6

PROTOTYPING THE DETECTION OF INTERACTIONS USING SWISH INFRASTRUCTURE

Reality is merely an illusion, albeit a very persistent one.

Albert Einstein

SWISH provides a general purpose collaborative infrastructure for applying Prolog reasoning over an RDF dataset together with features that facilitates prototyping Semantic Web applications. In this paper we report on the use of SWISH for efficiently developing a prototype for detection of clinical guideline interactions. These guidelines are a set of medical recommendations meant for supporting doctors on tackling a single disease. However, often guidelines need to be combined for treating patients that suffer from multiple diseases, and then a number of interactions can occur. The generic interaction rules are implemented in SWI-Prolog and the guideline RDF-data is enriched with clinical Linked Open Data (LOD) (e.g. Drugbank, Sider). We show the implementation of the proposed theory about interaction detection in a case-study on combining three guidelines. The experiment is interactively described using a SWISH notebook and the results are graphical visualised empowered by graphviz.

This chapter is based on *Zamborlini, V.; Wielemaker, J.; da Silveira, M.; Pruski, C.; ten Teije, A.; van Harmelen, F. "SWISH for prototyping Clinical Guideline Interactions Theory" in Proc. of the Workshop on Semantic Web Applications and Tools for Life Sciences, Amsterdam, Netherlands, 2016.*

6.1 INTRODUCTION

Building semantic web applications classically follows a three-tier model: **storage**, **application logic (reasoning)** and **presentation**. Many tools and languages have been proposed for supporting these tiers.

For the **storage** tier, LOD (Linked Open Data) triplestores typically provide their data through a SPARQL endpoint. The **application logic (reasoning)** tier can be implemented in many languages, where typical choices are Java or Python. However, implementing a rule based reasoning layer in these languages that bind to SPARQL is not trivial. Finally, the **presentation** tier can be implemented using the web-page generation facilities of languages such as Java, Python, Ruby or NodeJS, dedicated web programming systems such as PHP or ASP or in the client using JavaScript. The language boundaries between the tiers are relatively hard to maintain, which makes such as software stack mostly viable for large applications that are designed in a top-down fashion and implemented by a team.

Life Science researchers are currently benefiting and contributing to the Semantic Web [20]. For instance, LinkedLifeData (LLD) is an integrated linked data repository for health care and life sciences promoted by the W3C Semantic Web for Health Care and Life Sciences Interest Group[1]. In this context, semantic-web-based prototyping can favour reducing the overhead for connecting the three tiers, while speeding up and enhancing the interaction with domain experts. Moreover, to ensure that the results are verifiable, reproducible and reusable [59], code and data should be accessible in a user friendly way, so that scientists and domain experts can freely explore it.

To this end, SWISH, and in particular the SWISH package for ClioPatria [104], offers an infrastructure that supports all three tiers in an interactive, web-based and cooperative environment for prototyping linked data applications. ClioPatria is a SWI-Prolog based semantic web application framework. The RDF **storage tier** is implemented as a low-level C module that is tightly integrated into Prolog, a perfect language for the **application logic tier** due to its native support for rule based reasoning. SWISH enhances Cliopatria's support for prototyping[2] by providing a web-based development environment and, for the **presentation tier**, an infrastructure for user interfaces that act as a shared platform, similar to a *wiki*. It supports both programmers and domain experts with little or no programming experience and facilitates cooperation between them.

In the first implementation of our theory [114] we followed the classical path aforementioned: (i) we used Stardog for the **storage tier**, (ii) for the **application logic tier** a combination of OWL2 inferenc-

1 See http://linkedlifedata.com and https://www.w3.org/wiki/HCLSIG.

2 [104] presents ClioPatria as suitable infrastructure for prototyping linked data apps.

ing, Stardog SPARQL rules (a SWRL dialect) and custom SPARQL update queries to perform reasoning managed by a Python server-application and (iii) we used html+javaScript for the **presentation tier**. Besides the inherent complexity of assembling those tiers together, the limitations of OWL2 for detecting the interactions, forced the use of multiple knowledge representation languages. Altogether, the resulting system was expensive to build and maintain. Therefore, in [114] we chose for SWI-Prolog via SWISH, which gave us an integrated environment for prototyping our theory using a single language for expressing our inference rules, benefiting understandability and maintainability. And finally, in the present work we also benefit from SWISH for the **presentation tier** by automatically building a graphical representation for the results.

