A Microservice Architecture for the Design of Computer-Interpretable Guideline Processing Tools

Martin Chapman
Department of Population Health Sciences,
King's College London,
London, U.K.
Email: martin.chapman@kcl.ac.uk

Vasa Curcin
Department of Population Health Sciences,
King's College London,
London, U.K.
Email: vasa.curcin@kcl.ac.uk

***Abstract*—**
Several tools exist that are designed to process computer interpretable guidelines (CIGs), each with a distinct purpose, such as detecting interactions or patient personalisation. While it is desirable to use these tools as part of larger decision support systems (DSSs) doing so is often not straightforward, as their design does not often support external interoperability or account for the fact that other CIG tools may be running in parallel, a situation that will become increasingly more prevalent with the increased adoption of CIGs in different parts of the health system. This results in an integration overhead, system redundancy and a lack of flexibility in how these tools can be combined. To address these issues, we define a blueprint architecture to be used in the design of guideline processing tools, based on the conceptualisation of key components as RESTful microservices. In addition, we define the types of data endpoints that each component should expose, for both the communication between internal components and communication with external components that exist as a part of a DSS. To demonstrate the utility of our architecture, we show how an example guideline processing tool can be restructured according to these principles, in order to enable it to be flexibly integrated into the DSS used in the CONSULT project.

I. INTRODUCTION

Clinical practice guidelines (CPGs) are defined as 'systematically developed statements to assist practitioner and patient decisions about appropriate health care for specific clinical circumstances' [1]. When represented in a format that can be autonomously processed by a computer, CPGs are referred to as computer-interpretable guidelines (CIGs) [2]. A number of formalisms exist to represent CPGs as CIGs, including GLIF3 [3], Pro*forma* [4] and GLARE [5]. In addition to guideline representation, a number of tools exist that aim to process CIGs in order to derive additional information. These tools typically consist of two logical components: a reasoner, and a set of CPGs represented in a chosen CIG formalism, upon which the reasoner acts. For example, Zamborlini et al. introduce a tool that uses a logic-based *Prolog* reasoner to processes CIGs, which are defined using a set of semantic web languages (e.g. *OWL* [6]), in order to derive information about guideline interactions, such as when the action associated with one stored guideline counteracts the effects of the action associated with another stored guideline [7]. Alternatively, Riano et al. introduce a tool that aims to personalise CIGs by processing both general CIG information and patient-specific information, described in a state-decision-action (SDA) language, in order to produce additional information about how an individual should be treated [8]. Here, the reasoner takes the form a bespoke SDA processing tool, *SDA lab*. Isern et al. also use this SDA language for the purpose of generating patient-specific guidelines, however their derivation of personalised guideline information is based on an reasoner that uses agent-based execution [9].

A (clinical) decision support system (DSS) is designed to provide a clinician, or patient, with recommendations in order to assist with clinical decision making [10]. Many DSSs provide their recommendations by consulting a number of different types of data sources. One of these sources is naturally CIGs, and the output of CIG processing tools. It is therefore often the case that the designer of a decision support system (DSS) wishes to integrate the software components of their system with multiple CIG tools. Once integrated, there are two main types of components in a DSS that will interact with each CIG tool. Firstly, components that are designed to gather clinician input via a graphical user interface (GUI) will pass medical knowledge pertinent to the creation of guidelines to each CIG tool. Secondly, decision-making components, will, autonomously, as a part of their computational processes, trigger the processing of guideline information within each tool and collect the results, such as information about interactions between guidelines or information about patient specific guidelines. However, integrating these software components with each CIG tool is often a cumbersome task for a number of reasons. Firstly, CIG tools are often not designed to interact (autonomously) with other computational systems, but instead only directly with human beings, such as having their processing functionality triggered through, and providing outputs via, a GUI. For example, one interacts with Zamborlini's tool using a prolog-based web application front-end (*SWISH*), while the SDA processing tools found in Riano et al's work are also graphical. This means that a system designer must develop additional software-level interfaces for each tool in order to integrate it for the purpose of (autonomous) decision support. Secondly, the type of communication between the two

