WebQR: Building a Knowledge Representation Application on the Semantic Web

Wouter Beek[1], Sander Latour[2], and Stefan Schlobach[1]

[1] VU University Amsterdam
[2] Perceptum BV

Abstract. The Semantic Web (SW) was originally positioned as a combination of Knowledge Representation (KR) and the Web. However, most applications that use SW data today lean more towards the Information Retrieval spectrum. The reason for this is that traditional KR systems are designed to work with datasets that are small, curated, homogeneous, and application-specific. However, the SW is large-scale, messy, and heterogeneous. We present a software project called WebQR in which we take a traditional KR application: Qualitative Reasoning, but implement it on top of the SW. We show that reimagining a traditional KR system in the context of the SW opens up many opportunities but presents several problems for application development as well.

1 Introduction

The Semantic Web (SW) was originally positioned by Tim Berners-Lee as a combination of Knowledge Representation (KR) and the Web [2]. However, most applications that use SW data today lean more towards the Information Retrieval and/or Databases spectrum (e.g., search, recommendation, data integration), than to the original gamut of KR. The reason for this is that traditional KR applications have worked with datasets that are small, curated, homogeneous, and optimized for a specific system. In sharp contrast to that, the SW is large-scale, messy, heterogeneous, and application-independent. We present WebQR, which takes a traditional KR application: Qualitative Reasoning (QR), but implements this on top of the SW. As such, WebQR provides a significant departure from the way in which contemporary QR modeling tools are designed. It also provides an interesting use case for implementing a traditional KR system on the SW (denoted as KR+SW systems from now on).

Overview

Section 2 gives an overview of existing QR modeling tools, and identifies their major culprits. Section 3 enumerates the design principles that guide WebQR developed. Section 4 explains how these abstract design principles are made concrete. Key lessons that were learned during development are enumerated in Section 5.

2 Related work

Qualitative Reasoning [5] is a traditional KR approach in which declarative knowledge about the continuous behavior of a physical system, e.g. the combustion engine of a car, is used in order to simulate the future behavior of that system. The behavior of the system is not represented numerically, but symbolically, making it easy to process by humans. By encoding knowledge declaratively, a QR model provides handles for giving advanced forms of user feedback such as allowing simulation results to be explained.

Betty's Brain [8], VModel [6] and Garp/DynaLearn [3] are QR modeling tools that target ground school, middle school, and university-level students, respectively. Creating a causal model in Betty's Brain is as simple as creating a traditional concept map. Modeling in the latter two tools requires the modeler to learn a tool-specific language. Formulating a model in these languages requires various ontological distinctions to be made by the modeler, e.g. whether something is an object (stable) or a quantity (varying over time). In line with this, Betty's Brain can only perform relatively simple simulations, while the latter two are more advanced in this respect. Qualitative Concept Map (QCM) [4] and DAE-hybrid [7] are QR modeling tools that target professional modelers rather than students. The latter tool makes smart reuse of existing software projects such as Modelica[3].

In general, we see that existing QR modeling tools are either expressive and difficult to use, or easy to use but limited in terms of the simulations they can perform. Despite their similarity, these tools make no use of each others software components, nor do they support each others modeling languages, thereby limiting model interchange and reuse. Another commonality is that they all use WIMP (Windows, Icons, Menu, Pointer) GUIs, and none of them (re)uses modern Web standards and/or Web services. With the exception of DAE, they do not make use of related projects such as concept mapping tools and visual editors. With the exception of Betty's Brain and DAE existing QR tools seem to have a small userbase and have problems with being sustained over a longer period of time.

3 Development principles

The development of WebQR is guided by a core set of design principles, which we posit are generic enough to transfer to the development of other KR+SW systems.

Principle 1 *Represent QR models as 5-star [1] Linked Data.*

Principle 1 makes it possible for QR models to be exchanged as Linked Data. Even if different tools define different schemas some of the core QR concepts can still be interlinked thereby promoting model reuse.

[3] `https://www.openmodelica.org/`

Principle 2 *Replace QR application components with generic SW services.*

As we saw in Section 2, existing KR systems are often developed in isolation from other projects, creating their own model editor, representation language, reasoning engine, etc. As a consequence KR systems are renown for being code giants that are difficult to maintain and reuse. If a QR model is represented as Linked Data existing (generic) SW services can be used to replace custom QR application components (Principle 2). For example, semantic recommendation systems can guide QR model construction, SW reasoners can perform (parts of) the QR simulation process, Web-based concept modeling tools serve as the basis for the modeling interface, visualization widgets can give views on simulation outcomes. Reusing (Semantic) Web services has many benefits such as reducing the code base and distributing the development effort.