This paper reports on the use of SWISH for prototyping a theory about representing knowledge underlying clinical guidelines with rules for detecting interactions among recommendations [113] (TMR - Transition-based Medical Recommendation). It allowed for efficiently developing a prototype that demonstrate the applicability of the theory using both some clinical guideline data manually modelled in RDF and existing clinical knowledge available as LOD such as Drugbank and Sider[3]. This online environment allows for scientists to check, reuse, and modify the code and data. The use of notebooks and graph visualizations favors the communication with scientists and domain-experts.

The remainder of this paper describes background knowledge about SWISH features (Sect. 6.2.1) and TMR theory (Sect. 6.2.2). Then Sect. 6.3 describes the use of SWISH for prototyping the TMR theory and Sect. 6.4 provides the final considerations.

6.2 BACKGROUND

6.2.1 *SWISH*

6.2.1.1 *The architecture of SWISH.*

SWISH started life as a Prolog oriented alternative to jsfiddle,[4] allowing people to write, save and share Prolog programs on the web. It was further developed for prototyping data fusion tasks on relational databases in a project called *DataLab*. For documentation and tutorial purposes we added *notebooks*, inspired by iPython notebook[5].

3 http://www.drugbank.ca and http://sideeffects.embl.de
4 https://jsfiddle.net
5 now called Jupyter http://jupyter.org

SWISH is based on *Pengines* [54], a SWI-Prolog infrastructure to create optionally sandboxed *P*rolog *engines* on a server and to interact with them similarly to the Prolog top level. A pengine is represented by a Prolog thread [103] and a temporal *module* for isolating the program. Prolog threads can be created quickly while they can access a shared Prolog and RDF database. SWISH adds a web interface and a versioned store for programs and notebooks to Pengines. Most of the web interface runs in the browser and is written in JavaScript. The server side is written in Prolog and provides a *semantic highlighting* service, documentation services, (optionally) authentication and a the program store (for details about the architecture see [103]).

6.2.1.2 *Using SWISH to prototype LOD applications*

SWISH is essentially a *wiki* providing collaborative editing of programs as well as user interfaces ranging from plain Prolog to full HTML5 web applications. At its core, Prolog allows a programmer to define abstraction over the *atomic* RDF relations that model the data at a level that is more suitable for defining the **application logic**. For example, if there are several datasets, e.g., d1 and d2, which describe *symptoms*, we can define a predicate that unifies this data, as showed in the code below. In the remainder of the application logic one could simply refer to `symptom(S)` which makes the application logic clearer and more compact while it greatly simplifies adding new datasets.

```
symptom(S) :- rdf(S, rdf:type, d1:'Symptom').
symptom(S) :- rdf(I, d2:hasSymptom, S),
              rdf(I, rdf:type, d2:'Disease').
```

The basic abstraction layer can be included into one or multiple programs that define the application logic. The abstraction layer as well as the application logic can be tested interactively and incrementally using one of the several interface methods provided by SWISH and described below.

- While a programmer is exploring the data to find suitable abstractions and prototype rules, the web-based version of the Prolog interactive *top level* is used to run queries on these predicates. Note that, in contrast with e.g., SPARQL, queries can be developed and tested in a modular fashion.
- Results can be presented in a domain specific format using *rendering plugins*. Standard plugins may be used to create tables,

charts and graphs (example in Fig. 6.3). After being loaded, these plugins are triggered by specifically shaped Prolog terms. For example, the *graphviz*[6] plugin creates a simple graph with two nodes from a term `digraph([a->b])`. Domain specific plugins can be added. These plugins can exploit all facilities of modern browsers, notably HTML5, SVG and JavaScript.

- Once domain experts need to be involved, tasks may be documented and scripted using a *notebook*. A notebook is an interactive document consisting of *markdown* text for explaining the notebook, canned *Prolog queries* that are executed by clicking a button and may use one of the above described rendering plugins to present results in an attractive form. Finally, there are program fragments that include shared programs and bind these together.
- Notebooks provide HTML cells that allow writing full web applications. Such applications may be implemented fully interactively using the CodeMirror based HTML/JavaScript editor. It is also possible to reuse externally developed JavaScript libraries and use the SWISH environment merely to configure the interface and bind it to the application logic.
- Finally, predicates can be accessed through an API. Currently there are clients for this API for Prolog, bash (Unix shell), JavaScript (browsers and NodeJS), Java and Ruby. Except for the bash client, all clients can request additional answers incrementally while multiple answers can be returned in *chunks* for optimal performance when downloading large sets of answers.