978-1-5386-9301-8/19/$31.00 ©2019 IEEE

components of a CIG tool – the reasoner and the guideline data stored using a representation formalism – does not often occur in a standard way across tools. This is the case even when the same CIG formalisms are used. For example, the logical reasoner in Zamborlini et al's tool communicates with serialisations of CIGs (e.g. *Turtle* serialisations [11]), and associated knowledge, using local text file reads, while the reasoners in the tools introduced by Riano et al. and Isern et al., although both using SDA structured data, communicate with this data in different ways, the former interacting directly from the SDA lab tool, and the latter via a local API [8], [9]. This often means that a designer must maintain multiple stores for the same guideline data, even when those stores use the same format, as the same store cannot be used by multiple reasoners. Moreover, the designer cannot arbitrarily combine reasoners and formatted guideline stores to suit their purposes, such as removing the semantic formalism used by Zamborlini et al., and replacing it with an SDA formalism, if they prefer this combination. A number of other difficulties exist, including difficult technical deployment, as well issues with resilience and scalability.

To address these issues, in this paper we propose that the components of a CIG tool be split into a number of standard *microservices* [12], each with a well-defined *REST*ful interface [13] that exposes certain types of endpoints, allowing for a communication structure that better accommodates external communication with the components of a DSS, a structure in which there is limited software redundancy and a structure in which the software supporting different types of services (e.g. guideline stores) can be arbitrarily swapped within a given tool. The remainder of this paper is therefore structured as follows. We first introduce a blueprint microservice architecture, and discuss which services CIG tools developed using this architecture ought to contain. In doing so, we also define the categories of endpoints each type of microservice should offer. Next, we illustrate the typical communication flows that are present in CIG tools designed under our architecture, when interacting with the components of a DSS. Finally, we close with a case study that illustrates the utility of our approach by redesigning an existing, monolithic CIG tool under our architecture, and show the improvements this brings to the interoperability and modularity of the system when integrated into a DSS, specifically the CONSULT system [14].

II. MICROSERVICES

A microservice architecture separates the features of a system into individual services, which are typically composed of one or more applications, including a server, which may then be spread over one or more physical servers [12]. Microservices are often also *REST*ful (Representational State Transfer), which means that the functionality of the systems they contain can be invoked, and information both extracted and entered, using URIs that are defined as a part of that service's Application Programming Interface (API) [13]. For example, a client might issue a self-contained *HTTP GET* request to the URI */data/all* of a microservice in order to obtain the contents of a database contained within that service, and this information will be returned in the body of a HTTP response.

TABLE I
MICROSERVICE ENDPOINTS

Microservice	Endpoint	Endpoint Description
Interaction	/guideline/create	Create a guideline set.
	/guideline/(add|delete)	Add or delete information from a guideline set.
	/guideline/[knowledge]/(add|delete)	Add or delete supporting guideline knowledge.
	/guideline/[data]+/get	Get data from a guideline.
	/guidelines/[action]	Perform a processing action on a guideline set.
Reasoner	/guidelines/.+	All paths with the plural word *guidelines* are handled by the reasoner, as these relate to the processing of guidelines.
Store	/guideline/.+	All paths with the singular word *guideline* are handled by the store, as these relate to the storage and retrieval of guideline information.

The RESTful microservice approach allows us to design a blueprint architecture for guideline processing tools that addresses many of the issues discussed, including *interoperability*, which enables flexible integration with the external component of a DSS, with those components only having to issue simple (autonomous) HTTP requests to the RESTful endpoints provided, and receive machine-processable responses, in order to achieve integration; *reusability*, with the separation into stateless services enabling one service to be easily used by multiple other services, such as using the same software to represent and store the CIGs processed by multiple reasoners; *technological heterogeneity*, which allows the system contained within a service to be changed without this affecting the other services using it, such as arbitrarily changing the software used to store CIGs but retaining the same API structure; and a number of additional benefits, including ease of deployment, resilience and scalability.

Under our CIG tool architecture, we specify three different types of RESTful microservices, which we detail in the following sections while expanding upon the above properties.

A. Interaction

The *interaction* service contains a server designed to expose all of the functionality offered by the tool to the integrating components of a DSS. As noted, exposing this functionality as a set of REST endpoints, rather than as a GUI, makes a tool interoperable with DSS software components – such as those that gather clinician input or perform autonomous processing – via HTTP requests.

