PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A

Phil. Trans. R. Soc. A (2010) **368**, 4193–4208
doi:10.1098/rsta.2010.0157

The design and implementation of a workflow analysis tool

By Vasa Curcin*, Moustafa Ghanem and Yike Guo

Department of Computing, Imperial College London, 180 Queen's Gate, London SW7 2AZ, UK

Motivated by the use of scientific workflows as a user-oriented mechanism for building executable scientific data integration and analysis applications, this article introduces a framework and a set of associated methods for analysing the execution properties of scientific workflows. Our framework uses a number of formal modelling techniques to characterize the process and data behaviour of workflows and workflow components and to reason about their functional and execution properties. We use the framework to design the architecture of a customizable tool that can be used to analyse the key execution properties of scientific workflows at authoring stage. Our design is generic and can be applied to a wide variety of scientific workflow languages and systems, and is evaluated by building a prototype of the tool for the Discovery Net system. We demonstrate and discuss the utility of the framework and tool using workflows from a real-world medical informatics study.

Keywords: scientific workflows; program analysis; program verification; e-Science

1. Introduction: scientific workflows today

Over the past decade the scientific workflow paradigm has gained wide popularity as a user-oriented approach for developing large-scale and distributed scientific applications. The key advantages of the paradigm are twofold. In the broadest sense a workflow provides a visual representation of a scientific process in terms of its subtasks and the dependencies between them. This abstract representation can be used by scientists easily as a mechanism for developing, and reasoning about, the high-level application logic of their tasks. Moreover, in many cases, the workflow description itself also maps naturally into an executable program orchestrating the flow of information between subtasks executing on remotely accessible distributed resources. The technical details of how such resources are accessed can be usually hidden from the user by an underlying workflow system.

(*a*) *Scientific workflow systems*

Building on these two advantages, a large number of scientific workflow systems have been developed to support what has become to be known as user-oriented e-Science (Hey & Trefethen 2002). Examples of such systems

*Author for correspondence (vasa.curcin@imperial.ac.uk).

One contribution of 15 to a Theme Issue 'e-Science: past, present and future II'.

include Discovery Net and InforSense (Ghanem *et al.* 2008), Taverna (Hull *et al.* 2006), Triana (Taylor *et al.* 2007) and Kepler (Ludäscher *et al.* 2006).

Most scientific systems support a visual front-end that enables scientists to code their workflows interactively using a graph-based metaphor where nodes represent individual tasks (whether local processing operations or remote services) and arcs represent the data and control-flow dependencies between them. The execution semantics of nodes and arcs varies across systems. Under a control-flow dependency model, an arc strictly defines node sequencing, i.e. that the source node of the arc must execute before the destination node. Under a data-flow dependency model, the arc just defines the composition of data transformation operations represented by the nodes in the graph. In this case, the order of the composition could be changed by an optimizer based on the functional properties of the individual nodes. Semantics of other workflow graph patterns, such as branching, merging and looping, also vary from one system to another. The detailed comparison is beyond the scope of this article, and a review of execution semantics of several workflow systems can be found in Curcin & Ghanem (2008).

(*b*) *Motivation*

A key assumption underlying all scientific workflow systems is that the scientists themselves will be able to use a workflow system to develop their applications based on some practical familiarity with programming, but with little fundamental knowledge of concepts such as architectures, reusability, generality and other software engineering topics. While valid for producing simple applications, and prototypes, this assumption breaks down rapidly as users attempt to develop larger and more complex workflows with more nodes, more dependencies and using more resources. Bugs and unexpected features start to appear rapidly as with any traditional programming environment, especially as more users attempt to share, re-use and re-purpose existing workflows to more real-world scenarios (Goderis *et al.* 2009), as is performed in the myExperiment initiative (Roure *et al.* 2007) which collects and disseminates users' workflows authored in Taverna. Furthermore, in cases when users attempt to port, or re-code from scratch, a simple workflow developed on one workflow system to another platform, the exercise reveals, often too late, the effects of the many hidden assumptions, limitations and unsuitability of a particular workflow language/system for the intended task, as was demonstrated in the SIMDAT project (Ghanem *et al.* 2006).

Introducing program analysis and verification into the workflow world requires detailed understandings of the execution semantics of each workflow language, including the execution properties of nodes and arcs in the workflow graph, understanding of the functional equivalencies between workflow patterns, of data type safety and many other issues. Doing such analysis manually is difficult, and addressing these issues therefore requires building on formal methods typically used in computer science research. Addressing them from a practical perspective requires building on these formal methods to develop user-level tools to reason about the properties of both workflows and workflow systems. It is the lack of such tools that is stopping workflows' evolution from nice-to-have academic toys to production-level tools used outside the narrow circle of early adopters and workflow enthusiasts.

Our work presented in this article builds on, and extends, the formal methods developed to analyse the properties of the Discovery Net scientific workflow system. In particular it combines a theoretical formal analysis framework for the verification and profiling of the control-flow aspects of scientific workflows, a theoretical polymorphic type framework for analysing workflows based on a relational data model, and the concept of embedding. The contribution of the article is twofold. First, we argue that using formal methods to model such properties holds many benefits to both the system designer and system user. Second, we demonstrate how using such formal methods goes beyond being a theoretical exercise to form the basis of developing practical user-oriented tools that can be used by both scientific workflow system designer and system user.

