprov:SelectedServiceDescription</strong>: is-a prov:Collection that represents the selected service descriptions that were generated by the serviceprov:ServiceSelection activity based on a number of QoSRequirements.</td>
<td>prov:wasGeneratedBy prov:used prov:value</td>
</tr>
<tr>
<td><strong>serviceprov:WorkPlan</strong>: is-a prov:Plan that represents the actual plan to be executed. It is an executable business process description (e.g., WS-BPEL description) of services considering their functional and non-functional properties.</td>
<td>prov:wasGeneratedBy prov:used prov:value</td>
</tr>
<tr>
<td><strong>serviceprov:AbstractProcessPlan</strong>: is-a prov:Plan that represents an abstract process plan (e.g., BPMN description) that specifies the control flow of services considering only the functional requirements of them. This was prov:used by the serviceprov:ServicePlanning to generate a serviceprov:ServiceRequest.</td>
<td>prov:used prov:value</td>
</tr>
<tr>
<td><strong>serviceprov:NFP</strong>: represents a non-functional property value of a serviceprov:ServiceExecution such as :Performance (:Throughput, :ResponseTime, :ExecutionTime, :ProcessingTime, :RoundTripTime, :Latency, :Scalability) serviceprov:Cost, :Security, or of a serviceprov:Service, for a number of its executions in the past, such as :Reliability (:MTTF,:MTBF), :Dependability (:Availability, :Robustness, :Accuracy, :ReliableMessaging) and :Trust.</td>
<td>prov:used prov:wasGeneratedBy serviceprov:hadNFP serviceprov:NFPvalue</td>
</tr>
<tr>
<td><strong>serviceprov:ResourceCollection</strong>: is-a prov:Collection of the available serviceprov:Resources prov:used during serviceprov:ServiceExecution.</td>
<td>serviceprov:usedResource prov:used prov:hadMember serviceprov:wasGeneratedBy</td>
</tr>
<tr>
<td><strong>serviceprov:Resource</strong>: represents a member of the serviceprov:ResourceCollection made available for a ServiceExecution. It contains a number of subtypes such as CPU, Storage (Memory and Disk), NetworkAdapter, OperatingSystem, concepts being reused as part of the SEALS Computing Resource Ontology.</td>
<td>prov:value usedResource</td>
</tr>
<tr>
<td>Class Definition</td>
<td>Properties</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td><strong>serviceprov:ServiceExecutionServer</strong></td>
<td>is-a prov:Role</td>
</tr>
<tr>
<td>that captures the role played by a service-</td>
<td>serviceprov:wasRoleIn</td>
</tr>
<tr>
<td>prov:Server in a qualified association with the</td>
<td></td>
</tr>
<tr>
<td>serviceprov:ServiceExecution of a service.</td>
<td></td>
</tr>
<tr>
<td><strong>serviceprov:ServiceProvider</strong></td>
<td>is-a prov:Role</td>
</tr>
<tr>
<td>that captures the role played by a service-</td>
<td>serviceprov:wasRoleIn</td>
</tr>
<tr>
<td>prov:Server in a qualified association with the</td>
<td></td>
</tr>
<tr>
<td>serviceprov:ServiceInterfacePublishing of a service-</td>
<td>prov:wasRoleIn</td>
</tr>
<tr>
<td>prov:ServiceDescriptionPublishedItem through which a</td>
<td></td>
</tr>
<tr>
<td>service is published and made discoverable.</td>
<td></td>
</tr>
<tr>
<td><strong>serviceprov:Orchestrator</strong></td>
<td>is-a prov:Role</td>
</tr>
<tr>
<td>that captures the role played by a service-</td>
<td>serviceprov:wasRoleIn</td>
</tr>
<tr>
<td>prov:Server in a qualified association with the</td>
<td></td>
</tr>
<tr>
<td>serviceprov:OrchestratorExecution.</td>
<td></td>
</tr>
<tr>
<td><strong>serviceprov:Aggregator</strong></td>
<td>is-a prov:Role</td>
</tr>
<tr>
<td>that captures the role played by a service-</td>
<td>prov:wasRoleIn</td>
</tr>
<tr>
<td>prov:Server in a qualified association with both the</td>
<td></td>
</tr>
<tr>
<td>serviceprov:ServiceSelection and the</td>
<td></td>
</tr>
<tr>
<td>serviceprov:Planning types of prov:Activities. It</td>
<td></td>
</tr>
<tr>
<td>discovers and groups services that are provided by</td>
<td></td>
</tr>
<tr>
<td>other service providers into a distinct value added</td>
<td></td>
</tr>
<tr>
<td>service. Service aggregators may become service</td>
<td></td>
</tr>
<tr>
<td>providers by publishing the composite service</td>
<td></td>
</tr>
<tr>
<td>descriptions they create.</td>
<td></td>
</tr>
<tr>
<td><strong>serviceprov:Registry</strong></td>
<td>is-a type of prov:Role</td>
</tr>
<tr>
<td>that represents the role played by a service-</td>
<td>prov:wasRoleIn</td>
</tr>
<tr>
<td>prov:Server in a qualified attribution with a</td>
<td></td>
</tr>
<tr>
<td>serviceprov:ServiceDescriptionDiscoveredItem sent</td>
<td></td>
</tr>
<tr>
<td>in a message by the registry in response to a query</td>
<td></td>
</tr>
<tr>
<td>for that description. It maintains a number of</td>
<td></td>
</tr>
<tr>
<td>service descriptions which were made available to</td>
<td></td>
</tr>
<tr>
<td>the registry through the serviceprov:ServiceInterfacePublishing activity.</td>
<td></td>
</tr>
<tr>
<td><strong>serviceprov:ServiceBroker</strong></td>
<td>is-a type of prov:Role</td>
</tr>
<tr>
<td>that represents the role played by a service-</td>
<td>prov:wasRoleIn</td>
</tr>
<tr>
<td>prov:Server in a qualified association with the</td>
<td></td>
</tr>
<tr>
<td>serviceprov:ServiceDiscovery process. It prov:used</td>
<td></td>
</tr>
<tr>
<td>a serviceprov:ServiceRequest and prov:generated a</td>
<td></td>
</tr>
<tr>
<td>serviceprov:ServiceCollection of service</td>
<td></td>
</tr>
<tr>
<td>descriptions.</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 15

Appendix C

15.1 Queries - Travel Planner Case Study

In this section, we present the full list of provenance question (query) type patterns for the travel planner use case study expressed in SPARQL query language to validate the expressiveness of the proposed provenance query patterns in Chapter 8. The SPARQL provenance query patterns expressed for the travel planner use case are listed in the same order as per the proposed formal provenance query type patterns introduced in Chapter 8 so that there is a one-to-one match for validating each query type.
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX p1: <http://www.w3.org/ns/prov#>
PREFIX serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>

select ?serviceName ?provider ?ip ?port
where {
  ?provider serviceprov:hadPort ?port.
  ?provider serviceprov:hadIPaddress ?ip.
  ?association p1:hadRole ?role.
  ?role a serviceprov:ServiceProvider.
  ?servdescriptionPublishedItem p1:wasGeneratedBy ?servInterfacePublishing.
  ?servInterfacePublishing.
  ?servdescriptionDiscoveredExceptionItem p1:wasDerivedFrom ?servdescriptionPublishedItem.
  ?servdescription PublishedItem.
  ?servdescription p1:hadMember ?servdescriptionDiscoveredExceptionItem.
  ?service serviceprov:hadDescription ?servdescription.
  FILTER ( regex ( str (?serviceName), "flight", "i"))
  ?servCollection p1:hadMember ?servdescription.
  ?servCollection p1:wasGeneratedBy ?discovery.
  ?discovery p1:used ?servRequest.
  ?servRequest p1:wasGeneratedBy ?planning.
  ?workplan p1:wasGeneratedBy ?planning.
  ?orchestratorExecution p1:used ?workplan.
  ?orchestratorExecution p1:value ?orchestratorExecutionID.
  FILTER (regex(str(?orchestratorExecutionID), "travelPlannerNWX1UX", "i"))
}

Listing 15.1: Querying Service Provider's Identity
Listing 15.2: Querying Service Execution Identity

Listing 15.3: Querying Service Execution Server Identity
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX p1: <http://www.w3.org/ns/prov#>
PREFIX serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>

where {
?registry serviceprov:hadPort ?port .
?registry serviceprov:hadIPaddress ?ip .
?servdescriptionDiscoveredItem a serviceprov:ServiceDescriptionDiscoveredItem .
?attribution p1:hadRole ?role .
?role a serviceprov:Registry .
?servdescription p1:hadMember ?servdescriptionDiscoveredItem .
?service serviceprov:hadDescription ?servdescription .
?service serviceprov:serviceName ?serviceName .
FILTER ( regex (str(?serviceName), "easyJet_flight", "i"))
?servCollection p1:hadMember ?servdescription .
?servCollection p1:wasGeneratedBy ?discovery .
?discovery p1:used ?servRequest .
?servRequest p1:wasGeneratedBy ?planning .
?workplan p1:wasGeneratedBy ?planning .
?orchestratorExecution p1:used ?workplan .
FILTER (regex (str(?orchestratorExecutionID), "travelPlannerNWX1UX", "i"))
}