To improve the interoperability of the interaction service further, we specify five main types of endpoints that a system designer should expect to be able to embed programmatic calls to within their software components when integrating a DSS with a CIG tool developed under our architecture. The types of

TABLE II
MICROSERVICE ENDPOINT REQUEST BODY FORMATS

Endpoint	Body field	Description
/guideline/create	id	The id of this guideline group.
	description	A text description of this guideline group.
/guideline/[knowledge]/(add|delete)	id	The id of this knowledge item.
	[knowledge]_id	A reference to a supporting knowledge item.
	[property]	A property of this knowledge item.
/guideline/(add|delete)	id	The id of this guideline.
	group_id	The group to which this guideline belongs.
	[knowledge]_id	The id of a knowledge entity.
	[property]	A property of this guideline.
	author	The author of this guideline.
/guideline/[data]+/get	guideline_id	The guideline from which to acquire the data specified.
	guideline_group_id	The guideline group within which the target guideline is contained.
/guidelines/[action]	guideline_group_id	The guideline group upon which to perform the specified action.

endpoints chosen are based on the observation that guidelines are often stored in *sets*, such as those relating to a particular condition. Moreover, repositories of CIG information also typically allow for the storage of more general types of knowledge, which support the data held in the guidelines themselves. As such, our endpoints include: those that allow the creation of a guideline set, the addition of new guideline information to a set or the addition of supporting knowledge, the deletion of existing guideline information from a set or the deletion of supporting knowledge, the retrieval of information from a guideline and the invocation of internal processing upon the guidelines in a set. This specification is in the form of *template* endpoint paths, as shown in Table I. Here, we indicate variables using square brackets, options using parenthesis and the arbitrary repetition of variable types using the plus notation. These template paths are designed to be instantiated in order to construct actual paths upon the implementation of a tool under our architecture, and detailed in API documentation provided for that tool. Similarly, we specify the types of information that a DSS software component should provide in the *body* of a request to each of the endpoints shown. This information is shown in Table II. Like the endpoints themselves, we include several variables here, which are designed to be fully specified in the API documentation of each tool.

B. Reasoner

The *reasoner* service is an internal service that encapsulates a tool's reasoner, and is designed to be invoked by the interaction service, which in turn offers reasoning functionality to DSS components. The endpoints offered by the reasoner service are thus the subset of those used to offer processing functionality by the interaction service. By using the interaction service as a proxy to the reasoner service, as opposed to allowing DSS components to call it directly, we ensure that the availability of the tool is consistent, even in the presence of a reasoner that enters a failing state. This increases the resilience of the tool. Moreover, one could instantiate multiple copies of the reasoning service to serve the requests received by a single interaction service – which we assume makes a choice about which instantiation of the reasoner service to call – in order to improve the scalability of a CIG processing tool.

C. Store

Our final microservice, *store*, encapsulates the software required to store CPG information in a given format. It is again an internal service that is designed to be called by both the interaction service and the reasoner, with the former forwarding guideline information onto, and providing guideline information from, DSS components, and the latter processing the guidelines in order to provide the tool's main functionality. The endpoints offered by the store are thus the subset of those used to offer guideline storage and retrieval functionality by the interaction service. The interoperability brought by the encapsulation of the CIG store in a RESTful microservice is, in this instance, beneficial not to external integrating components, but instead in enabling different reasoners to interact with the same information store (reusability). Moreover, specifying a standard set of REST endpoints (Table I), means that other services, such as the reasoner, become agnostic to the type of guideline representation software used within the service to store CIGs. This enables a system designer to use different CIG representation software with the same reasoner (technological heterogeneity). Like the reasoner service, using the interaction service as a proxy to the store ensures that the tool is always able to provide a response, even if there are internal issues with the store software.

III. COMMUNICATION

We now provide an example of the communication between the microservices in our architecture in order to facilitate the two types of interaction listed in section I: the scenario in which a clinician wishes to add one or more new guidelines to the system and then a decision-making component autonomously invokes processing upon those guidelines.

