Toward a Reference Architecture for Traceability in SBVR-based Systems

Lloyd Rutledge
Open University of the Netherlands
Heerlen, The Netherlands
Lloyd.Rutledge@ou.nl

Rudy Italiaander
Tax and Customs Administration
of The Netherlands
Utrecht, The Netherlands

Abstract

We present an initial reference architecture for traceability in SBVR-based systems. It facilitates rule-based development that lets end users trace interface behavior back to the human decision points that lead to it. This closes a feedback loop, which facilitates agile development. The architecture is based on Web standards to generalize comparison and implementation.

It begins with the human process of linking document excerpts to the SBVR code that defines them. The next step transforms SBVR into computer code that implements it. Reasoners then form conclusions by applying the rules to data. They can also provide rudimentary explanations for these conclusions. The system then provides an end-user interface to all this rule-derived information. The core challenge here is maintaining the data needed to trace back through these layers, so end-user feedback can improve the entire development process.

1 Introduction

Semantics of Business Vocabulary and Business Rules (SBVR) provides a controlled natural language (CNL) for data models and business rules (Group, 2006). It can be mapped to more easily readable equivalents such as RuleSpeak (Spreeuwenberg and Healy, 2010). Such CNLs provide a step between stakeholders and programmers that helps designers communicate with both.

Rule-based systems constrain users to follow rules but often cannot explain where the rules come from or why they exist. This is because development of such systems often follows a sequential, or waterfall, approach. Analysis of legal text or policy documents leads to descriptions in SBVR. Other people then write program code for this SBVR. However, nothing in the code, and thus nothing in the resulting system interface, necessarily leads back to its source in SBVR or further back to the source documentation. The original motivation for the rules becomes effectively forgotten. End users end up several layers of one-way traffic away from understanding or potentially influencing change in the documents and discussions that form the rules they must follow.

Traceability in rapid prototypes of rule-based systems adds resilience to the process of forming laws and implementing them (Ausems et al., 2021). However, the mapping from human-readable business rules to computer-processed logic on the Semantic Web is complex (Spreeuwenberg and Gerrits, 2006). A reference architecture could help implement traceability in this complex mapping.

We propose forming a reference architecture in which this software development for business rule systems is less waterfall and more agile. It provides end users with transparency to the origins of rules in their systems so they can take part in their longer-term maintenance. In addition, development can then more easily include end-user representatives who provide short-term feedback over the effectiveness of both the documented rules and how the SBVR rules and resulting business systems implement them. Problems involving mismatches between high-level agreements and how well end-users of the system can appreciate and execute them can then be detected and addressed earlier and more often.

2 Reference Architecture

Fig. 1 illustrates our reference architecture. It has three broader components. First is the *development* of the specifications and code by people. Then comes the *automation*, which compiles the code into the business system. Finally, automation provides the system *interface* that the *end user* interacts with. The first step in development is *discussion* between human stakeholders, which results in *documents*, such as legislation or contracts, that define agreements reached. An *analysis* of the documents provides *annotation* of it, which shows the

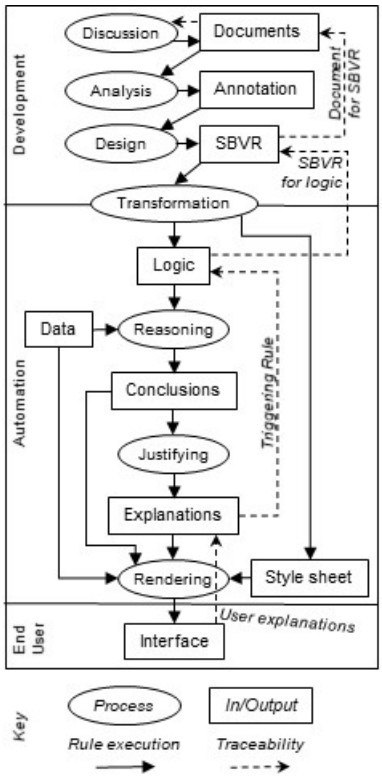

Figure 1: Our reference architecture for traceability in SBVR-based systems.

phrases for primary components in the system's data model and rules. The next process is the *design* of *SBVR* code for the model and rules.

With the SBVR code established, the *automation* of it is ready to start. The *transformation* of human-readable SBVR to machine-processable code can be human, automated or a combination of both. The resulting *logic* code defines the model and rules in a format a computer can process. The *reasoning* process applies the logic to the *data* in the system to generate *conclusions*. Conclusions can infer additional data, or find violations that the given data triggers. Core to traceability is automatically *justifying* these conclusions to provide *explanation* of them to the end-user.