(*c*) *Related work*

Formal modelling of user interaction with individual Web-based applications originated with the introduction of Web service concepts (Cardelli & Davies 1999), which included parallel and sequential service executions, and notions of failure and fallback. The ability of distributed configurations to change their structure under abnormal conditions was explored in Magee & Kramer (1996). With the advent of the workflow paradigm as a mechanism for composing service-based applications, increasing effort has gone towards formal analysis of the properties of workflows and workflow systems. An operational semantics of Taverna workflow system was developed and presented in Turi *et al.* (2007) using computational lambda calculus, providing a formal perspective on different iterations and error handling procedures present in the system. Control and data modelling of Discovery Net have both been presented (Curcin *et al.* 2009, 2010). Generic applicability of lambda calculus to workflows was investigated in Kelly *et al.* (2008). The behaviour of Kepler workflow system is described using Kahn process networks, which model the data token passing in each of its supported execution behaviours. Various researchers have also modelled the execution semantics of traditional workflow languages in π-calculus, including a segment of BPEL. Similarly, Web service choreographies were also modelled in process algebras (Brogi *et al.* 2004; Foster *et al.* 2007). Static verification in Petri net-based YAWL workflow system was described in van der Aalst & ter Hofstede (2005). Finally, the embedding paradigm used in our analysis tool for constructing pure control and pure data models for a workflow with heterogeneous segments addresses a similar issue as other configuration formalisms that need to ensure type safety between isolated fragments; the Links (Cooper *et al.* 2006) initiative looked into this matter. In our work, the embedded structures are (so far) guaranteed to be in the same language, so type composition is simplified.

2. Key properties in workflow life cycle

Before demonstrating our approach and its benefits, it is important to review the generic life cycle of scientific workflow development, refinement and usage so as to highlight where different workflow analysis methods are needed.

The workflow is initially created by a user designing an informal abstract description, defining the key tasks and interactions between them at a high level without worrying how individual tasks map to executable components. Existing workflows or workflow fragments may be considered for reuse or repurposing. Following that, the workflow authoring client is used to locate executable components that implement the individual tasks and to compose them visually using the workflow graph model and its primitives. Once this is complete, the user may submit individual fragments of the workflow for execution to the enactment engine to review their execution behaviour, which may lead to an iterative refinement process—either for modifying the functionality or improving the performance. These refinements may require changing some of the components used in the workflow and/or refining the graph structure itself. Once the workflow development is complete, it can be deployed for execution and also published in a repository so that its definition can be shared with other users. Such users can retrieve it, and re-purpose it for their own applications using a similar iterative development cycle.

There are several key properties of workflows that are relevant to this process.

— *Resource usage.* The majority of nodes in a scientific workflow are components that execute on some computational resource. Efficiently managing resources for cost and/or performance requires estimating how many times specific components will be executed. Workflow process properties such as task parallelism and sequential orderings need to be analysed before determining these estimates. Process algebras and various state machines, such as Petri nets, are formalisms that are typically used to analyse such problems.

— *Process safeness.* The branching structure of workflow graphs, combined with execution in a distributed environment, may cause deadlocks and/or livelocks during execution. Establishing guarantees that the process will, or can, terminate is another type of analysis performed with process and state formalisms.

— *Functional correctness.* Scientific workflows transform some input data into one or more outputs. Therefore, establishing whether the transformation is correct in representing some function is highly important. In the workflow world, this involves defining the functional representation of the workflow, and comparing it to its functional specification. Functional languages, such as lambda calculus, are designed to model these problems.

— *Functional equivalence/similarity.* Establishing the level of similarity between multiple workflows is essential in determining the most efficient and effective solution to a problem. In functional terms, this denotes whether, given identical data, the workflows will produce the same result. This is another area explored by functional abstractions.

— *Process equivalence/similarity.* In a similar vein, understanding whether two processes will always execute in the same way, or whether they will reach a particular point in their execution, is relevant for performing workflow optimization and reasoning about resource usage. Various flavours of process bisimilarity can be used to express different levels of process identity and similarity.

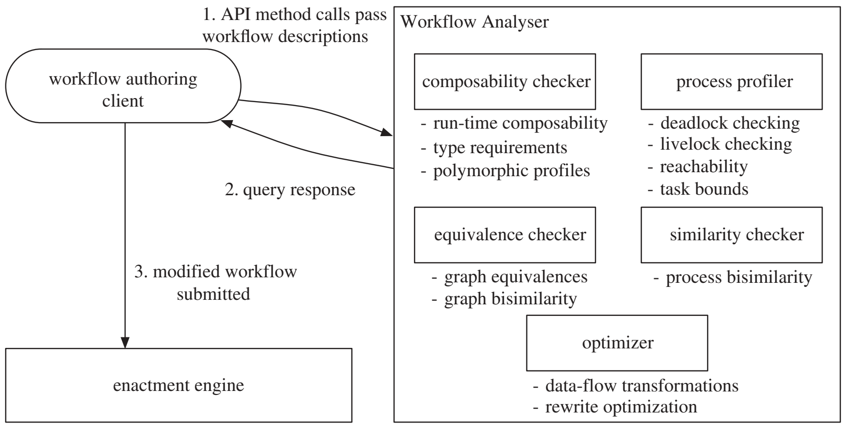


Figure 1. Usage of Workflow Analyser in workflow construction.

— *Data-type correctness.* When reusing a workflow or a workflow segment, a user will usually provide a new dataset to run the task on. In order to know whether these data will run in this workflow, there needs to be some input specification that allows this to be checked statically. Similarly, workflow output information is valuable for knowing what will be the type of the result once the execution is complete, and for connecting the component to further downstream nodes. Type checking systems have been designed to efficiently answer these questions, and they are readily applicable to workflows with type information present.

Figure 1 illustrates how a workflow analysis tool addressing these properties can be used during workflow construction to examine different aspects of the workflow. *Composability checker* performs type checking of two nodes or workflow fragments to determine whether their inputs and outputs can be connected, based on the type formulas of the atomic workflow components. *Process profiler* determines the presence of illegal conditions, such as deadlocks and livelocks, as well as the reachability of nodes in the workflow and the bounds on the number of parallel copies that can be spawned (used for monitoring resource usage). *Equivalence checker* determines if the two data flows always produce equivalent results, given the execution rules for the data flows. *Similarity checker* determines the equivalence of process executions, for some defined observable behaviour. Finally, *Optimizer* uses predefined heuristics together with rules defining allowed graph transformations to suggest improvements to the workflow.