Figure 1 first shows the communication path between the clinician and the tool. The clinician enters the required guideline information into the GUI of an external DSS component designed to elicit clinical (guideline) knowledge, and this data is then forwarded to the interaction microservice of the tool by issuing a HTTP request to one of the */add* endpoints, which in turn forwards this information to the CIG store using an additional request. Recall that DSS components do not interact with the reasoner or the store directly, to ensure availability of the tool even in the presence of errors with either of these services. Confirmation is received back from

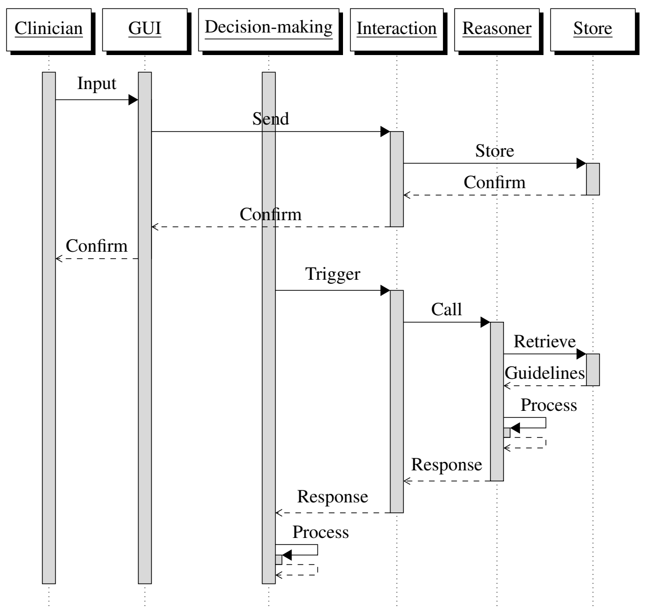

Fig. 1. An overview of the communication between services to supply information about a new guideline to a tool, and to then process that guideline. Calls between services are HTTP POST requests.

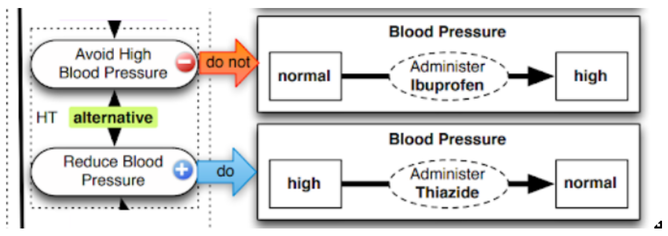

Fig. 2. Two guidelines, *Diueretic2* and *Diuretic*, formatted according to the TMR model for use in the guideline interaction tool.

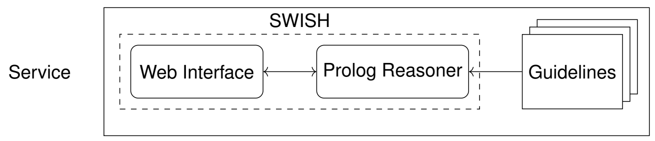

Fig. 3. The original structure of the guideline interaction tool.

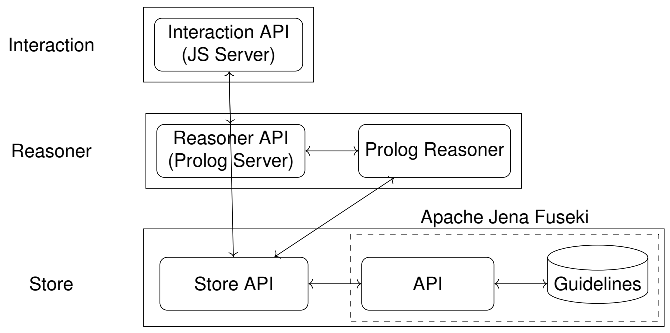

Fig. 4. The guideline interaction tool (Figure 3) reorganised into the set of microservices dictated by our architecture.

the store, which in turn is passed back to the GUI component and then to the clinician. This processes is repeated for each piece of guideline knowledge to be stored. At some point in the future, a decision-making component in the DSS, triggered by a perceived changed in the environment of the system, requires additional information about the guidelines that have been previously stored, which is provided by the CIG processing tool. It therefore makes an autonomous request directly to the interaction service, using a */guidelines/* endpoint, which in turn calls the reasoner service. This service then makes an internal call to the store in order to retrieve the priorly stored guidelines. It then processes these guidelines in order to generate additional information (such as personalised patient information), and returns this back to the component, via the interaction service, which can then perform its own processing.

IV. Case Study

To illustrate the impact of our architecture on the interoperability of a guideline processing system, we now transform an existing CIG tool so that it adheres to our architecture; separating the existing components of the system into individual services, and defining a set of communication endpoints that are compliant with our specification. We then show how the new architecture of the tool supports interactions with a DSS such as the Consult system [14]. Our example implementation is available open source: https://github.com/martinchapman/guideline-interaction-tool.