Listing 15.4: Querying Registry Identity
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX p1: <http://www.w3.org/ns/prov#>
PREFIX serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>

select ?serviceName ?broker ?ip ?port
where {
?broker serviceprov:hadPort ?port.
?broker serviceprov:hadIPAddress ?ip.
?discovery a serviceprov:ServiceDiscovery.
?association p1:hadRole ?role.
?servCollection p1:wasGeneratedBy ?discovery.
?servCollection p1:hadMember ?servdescription.
?servdescription p1:hadMember
?servdescriptionDiscoveredItem.
?service serviceprov:hadDescription ?servdescription.
?service serviceprov:serviceName ?serviceName.
FILTER (regex (str(?serviceName), "easyJet_flight","i"))
?discovery p1:used ?servRequest.
?servRequest p1:wasGeneratedBy ?planning.
?workplan p1:wasGeneratedBy ?planning.
?orchestratorExecution p1:used ?workplan.
?orchestratorExecution p1:value ?orchestratorExecutionID.
FILTER (regex (str(?orchestratorExecutionID), "travelPlannerNWX1UX","i"))
}

Listing 15.5: Querying Broker Identity
Listing 15.6: Querying Aggregator Identity

```sql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX p1: <http://www.w3.org/ns/prov#>
PREFIX serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>

select ?serviceName ?aggregator ?ip ?port
where {
  ?aggregator serviceprov:hadPort ?port.
  ?aggregator serviceprov:hadIPAddress ?ip.
  ?servSelection a serviceprov:ServiceSelection.
  ?association p1:hadRole ?role.
  ?role a serviceprov:Aggregator.
  ?servSelection p1:used ?servCollection.
  ?servCollection p1:hadMember ?servdescription.
  ?servdescription p1:hadMember ?servdescriptionDiscoveredItem.
  ?servCollection p1:hadMember ?servdescription.
  ?servSelection p1:used ?servCollection.
  ?servSelection p1:used ?servCollectionDiscoveredItem.
  ?service serviceprov:serviceName ?serviceName.
  FILTER ( regex (str(?serviceName), "easyJet_flight", "i"))
  ?service serviceprov:hadDescription ?servdescription.
  ?servdescription p1:hadMember ?servdescriptionDiscoveredItem.
  ?servCollection p1:hadMember ?servdescription.
  ?servSelection p1:used ?servCollection.
  ?selectedServiceDescription p1:wasGeneratedBy ?servSelection.
  ?selectedServiceDescription a serviceprov:SelectedServiceDescription.
  ?p1:planning p1:used ?selectedServiceDescription.
  ?p1:workplan p1:wasGeneratedBy ?planning.
  ?p1:orchestratorExecution p1:value ?orchestratorExecutionID.
  FILTER ( regex (str(?orchestratorExecutionID), "travelPlannerNWX1UX", "i"))
}
```

Listing 15.7: Querying Selected Service Identity

```sql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX p1: <http://www.w3.org/ns/prov#>
PREFIX serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>

select ?serviceName ?service ?selectedServiceDescription
where {
  ?service serviceprov:serviceName ?serviceName.
  FILTER ( regex (str(?serviceName), "flight", "i"))
  ?service serviceprov:hadDescription ?servdescription.
  ?servdescription p1:hadMember ?servdescriptionDiscoveredItem.
  ?servCollection p1:hadMember ?servdescription.
  ?servSelection p1:used ?servCollection.
  ?selectedServiceDescription p1:wasGeneratedBy ?servSelection.
  ?selectedServiceDescription a serviceprov:SelectedServiceDescription.
  ?p1:planning p1:used ?selectedServiceDescription.
  ?p1:workplan p1:wasGeneratedBy ?planning.
  ?p1:orchestratorExecution p1:value ?orchestratorExecutionID.
  FILTER ( regex (str(?orchestratorExecutionID), "travelPlannerNWX1UX", "i"))
}
```
Listing 15.8: Querying Input Message Parameters

Listing 15.9: Querying Output Message Parameters
Listing 15.10: Querying Resource Availability

Listing 15.11: Querying Resource CPU Availability
Listing 15.12: Querying Time Provenance

Listing 15.13: Querying Service Availability
Listing 15.14: Querying Service Reliability

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX p1: <http://www.w3.org/ns/prov#>
PREFIX serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>

select ?value
where {?reliability serviceprov:NFPvalue ?value.
  ?reliability a serviceprov:ReliableMessaging.
  ?service serviceprov:hadNFP ?reliability.
  ?service p1:invalidatedAtTime ?stend.
  ?service serviceprov:serviceName ?serviceName.
  FILTER ( regex (str(?serviceName), "easyJet_flight","i"))
  ?execution p1:used ?inMessage.
  ?inMessage p1:wasGeneratedBy ?orchestratorExecution.
  ?orchestratorExecution p1:value ?orchestratorExecutionID.
  FILTER ( regex(str(?orchestratorExecutionID), "travelPlannerNWX1UX", "i"))
  ?orchestratorExecution p1:startedAtTime ?otstart.
  FILTER (?ststart <= ?otend)
  FILTER (?stend >= ?otstart)
}
```

Listing 15.15: Querying Service Security

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX p1: <http://www.w3.org/ns/prov#>
PREFIX serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>

select ?value
where {?security serviceprov:NFPvalue ?value.
  ?service serviceprov:hadNFP ?security.
  ?service p1:invalidatedAtTime ?stend.
  ?service serviceprov:serviceName ?serviceName.
  FILTER ( regex (str(?serviceName), "easyJet_flight","i"))
  ?execution p1:used ?inMessage.
  ?inMessage p1:wasGeneratedBy ?orchestratorExecution.
  ?orchestratorExecution p1:value ?orchestratorExecutionID.
  FILTER ( regex(str(?orchestratorExecutionID), "travelPlannerNWX1UX", "i"))
  ?orchestratorExecution p1:startedAtTime ?otstart.
  FILTER (?ststart <= ?otend)
  FILTER (?stend >= ?otstart)
}
```
Listing 15.16: Querying Service Reputation

Listing 15.17: Querying Service Cost
Listing 15.18: Querying Service Penalty

Listing 15.19: Querying Service Execution Time
Listing 15.20: Querying Service Response Time

Listing 15.21: Querying QoS (Descriptions) Offered
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX p1: <http://www.w3.org/ns/prov#>
PREFIX serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>

select ?serviceName ?service ?servdescription
where {
  ?service serviceprov:serviceName ?serviceName.
  FILTER ( regex (str(?serviceName), "easyJet_flight", "i"))
  ?service serviceprov:hadDescription ?servdescription.
  ?servCollection p1:hadMember ?servdescription.
  ?servCollection p1:wasGeneratedBy ?discovery.
  ?discovery p1:used ?servRequest.
  ?servRequest p1:wasGeneratedBy ?planning.
  ?workplan p1:wasGeneratedBy ?planning.
  ?orchestratorExecution p1:used ?workplan.
  ?orchestratorExecution p1:value ?orchestratorExecutionID.
  FILTER (regex (str(?orchestratorExecutionID), "travelPlannerNWX1UX", "i"))
}

Listing 15.22: Querying Service Description

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX p1: <http://www.w3.org/ns/prov#>
PREFIX serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>

select ?QoSDescription ?QoSvalue
where {
  ?servSelection p1:used ?QoSDescription.
  ?QoSDescription p1:value ?QoSvalue.
  ?servSelection a serviceprov:ServiceSelection.
  ?selectedServiceDescription p1:wasGeneratedBy ?servSelection.
  ?selectedServiceDescription a serviceprov:SelectedServiceDescription.
  ?workplan p1:wasGeneratedBy ?planning.
  ?orchestratorExecution p1:used ?workplan.
  ?orchestratorExecution p1:value ?orchestratorExecutionID.
  FILTER (regex (str(?orchestratorExecutionID), "travelPlannerNWX1UX", "i"))
}

Listing 15.23: Querying QoS Requirements
15.2 Queries - Industrial Case Study

In this section we present the full list of provenance questions and provenance query patterns for the ORBI system industrial case study expressed in SPARQL query language. This list is used to validate realism of the proposed facets as presented Chapter 6 and how realistic are the proposed provenance question type (query) patterns as presented Chapter 8.

Queries Exhibiting Service and Provider Identity Provenance (facet 1)

Which server executed the request contact service? (Listing 15.24)

Which server executed the arrange/create appointment service? (Listing 15.25)

Which server executed the calendar service? (Listing 15.26)

Which server executed the RSS service? (Listing 15.27)

Which server executed the weather service? (Listing 15.28)

Which server executed the update appointment service? (Listing 15.29)

Which server executed the ItemServices service? (Listing 15.30)

Which server executed the TODO service? (Listing 15.31)

Which server executed the Collections service? (Listing 15.32)

Which server executed the create JOURNAL service? (Listing 15.33)

Which server executed the TENANTCOMPANIES service? (Listing 15.34)

Which server executed the create purchase service? (Listing 15.35)
Which server executed the create sales service? (Listing 15.36)

Which server executed the create projects service? (Listing 15.37)

Which server executed the create payments service? (Listing 15.38)

Which server executed the create order service? (Listing 15.39)

Which server executed the make payment service? (Listing 15.40)

Which was the service provider of the make payment service? (Listing 15.41)
WHERE {?server serviceprov:hadPort ?port.
?server serviceprov:hadIPAddress ?ip.
?servExecution a serviceprov:AtomicServiceExecution.
?association p1:hadRole ?role.
?role a serviceprov:ServiceExecutionServer.
?service serviceprov:serviceName ?serviceName.
FILTER ( regex (str(?serviceName), "CALENDAR", "i"))
?servExecution p1:used ?inMessage.
?inMessage p1:wasGeneratedBy ?orchestratorExecution.
?orchestratorExecution p1:value ?orchestratorExecutionID.
FILTER ( regex (str(?orchestratorExecutionID),
"app.orbi.gr/client/dispatch.php", "i")))