It is important to note that the models used in the analysis tool are independent of the workflow system used, so they can establish equivalences and type comparisons between different tools, provided appropriate translators are implemented.

3. Control and data models for workflow analysis

In this section, we will briefly look into the underlying models that can be used to implement a workflow analysis tool. The workflow model is separated into two aspects: control (looking into the process execution behaviour) and data (concerned with the transformation of input data into outputs). Notation used for control-flow modelling is fully described in Curcin *et al.* (2009) and the notation used for data-flow-type transformations is presented in Curcin *et al.* (2010).

(*a*) *Control-flow model*

The formal model for the control-flow layer is based on asynchronous π-calculus which models processes, denoted by W, A, B, $C, \dots$, and communication channels between them, denoted by $m, n, o, \dots$, that are used to exchange messages. The semantics consist of a set of rules that allow transitions from one process state to another using actions: messages sent and received on the channels, and silent actions that the processes do internally. Bisimilarity is the equivalence relation that establishes that two processes have the same set of available actions in equivalent states.

In the model we developed, the full workflow process consists of a parallel composition of individual node processes, $W = A \mid B \mid C \mid \dots$. Each node process has some unidirectional input and output channels, and upon receiving a signal, the node executes and, if successful, sends signals to one or more of its downstream nodes. A simple node A receiving input on channel m, executing some internal action τ_A and sending output on channel n may take the form $A = !m(t).\tau_A.\bar{n}(t)$, where t is some content being passed. A replication operator ! is used to denote that the node will react to any number of inputs received on channel m by launching a new copy of itself each time.

Based on this process model, a state-space model was developed for reasoning about all possible execution states that a workflow may go through, and investigating reachability of individual configurations, which allows detection of deadlocks and livelocks. The model defines state as being the number of currently executing instances of each node, and as such is well suited to modelling resource usage of a workflow. Computational tree logic (CTL; Clarke *et al.* 1986) is used to reason about these states and the execution paths between them using its temporal operators, **A/E** for denoting all paths and existence of the path and **G/F** representing all/some future states on the path. For example, **AF** C_τ states that node C will execute its silent action τ at some future point in all executions. The full model was published in Curcin *et al.* (2009).

(*b*) *Data-flow model*

Data-flow modelling is concerned with establishing the correctness of data transformations within a workflow. The formal model used in the analyser tool is based on lambda calculus, with nodes within a workflow represented as functions that operate on input data to produce data outputs. Workflow fragments are then represented as function compositions by using the links in the workflow to define the substitutions of one function's result for the inputs of downstream functions. The exact nature of substitutions depends on whether the workflow system uses lazy or eager execution strategy, with the former using a *let* construct, and the latter a straightforward β-conversion.

Thus, functional equivalence between different data flows can be established by applying term reduction operations, and compiling the associated lambda terms to their normal form. Two data flows that reduce to the same normal form are guaranteed to be result-equivalent, i.e. they will always produce the same output value. When dealing with a particular workflow system, equivalences between sets of data-flow nodes need to be established only once, to obtain the sets of rewrite rules that preserve functional equality, based on term-graph transformations of workflow graphs. These rewrite rules, combined with some weighting heuristics, produce optimization strategies for workflow graphs. A typical example are early projections and early selections in workflows based on the relational data model.

In addition to the term graph view, each node and workflow fragment have some requirements of the input data type, and provide some guarantees on the output data. The type formulas representing these data constraints can be checked for composability, and to infer the type of output data, given the input. A full model and the type inference engine for relational data model and its operators are described in Curcin *et al.* (2010).

(*c*) *Embedding control and data*

Within the context of scientific workflows, the control and data aspects cannot always be cleanly separated. Data flows may have computational steps which require non-functional logic or steps that are not directly linked to the data model, such as indexing the data or making back-up copies of interim results. Control flows, on the other hand, may want to allow sharing of data between tasks in a meaningful manner, and passing some data inputs and outputs from and to an external entity: a user or an enclosing application. The workflow-analysis model uses embedding, or hierarchical composition, of data and control models as a way of combining the two aspects within a single layered structure without violating the principles of either. In such a representation, a functional node in the data flow may have an internal computation based on the control flow, and an execution node in the control flow may use the data flow to perform a computation.

Combining these multiple types of embedding results in a hierarchical structure in which both models can contain instances of the other. In practical terms, this is achieved by giving a process interpretation of the structure defined by the data flow and any embedded control flow, thereby enabling the process analysis to be performed in the same way as for the control model. For control flows, a new construct is added to the model—a shared variable space that is used for passing data between the embedded data flows and from internal data flows to external ones. This enables data-flow analysis, in which type variables are unified via the shared space. The importance of embedding approach is that it allows data modelling of the workflows which have data-ignorant steps, as well as the process modelling of workflows containing data-processing nodes with trivial control behaviour.

4. Workflow Analyser architecture

The architecture of the Workflow Analyser, depicted in figure 2, consists of five externally accessible tools, described previously, and five internal engines that perform model transformations, and reason about the properties of those models.