Principle 3 *Replace QR content libraries with SW data.*

In traditional QR systems it takes a lot of effort to build a model that is able to display non-trivial behavior. Only some QR environments come with interesting content out-of-the-box. Since existing content libraries have to be created by knowledgable experts and are not reusable across systems, they are typically quite small. Adherence to Principle 3 would bring an unprecedented amount of content to a QR application. The SW contains elaborate descriptions of millions of concepts that could be of interest to a QR modeler. Even if only common sense knowledge would be extracted from the SW (e.g., that water is a liquid that freezes at 0 and boils at 100 degrees Celsius) this would already reduce the more repetitive parts of QR modeling, thereby lowering the barrier towards actually using a QR application.

Principle 4 *Publish QR-created content as part of the LOD cloud.*

Principle 4 basically states that WebQR should not only read from the SW, but should also write to it. In a QR model causal relations are expressed between quantities, e.g. "if the volume of a gas decreases its pressure increases". Such causal knowledge of how physical systems behave is typically missing from today's SW. Therefore the collection of causal relations that are created in WebQR would be a valuable crowd-sourced contribution to the Linked Open Data (LOD) cloud.

4 Implementation

Client side: In line with Principle 2, the WebQR UI is post-WIMP. It does not have the complexity of multiple windows, toolbars, menus, status bars, etc. but has a Web-based (HTML5) UI that is build on top of a UI that is used for visually modeling advanced (modal) logical models.[4] This is where the user creates and

[4] `http://rkirsling.github.io/modallogic/` which in turn uses `http://d3js.org`

manipulates her model. In this UI QR models are visualized as Scalable Vector Graphics (SVG) with annotated styles and animations (CSS3) to make them look more appealing. User interaction with the canvas is tablet/touch-first and desktop/mouse-compatible. Touching the canvas adds a new concept. Touching a concept displays additional information about it. Dragging a line between two concepts adds a (directed) relation. The second UI component is a floating box that appears whenever the user needs to disambiguate an action. In line with Principle 3 it contains a machine-generated, ranked list of suggestions queried from the SW (e.g. a list of relevant relations after drawing a line between two concepts) and an input field for custom text. Use of the custom text field is minimized by supporting the user with intelligent suggestions most of the time (examples at the end of this section). For some user interactions it is difficult to give suggestions, e.g. the first concept a user creates cannot easily be guessed since there is no context. The client-side codebase is concise, mostly reusing existing JavaScript libraries.

Server-side: The server-side is where user-created concepts and relations are represented by SW resources (Principle 1). The LOD scraping library lodCache[5] enriches the server-side store by automatically the LOD cloud and adding relevant data about the user-created resources (Principle 3). The WebQR server is written in SWI-Prolog[6], a high-level relational programming language that allows the reasoning aspects of QR to be written concisely. The server-side is not a stand-alone component, but a plugin of ClioPatria[7], a triple store that has been used in both research and production environments. Content created within WebQR, e.g. the causal relations that have been defined by users, are exposed as an external service by using a SPARQL endpoint (Principle 4).

Client/server API: As can be glanced from the foregoing, the client and server sides of WebQR are very different. The strength of JavaScript to provide a dynamic light-weight Web-based UI is coupled with the strength of a knowledge-intensive triple store in order to produce an optimal overall product. This is realized by having an efficient client/server API that is based on REST principles. Data is interchanged between the client and server components seamlessly because of the use of JSON which is native to JavaScript and is converted to/from SWI-Prolog-specific dictionaries.[9]

SW reasoning: WebQR uses SW reasoning techniques (Principle 2) in order to derive new knowledge throughout its modeling and simulation components. For example, Statements 1 and 2 are part of the WebQR vocabulary. Suppose that Statement 3 is added in the user interface by drawing a line from *Temperature* to *Pressure*. Statements 4 and 5 are now inferred by WebQR. Although the deduction may be considered trivial, existing tools require the user to explicitly specify that *Temperature* and *Pressure* are quantities while WebQR infers this. Notice that such inference requires no custom code since these are generic,

[5] `https://github.com/wouterbeek/lodCache`
[6] `http://www.swi-prolog.org`
[7] `http://cliopatria.swi-prolog.org`

application-independent RDF 1.1 entailments.

$$\langle \texttt{qr:positivelyInfluences}, \texttt{rdfs:domain}, \texttt{qr:Quantity} \rangle \quad (1)$$

$$\langle \texttt{qr:positivelyInfluences}, \texttt{rdfs:range}, \texttt{qr:Quantity} \rangle \quad (2)$$

$$\langle \texttt{dbpedia:Temperature}, \texttt{qr:positivelyInfluences}, \texttt{dbpedia:Pressure} \rangle \quad (3)$$

$$\langle \texttt{dbpedia:Temperature}, \texttt{rdf:type}, \texttt{qr:Quantity} \rangle \quad (4)$$

$$\langle \texttt{dbpedia:Pressure}, \texttt{rdf:type}, \texttt{qr:Quantity} \rangle \quad (5)$$