Listing 15.26: Querying Server Executor’s Identity - Calendar Service

WHERE {?server serviceprov:hadPort ?port.
?server serviceprov:hadIPAddress ?ip.
?servExecution a serviceprov:AtomicServiceExecution.
?association p1:hadRole ?role.
?role a serviceprov:ServiceExecutionServer.
?service serviceprov:serviceName ?serviceName.
FILTER ( regex (str(?serviceName), "RSS", "i"))
?servExecution p1:used ?inMessage.
?inMessage p1:wasGeneratedBy ?orchestratorExecution.
?orchestratorExecution p1:value ?orchestratorExecutionID.
FILTER ( regex (str(?orchestratorExecutionID),
"app.orbi.gr/client/dispatch.php", "i")))

Listing 15.27: Querying Server Executor’s Identity - RSS Service

What was the service execution ID for the create order service execution? (Listing 15.42)

Queries Exhibiting Data Flow Provenance (facet 2)

What were the parameters of the input/output messages of the request contact
service? (Listing 15.43, Listing 15.44)

What were the parameters of the input/output messages of the arrange ap-
pointment service? (Listing 15.45)

What were the parameters of the input/output messages of the calendar request
service? (Listing 15.46)
WHERE { ?server serviceprov:hadPort ?port .
?server serviceprov:hadIPAddress ?ip .
?servExecution a serviceprov:AtomicServiceExecution .
?association pl:hadRole ?role .
?service serviceprov:serviceName ?serviceName .
FILTER (regex (str (?serviceName), "WEATHER", "i"))}

Listing 15.28: Querying Server Executor’s Identity - Weather Service

WHERE { ?server serviceprov:hadPort ?port .
?server serviceprov:hadIPAddress ?ip .
?servExecution a serviceprov:AtomicServiceExecution .
?association pl:hadRole ?role .
?service serviceprov:serviceName ?serviceName .
FILTER (regex (str (?serviceName), "UPDATEAPPOINTMENTS", "i"))}

Listing 15.29: Querying Server Executor’s Identity - Update Appointment Service

What were the parameters of the input messages of the RSS service? (Listing 15.47)

What were the parameters of the input/output messages of the weather service? (Listing 15.48, Listing 15.49)

What were the parameters of the input/output messages of the update appointment service? (Listing 15.50)

What were the parameters of the input/output messages of the create Item-Service service? (Listing 15.51)
WHERE {{
  ?server serviceprov:hadPort ?port.
  ?server serviceprov:hadIPAddress ?ip.
  ?servExecution a serviceprov:AtomicServiceExecution.
  ?association pl:hadRole ?role.
  ?role a serviceprov:ServiceExecutionServer.
  ?service serviceprov:serviceName ?serviceName.
} FILTER (regex (str(?serviceName), "ITEMSERVICES", "i"))}

Listing 15.30: Querying Server Executor’s Identity - Item Service

WHERE {{
  ?server serviceprov:hadPort ?port.
  ?server serviceprov:hadIPAddress ?ip.
  ?servExecution a serviceprov:AtomicServiceExecution.
  ?association pl:hadRole ?role.
  ?role a serviceprov:ServiceExecutionServer.
  ?service serviceprov:serviceName ?serviceName.
} FILTER (regex (str(?serviceName), "TODO", "i"))}

Listing 15.31: Querying Server Executor’s Identity - TODO Service

What were the parameters of the input/output messages of the TODO service? (Listing 15.52, Listing 15.53)

What were the parameters of the input/output messages of the create Collections service? (Listing 15.54)

What were the parameters of the input/output messages of the create journal service? (Listing 15.55)

What were the parameters of the input/output messages of the create tenant companies service? (Listing 15.56)
WHERE {?server serviceprov:hadPort ?port.
?server serviceprov:hadIPAddress ?ip.
?servExecution a serviceprov:AtomicServiceExecution.
?association pl:hadRole ?role.
?role a serviceprov:ServiceExecutionServer.
?service serviceprov:serviceName ?serviceName.
FILTER ( regex (str(?serviceName), "COLLECTIONS", "i"))
?servExecution pl:used ?inMessage.
?inMessage pl:wasGeneratedBy ?orchestratorExecution.
?orchestratorExecution pl:value
?orchestratorExecutionID.
FILTER (regex(str(?orchestratorExecutionID), "app.orbi.gr/client/dispatch.php", "i")))

Listing 15.32: Querying Server Executor’s Identity - Collections Service

WHERE {?server serviceprov:hadPort ?port.
?server serviceprov:hadIPAddress ?ip.
?servExecution a serviceprov:AtomicServiceExecution.
?association pl:hadRole ?role.
?role a serviceprov:ServiceExecutionServer.
?service serviceprov:serviceName ?serviceName.
FILTER (regex(str(?serviceName), "JOURNAL", "i"))
?servExecution pl:used ?inMessage.
?inMessage pl:wasGeneratedBy ?orchestratorExecution.
?orchestratorExecution pl:value
?orchestratorExecutionID.
FILTER (regex(str(?orchestratorExecutionID), "app.orbi.gr/client/dispatch.php", "i")))

Listing 15.33: Querying Server Executor’s Identity - JOURNAL Service

What were the parameters of the input/output messages of the create purchases service? (Listing 15.57)

What were the parameters of the input/output messages of the create sales service? (Listing 15.58)

What were the parameters of the input/output messages of the create projects service? (Listing 15.59)

What were the parameters of the input/output messages of the create payments service? (Listing 15.60)
What were the parameters of the input/output messages of the create order service? (Listing 15.61)

What were the parameters of the input/output messages of the make payment service? (Listing 15.62)

Queries Exhibiting Resources Availability/Usage (facet 3)

What was the CPU available for the Viva payment service of ORBI application? (Listing 15.63)
WHERE {?server serviceprov:hadPort ?port .
?server serviceprov:hadIPAddress ?ip .
?servExecution a serviceprov:AtomicServiceExecution .
?association pl:hadRole ?role .
?role a serviceprov:ServiceExecutionServer .
?service serviceprov:serviceName ?serviceName .
FILTER ( regex (str (?serviceName), "SALES", "i"))
?servExecution pl:used ?inMessage .
?inMessage pl:wasGeneratedBy ?orchestratorExecution .
FILTER (regex (str (?orchestratorExecutionID), "app.orbi.gr/client/dispatch.php", "i"))}

Listing 15.36: Querying Server Executor’s Identity - Create Sales Service

WHERE {?server serviceprov:hadPort ?port .
?server serviceprov:hadIPAddress ?ip .
?servExecution a serviceprov:AtomicServiceExecution .
?association pl:hadRole ?role .
?role a serviceprov:ServiceExecutionServer .
?service serviceprov:serviceName ?serviceName .
FILTER (regex (str (?serviceName), "PROJECTS", "i"))
?servExecution pl:used ?inMessage .
?inMessage pl:wasGeneratedBy ?orchestratorExecution .
FILTER (regex (str (?orchestratorExecutionID), "app.orbi.gr/client/dispatch.php", "i"))}

Listing 15.37: Querying Server Executor’s Identity - Create Projects Service

What was the RAM/memory available for the Viva payment service of ORBI application? (Listing 15.64)

What was the Disk available for the Viva payment service of ORBI application? (Listing 15.65)

Queries Exhibiting Timestamps Provenance (facet 4)

When did the request contact service execution started? (Listing 15.66)

When did the create appointment service execution started? (Listing 15.67)
Listing 15.38: Querying Server Executor’s Identity - Create Payments Service

Listing 15.39: Querying Server Executor’s Identity - Create Order Service

When did the calendar service execution started? (Listing 15.68)

When did the RSS service execution started? (Listing 15.69)

When did the weather service execution started? (Listing 15.70)

When did the update appointment service execution started? (Listing 15.71)

When did the Item services execution started? (Listing 15.72)

When did the create Collections service execution started? (Listing 15.73)

When did the create journal service execution started? (Listing 15.74)
SELECT ?service ?server ?serverID
WHERE {?server p1:value ?serverID .
?servExecution a serviceprov:AtomicServiceExecution .
?association p1:hadRole ?role .
?role a serviceprov:ServiceProvider .
?service serviceprov:serviceName ?serviceName .
FILTER ( regex ( str(?serviceName), "VIVAPayment", "i"))
?servExecution p1:used ?inMessage .
?inMessage p1:wasGeneratedBy ?orchestratorExecution .
FILTER ( regex(str(?orchestratorExecutionID), "app.orbi.gr/client/dispatch.php", "i")))

Listing 15.40: Querying Server Executor’s Identity - Make Payment Service

SELECT ?service ?server ?serverID
WHERE {?server p1:value ?serverID .
?servExecution a serviceprov:AtomicServiceExecution .
?association p1:hadRole ?role .
?role a serviceprov: ServiceProvider .
?service serviceprov: serviceName ?serviceName .
FILTER ( regex (str(?serviceName), "VIVAPayment", "i"))
?servExecution p1:used ?inMessage .
?inMessage p1:wasGeneratedBy ?orchestratorExecution .
FILTER ( regex(str(?orchestratorExecutionID), "app.orbi.gr/client/dispatch.php", "i")))

Listing 15.41: Querying Service Provider’s Identity - Make Payment Service

When did the create tenant companies service execution started? (Listing 15.75)

When did the create purchases service execution started? (Listing 15.76)

When did the create sales service execution started? (Listing 15.77)