The different interfaces allow a variety of users to cooperate. Initially, the Prolog programmer typically wants access to the raw data rather than having to deal with complex domain specific representations. The SWISH *URI plugin* renders URIs as links that opens a page in the ClioPatria interface showing the URI in context. At the other end of the spectrum, e.g., in the use case application described in this article, we have computed interactions between medical guidelines and show these in a way that is understandable to a doctor. Once the structure of the data and the notion of guideline interaction diagrams becomes clear it is worthwhile exploiting the Graphviz plugin to make a somewhat crude but usable graphical representation

6 http://www.graphviz.org/

(see Fig. 6.3). This representation helps the programmer evaluating the ruleset and can be used in discussions with domain experts. In this scenario the programmer creates queries collaboratively with a doctor and discusses the result using the presented diagram. HTML notebook cells can be used to realised a modern web application that allows non-programmers to interact with the system. HTML notebook cells will be used in future work.

6.2.2 *Clinical Guideline Interactions Theory*

This section briefly introduces the main concepts of the TMR model, the interaction rules and the applied nanopublication structure (see [113] for more details).

The graphical schema illustrated in Fig. 6.1 introduces the main concepts, including one detected interaction. In the right side, the big rectangles (boxes) represent **causation beliefs** about **transitions** regarding a **property**, which are promoted by executing a **care action** (e.g., *blood pressure* does change from *high to normal* by *administering Thiazide*). A care action is represented as an ellipse within the causation-box, whilst a transition is described by the affected property (e.g., *blood coagulation* described on the top) and by the **pre- and post-situations** (whose values are described in the small boxes connected via directed arrow from pre to post). They either contain the values for the referred property (e.g. *normal* or *low*) or a question mark ('?') which indicates that the initial or final values are not in the data. The gray-shaded boxes indicate that the causation is imported from an external knowledge source, in this example, *Sider*. A **recommendation** is represented as a rounded box, with an identifier for reference (e.g., HT.1), and its deontic strength (e.g., *should* or

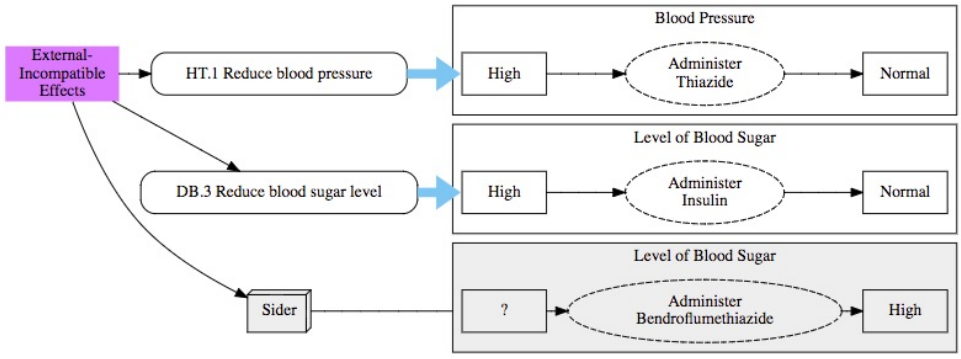

Figure 6.1: TMR graphical schema

should not) is indicated by a thick directed arrow connecting it to the causation-box, blue for positive and red for negative.

Finally, when combining clinical knowledge from different guidelines and external sources, a number of **interactions** can be identified, e.g., drugs recommended more than once or recommended drugs that have incompatible effects. These interactions can be described in the form of rules, exemplified here in an intuitive simplified format:

```
IF positive recommendation R1 to action A1 to change sit. S1 &
positive recommendation R2 to action A2 to change sit. S2 (≠S1) &
action A2' is believed to bring about S1 (A2'≠A1, A2'⊆A2 or A2'⊃A2)
THEN R1 and R2 have 'external' incompatible effect
```

When applying this rule to the aforementioned example we have the positive recommendation DB.3 (as **R1**) about *Administering Insulin* (**A1**) for changing *High Blood Sugar Level* (**S1**). Next, the recommendation HT.1 (as **R2**) advises *Thiazide Administration* (**A2**) to change something else. However, a subtype of *Thiazide* called *Bendroflumethiazide* (**A2'**) is known to bring about *High Blood Sugar Level* (**S1**) according to Sider. Therefore, there exists an **interaction** labeled in Fig. 6.1 as '*External Incompatible Effects*' connecting the interacting recommendations and refering to the external causation. Observe that this **interaction rule** is **generic**, i.e., it applies not only to this example but to whatever other similar scenario that might happen when combining multiple guidelines. This and other generic rules are formally defined in [113] for detecting other **external interactions** ('External Alternative Action' and 'External Incompatible Action'), which also use external knowledge sources, and **internal interactions**, i.e., the detection rules use only knowledge provided within the guideline ('Repeated Action', 'Alternative Actions', 'Contradictory Norms', 'Repairable Transitions').