Our target is Zamborlini et al's CIG tool, which is designed to identify interactions between guidelines [7]. We thus refer to this as the *guideline interaction tool*. An example output of this tool – two interacting guidelines – is shown in Figure 2. Here, the guidelines interact by one being a *reparable transition* (*alternative*) of the other: administering Thiazide (*Diuretic*) reverses the effects of administering Ibuprofen (*Diuertic2*). The structure of the information shown in Figure 2, including the interaction, is specified by the drug interaction tool's companion model, the *Transition-based Medical Recommendation* model (TMR) [15]. As can be seen in Figure 2, TMR pertains not only to specific guideline information, but also more general world information, such as the existence of drugs; the concept of a patient having a certain medical state, such as a blood pressure level; the concept of altering a person's medical state, such as their blood pressure; and the effects of taking a drug. As discussed, this information is specified prior to the execution of the system using semantic web languages such as OWL, and is then serialised and stored in a set of text files. It is then processed as part of a monolithic Prolog application. Interaction with the application occurs via the use of a web front-end. Everything is contained within a single conceptual service, with each component requiring different deployment steps, and robustness and scalability fall outside of the focus of the work. An overview of the original architecture of this tool is given in Figure 3.

TABLE III
ENDPOINTS FOR THE RECONFIGURED INTERACTION TOOL

Microservice	Endpoints	Instantiation
Interaction	/guideline/create	N
	/guideline/add	N
	/guideline/drug/add	Y
	/guideline/transition/add	Y
	/guideline/belief/add	Y
	/guideline/.+/delete	N
	/guideline/drug/get	Y
	/guideline/drug/effect/get	Y
	/guidelines/interactions	Y
Reasoner	/guidelines/.+	-
Store	/guideline/.+	-

TABLE IV
AN EXAMPLE BODY FORMAT FOR THE RECONFIGURED INTERACTION TOOL: /GUIDELINE/ADD

Body field	Description
guideline_group_id	The id of the group to which this guideline is to belong
guideline_id	The id of the new guideline
drug_id	The drug to which this guideline pertains
belief_id	The belief to which this guideline pertains
aim	A description of the aim of this guideline
administer_or_avoid	Whether the drug to which this guideline pertains should be administered or avoided in order to achieve the aim.
author	The author of this guideline.

A. Microservices and Endpoints

In order to reorganise the architecture of this system to adhere to our own, we first split the guideline interaction tool into the three types of service given in section II. The separation of these components is shown in Figure 4. The web interface is replaced with an interaction service, realised as a external-facing Javascript server, designed to interact with the external GUI and decision-making components of a DSS. This service acts as a proxy to the reasoner service, which consists of a Prolog server and an associated reasoning component, which is also realised in Prolog. Both the interaction service and the reasoner source guideline information from and, in the case of the interaction service, add guideline information to, a structured *n-triple* store, *Apache Jena Fuseki* [16]. As Jena has its own API endpoints, which may not necessarily be consistent with the endpoints specified for a store service as a part of our architecture (Table I), Jena is proxied by an additional API within the store service.

A list of the endpoints provided by our new set of microservices is given in Table III. Here, we instantiate the path variables given in Table I in order to derive three individual knowledge paths for the supporting knowledge required for the interaction tool to represent guidelines (drugs, medical state transitions, and beliefs on the effects of drugs), as well as example paths to retrieve the drug mentioned in a guideline and to retrieve the effect of administering a drug, and a path to initiate the processing of the stored guidelines in order to identify any interactions. In addition, we illustrate the data that can be sent in the body of a request to one of these endpoints

```
curl --request POST \
  --url https://consult.hscr.kcl.ac.uk/
      guideline/add \
  --data 'guideline_group_id=HT&guideline_id=
      Diuretic2&drug_id=Ibuprofen&belief_id=
      IbuprofenBP&label=avoid high blood
      pressure&administer_or_avoid=avoid&
      author=martin'
```

Fig. 5. A HTTP request to store information on the guideline *Diuretic2*.

```
curl --request POST \
  --url https://consult.hscr.kcl.ac.uk/
      guidelines/interactions \
  --data guideline_group_id=HT

[interaction(ReparableTransitionRecHT-
    Diuretic2RecHT-Diuretic, Reparable
    Transition, [RecHT-Diuretic, RecHT-
    Diuretic2])]
```

Fig. 6. A HTTP request to list the interactions between stored guidelines.