When did the create projects service execution started? (Listing 15.78)

When did the create payments service execution started? (Listing 15.79)

What was the start time service execution timestamp of the create order service? (Listing 15.80)
---

SELECT ?service ?servExecutionID
WHERE {?server serviceprov:hadPort ?port .
?server serviceprov:hadIPAddress ?ip .
?servExecution a serviceprov:AtomicServiceExecution .
?association pl:hadRole ?role .
?role a serviceprov:ServiceExecutionServer .
?service serviceprov:serviceName ?serviceName .
FILTER ( regex ( str (?serviceName), "www.orbi.gr/pay", "i") )
?servExecution pl:used ?inMessage .
?inMessage pl:wasGeneratedBy ?orchExecution .
FILTER ( regex ( str (?orchExecID), "app.orbi.gr/client/dispatch.php", "i") )}

**Listing 15.42:** Querying Service Identity - Create Order Service

---

SELECT ?serviceName ?parameterIn ?inputValue
WHERE { ?parameterIn pl:value ?inputValue .
?inMessage pl:hadMember ?parameterIn .
?service serviceprov:serviceName ?serviceName .
FILTER ( regex ( str (?serviceName), "CONTACTS", "i") )
?servexecution pl:used ?inMessage .
?inMessage pl:wasGeneratedBy ?orchExecution .
FILTER ( regex ( str (?orchExecID), "app.orbi.gr/client/dispatch.php", "i") )}

**Listing 15.43:** Querying Input Data Flow - Request Contact Service

---

What was the start time service execution timestamp of the make payment service? (Listing 15.81)

**Queries Exhibiting Past History Data Provenance (facet 5)**

What was the availability of the ORBI execution service? (Listing 15.82)

What was the reliability / failure rate of the ORBI execution service the last month? (Listing 15.83)

**Queries Exhibiting NFPs and QoS Provenance (facet 7)**

What was the execution time for the particular ORBI execution service? (Listing 15.84)
SELECT ?serviceName ?parameterOut ?outputValue
  ?outMessage p1:hadMember ?parameterOut.
  ?service serviceprov:serviceName ?serviceName.
  FILTER ( regex (str(?serviceName), "CONTACTS", "i"))
  ?outMessage p1:wasGeneratedBy ?servexecution.
  ?orchExecution p1:value ?orchExecID.
  FILTER ( regex (str(?orchExecID), "app.orbi.gr/client/dispatch.php", "i")))

Listing 15.44: Querying Output Data Flow - Request Contact Service

SELECT ?serviceName ?parameterIn ?inputValue
WHERE { ?parameterIn p1:value ?inputValue.
  ?inMessage p1:hadMember ?parameterIn.
  ?service serviceprov:serviceName ?serviceName.
  FILTER ( regex (str(?serviceName), "APPOINTMENTS", "i"))
  ?servexecution p1:used ?inMessage.
  ?inMessage p1:wasGeneratedBy ?orchExecution.
  ?orchExecution p1:value ?orchExecID.
  FILTER ( regex (str(?orchExecID), "app.orbi.gr/client/dispatch.php", "i")))

Listing 15.45: Querying Input Data Flow - Arrange Appointment Service

What was the response time of the ORBI execution service? (Listing 15.85)

What was the latency of the ORBI execution service? (Listing 15.86)

Queries Exhibiting Actors Provenance (facet 8)

Which was the broker for the make payment service? (Listing 15.87)

SELECT ?serviceName ?parameterIn ?inputValue
WHERE { ?parameterIn p1:value ?inputValue.
  ?inMessage p1:hadMember ?parameterIn.
  ?service serviceprov:serviceName ?serviceName.
  FILTER ( regex (str(?serviceName), "CALENDAR", "i"))
  ?servexecution p1:used ?inMessage.
  ?inMessage p1:wasGeneratedBy ?orchExecution.
  ?orchExecution p1:value ?orchExecID.
  FILTER ( regex (str(?orchExecID), "app.orbi.gr/client/dispatch.php", "i")))

Listing 15.46: Querying Input Data Flow - Calendar Request Service
SELECT ?serviceName ?parameterIn ?inputValue
WHERE { ?parameterIn p1:value ?inputValue.
?inMessage p1:hadMember ?parameterIn.
?service serviceprov:serviceName ?serviceName.
FILTER ( regex (str(?serviceName), "RSS", "i"))
?servexecution p1:used ?inMessage.
?inMessage p1:wasGeneratedBy ?orchExecution.
?orchExecution p1:value ?orchExecID.
FILTER ( regex (str (?orchExecID), "app.orbi.gr/client/dispatch.php", "i")))

Listing 15.47: Querying Data Flow - RSS Service

SELECT ?serviceName ?parameterIn ?inputValue
WHERE { ?parameterIn p1:value ?inputValue.
?inMessage p1:hadMember ?parameterIn.
?service serviceprov:serviceName ?serviceName.
FILTER ( regex (str(?serviceName), "WEATHER", "i"))
?servexecution p1:used ?inMessage.
?inMessage p1:wasGeneratedBy ?orchExecution.
?orchExecution p1:value ?orchExecID.
FILTER ( regex (str (?orchExecID), "app.orbi.gr/client/dispatch.php", "i")))

Listing 15.48: Querying Input Data Flow - Weather Service

Which orchestrator executed the ORBI execution payment service? (Listing 15.88)

Which was the service provider of the make Payment service? (Listing 15.89)

Note: The facet categories of design information and routes not followed have not been captured in the particular case study as those comprise design information that do not necessarily need to be collected at run-time.

SELECT ?serviceName ?parameterOut ?outputValue
?outMessage p1:hadMember ?parameterOut.
?service serviceprov:serviceName ?serviceName.
FILTER ( regex (str(?serviceName), "WEATHER", "i"))
?outMessage p1:wasGeneratedBy ?servexecution.
?servexecution p1:used ?outMessage.
?orchExecution p1:value ?orchExecID.
FILTER ( regex (str (?orchExecID), "app.orbi.gr/client/dispatch.php", "i")))

Listing 15.49: Querying Output Data Flow - Weather Service
Listing 15.50: Querying Input Data Flow - Update Appointment Service

Listing 15.51: Querying Input Data Flow - Item Service

Listing 15.52: Querying Input Data Flow - TODO Service

Listing 15.53: Querying Output Data Flow - TODO Service
SELECT ?serviceName ?parameterIn ?inputValue
WHERE { ?parameterIn p1: value ?inputValue .
?inMessage p1: hadMember ?parameterIn .
?service serviceprov: serviceName ?serviceName .
FILTER ( regex (str(?serviceName),"COLLECTIONS", "i"))
?servexecution p1: used ?inMessage .
?inMessage p1: wasGeneratedBy ?orchExecution .
FILTER ( regex (str (?orchExecID), "app.orbi.gr/client/dispatch.php", "i"))
}

Listing 15.54: Querying Input Data Flow - Collections Service

SELECT ?serviceName ?parameterIn ?inputValue
WHERE { ?parameterIn p1: value ?inputValue .
?inMessage p1: hadMember ?parameterIn .
?service serviceprov: serviceName ?serviceName .
FILTER (regex (str(?serviceName),"JOURNAL", "i"))
?servexecution p1: used ?inMessage .
?inMessage p1: wasGeneratedBy ?orchExecution .
FILTER (regex (str (?orchExecID), "app.orbi.gr/client/dispatch.php", "i"))
}

Listing 15.55: Querying Input Data Flow - JOURNAL Service

SELECT ?serviceName ?parameterIn ?inputValue
WHERE { ?parameterIn p1: value ?inputValue .
?inMessage p1: hadMember ?parameterIn .
?service serviceprov: serviceName ?serviceName .
FILTER (regex (str(?serviceName),"TENANTCOMPANIES", "i"))
?servexecution p1: used ?inMessage .
?inMessage p1: wasGeneratedBy ?orchExecution .
FILTER (regex (str (?orchExecID), "app.orbi.gr/client/dispatch.php", "i"))
}

Listing 15.56: Querying Input Data Flow - Tenant Companies Service

SELECT ?serviceName ?parameterIn ?inputValue
WHERE { ?parameterIn p1: value ?inputValue .
?inMessage p1: hadMember ?parameterIn .
?service serviceprov: serviceName ?serviceName .
FILTER (regex (str(?serviceName),"PURCHASES", "i"))
?servexecution p1: used ?inMessage .
?inMessage p1: wasGeneratedBy ?orchExecution .
FILTER (regex (str (?orchExecID), "app.orbi.gr/client/dispatch.php", "i"))
}

Listing 15.57: Querying Input Data Flow - Create Purchase Service

414
Listing 15.58: Querying Input Data Flow - Create Sales Service

```
SELECT ?serviceName ?parameterIn ?inputValue
WHERE { ?parameterIn p1:value ?inputValue .
?inMessage p1:hadMember ?parameterIn .
?service serviceprov:serviceName ?serviceName .
FILTER ( regex (str(?serviceName),"SALES", "i"))
?servexecution p1:used ?inMessage .
?inMessage p1:wasGeneratedBy ?orchExecution .
FILTER ( regex (str(?orchExecID),
"app.orbi.gr/client/dispatch.php", "i")))
```

Listing 15.59: Querying Input Data Flow - Create Projects Service

```
SELECT ?serviceName ?parameterIn ?inputValue
WHERE { ?parameterIn p1:value ?inputValue .
?inMessage p1:hadMember ?parameterIn .
?service serviceprov:serviceName ?serviceName .
FILTER ( regex (str(?serviceName),"PROJECTS", "i"))
?servexecution p1:used ?inMessage .
?inMessage p1:wasGeneratedBy ?orchExecution .
FILTER ( regex (str(?orchExecID),
"app.orbi.gr/client/dispatch.php", "i")))
```