Initial translator engine, which acts as the embedding interpreter, separates the process and data flows to form a hierarchical structure. The embedding relationships in the resulting structure are used to construct an overall process model, and build data and type dependencies between components. This results

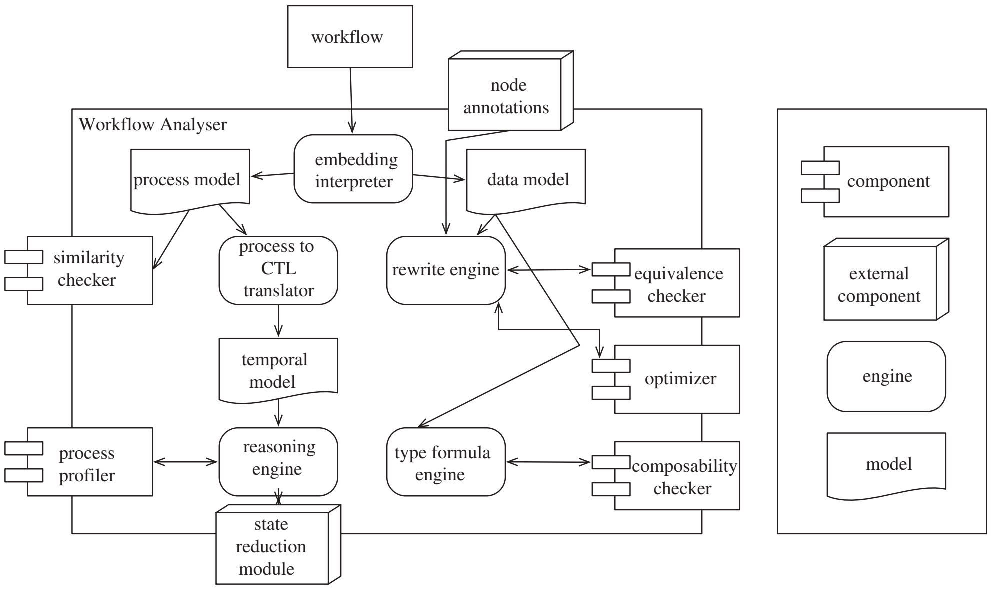


Figure 2. Architecture of Workflow Analyser.

in pure control and pure data models suitable for analysis. The former replace the data-flow segments with their process calculus representation, while the latter establish any data transformations performed by a control segment (and any data flows embedded into it) via a shared object space attached to that segment.

The process-to-CTL translator engine uses the control-flow process model, defined in π-calculus, as a basis for a finite state system based on the number of concurrently executing components and a temporal model using CTL. The model is then fed into the underlying reasoning engine which accepts CTL queries about the system. Rewrite engine uses the data-flow system graph translation rules to determine the valid set of transformations that preserve the functional meaning of the workflow. The transformations are defined for each workflow system and used for optimization and equivalence checking. Type formula engine establishes the type properties of the data flow in terms of type constraints on the inputs and the type properties of the output. Composability checker then uses it to determine the type-safety of the data-flow compositions.

The prototype of the workflow analysis tool has been implemented as a generic module that can be plugged into any workflow system to perform static analysis of the workflows. In this use case, it has been connected to the Discovery Net workflow system to perform an adverse drug reaction (ADR) study.

(a) *Discovery Net*

The Discovery Net system (Ghanem *et al.* 2008) has been designed around a scientific workflow model for integrating distributed data sources and analytical tools within a grid computing framework. Many of the research ideas developed within the system have also been incorporated within the InforSense Knowledge

Discovery Environment system (InforSense 2003), a commercial workflow management and data mining system that has been widely used for business-oriented applications. The system is based on a multi-tier architecture, with a workflow server providing a number of supporting functions needed for workflow authoring and execution, such as integration and access to remote computational and data resources, collaboration tools, visualizers and publishing mechanisms. The workflows are represented and stored using Discovery Process Markup Language (Syed *et al.* 2007), an XML-based representation language for workflow graphs supporting both a strongly typed data-flow model of computation (for analytical workflows) and a control-flow model (for orchestrating multiple disjoint workflows).

(*b*) *Prototype implementation*

The architecture described above has been prototyped through proof-of-concept tools for examining process bisimilarity and CTL checking, a draft implementation of the type inference algorithm and composability checker that has been provided for Discovery Net, and the rewrite engine and data-flow optimizer that have been assembled for its relational subset.

The process bisimilarity checks are performed by encoding input control flows into the π-calculus format understandable by the ABC tool (Briais 2007), and the checks are then executed by passing script files, containing the processes and the equivalence queries, to the tool. The CTL encoding is performed by converting the control-flow model into a Kripke frame representation used in the NuSMV tool (Cimatti *et al.* 2002). This is achieved by encoding NuSMV modules for each type of input and output behaviour, and constructing nodes as instances of those modules, with each node knowing the number of its replications present in the frame. The workflow Kripke frame is passed to the tool as an input file, initializing the state space, and allowing the execution of analytical queries, implemented as predefined CTL statements parametrized with node names and evaluated within NuSMV. The state reduction module, which prevents the state explosion, is not yet fully implemented, so each node instance is only being considered up to ten replications.

The type inference tool has been implemented in Java, by providing the object representation of the node polymorphic type constraints and a composition toolkit that performs composability checks and generates type formulas of composed nodes. To apply this to Discovery Net, the type constraints of each relational model node have been represented in a type formula, and the required data-flow typing is constructed by invoking methods on individual type formulas to produce the joint construct. The rewrite package, also written in Java, checks the data-flow graphs for the optimization patterns specified. In the case of Discovery Net, these patterns consist of moving projections and selections before union and join operations where possible. The tool points out instances of these patterns in the input data-flow graph and, if required, rearranges it by modifying the links between the nodes as specified by the rewrite rule.

Both tools perform well on workflows of up to 100 or so nodes that we have tested them on using a standard Macbook laptop. The type inference tool scales linearly with the number of nodes, while the process model requires some state reduction to be performed once more than 100 processes are being observed.

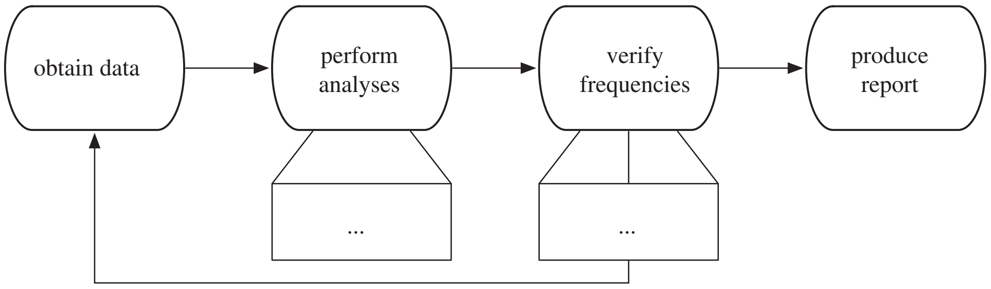


Figure 3. Abstract workflow for ADR study.