– the guideline addition endpoint – in Table IV. This endpoint accepts the ids of a priorly defined drug and belief, as well as a guideline group and properties such as the guideline aim, in order to construct guidelines such as the ones seen in Figure 2.

B. DSS Integration

With the guideline interaction tool now separated into the set of microservices dictated by our architecture, each supporting the specified types of endpoints, we now illustrate its interoperability with a DSS, specifically the CONSULT system. CONSULT aims to support the autonomous generation of treatment plans for stroke patients by gathering data about those patients from a variety of sources, including guidelines and information about the interactions between guidelines, and processes these datasources in a centralised computational argumentation engine [17]. We consider a scenario similar to the one show in in Figure 1 in which the designer of the CONSULT system wishes to enable a clinician to add guideline knowledge to the tool, specifically the guideline information shown in Figure 2, and wishes to enable the argumentation engine to autonomously issue requests to the tool for interaction information when needed.

First of all, it is now trivial for the designer to deploy the interaction tool, as it consists of a set of lightweight services that have minimal platform dependencies. Once the three services required to provide the interaction tool functionality are running, then a request like the one shown in Figure 5 is all that a component supporting clinician input is required to send in order to add information about a guideline to the tool (assuming prior requests have been issued to provide knowledge about the associated drug (*Inbuprofen*), belief (*IburpofenBP*) and guideline group (*HT*)). This request is sent to the */guideline/add* endpoint (Table III) at an address provided by the CONSULT system, and provides data according to required body structure of this request (Table IV) pertaining to the guideline *Diuretic2* (Figure 2). Note that this information can now be specified as a lightweight HTTP request, with simple parameters, once the system is running, as opposed to having to priorly specifying this information in text files directly in the semantic web formalisms used. Assuming a similar request is made to specify the *Diuretic* guideline, a request like the one shown in Figure 6 is then all that is required for the argumentation engine to invoke processing upon any stored guidelines. The response obtained from this request lists the interactions between the guidelines in a machine-processable form, giving, for each interaction, the type of interaction (e.g. reparable transition) and the unique IDs of the interacting guidelines.

As previously discussed, in addition to interoperability, there are a number of additional advantages to a DSS such as the CONSULT system of utilising a set of CIG processing tools designed under our architecture. If the designers wish to introduce an additional reasoner, they can do so by simply running the reasoning service of another CIG processing tool designed under our architecture, and configure both reasoners to direct their store requests to the address of the existing store service. For example, one could introduce the reasoner developed by Riano et al. and have it also communicate with the Jena store used by the interaction tool. Similarly, the designers could, if they deem the Jena store not be fit for purpose, simply remove it and replace it with a different formalism for storing CIGs. As the Store API remains unchanged, this will leave the rest of the system unaffected. Additionally, the designers may wish to duplicate the reasoning service to serve higher load.

V. RELATED WORK

Microservices have been used to good effect in a number of different domains, including smart cities [18], IoT [19] and automated assessment in education [20]. In addition, the KGrid initiative [21] investigates building a microservice interface around generic knowledge objects, which could be applied to guideline representation and reasoning. Some attempts to standardise elements of the interactions within guideline systems do exist, such as the local API proposed by Isern et al. [9], but systems like this are not typically designed to promote interoperability between tools.

VI. CONCLUSION AND FUTURE WORK

In this paper, we introduce an architecture for the design of CIG processing tools, which centres around three types of RESTful microservices, each with a set of well-defined endpoints. By adopting this architecture, tool developers increase the interoperability of their systems with DSSs, ensure DSS designers are not forced to run redundant copies of software, have the flexibility to arbitrarily combine components of their tools with the components of other tools, and have tools available that are easy to deploy, resilient and scalable.

Future work will focus on further defining the interaction between components in the architecture. For example, in systems like the guideline interaction tool (Figure 4), a formal mechanism may be needed to translate requests to the Store API into requests to the underlying storage platform.

ACKNOWLEDGMENT

This work is funded by the UK Engineering & Physical Sciences Research Council (EPSRC) under grant #EP/P010105/1.

REFERENCES

[1] Institute of Medicine, “Clinical Practice Guidelines: Directions for a New Program,” Institute of Medicine, Tech. Rep. 8, 1990.

[2] M. Peleg, “Computer-interpretable clinical guidelines: A methodological review,” pp. 744–763, aug 2013.