Listing 15.60: Querying Input Data Flow - Create Payments Service

```
SELECT ?serviceName ?parameterIn ?inputValue
WHERE { ?parameterIn p1:value ?inputValue .
?inMessage p1:hadMember ?parameterIn .
?service serviceprov:serviceName ?serviceName .
FILTER ( regex (str(?serviceName),"PAYMENTS", "i"))
?servexecution p1:used ?inMessage .
?inMessage p1:wasGeneratedBy ?orchExecution .
FILTER ( regex (str(?orchExecID),
"app.orbi.gr/client/dispatch.php", "i")))
```

Listing 15.61: Querying Input Data Flow - Create Order Service

```
SELECT ?serviceName ?parameterIn ?inputValue
WHERE { ?parameterIn p1:value ?inputValue .
?inMessage p1:hadMember ?parameterIn .
?service serviceprov:serviceName ?serviceName .
FILTER ( regex (str(?serviceName),"www.orbi.gr/pay", "i"))
?servexecution p1:used ?inMessage .
?inMessage p1:wasGeneratedBy ?orchExecution .
FILTER ( regex (str(?orchExecID),
"app.orbi.gr/client/dispatch.php", "i")))
```
SELECT ?serviceName ?parameterIn ?inputValue
WHERE { ?parameterIn p1:value ?inputValue.
?inMessage pl:hadMember ?parameterIn.
?service serviceprov:serviceName ?serviceName.
FILTER ( regex (str(?serviceName), "VIVApayment", "i"))
?servexecution pl:used ?inMessage.
?inMessage pl:wasGeneratedBy ?orchExecution.
?orchExecution pl:value ?orchExecID.
FILTER ( regex (str(?orchExecID), "app.orbi.gr/client/dispatch.php", "i"))}

Listing 15.62: Querying Input Data Flow - Make Payment Service

SELECT ?serviceName ?CPUvalue
?resource a serviceprov:CPU.
?servExecution a serviceprov:AtomicServiceExecution.
?service serviceprov:serviceName ?serviceName.
FILTER (regex (str(?serviceName), "VIVApayment", "i"))}

Listing 15.63: Querying Resource Availability - CPU

SELECT ?serviceName ?RAMvalue
?resource a serviceprov:Memory.
?servExecution a serviceprov:AtomicServiceExecution.
?service serviceprov:serviceName ?serviceName.
FILTER (regex (str(?serviceName), "VIVApayment", "i"))}

Listing 15.64: Querying Resource Availability - Memory

SELECT ?serviceName ?Diskvalue
?resource a serviceprov:Disk.
?servExecution a serviceprov:AtomicServiceExecution.
?service serviceprov:serviceName ?serviceName.
FILTER (regex (str(?serviceName), "VIVApayment", "i"))}

Listing 15.65: Querying Resource Availability - Disk
Listing 15.66: Querying Timestamps - Contact Service

Listing 15.67: Querying Timestamps - Create Appointment Service

Listing 15.68: Querying Timestamps - Calendar Service

Listing 15.69: Querying Timestamps - RSS Service
SELECT ?serviceName ?ststart
WHERE {
  ?servExecution a serviceprov:AtomicServiceExecution .
  ?service serviceprov:serviceName ?serviceName .
  FILTER ( regex ( str(?serviceName) , "WEATHER" , "i" ) )
  ?servExecution p1:startedAtTime ?ststart .
  ?servExecution p1:used ?inMessage .
  ?inMessage p1:wasGeneratedBy ?orchExecution .
  FILTER ( regex ( str ( ?orchExecID ) , "app.orbi.gr/client/dispatch.php" , "i" ) )
}

Listing 15.70: Querying Timestamps - Weather Service

SELECT ?serviceName ?ststart
WHERE {
  ?servExecution a serviceprov:AtomicServiceExecution .
  ?service serviceprov:serviceName ?serviceName .
  FILTER ( regex ( str(?serviceName) , "UPDATEAPPOINTMENT" , "i" ) )
  ?servExecution p1:startedAtTime ?ststart .
  ?servExecution p1:used ?inMessage .
  ?inMessage p1:wasGeneratedBy ?orchExecution .
  FILTER ( regex ( str ( ?orchExecID ) , "app.orbi.gr/client/dispatch.php" , "i" ) )
}

Listing 15.71: Querying Timestamps - Update Appointment Service

SELECT ?serviceName ?ststart
WHERE {
  ?servExecution a serviceprov:AtomicServiceExecution .
  ?service serviceprov:serviceName ?serviceName .
  FILTER ( regex ( str(?serviceName) , "ITEMSERVICES" , "i" ) )
  ?servExecution p1:startedAtTime ?ststart .
  ?servExecution p1:used ?inMessage .
  ?inMessage p1:wasGeneratedBy ?orchExecution .
  FILTER ( regex ( str ( ?orchExecID ) , "app.orbi.gr/client/dispatch.php" , "i" ) )
}

Listing 15.72: Querying Timestamps - Item Service

SELECT ?serviceName ?ststart
WHERE {
  ?servExecution a serviceprov:AtomicServiceExecution .
  ?service serviceprov:serviceName ?serviceName .
  FILTER ( regex ( str(?serviceName) , "COLLECTIONS" , "i" ) )
  ?servExecution p1:startedAtTime ?ststart .
  ?servExecution p1:used ?inMessage .
  ?inMessage p1:wasGeneratedBy ?orchExecution .
  FILTER ( regex ( str ( ?orchExecID ) , "app.orbi.gr/client/dispatch.php" , "i" ) )
}

Listing 15.73: Querying Timestamps - Collections Service
Listing 15.74: Querying Timestamps - JOURNAL Service

Listing 15.75: Querying Timestamps - Tenant Companies Service

Listing 15.76: Querying Timestamps - Create Purchases Service

Listing 15.77: Querying Timestamps - Create Sales Service

419
```
SELECT ?serviceName ?ststart
WHERE {
?servExecution a serviceprov:AtomicServiceExecution .
?service serviceprov:serviceName ?serviceName .
FILTER ( regex ( str(?serviceName),"PROJECTS", "i") )
?servExecution p1:startedAtTime ?ststart .
?servExecution p1:used ?inMessage .
?inMessage p1:wasGeneratedBy ?orchExecution .
FILTER ( regex ( str ( ?orchExecID ),
"app.orbi.gr/client/dispatch.php", "i")))
}
```

Listing 15.78: Querying Timestamps - Create Projects Service

```
SELECT ?serviceName ?ststart
WHERE {
?servExecution a serviceprov:AtomicServiceExecution .
?service serviceprov:serviceName ?serviceName .
FILTER ( regex ( str(?serviceName),"PAYMENTS", "i") )
?servExecution p1:startedAtTime ?ststart .
?servExecution p1:used ?inMessage .
?inMessage p1:wasGeneratedBy ?orchExecution .
FILTER ( regex ( str ( ?orchExecID ),
"app.orbi.gr/client/dispatch.php", "i")))
}
```

Listing 15.79: Querying Timestamps - Create Payments Service

```
SELECT ?serviceName ?ststart
WHERE {
?servExecution a serviceprov:AtomicServiceExecution .
?service serviceprov:serviceName ?serviceName .
FILTER ( regex ( str(?serviceName),"www.orbi.gr/pay", "i") )
?servExecution p1:startedAtTime ?ststart .
?servExecution p1:used ?inMessage .
?inMessage p1:wasGeneratedBy ?orchExecution .
FILTER ( regex ( str ( ?orchExecID ),
"app.orbi.gr/client/dispatch.php", "i")))
}
```

Listing 15.80: Querying Timestamps - Create Order Service

```
SELECT ?serviceName ?ststart
WHERE {
?servExecution a serviceprov:AtomicServiceExecution .
?service serviceprov:serviceName ?serviceName .
FILTER ( regex ( str(?serviceName),"VIVApayment", "i") )
?servExecution p1:startedAtTime ?ststart .
?servExecution p1:used ?inMessage .
?inMessage p1:wasGeneratedBy ?orchExecution .
FILTER ( regex ( str ( ?orchExecID ),
"app.orbi.gr/client/dispatch.php", "i")))
}
```

Listing 15.81: Querying Timestamps - Make Payment Service
SELECT ?orchExecution ?serviceName ?value
WHERE {?availability serviceprov:NFPvalue ?value.
?availability a serviceprov:Availability.
?orchExecution serviceprov:hadNFP ?availability.
?service serviceprov:serviceName ?serviceName.
FILTER ( regex (str(?serviceName), "ORBI", "i"))
?orchExecution a serviceprov:OrchestratorExecution.
?association p1:hadRole ?role.
?role a serviceprov:Orchestrator.
?orchExecution p1:value ?orchestratorExecutionID.
FILTER ( regex(str(?orchestratorExecutionID), "app.orbi.gr/client/dispatch.php","i")))

Listing 15.82: Querying Past History Data - Availability

SELECT ?orchExecution ?serviceName ?value
WHERE {?MTTF serviceprov:NFPvalue ?value.
?MTTF a serviceprov:Reliability.
?orchExecution serviceprov:hadNFP ?MTTF.
?service serviceprov:serviceName ?serviceName.
FILTER ( regex (str(?serviceName), "ORBI", "i"))
?orchExecution a serviceprov:OrchestratorExecution.
?association p1:hadRole ?role.
?role a serviceprov:Orchestrator.
?orchExecution p1:value ?orchestratorExecutionID.
FILTER ( regex(str(?orchestratorExecutionID), "app.orbi.gr/client/dispatch.php","i")))