The results of this automated processing need, of course, to be presented to the end user. The transformation of SBVR into system code includes not just the logic for data processing but also the *style* of this presentation. The *rendering* step processes this style code to determining how the data, conclusions and explanations should appear in the *interface*. This interface should not only let the user see all this information, but also trace back from it to the SBVR code and fragments of the documents from which the information originates.

3 Related Work

The software Cognitatie supports annotation of documents for conversion into knowledge models, including rules, placing it in the development block of our architecture (PNA Group). The tool RuleXpress processes business rules against a data set, functioning as the automation block (RuleArts). The Web service s2o[1] is a converter from SBVR to the Semantic Web ontology language OWL-2, thus automating the transformation process (Karpovic et al., 2014). Also on the Semantic Web, the ontology editor Protégé provides all automation processes, from reasoning on logic and data, to providing explanations (Musen and Protégé, 2015).

The Fresnel vocabulary for Semantic Web browsers is a format for style sheets mapping Semantic Web data to end-user interfaces to it (Pietriga et al., 2006). In earlier work, we developed software for generating Fresnel code for any given ontology (Rutledge et al., 2016).

HTML provides, perhaps obviously, a well applicable standard for source document descriptions of rules. We propose also having HTML browsers in the end user interface of our architecture provide direct user access to these source documents. For the system between rule documents and end users, the Semantic Web provides standards for data, its structure and rule-based logic applied to it. HTML and the Semantic Web both use URLs, which allows processing source document excerpts like any other Semantic Web concepts. Davis gives an overview of different ways to annotate HTML documents in the context of CNLs (Davis, 2013). Some techniques do not require editing of the annotated documents. The text to be annotated can be both CNL and the source document text they encode.

The RACE Reasoner for Attempto Controlled English processes input CNL logic and facts and outputs conclusions and reasoning explanations in CNL as well (Fuchs et al., 2008). The reference architecture we propose here models much of this processing, including processing CNL text to form conclusions. One difference is that our architecture does not explicitly account for generating explanations of conclusions in CNL. What the architecture does do here is provide links to the source CNL and document texts that defines the rules involved in a reasoning. As such, either the user can navigate to those document sources, or a natural language

[1] https://s2o.isd.ktu.lt/

generator can use these links to get the information needed to provide a readable explanation.

The course Rule-based Design that we teach at the Open University in the Netherlands uses the business rule reasoner Ampersand, which specializes in legal reasoning (Joosten, 2017). While Ampersand has no automatic generation of explanations behind its reasoning as RACE does, the Ampersand syntax provides the human editor some constructs for traceability (AmpersandTarski GitBook project). One is the RULE construct, which provides template-based explanation text, which the interface shows the user when the given rule is violated. This template text has placeholders for labels of specified components of the rule. In addition, the PURPOSE construct offers a human-readable citation of the document source for a rule or component of the data model. We propose here a PURPOSE-like citation for ontology components, but one that is machine-readable, and can link to source CNL chunks as well as document fragments.

Much real-world data exists on the Semantic Web, which matters for both practical application and academic validation of research results. For example, in earlier work, we created large amounts of data for applying Semantic Web logic to in order to generate statistics about the efficiency of that logic (Italiaander, 2019). Including the Semantic Web in our rule-based system development architecture facilitates testing and analysis with large amounts of existing or synthetic data.

4 Illustrative Scenario

We illustrate our reference architecture by applying it in an example scenario. This scenario derives from the fictional business EU-Rent, with its text descriptions and SBVR code (Object Management Group, 2016). In particular, we use a rule in EU-Rent's section G.6.7 "Car Movements". This EU-Rent section starts with the text "Car movements meet the business requirement that a car of a given group has to be moved between branches". We treat this as source text in our reference architecture. EU-Rent then provides the following SBVR rule it derives from that text: "*Necessity: Each car movement includes exactly one receiving branch*".

We convert this SBVR rule into machine-processable logic standard OWL with the tool s2o (Karpovic et al., 2014). Its equivalent in s2o's SBVR dialect that this scenario puts in s2o's rules field is "*It is necessary that car_movement has_receiving_branch exactly 1 branch*". The conversion's output is OWL code that makes car movements a subclass of the class of things with exactly one assignment for the property has_receiving_branch. The OWL constructs it uses are restrictions and qualified cardinality. This code corresponds to the logic code in the architecture.