5. Use case: adverse drug reaction workflow in Discovery Net

We demonstrate the use of the Workflow Analyser to a scientific workflow from a medical informatics study. The study is based on extracting and analysing the associations between drugs and adverse events by analysing large collections of primary care data records. The workflow encodes a case-crossover design, where the rates of adverse events are observed in the same individuals while they are exposed and unexposed to the drug. There were two objectives to the study. First was to use the case-crossover method to identify how long after the introduction of a drug can an ADR be identified. Second was to do this in a reusable fashion that can be adapted to different data sources and different conditions and reduce the analysis time from several weeks to a couple of days. The findings of the study were published in Molokhia *et al.* (2008).

Figure 3 shows the conceptual workflow. After obtaining the primary care data from the source (patient information, prescriptions, therapies and observations), there are two key stages in the analysis, each of which can itself be further expressed as a workflow. The first is conducting statistical analysis to extract the associations for a particular drug and a control drug. The second is a verification stage used to evaluate the statistical significance of the results, and to decide whether further iterations of data collection and analysis are needed. Once all the analysis is conducted a report is generated for the user.

The workflow shown can generally be mapped to an executable workflow in a number of existing scientific workflow systems. Figure 4 shows the implementation in the Discovery Net system, consisting of more than 100 nodes in the unfolded form. It will now be shown how the analysis tool is used to investigate the properties of interest.

(*a*) *Process profiler*

In figure 4, the control-flow layer contains five *Execute* tasks. *Get data* (GD) involves obtaining the data from source database, loading them into the target database and creating indices over relevant tables. This is a manual step that is not modelled by a data flow. *Statin analysis* (SA) and *Fibrate analysis* (FA) use the same underlying data flow and are performed automatically. *Verify frequencies* (VF) also uses a data flow and it compares the results from both analysis to known prescription frequencies for both types of drugs in each year

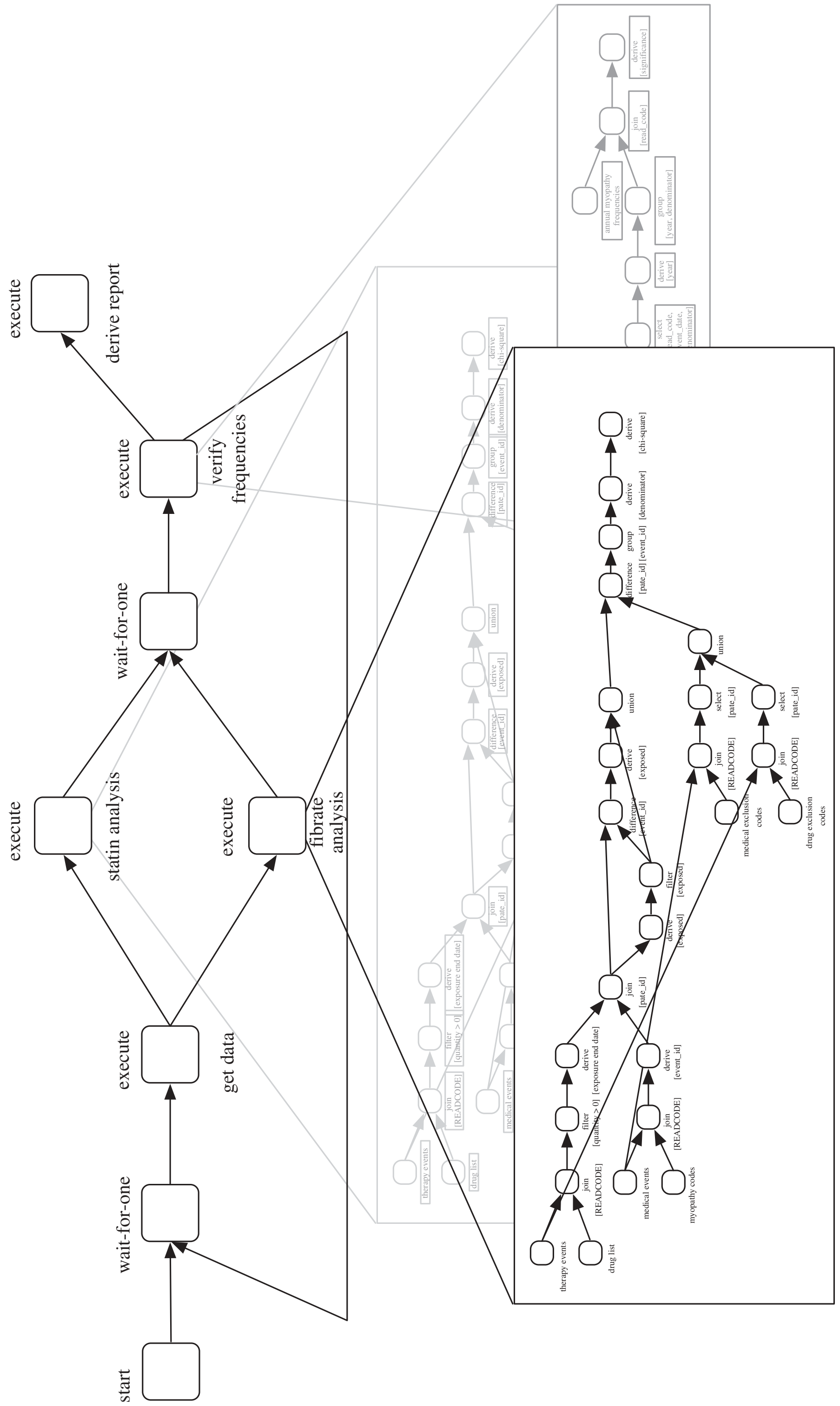


Figure 4. Full workflow for the ADR study.

of the study to ensure that no errors have crept into the analysis. *Derive report* (DR) is a manual step in which the data are formatted by the user and placed in a report.