Listing 15.83: Querying Past History Data - Reliability/Failure Rate

SELECT ?orchExecution ?serviceName ?value
WHERE {?executionTime serviceprov:NFPvalue ?value.
?executionTime a serviceprov:ExecutionTime.
?service serviceprov:serviceName ?serviceName.
FILTER ( regex (str(?serviceName), "ORBI", "i"))
?orchExecution a serviceprov:OrchestratorExecution.
?association p1:hadRole ?role.
?role a serviceprov:Orchestrator.
?orchExecution p1:value ?orchestratorExecutionID.
FILTER ( regex(str(?orchestratorExecutionID), "app.orbi.gr/client/dispatch.php","i")))

Listing 15.84: Querying NFPs - Execution Time
SELECT ?orchExecution ?serviceName ?value
WHERE {?responseTime serviceprov:NFPvalue ?value.
?service serviceprov:serviceName ?serviceName.
FILTER ( regex (str(?serviceName), "ORBI", "i"))
?orchExecution a serviceprov:OrchestratorExecution.
?association p1:hadRole ?role.
?role a serviceprov:Orchestrator.
?orchExecution p1:value ?orchestratorExecutionID.
FILTER (regex(str(?orchestratorExecutionID), "app.orbi.gr/client/dispatch.php", "i")))

Listing 15.85: Querying NFPs - Response Time

SELECT ?orchExecution ?serviceName ?value
WHERE {?latency serviceprov:NFPvalue ?value.
?latency a serviceprov:Latency.
?orchExecution serviceprov:hadNFP ?latency.
?service serviceprov:serviceName ?serviceName.
FILTER (regex(str(?serviceName), "ORBI", "i"))
?orchExecution a serviceprov:OrchestratorExecution.
?association p1:hadRole ?role.
?role a serviceprov:Orchestrator.
?orchExecution p1:value ?orchestratorExecutionID.
FILTER (regex(str(?orchestratorExecutionID), "app.orbi.gr/client/dispatch.php", "i")))

Listing 15.86: Querying NFPs - Latency
Listing 15.87: Querying Broker’s Identity - Make Payment Service

Listing 15.88: Querying Orchestrator’s Identity - Payment Service

Listing 15.89: Querying Service Provider’s Identity - Payment Service
Chapter 16

Appendix D

16.1 Query Patterns - Prototype Tool Implementation

In this section we present the full list of provenance query type patterns that have been used for the realisation of both the travel planner and industrial (orbi) use case studies through our prototype tool implementation as presented in Section 11.3.5. The latter has been developed as a proof concept to exhibit the genericity/reusability, computability and realism of our proposed modelling and analysis framework of provenance awareness for compositions of services as a whole (Hypothesis 1.5.1 and Hypothesis 1.5.2).

```
1  prefix prov: <http://www.w3.org/ns/prov#>
2  prefix serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>
3  prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4  prefix owl: <http://www.w3.org/2002/07/owl#>
5  prefix xsd: <http://www.w3.org/2001/XMLSchema#>
6  prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
7
8  SELECT ?servexecution service ?name
9  WHERE {
12  ?service a serviceprov:Service .
13  ?service serviceprov:serviceName ?name .
14  }
```

Listing 16.1: Querying Service Execution Identity
SELECT ?serviceName ?provider ?ipAddress ?port
WHERE {
  ?provider serviceprov:hadPort ?port .
  ?provider serviceprov:hadIPaddress ?ipAddress .
  ?association prov:hadRole ?role .
  ?role rdf:type serviceprov:ServiceProvider .
  ?serviceDescriptionPublishedItem prov:wasGeneratedBy ?serviceInterfacePublishing .
  ?serviceDescriptionPublishedItem rdf:type serviceprov:ServiceDescriptionPublishedItem .
  ?serviceDescriptionDiscoveredItem prov:wasDerivedFrom ?serviceDescriptionPublishedItem .
  ?serviceDescriptionDiscoveredItem rdf:type serviceprov:ServiceDescriptionDiscoveredItem .
  ?service serviceprov:hadDescription ?servdescription .
  ?servdescription rdf:type serviceprov:ServiceDescription .
  ?serv.Collection prov:wasGeneratedBy ?discovery .
  ?discovery prov:used ?servRequest .
  ?servRequest rdf:type serviceprov:ServiceRequest .
  ?servRequest prov:wasGeneratedBy ?planning .
  ?servdescription prov:hadMember ?servdescriptionDiscoveredItem .
  ?servdescriptionDiscoveredItem rdf:type serviceprov:ServiceDescriptionDiscoveredItem .
  FILTER ( regex ( str (?servdescriptionDiscoveredItem ), "FlightBookingService", "i" ) )
}
PREFIX PROV: <http://www.w3.org/ns/prov#>
PREFIX SERVICEPROV: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>
PREFIX RDF: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX OWL: <http://www.w3.org/2002/07/owl#>
PREFIX XSD: <http://www.w3.org/2001/XMLSchema#>
PREFIX RDFS: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?serviceName ?registry ?ip ?port
WHERE
{ ?registry serviceprov: hadPort ?port .
  ?registry serviceprov: hadIPAddress ?ip .
  ?attribution rdf: type prov: Attribution .
  ?attribution prov: hadRole ?role .
  ?role rdf: type serviceprov: Registry .
  ?servdescriptionDiscoverdItem prov: wasAttributedTo ?registry .
  ?servdescriptionDiscoverdItem rdf: type serviceprov: ServiceDescriptionDiscoverdItem .
  ?servdescription prov: hadMember ?servdescriptionDiscoverdItem .
  ?service serviceprov: serviceName ?serviceName .
  ?servCollection prov: hadMember ?servdescription .
  ?servCollection prov: wasGeneratedBy ?discovery .
  ?servRequest rdf: type serviceprov: ServiceRequest .
  ?discovery prov: used ?servRequest .
  ?servRequest prov: wasGeneratedBy ?planning .
  ?workplan prov: wasGeneratedBy ?planning .
  ?orchestratorExecution prov: used ?workplan .
  BIND ( regex ( str (?orchestratorExecutionID), "orchestratorExecutionID", "i") )
}

Listing 16.3: Querying Service Registry Identity
prefix prov: <http://www.w3.org/ns/prov#>
prefix serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?serviceName ?registry ?ip ?port
WHERE
{
?registry serviceprov:hadPort ?port.
?registry serviceprov:hadIPAddress ?ip.
?attribution rdf:type prov:Attribution.
?attribution prov:hadRole ?role.
?role rdf:type serviceprov:Registry.
?servdescriptionDiscoverdItem prov:wasAttributedTo ?registry.
?servdescriptionDiscoverdItem rdf:type serviceprov:ServiceDescriptionDiscoveredItem.
?servdescription prov:hadMember ?servdescriptionDiscoverdItem.
?service serviceprov:hadDescription ?servdescription.
?servdescription rdf:type serviceprov:ServiceDescription.
?servCollection prov:hadMember ?servdescription.
?servCollection prov:wasGeneratedBy ?discovery.
?discovery prov:used ?servRequest.
?servRequest rdf:type serviceprov:ServiceRequest.
?servRequest prov:wasGeneratedBy ?planning.
?workplan prov:wasGeneratedBy ?planning.
?orchestratorExecution prov:used ?workplan.
?orchestratorExecution prov:value ?orchestratorExecutionID.
FILTER (regex(str(?orchestratorExecutionID), "orchestratorExecutionID", "i"))
}

Listing 16.4: Querying Service Broker Identity
prefix prov: <http://www.w3.org/ns/prov#>
prefix serviceprov: <https://sourceforge.net/projects/serviceprov#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

WHERE {
  ?aggregator serviceprov:hadPort ?port .
  ?aggregator serviceprov:hadIPAddress ?ip .
  ?association prov:hadRole ?role .
  ?role rdf:type serviceprov:Aggregator .
  ?servSelection prov:wasAssociatedWith ?aggregator .
  ?servSelection prov:used ?servCollection .
  ?servCollection prov:hadMember ?servdescription .
  ?servdescription rdf:type serviceprov:ServiceDescription .
  ?servdescription prov:hadMember ?servdescriptionDiscoveredItem .
  ?service serviceprov:serviceName ?serviceName .
  ?servRequest prov:wasGeneratedBy ?servSelection .
  ?servRequest rdf:type serviceprov:ServiceRequest .
  ?planning prov:used ?selectedServiceDescription .
  ?selectedServiceDescription rdf:type serviceprov:SelectedServiceDescription .
  FILTER ( regex ( str (?orchestratorExecutionID ), "orchestratorExecutionID", "i") )
}

Listing 16.5: Querying Service Aggregator Identity
prefix prov: <http://www.w3.org/ns/prov#>
prefix serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?serviceName ?service
WHERE {
  ?service serviceprov:serviceName ?serviceName .
  ?service serviceprov:hadDescription ?servdescription .
  ?servdescription rdf:type serviceprov:ServiceDescription .
  ?servdescription prov:hadMember ?servdescriptionDiscoveredItem .
  ?servCollection prov:hadMember ?servdescription .
  ?servdescriptionDiscoveredItem rdf:type serviceprov:ServiceDescriptionDiscoveredItem .
  ?servCollectionDiscoveredItem rdf:type serviceprov:ServiceDescriptionDiscoveredItem .
  ?servCollectionDiscoveredItem prov:value ?value .
  ?servSelection prov:used ?servCollection .
  ?selectedServiceDescription prov:wasGeneratedBy ?servSelection .
  ?selectedServiceDescription rdf:type serviceprov:SelectedServiceDescription .
  ?planning prov:used ?selectedServiceDescription .
  ?orchestratorExecution prov:used ?workplan .
  FILTER (regex (str(?orchestratorExecutionID), "orchestratorExecutionID", "i"))
}