[3] M. Peleg, A. A. Boxwala, O. Ogunyemi, Q. Zeng, S. Tu, R. Lacson, E. Bernstam, N. Ash, P. Mork, L. Ohno-Machado, E. H. Shortliffe, and R. A. Greenes, “GLIF3: the evolution of a guideline representation format.” *AMIA Symposium*, pp. 645–9, 2000.

[4] D. R. Sutton and J. Fox, “The Syntax and Semantics of the PROforma Guideline Modeling Language,” *JAMIA*, vol. 10, no. 5, pp. 433–443, sep 2003.

[5] P. Terenziani, P. Terenziani, S. Montani, S. Montani, A. Bottrighi, A. Bottrighi, M. Torchio, M. Torchio, G. Molino, G. Molino, G. Correndo, and G. Correndo, “The GLARE approach to clinical guidelines: Main features,” in *Studies in HTI*, vol. 101, 2004, pp. 162–166.

[6] D. L. McGuinness and F. Van Harmelen, “OWL Web Ontology Language Overview,” World Wide Web Consortium, Tech. Rep., 2004.

[7] V. Zamborlini, J. Wielemaker, M. Da Silveira, C. Pruski, A. Ten Teije, and F. Van Harmelen, “SWISH for prototyping clinical guideline interactions theory,” in *CEUR*, vol. 1795, 2016.

[8] D. Riaño, F. Real, J. A. López-Vallverdú, F. Campana, S. Ercolani, P. Mecocci, R. Annicchiarico, and C. Caltagirone, “An ontology-based personalization of health-care knowledge to support clinical decisions for chronically ill patients,” *JBI*, vol. 45, no. 3, pp. 429–446, 2012.

[9] D. Isern, A. Moreno, D. Sánchez, Á. Hajnal, G. Pedone, and L. Z. Varga, “Agent-based execution of personalised home care treatments,” *Applied Intelligence*, vol. 34, no. 2, pp. 155–180, 2011.

[10] D. L. Hunt, R. B. Haynes, S. E. Hanna, and K. Smith, “Effects of Computer-Based Clinical Decision Support Systems on Physician Performance and Patient Outcomes,” *JAMA*, vol. 280, no. 15, p. 1339, oct 1998.

[11] D. Beckett and T. Berners-Lee, “Turtle: Terse RDF triple language,” World Wide Web Consortium, Tech. Rep., 2005.

[12] S. Newman, *Building Microservices*, 1st ed. O’Reilly Media, Inc., 2015.

[13] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” Ph.D. thesis, University of California Irvine, 2000.

[14] N. Kokciyan, I. Sassoon, A. Young, M. Chapman, T. Porat, M. Ashworth, V. Curcin, S. Modgil, S. Parsons, and E. Sklar, “Towards an Argumentation System for Supporting Patients in Self-Managing their Chronic Conditions,” in *W3PHIAI*, 2018.

[15] V. Zamborlini, R. Hoekstra, M. Da Silveira, C. Pruski, A. Ten Teije, and F. Van Harmelen, “Inferring recommendation interactions in clinical guidelines,” in *Semantic Web*, vol. 7, no. 4, 2016, pp. 421–446.

[16] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, “Jena: Implementing the Semantic Web Recommendations,” in *WWW’04*, 2004, pp. 74–83.

[17] I. Sassoon, N. Kokciyan, A. P. Young, S. Parsons, S. Modgil, and E. Sklar, “Argumentation-based Decision Support for Patient Self-Management,” in *ArgSoc, COMMA*, 2018.

[18] A. Krylovskiy, M. Jahn, and E. Patti, “Designing a Smart City Internet of Things Platform with Microservice Architecture,” in *FiCloud and OBD*, 2015, pp. 25–30.

[19] B. Familiar, *Microservices, IoT and Azure: Leveraging DevOps and Microservice*. Apress, 2015.

[20] S. Zschaler, S. White, K. Hodgetts, and M. Chapman, “Modularity for automated assessment: A design-space exploration,” in *CEUR*, vol. 2066, 2018, pp. 57–61.

[21] A. J. Flynn, P. Boisvert, N. Gittlen, C. Gross, B. Iott, C. Lagoze, G. Meng, and C. P. Friedman, “Architecture and Initial Development of a Knowledge-as-a-Service Activator for Computable Knowledge Objects for Health.” *Studies in HTI*, vol. 247, pp. 401–405, 2018.