Process profiler connects to the CTL reasoning engine to investigate bounds on the number of parallel executions of any node in the system, as well as the total number of executions in the system in different scenarios. It also investigates the causal relationships between nodes—whether one execution implies another one. Special cases of causality are livelock situations, where the node execution always implies another execution of the same node. Finally, deadlock situations are also checked, where the system cannot proceed, and there are still nodes executing.

The test of whether termination is possible is performed by searching for a state in some execution in which there will be no more node instances in any possible future execution:

$$\mathbf{EF}\ \mathbf{AG}\neg\mathrm{GD}_\tau \wedge \neg\mathrm{SA}_\tau \wedge \neg\mathrm{FA}_\tau \wedge \neg\mathrm{VF}_\tau \wedge \neg\mathrm{DR}_\tau.$$

This query returns a positive answer from the process profiler, so there is some execution which will terminate. The next thing to check is if in *all* possible executions, there is such a state as follows:

$$\mathbf{AF}\ \mathbf{AG}\neg\mathrm{GD}_\tau \wedge \neg\mathrm{SA}_\tau \wedge \neg\mathrm{FA}_\tau \wedge \neg\mathrm{VF}_\tau \wedge \neg\mathrm{DR}_\tau.$$

And the query fails, listing the path in which the workflow enters an infinite cycle. Therefore, there is a possibility that the workflow will never terminate, if the frequency verification step always fails. The workflow designer would consider whether this is an acceptable risk and could potentially provide a mechanism to restrict the number of times the check can occur.

Deadlocks and livelocks are checked by querying whether in every state there exists an execution that leads towards the termination state:

$$\mathbf{AG}\ \mathbf{EF}\ \mathbf{AG}\neg\mathrm{GD}_\tau \wedge \neg\mathrm{SA}_\tau \wedge \neg\mathrm{FA}_\tau \wedge \neg\mathrm{VF}_\tau \wedge \neg\mathrm{DR}_\tau.$$

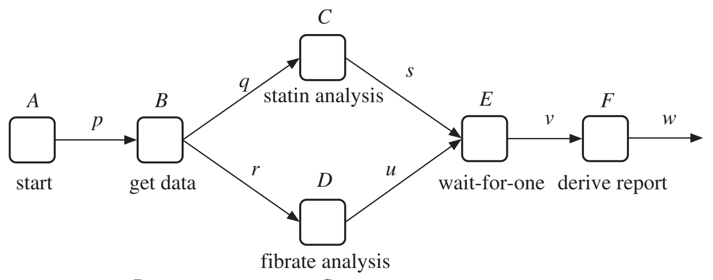
This query also succeeds in the process profiler tool. Therefore, there is no deadlock or livelock situation, and from any state the workflow can always move to completion.

(*b*) *Similarity checker*

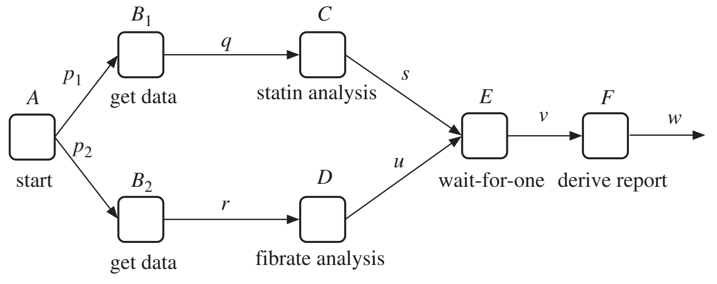
The similarity checker operates directly on the π-calculus process model, as shown on a variant of the ADR workflow in figure 5. The figure depicts two workflow graphs with their π-calculus representation, as detailed in Curcin *et al.* (2009). The checker establishes the equivalence relations between the two input processes, by either reporting that they are equivalent, or listing the states in which the behaviour diverges. Based on this, the workflow author may choose to replace one implementation for the other, due to efficiency or cost constraints (e.g. top example executes only node B instead of B_1 and B_2 which may affect the maximum resource load).

(*c*) *Equivalence checker*

The equivalence checker uses the transformation rules to determine if two data flows produce an identical result upon execution. This is done purely based on allowed graph transformations that have been defined for the given data-flow



$$
\begin{aligned}
W &= (\upsilon\, p, q, r, s, u)\\
&\quad (A \mid B \mid C \mid D \mid E \mid F)\\
A &= \tau_A . \bar{p}(t)\\
B &= !p(t).\tau_B.(\bar{q}(t) \mid \bar{r}(t))\\
C &= !q(t).\tau_C.\bar{s}(t)\\
D &= !r(t).\tau_D.\bar{u}(t)\\
E &= !s(t).u(t).\tau_E.\bar{v}(t)\\
F &= !v(t).\tau_F.\bar{w}(t)
\end{aligned}
$$



$$
\begin{aligned}
W &= (\upsilon\, p_1, p_2, q, r, s, u)\\
&\quad (A \mid B_1 \mid B_2 \mid C \mid D \mid E \mid F)\\
A &= \tau_A.(\bar{p}_1(t) \mid \bar{p}_2(t))\\
B_1 &= !p_1(t).\tau_B.\bar{q}(t)\\
B_2 &= !p_2(t).\tau_B.\bar{r}(t)\\
C &= !q(t).\tau_C.\bar{s}(t)\\
D &= !r(t).\tau_D.\bar{u}(t)\\
E &= !s(t).u(t).\tau_E.\bar{v}(t)\\
F &= !v(t).\tau_F.\bar{w}(t)
\end{aligned}
$$

Figure 5. Two bisimilar varieties of the same workflow.