Listing 16.6: Querying Service Selected Identity
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>

SELECT ?QoSDescription ?QoSvalue
WHERE {
?servSelection prov:used ?QoSDescription .
?QoSDescription prov:value ?QoSvalue .
?selectedServiceDescription prov:wasGeneratedBy ?servSelection .
?selectedServiceDescription rdf:type serviceprov:SelectedServiceDescription .
?planning prov:used ?selectedServiceDescription .
?workplan prov:wasGeneratedBy ?planning .
?orchestratorExecution prov:used ?workplan .
FILTER ( regex (str (? orchestratorExecutionID), "orchestratorExecutionID", "i"))
}

Listing 16.7: Querying QoS Requirements

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>

SELECT ?serviceName ?parameterout
WHERE {
?outMessage prov:hadMember ?parameterout .
?outMessage prov:wasGeneratedBy ?servexecution .
?service serviceprov:serviceName ?serviceName .
FILTER ( regex (str (? orchestratorExecutionID), "orchestratorExecutionID", "i"))
}

Listing 16.8: Querying Output Messages of Executed Services
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>

SELECT ?serviceName ?parameterout
WHERE {
?outMessage prov:hadMember ?parameterout.
?outMessage prov:wasGeneratedBy ?servexecution.
?service serviceprov:serviceName ?serviceName.
?orchestratorExecution prov:value ?orchestratorExecutionID.
FILTER ( regex (str(?orchestratorExecutionID), "orchestratorExecutionID", "i"))
}

Listing 16.9: Querying Input Messages of Executed Services

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>

SELECT ?service ?serviceName ?value
WHERE {
?service serviceprov:hadNFP ?availability.
?service serviceprov:serviceName ?serviceName.
?servexecution prov:used ?inMessage.
?inMessage prov:wasGeneratedBy ?orchestratorExecution.
?orchestratorExecution prov:value ?orchestratorExecutionID.
FILTER ( regex (str(?orchestratorExecutionID), "orchestratorExecutionID", "i"))
}

Listing 16.10: Querying Service Availability
Listing 16.11: Querying Service Reliability

Listing 16.12: Querying Service Security
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>

WHERE
{
	?service serviceprov:hadNFP ?reputation .
	?service serviceprov:serviceName ?serviceName .
	?servexecution prov:used ?inMessage .
	?inMessage prov:wasGeneratedBy ?orchestratorExecution .
	FILTER ( regex ( str (?orchestratorExecutionID ), "orchestratorExecutionID", "i") )
	?server serviceprov:hadPort ?port .
	?server serviceprov:hadIPAddress ?ip .
}

Listing 16.13: Querying Service Reputation
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>

?service serviceprov:hadNFP ?cost .
?service serviceprov:serviceName ?serviceName .
?servexecution prov:used ?inMessage .
?inMessage prov:wasGeneratedBy ?orchestratorExecution .
?association prov:hadRole ?role .
?server serviceprov:hadPort ?port .
?server serviceprov:hadIPAddress ?ip .
FILTER (regex (str (?orchestratorExecutionID), "orchestratorExecutionID", "i"))
}

Listing 16.14: Querying Service Cost
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>

WHERE
{
  ?service serviceprov:hadNFP ?penalty .
  ?service serviceprov:serviceName ?serviceName .
  ?servexecution prov:used ?inMessage .
  ?inMessage prov:wasGeneratedBy ?orchestratorExecution .
  FILTER ( regex ( str (?orchestratorExecutionID), "orchestratorExecutionID", "i") )
  ?association prov:hadRole ?role .
  ?server serviceprov:hadPort ?port .
  ?server serviceprov:hadIPAddress ?ip .
}

Listing 16.15: Querying Service Penalty
Listing 16.16: Querying Service Execution Time
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov/>

WHERE
{
?service serviceprov:serviceName ?serviceName.
?servexecution prov:used ?inMessage.
?inMessage prov:wasGeneratedBy ?orchestratorExecution.
?orchestratorExecution prov:value ?orchestratorExecutionID.
FILTER ( regex (str(?orchestratorExecutionID), "orchestratorExecutionID", "i"))
?servexecution prov:wasAssociatedWith ?server.
?association prov:hadRole ?role.
?server serviceprov:hadPort ?port.
?server serviceprov:hadIPAddress ?ip.
}

Listing 16.17: Querying Service Response Time
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>

SELECT ?serviceName ?ststart ?stend
WHERE
{
?service serviceprov:serviceName ?serviceName.
?servexecution prov:used ?inMessage.
?inMessage prov:wasGeneratedBy ? orchestratorExecution.
? orchestratorExecution prov:value ? orchestratorExecutionID.
FILTER ( regex ( str (? orchestratorExecutionID ), " orchestratorExecutionID", "i" ))
}

Listing 16.18: Querying Time

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>

SELECT ?serviceName ?service ?servdescriptionItem ?value
WHERE
{
?service serviceprov:serviceName ?serviceName.
?service serviceprov:hadDescription ?servdescription.
?servdescription rdf:type serviceprov:ServiceDescription.
?servdescription prov:hadMember ?servdescriptionItem.
?servdescriptionItem rdf:type serviceprov:ServiceDescriptionDiscoveredItem.
?servCollection prov:hadMember ?servdescription.
?servCollection prov:wasGeneratedBy ? discovery.
? discovery prov:used ? servRequest.
?servRequest rdf:type serviceprov:ServiceRequest.
?workplan prov:wasGeneratedBy ?planning.
? orchestratorExecution prov:used ?workplan.
? orchestratorExecution prov:value ? orchestratorExecutionID.
FILTER ( regex ( str (? orchestratorExecutionID ), " orchestratorExecutionID", "i" ))
}

Listing 16.19: Querying QoS Offered

438
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>

SELECT ?serviceName ?service ?servdescription
WHERE {
?service serviceprov:serviceName ?serviceName.
?servdescription rdf:type serviceprov:ServiceDescription.
?servCollection prov:hadMember ?servdescription.
?discovery prov:used ?servRequest.
?servRequest rdf:type serviceprov:ServiceRequest.
?workplan prov:wasGeneratedBy ?planning.
?orchestratorExecution prov:used ?workplan.
FILTER ( regex (str (?orchestratorExecutionID), "orchestratorExecutionID", "i"))
}

Listing 16.20: Querying Discovered Services’ Identity

prefix prov: <http://www.w3.org/ns/prov#>
prefix serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix seals: <http://www.seals-project.eu/ontologies/SEALSComputingResource.owl#>

SELECT ?serviceName ?resource
WHERE {
?servexecution prov:used ?inMessage.
?inMessage rdf:type serviceprov:MessageIn.
?servexecution serviceprov:serviceName ?serviceName.
?inMessage prov:wasGeneratedBy ?orchestratorExecution.
FILTER ( regex (str (?orchestratorExecutionID), "orchestratorExecutionID", "i"))
}

Listing 16.21: Querying Resource Availability
SELECT ?serviceName ?resource ?value 
WHERE 
{
?resourceCollection prov:hadMember ?resource .
?resource seals:numberOfCores ?value .
?servexecution prov:used ?inMessage .
?inMessage rdf:type serviceprov:MessageIn .
?servservice serviceprov:serviceName ?serviceName .
?inMessage prov:wasGeneratedBy ?orchestratorExecution .
FILTER ( regex ( str(?orchestratorExecutionID), "orchestratorExecutionID", "i") )
}

Listing 16.22: Querying Resource CPU
```sparql
prefix prov: <http://www.w3.org/ns/prov#>
prefix serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix seals: <http://www.seals-project.eu/ontologies/SEALSComputingResource.owl#>

SELECT ?serviceName ?resource ?value
WHERE {
?resourceCollection prov:hadMember ?resource.
?resource serviceprov:used ?resource
services seals:hadStorageSize ?value.
?servexecution prov:used ?inMessage.
?inMessage rdf:type serviceprov:MessageIn.
?inMessage prov:wasGeneratedBy ?orchestratorExecution.
?orchestratorExecutionValue serviceprov:value ?orchestratorExecutionID.
FILTER (regex (str(?orchestratorExecutionID), ".*orchestratorExecutionID", "i"))
}
```

Listing 16.23: Querying Resource Memory
prefix prov: <http://www.w3.org/ns/prov#>
prefix serviceprov: <https://sourceforge.net/projects/serviceprov/files/serviceprov#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix seals: <http://www.seals-project.eu/ontologies/SEALSComputingResource.owl#>

SELECT ?serviceName ?resource ?value
WHERE
{
?resourceCollection prov:hadMember ?resource.
?resource seals:osName ?value.
?servexecution prov:used ?inMessage.
?inMessage rdf:type serviceprov:MessageIn.
?service serviceprov:serviceName ?serviceName.
?inMessage prov:wasGeneratedBy ?orchestratorExecution.
?orchestratorExecution
rdfs:label "orchestratorExecution".
?orchestratorExecution prov:value ?orchestratorExecutionID.
FILTER ( regex (str(?orchestratorExecutionID), "orchestratorExecutionID", "i"))
}