6.2.3 *RDF Data as Nanopublications*

The aforementioned data is provided in RDF format using the *Nanopublication* structure, which presupposes the use of *Prov* vocabulary as explained in [113][7]. Roughly the goal is to look at recommendations and beliefs as assertions enriched with provenance data. To illustrate this, Fig. 6.2 presents a (simplified) schema for the nanopublication for the external causation belief mentioned in the previous section.

7 See http://nanopub.org/wordpress and http://www.w3.org/TR/prov-o.

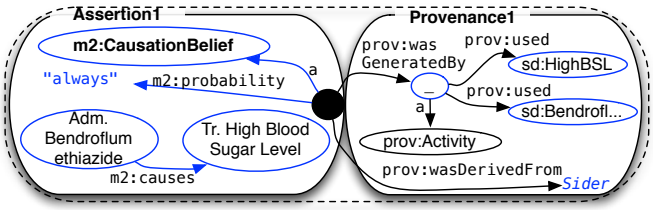

Figure 6.2: Nanopublication (simplified) schema proposed for causation belief imported from Sider.

On the left hand side the named-graph *Assertion1* is of type *Causation belief* with probability *always* about *Adm. Bendroflumethiazide* causing a transition to *High Blood Sugar Level*. On the right hand side, the named-graph *Provenance1* says the assertion was derived from *Sider* and was generated by an activity that used two resources from Sider, i.e., the drug and the side-effect URIs. Within this structure, this and other provenance information can be stored for further reference. For example, the beliefs imported from a certain source can be considered not trustworthy and thefore can be skipped by the inference rules. Or the external resources on which a certain causation relies can be inspected by the experts in order to decide "on the fly" whether to trust it or not. The graph visualization makes these resources accessible by clicking the external source.

6.3 A SWISH-BASED PROTOTYPE FOR CLINICAL GUIDELINES INTERACTIONS

This section describes the usage of SWISH to implement the aforementioned tiers and a case-study for illustrating the obtained results.

6.3.1 *Storage tier: TMR Model and Data as SWI-Prolog Facts*

SWISH relies on Cliopatria as a triple store where the OWL-RDF files for the TMR model and the guidelines' data are loaded via the web interface. The external data, retrieved from the internet in RDF format, is also loaded into the Cliopatria server. As previously mentioned, the tight integration of Cliopatria with the Prolog language makes the loaded RDF data available as normal Prolog facts. SWI-Prolog queries can be used for accessing the RDF data, such as the following one retrieves all the Event types that are believed to cause another one:

```
?- rdf(EventT1, vocab:'causes', EventT2, CBelief)
```

6.3.2 *Application logic tier: SWI-Prolog Rules*

Multiple Prolog programs are used to define rules for: (i) checking data properties: defining predicates that return True if a certain graph pattern is found and False otherwise (negation as failure); (ii) retrieving data: defining predicates that abstract from complex graph patterns matching; (iii) asserting data: defining rules that assert new RDF triples given given that a certain graph patterns is found.

The code is implemented using the SWISH editor that allows for modular testing of parts of the code. We divided the code into files that address different aspects of the proposed theory. We briefly describe herewith the main issues addressed by three relevant files and illustrate them with one rule.

`guidelines.pl` This file contains rules for querying and manipulating clinical data according to the TMR model. Below, we present one predicate for retrieving data about the causation relation between two event types:

```
causes(EventT1, EventT2, Prob, CB) :-
 rdfs_individual_of(CB, vocab:'CausationBelief'),
 rdf(CB, vocab:'probability', literal(type(xsd:string, Prob)), CB),
 rdf(EventT1, vocab:'causes', EventT2, CB).
```