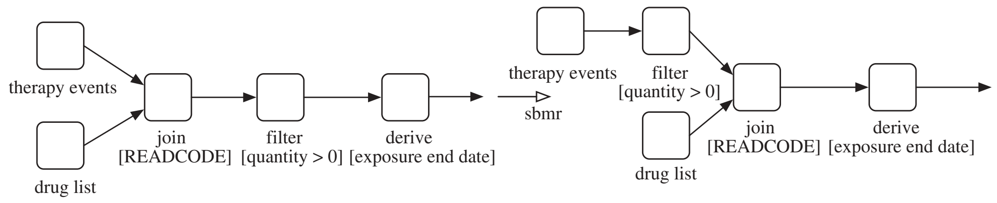


Figure 6. Rewriting a workflow segment.

system. In the example in figure 6, using the equivalence of the position of *Filter* node with respect to the *Join*, the checker decides that the operations represented by the two workflows produce the same result, and therefore rewrite is legal.

(*d*) *Data-flow optimizer*

The optimizer combines the transformation rules with the weighting rules for the system, which constitute the part of node annotations in the system. These rules determine what configurations of components are deemed ‘desirable’, in terms of processing cost. Combined with the transformation system, this module offers the user design–time advice in terms of workflow transformations that are available and potentially useful. The transformation in figure 6 uses the ‘early selection’ paradigm in relational algebra to reduce the size of data entering the join.

(*e*) *Composability checker*

The composability checker uses the typing engine to determine if the composition of two nodes in the workflow is valid. There are three main usages. Firstly, if the workflow is in the construction stage, it can allow the composition

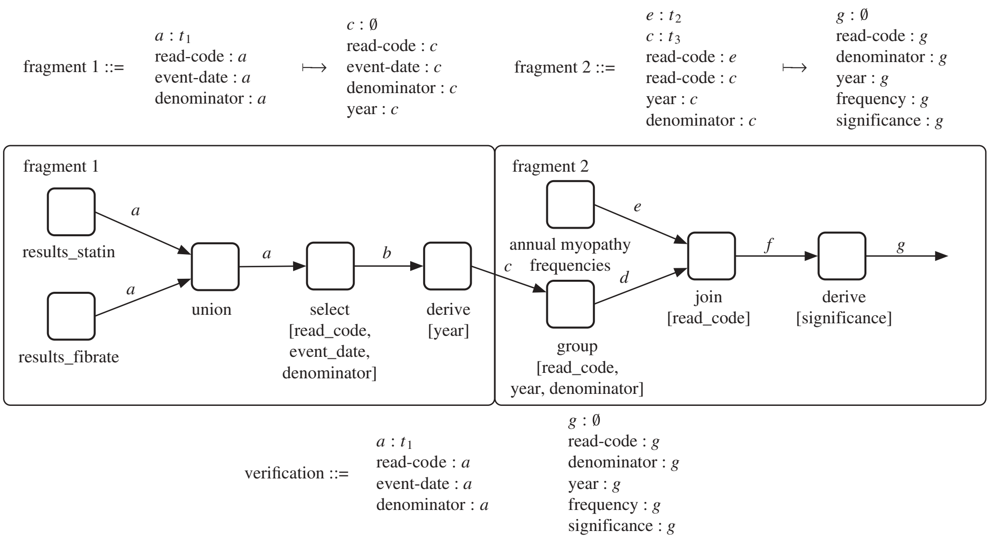

fragment 1 ::= a : t1 read-code : a event-date : a denominator : a ⟼ c : ∅ read-code : c event-date : c denominator : c year : c
fragment 2 ::= e : t2 c : t3 read-code : e read-code : c year : c denominator : c ⟼ g : ∅ read-code : g denominator : g year : g frequency : g significance : g
fragment 1
results_statin
results_fibrate
a
a
union
a
select [read_code, event_date, denominator]
b
derive [year]
c
fragment 2
annual myopathy frequencies
e
group [read_code, year, denominator]
d
join [read_code]
f
derive [significance]
g
verification ::= a : t1 read-code : a event-date : a denominator : a g : ∅ read-code : g denominator : g year : g frequency : g significance : g

Figure 7. Verification data-flow-type checking.

and present the type incompatibilities to be rectified. Secondly, if a new data input is being passed into a fixed, immutable workflow, it can refuse composition altogether. Thirdly, given a dataset, and a set of workflows, it can determine with which workflows the given dataset can be composed. The example in figure 7 shows two fragments of the verification workflow, with their type formulas. Each workflow fragment is given its type transformation, the notation for which can be found in Curcin *et al.* (2010). The composability checker combines the two, and obtains the joint formula (figure 7) that gives the type behaviour of the entire workflow.

6. Summary and conclusions

Previous and current efforts in scientific workflow research have focused on building workflows and workflow systems for different application domains. Achieving widespread use of such tools and enhancing their usability requires focusing on issues that arise when dealing with a large number of complex workflows. In this article, we have presented how a combination of formal methods can be used for the design and implementation of a practical tool for static checking of workflow execution properties at design time, including termination, optimization and type safety, among others, with the view of reducing the effort in workflow design and development process.

The key operations of the Workflow Analyser have been demonstrated on a medical workflow study. The framework is based on separating the analysis of control and data properties, and then designing formalisms for each. The approach can be applied to modelling workflows in any workflow system, once the process formulas for different nodes are specified, and node type properties and rewrite heuristics captured. However, the basic five engines remain the same, their implementation is generic and the overall design of the tool is modular so components can be used individually.

One future direction for the tool development involves adding a stochastic dimension to the control model to capture rates of execution of each component. This information, which can be extracted from execution logs, will allow us to include timing information in our temporal logic queries. A second direction comes from the reconfigurable aspect of workflows. While not explicitly present in the workflow control models we have developed, the formalism is capable of encoding failure conditions and alternative execution paths using process mobility.