Listing 16.24: Querying Resource Operating System
Chapter 17

Appendix E

17.1 Travel Planner WSDL Specification

```xml
<?xml version="1.0"?>
<definitions name="travelPlanner"

targetNamespace="http://travelPlanner.bpel.org/bpel/demo"

xmlns:tns="http://travelPlanner.bpel.org/bpel/demo"

xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"

xmlns="http://schemas.xmlsoap.org/wsdl/"

xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
>

<!-- TYPE DEFINITION - List of types participating in this BPEL process

The BPEL Designer will generate default request and response types
but you can define or import any XML Schema type and use them as part
of the message types. -->

<types>

<schema attributeFormDefault="unqualified"

elementFormDefault="qualified"

targetNamespace="http://travelPlanner.bpel.org/bpel/demo"

xmlns="http://www.w3.org/2001/XMLSchema">

<element name="travelPlannerRequest"/>
```
<complexType>
   <sequence>
      <element name="input" type="string"/>
   </sequence>
</complexType>

<element name="travelPlannerResponse">
   <complexType>
      <sequence>
         <element name="result" type="string"/>
      </sequence>
   </complexType>
</element>

<message name="travelPlannerRequestMessage">
   <part name="payload" element="tns:travelPlannerRequest"/>
</message>

<message name="travelPlannerResponseMessage">
   <part name="payload" element="tns:travelPlannerResponse"/>
</message>

<portType name="travelPlanner">
   <operation name="process">
      <input message="tns:travelPlannerRequestMessage" />
<output message="tns:travelPlannerResponseMessage"/>

</operation>
</portType>
<!--
PARTNER LINK TYPE DEFINITION
-->
<plnk:partnerLinkType name="travelPlanner">
  <plnk:role name="travelPlannerProvider" portType="tns:travelPlanner"/>
</plnk:partnerLinkType>
<!--
BINDING DEFINITION - Defines the message format and protocol
details for a web service.
-->
<binding name="travelPlannerBinding" type="tns:travelPlanner">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="process">
    <soap:operation
      soapAction="http://travelPlanner.bpel.org/bpel/demoprocess"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
<!--
SERVICE DEFINITION - A service groups a set of ports into
a service unit.
-->
Listing 17.1: WSDL Specification for the Travel Planner

17.2 Travel Planner BPEL Specification

---

<service name="travelPlannerService">
  <port name="travelPlannerPort" binding="tns:travelPlannerBinding">
    <soap:address location="http://localhost:8080/travelPlanner"/>
  </port>
</service>

<definitions>

Listing 17.1: WSDL Specification for the Travel Planner
myRole="travelPlannerProvider"

<!-- VARIABLES -->
<!-- List of messages and XML documents used within this BPEL process -->
<!-- Reference to the message passed as input during initiation -->

<!-- Reference to the message that will be returned to the requester -->

<!-- ORCHESTRATION LOGIC -->
<!-- Set of activities coordinating the flow of messages across the services integrated within this business process -->

<!-- Receive input from requester. Note: This maps to operation defined in travelPlanner.wsdl -->

<!-- Receive name="receiveInput" partnerLink="client"
portType="tns:travelPlanner"
operation="process" variable="input"
createInstance="yes"/-->
52  <bpel:empty name="FIX_ME-Add_Business_Locke_Here"></bpel:
53  empty>
54  <!-- Generate reply to synchronous request -->
55  <bpel:reply name="replyOutput"
56  partnerLink="client"
57  portType="tns:travelPlanner"
58  operation="process"
59  variable="output"
60  />
61  </bpel:sequence>
62  </bpel:process>

Listing 17.2: BPEL Specification for the Travel Planner

17.3 ORBI WSDL Specification

Figure 17.1: WSDL specification for ORBI Use Case Study

17.4 ORBI BPEL Specification

<!-- ORBI BPEL Process [Generated by the Eclipse BPEL Designer] -->
<!-- Date: Tue Oct 07 19:25:11 BST 2014 -->
<bpel:process name="ORBI"
  targetNamespace="http://orbi.org/"
  suppressJoinFailure="yes"
  xmlns:tns="http://ORBI.org/"
xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/ process/executable"

<!-- Import the client WSDL -->
<bpel:import namespace="http://www.example.org/CreateOrder/
location="CreateOrder.wsdl" importType="http://schemas.
xmlsoap.org/wsd1/" /></bpel:import>

<!-- PARTNERLINKS
  -->
<!-- List of services participating in this BPEL process
  -->

<!--
-->
<bpel:partnerLinks>
  <bpel:partnerLink name="client"
  partnerLinkType="tns:ORBI"
  myRole="ORBIProvider"
  partnerRole="partnerRole"/>

  <bpel:partnerLink name="ArrangeAppointment"
  partnerLinkType="tns:ArrangeAppointment" myRole="makeAppointments" initializePartnerRole="no"/>

  <bpel:partnerLink name="CreateOrder" partnerLinkType="tns:
  CreateOrder" initializePartnerRole="no" myRole="CreateOrder"
  partnerRole="CreateOrder"/>

  <bpel:partnerLink name="CalendarRequest" partnerLinkType="tns:
  CalendarService" partnerRole="calendarRequest" myRole="calendarRequest"/>

  <bpel:partnerLink name="Reporting" partnerLinkType="tns:
  Reporting" partnerRole="reporting" myRole="reporting"/>

  <bpel:partnerLink name="IssueInvoice" partnerLinkType="tns:
  IssueInvoiceCalculateDrivingTime" partnerRole="issueInvoice"
  myRole="issueInvoice"/>

  <bpel:partnerLink name="VIVAPayment" partnerLinkType="tns:
  PaymentService" myRole="paymentService" partnerRole=""
paymentService">

  <bpel:partnerLink name="MakePurchase" partnerLinkType="tns:MakePurchase" myRole="makePurchase" partnerRole="makePurchase"></bpel:partnerLink>

  <bpel:partnerLink name="MakeSales" partnerLinkType="tns:MakeSales" myRole="makeSales" partnerRole="makeSales"></bpel:partnerLink>
</bpel:partnerLinks>

<!-- VARIABLES -->

<!-- List of messages and XML documents used within this BPEL process -->

<!-- Reference to the message passed as input during initiation -->

  <bpel:variable name="ORBIRequest" messageType="tns:ORBIRequestMessage">
    <bpel:from partnerLink="client" endpointReference="myRole"></bpel:from>
  </bpel:variable>

<!-- Reference to the message that will be returned to the requester -->

  <bpel:variable name="ORBIResponse" messageType="tns:ORBIResponseMessage" />

  <bpel:variable name="ArrangeAppointmentRequest"></bpel:variable>

  <bpel:variable name="ArrangeAppointmentResponse"></bpel:variable>

  <bpel:variable name="CreateOrderRequest"></bpel:variable>

  <bpel:variable name="CreateOrderResponse"></bpel:variable>

  <bpel:variable name="CalendarRequest"></bpel:variable>

  <bpel:variable name="CalendarResponse"></bpel:variable>

  <bpel:variable name="ReportingRequest"></bpel:variable>
<bpel:variable name="ReportingResponse"></bpel:variable>
<bpel:variable name="IssueInvoiceRequest"></bpel:variable>
<bpel:variable name="IssueInvoiceResponse"></bpel:variable>
<bpel:variable name="MakePurchaseRequest"></bpel:variable>
<bpel:variable name="MakePurchaseResponse"></bpel:variable>
<bpel:variable name="MakeSalesRequest"></bpel:variable>
<bpel:variable name="MakeSalesResponse"></bpel:variable>
</bpel:variables>

<!-- ORCHESTRATION LOGIC -->
<!-- Set of activities coordinating the flow of messages across the services integrated within this business process -->

<!-- Receive input from requester. Note: This maps to operation defined in ORBI.wsdl -->
<bpel:receive name="receiveInput" partnerLink="client" portType="tns:ORBI" operation="process" variable="ORBIRequest" createInstance="yes">
<bpel:correlations>
<bpel:correlation initiate="no"></bpel:correlation>
</bpel:correlations>
</bpel:receive>

<!-- Generate reply to synchronous request -->
<bpel:assign validate="no" name="Assign"></bpel:assign>

<bpel:flow name="Flow">
<bpel:sequence name="Sequence">
<bpel:invoke name="InvokeCalendarRequestService">
</bpel:invoke>
</bpel:sequence>
</bpel:flow>
<bpel:invoke name="InvokeArrangeAppointmentService">
</bpel:invoke>

</bpel:sequence>
</bpel:flow>

<bpel:assign validate="no" name="Assign1"></bpel:assign>

<bpel:invoke name="InvokeReportingService"></bpel:invoke>

<bpel:sequence name="Sequence1">

<bpel:invoke name="InvokeCreateOrderService"></bpel:invoke>

<bpel:if name="If">
<bpel:sequence>
<bpel:invoke name="InvokeCreatePurchaseService"></bpel:invoke>
</bpel:sequence>
</bpel:if>

<bpel:assign validate="no" name="Assign2"></bpel:assign>
<bpel:invoke name="InvokeMakeSalesService"></bpel:invoke>
<bpel:invoke name="InvokeIssueInvoiceService"></bpel:invoke>
<bpel:assign validate="no" name="Assign3"></bpel:assign>
<bpel:reply name="replyOutput" partnerLink="client" portType="tns:ORBI" operation="process"/>

</bpel:sequence>
</bpel:process>

Listing 17.3: BPEL Specification for ORBI Use Case
Bibliography


456


[95] Q. H. Mahmoud, *Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI)*, (April 2005).