Reasoning with imperfect knowledge: As mentioned in Section 1 SW data is messy. This means that inferences like the one mentioned above cannot always be made and are sometimes made incorrectly. As mentioned in Section 3 we believe this provides a new challenge for traditional KR approaches. In line with Principles 1 and 2 WebQR implements non-standard reasoning capabilities in order to make non-trivial use of SW data. Again, suppose a user adds the concepts *Water* and *Temperature*. WebQR can prioritize the relations that may be added between those concepts based on knowledge available in the LOD cloud. This is done by finding the shortest path within the DBpedia SKOS categories hierarchy between the user-created concepts and the concepts `dbpedia:Category:Objects` and `dbpedia:Category:Quantity` that are aligned to the WebQR vocabulary. Depending on which shortest path is shorter, user-created concepts are asserted to be either objects (stable) or quantities (varying over time). Based on such assertions the relation `qr:hasQuantity` is assigned a higher likelihood than relation `qr:positivelyInfluences` thereby aiding the modeling process.

5 Lessons learned

We now enumerate the most important lessons that were learned during WebQR development.

Lesson 1 *Be aware that even some very simple (common sense) knowledge cannot be easily extracted from the SW today.*

Lesson 1 is a correction of Principle 3: today's LOD cloud does not contain some of the most obvious knowledge that one would like to use in a KR system. For example `dbpedia:Properties_of_water` contains the distance between the H and O atoms of H_2O (being 95.84pm), but does not include the boiling point of water. The boiling point of water is, however, necessary for simulating the changing behavior of a water substance that has a temperature quantity that can change its value over time. Such data can, of course, be added to the LOD cloud but this requires extra work on the side of the modeler.

Lesson 2 *Be aware that some straightforward operations are difficult to perform on the SW.*

Lesson 2 is a correction of Principle 2 and is best explained with an example. If a modeler adds a concept with the name "Monkey" it makes sense to show a

ranked list of suggested IRIs with the most common meaning appearing at the top. However, contemporary SW infrastructure (SPARQL in this case) does not support a notion of "most common meaning", returning a song about monkeys, a band, and the game Monkey Island before returning the animal species.

Lesson 3 *Do not expect all reasoning operations to be straightforwardly performed by generic SW reasoners.*

Even though we were able to perform some of the reasoning in terms of SW representations several rules in the simulation engine were difficult to translate to generic SW/OWL parlance while maintaining acceptable performance. Especially the introduction of temporal states in the simulation results, each describing a self-contained description of the modeled system, requires non-trivial extensions of OWL. Development of WebQR currently focuses on writing the remaining part of the simulation engine in terms of DL-calculable vocabularies.

6 Conclusion

We have been able to recreate some of the functionality that existing QR modeling environments have with a codebase that is tiny when compared to the functionally-equivalent subparts of those related projects. We claim that our approach towards reimagining existing KR systems in the context of the (Semantic) Web leads to application that are more lightweight, more sustainable, more reusable, and more usable. We hope to inspire other developers of KR+SW applications to follow a similar route and make optimal use of the rich data source the SW provides.

References

1. Berners-Lee, T.: Linked Data. `http://www.w3.org/DesignIssues/LinkedData.html` (2010)
2. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (2001)
3. Bredeweg, B., Liem, J., Beek, W., Linnebank, F., et al.: DynaLearn: An Intelligent Learning Environment for Learning Conceptual Knowledge . AI Magazine 34(4), 46–65 (2013)
4. Dehghani, M., Forbus, K.: QCM: A QP-Based Concept Map System. In: Proc. of the 23nd Int. Workshop on Qualitative Reasoning (2009)
5. Forbus, K.: Qualitative Modeling. In: van Harmelen, F., Lifschitz, V., Porter, B. (eds.) Handbook of Knowledge Representation, chap. 9, pp. 361–394. Elsevier (2008)
6. Forbus, K., Carney, K., Sherin, B., Ureel, L.: VModel: A Visual Qualitative Modeling Environment for Middle-School Students . In: Proc. of the 16th Innovative Applications of AI Conf. (July 2004)
7. Klenk, M., de Kleer, J., Bobrow, D., Janssen, B.: Qualitative Reasoning with Modelica Models. In: Proc. of the 28th AAAI Conference on AI (2014)
8. Leelawong, K., Biswas, G.: Designing Learning by Teaching Agents: The Betty's Brain System. Int. J. of AI in Education 18(3), 181–208 (2008)
9. Wielemaker, J.: SWI-Prolog Version 7 Extensions. In: Proc. of CICLOPS (2014)