The authors thank the anonymous reviewers for useful comments and suggested references.

References

Briais, S. 2007 ABC—another bisimulation checker. See http://lamp.epfl.ch/sbriais/abc/.

Brogi, A., Canal, C., Pimentel, E. & Vallecillo, A. 2004 Formalizing Web service choreographies. *Electron. Notes Theor. Comput. Sci.* **105**, 73–94. (doi:10.1016/j.entcs.2004.05.007)

Cardelli, L. & Davies, R. 1999 Service combinators for Web computing. *IEEE Trans. Softw. Eng.* **25**, 309–316. (doi:10.1109/32.798321)

Cimatti, A., Clarke, E. M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R. & Tacchella, A. 2002 NuSMV 2: an open source tool for symbolic model checking. In *CAV'02: Proc. 14th Int. Conf. on Computer Aided Verification*. Lecture Notes in Computer Science, no. 2404, pp. 359–364. Berlin, Germany: Springer.

Clarke, E. M., Emerson, E. A. & Sistla, A. P. 1986 Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.* **8**, 244–263. (doi:10.1145/5397.5399)

Cooper, E., Lindley, S., Wadler, P. & Yallop, J. 2006 Links: web programming without tiers. In *Proc. 5th Int. Symp. on Formal Methods for Components and Objects*. Lecture Notes in Computer Science, no. 4709, pp. 266–296. Berlin, Germany: Springer.

Curcin, V. & Ghanem, M. 2008 Scientific workflow systems—can one size fit all? In *Proc. 4th Int. Biomedical Engineering Conf., Cairo, Egypt, 18–20 December 2008*.

Curcin, V., Ghanem, M. M. & Guo, Y. 2009 Analysing scientific workflows with computational tree logic. *Cluster Comput.* **12**, 399–419. (doi:10.1007/s10586-009-0099-6)

Curcin, V., Ghanem, M. & Guo, Y. 2010 Polymorphic type framework for scientific workflows with relational data model. *Int. J. Bus. Process Integr. Manag.* **5**, 45–62. (doi:10.1504/IJBPIM.2010.033174)

Foster, H., Uchitel, S., Magee, J. & Kramer, J. 2007 WS-engineer: a model-based approach to engineering web service compositions and choreography. In *Test and analysis of web services* (eds L. Baresi & E. D. Nitto), pp. 87–119. Berlin, Germany: Springer.

Ghanem, M., Azam, N., Boniface, M. & Ferris, J. 2006 Grid-enabled workflows for industrial product design. In *Proc. 2nd IEEE Int. Conf. on e-Science and Grid Computing*, p. 96. Washington, DC: IEEE Computer Society.

Ghanem, M., Curcin, V., Wendel, P. & Guo, Y. 2008 Building and using analytical workflows in Discovery Net. In *Data mining on the grid* (ed. W. Dubitzky), pp. 119–140. Chichester, UK: Wiley.

Goderis, A., Fisher, P., Gibson, A., Tanoh, F., Wolstencroft, K., De Roure, D. & Goble, C. 2009 Benchmarking workflow discovery: a case study from bioinformatics. *Concurr. Comput. Pract. Exp.* **21**, 2052–2069. (doi:10.1002/cpe.1447)

Hey, A. & Trefethen, A. E. 2002 The UK e-Science core programme and the grid. *Future Gener. Comput. Syst.* **18**, 1017–1031. (doi:10.1016/S0167-739X(02)00082-1)

Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M. R., Li, P. & Oinn, T. 2006 Taverna: a tool for building and running workflows of services. *Nucleic Acids Res.* **34**, W729–W732. (doi:10.1093/nar/gkl320)

InforSense. 2003 Knowledge discovery environment. See http://www.inforsense.com.

Kelly, P. M., Coddington, P. D. & Wendelborn, A. L. 2008 Lambda calculus as a workflow model. In *Proc. 3rd Int. Conf. on Grid and Pervasive Computing*, pp. 15–22.

Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E. A., Tao, J. & Zhao, Y. 2006 Scientific workflow management and the Kepler system: research articles. *Concurr. Comput. Pract. Exp.* **18**, 1039–1065. (doi:10.1002/cpe.994)

Magee, J. & Kramer, J. 1996 Dynamic structure in software architectures. *SIGSOFT Softw. Eng. Notes* **21**, 3–14. (doi:10.1145/250707.239104)

Molokhia, M., McKeigue, P., Curcin, V. & Majeed, A. 2008 Statin induced myopathy and myalgia: time trend analysis and comparison of risk associated with statin class from 1991–2006. *PLoS ONE* **3**, e2522. (doi:10.1371/journal.pone.0002522)

Roure, D. D., Goble, C. A. & Stevens, R. 2007 Designing the myExperiment virtual research environment for the social sharing of workflows. In *Proc. 3rd Int. Conf. on e-Science and Grid Computing*, pp. 603–610. Washington, DC: IEEE Computer Society.

Syed, J., Ghanem, M. & Guo, Y. 2007 Supporting scientific discovery processes in Discovery Net. *Concurr. Comput. Pract. Exp.* **19**, 167–179. (doi:10.1002/cpe.1049)

Taylor, I., Shields, M., Wang, I. & Harrison, A. 2007 The Triana workflow environment: architecture and applications. In *Workflows for e-Science* (eds I. Taylor, E. Deelman, D. Gannon & M. Shields), pp. 320–339. Secaucus, NJ: Springer.

Turi, D., Missier, P., Goble, C. A., Roure, D. D. & Oinn, T. 2007 Taverna workflows: syntax and semantics. In *Proc. 3rd Int. Conf. on e-Science and Grid Computing*, pp. 441–448. Washington, DC: IEEE Computer Society.

van der Aalst, W. M. P. & ter Hofstede, A. H. M. 2005 YAWL: yet another workflow language. *Inform. Syst.* **30**, 245–275. (doi:10.1016/j.is.2004.02.002)