`externalSources.pl` External sources may adopt different vocabularies for describing similar data. To allow for the interaction rules to be independent of specific vocabularies, we provide importing rules for each source that reinterprets the data as beliefs according to the TMR model. Naturally, we assume that the local data and the external data share some properties (e.g., the UMLS code) that can be used to align the entities. Below we illustrate the rule for asserting new data by importing side effects from Sider as causation beliefs:

```
siderAssertCausationSideEffect :-
 forall(
  (( rdf(_, vocab:'hasTransformableSituation', Situation)
    ; rdf(_, vocab:'hasExpectedSituation', Situation)),
    rdf(Situation, owl:sameAs, SideEffect),
    rdf(DrugSider, sider:'sideEffect', SideEffect),
    rdf(DrugSider, owl:sameAs, DrugType),
    rdfs_individual_of(DrugType, vocab:'DrugType'),
    rdf(Action, vocab:'administrationOf', DrugType)
  ),
  ( assertPartialTransition(Situation, NewTr),
    rdf_global_id(data:'sider', SourceURI),
    assertCausation(Action, NewTr, 'always', SourceURI, NanopubURI),
```

```
    assertProvResourceUsed(NanopubURI, DrugSider),
    assertProvResourceUsed(NanopubURI, SideEffect))).
```

interactionRules.pl Internal and external interaction rules are described in this file, where different types of interactions are inferred based on patterns of recommendations and beliefs. Hereby we illustrate the rule for asserting interactions of type *External Incompatible Effects* (discussed as a simplified format in Sect 6.2.2):

```
detectExternalIncompatibleEffect :-
forall(
(regulates(Reg, Norm1, Act1, Strength, CB1),
 causes(Act1, Tr1, 'always', CB1),
 ((Strength = 'should', rdf(Tr1, vocab:'hasTransformableSituation', St1))
 ;(Strength = 'should-not', rdf(Tr1, vocab:'hasExpectedSituation',St1))),
 causes(Act, Tr, 'always', CB),  different(Act, Act1),
 rdf(Tr, vocab:'hasExpectedSituation', St1),
 regulates(Reg, Norm2, Act2, 'should', CB2),
 relatedTypes(Act, Act2),    different(CB2, CB)
),
(existsInteraction('ExternalIncompatibleEffects', Norm1, Norm2, CB))).
```

6.3.3 *Presentation tier: Notebook and Graph Visualization*

For the presentation tier we used graphviz to produce a graph visualization that resembles the graphical schema manually designed in previous work and that was successfully used for communication with domain experts. The Prolog code required to extract the clinical data to be plotted and create a representation suitable for the graphviz plugin is accessible via the main file **interaction_graph.pl**.

To enhance the communication with other researchers and domain experts we composed a SWISH notebooks[8] that explains step by step the initialization (data acquisition and inferencing) of a case study described in [113] on combining parts from 3 guidelines: Diabetes, Hypertension and Osteoarthritis. The notebook allows for users to access partial results achieved in each step of the experiment. Fig. 6.3 presents a screenshot of the notebook containing a graph visualization of the case-study described in this section. Each of the elements in the graph can be clicked to explore its related data in the ClioPatria interface. In particular, the small box representing external data redirects a page referring to the used external resources.

8 See https://tinyurl.com/CGGraph and https://tinyurl.com/CGMainteance

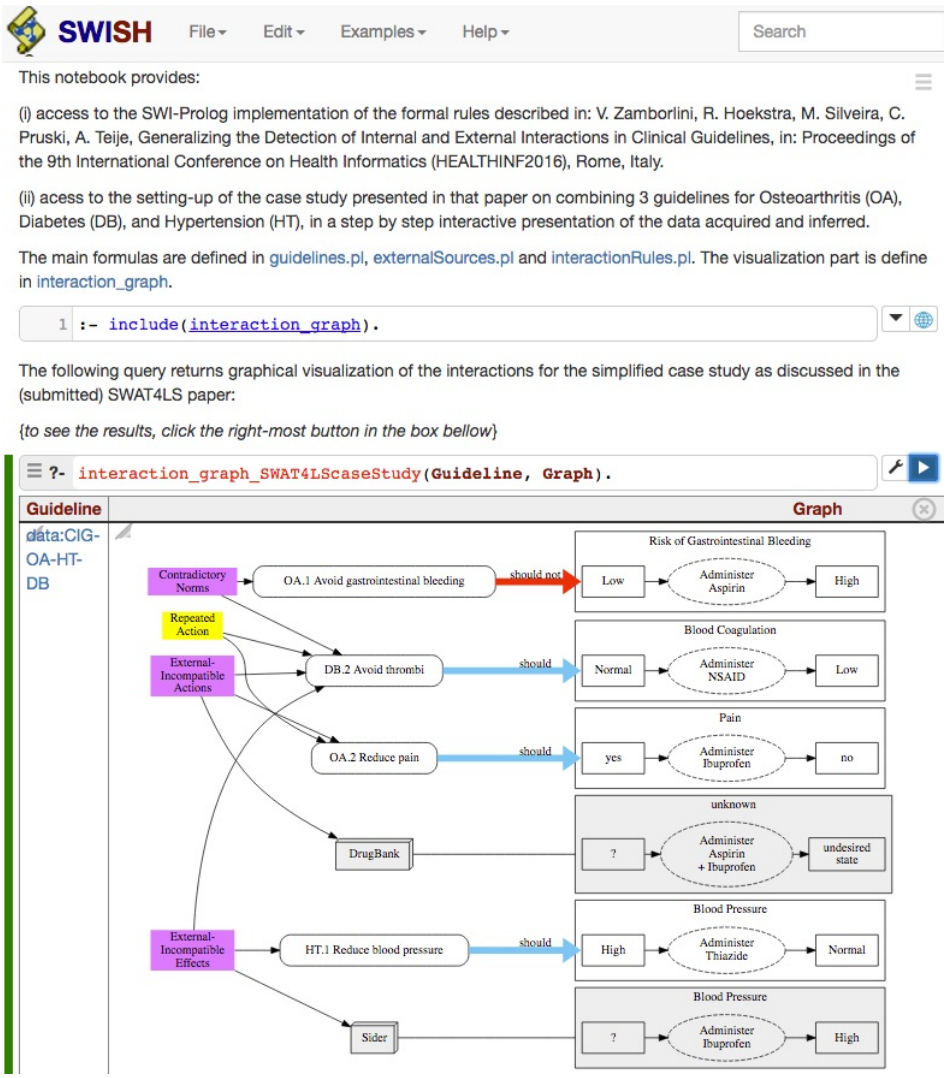

Figure 6.3: SWISH notebook with visualization of case study on combining guidelines for OA+HT+DB

The case-study depicted in Fig. 6.3 demonstrates the detection of interactions[9] of the original case study [113]. The presented recommendations are:

DIABETES (DB) :
(DB.2) *Should administer NSAID to reduce blood coagulation*
OSTEOARTHRITIS (OA) :
(OA.1) *Should NOT adm. Aspirin to avoid increasing risk of gastrointestinal bleeding*. (OA.2) *Should administer Ibuprofen to reduce pain*

9 The example is reduced due to space restrictions, but recommendations' ID's are the original.

HYPERTENSION (HT) :
(HT.1) *Should administer Thiazide to reduce the blood pressure*

Finally, the following interactions are detected by applying the interaction rules:

1. DB.2 and OA.1 can be **contradictory norms** since the *NSAID* in DB.2 can be prescribed as *Aspirin*, which is non-recommended by OA.1.
2. DB.2 and OA.2 can be **repeated action** since the *NSAID* in DB.2 can be prescribed as *Ibuprofen*, which is already recommended by OA.2.
3. DB.2 and OA.2 potentially interact since the *NSAID* in DB.2 can be prescribed as *Aspirin* which is said in *Drugbank* to be **incompatible** with *Ibuprofen* recommended by OA.2
4. HT.1 and DB.2 potentially interact because the situation that is meant to be changed by HT.1 (*high blood pressure*), can be promoted as **side-effect** if *NSAID* in DB.2 is prescribed as *Ibuprofen*, according to *Sider*.

6.4 DISCUSSION & CONCLUSION

This paper shows that SWISH was successfully applied for prototyping the Clinical Guideline Interaction theory [113] (extended version in [118]) by providing a single environment that supports the three tiers of a semantic web application: storage, application logic and presentation. This allows for the medical informatics researcher to focus on the solution to be provided rather than on connecting the tiers together using different languages and tools. Prolog alone now provides the rule-based reasoning required for our purpose, as opposed to the previous implementation when we tried to be compliant to the "Semantic Web Standard Languages" such as OWL and SPARQL (see Sect. 6.1). Finally, the resulting prototype favors the communication with other researchers and domain experts by providing (i) a notebook where the case-study experiment is described step by step and (ii) the dynamic graphical visualization of the inferred interactions. In the future we will improve the implementation to calculate the relevance of interactions so that they can be filtered to favor readability, specifically when many interactions are inferred. We will provide a more user-friendly interface prototype by adding HTML cells to the notebook that allows domain experts to interact with the system. A dedicated web application can be built using any web application framework that reuses our application logic tier by means of the API provided by the Prolog/SWISH.