A trigger for this rule is a car movement that is assigned to more than one different branch, resulting in an OWL inconsistency. In OWL, an inconsistency is a collection of facts and rules that cannot all be true. Fig. 2 shows a display from Protégé for such an inconsistency in our scenario. The top right of this display shows these two conflicting receiving branch assignments in red lettering, which in Protégé indicates an inconsistency. In the reference architecture, this shows a conclusion resulting from Protégé's OWL-defined reasoning.

When recognizing such an inconsistency, Protégé announces it to the user and then lets the user request an explanation for it. The explanation that Protégé then provides for our scenario appears in the lower part of Fig. 2. In this display under "Explanation 1", the first line shows the OWL code for the rule that is triggered. The other lines show the data that collectively triggers this rule. What traceability demands here is that the end-user be able to browse from the top line back to the SBVR code that derived it, and then back further to the document text that SBVR code defines.

5 Traceability

The solid arrows in the reference architecture diagram in Fig. 1 show the traditional view without traceability: we generate systems from rules, but can then easily forget the rules we generated the systems from. The dotted arrows show the primary places where traceability needs implementing. Each has a label describing which layer of traceability it provides. We describe them here in the order in which tracing happens here: from lower to higher in the diagram.

User explanations are the presentation of the explanations behind logical conclusions that appear in the interface for the end user. First, we should note that the process of justifying conclusions to provide explanations for them provides a layer of traceability. Logic systems such as Protégé implemented justification later than the simple display of conclusions, as it is a substantial technical challenge beyond generating the conclusions. Fig. 2

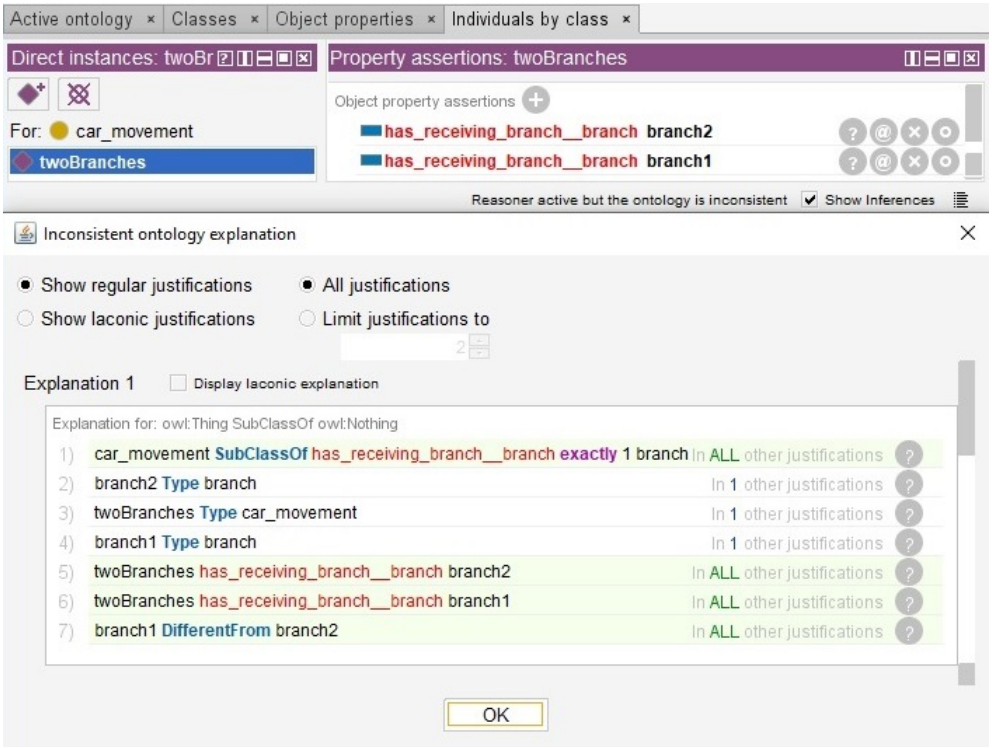

Figure 2: Protégé display of an inconsistency with its explanation.

shows how, in our scenario, Protégé informs the user that an inconsistency has occurred and which data it involves. Protégé shows an explanation only when and if the user requests it.

As this demo shows, this aspect of traceability is substantially implemented in Semantic Web systems. However, most end users cannot understand it, as it is an unstructured collection of all the data involved in forming the logical conclusion. It does have all the relevant information for an explanation, and software could process it for a CNL explanation, such as RACE does for its internal logic. However, the focus in this reference architecture is on how to access this information instead of how to present it understandably. In addition, our architecture focuses more on traceability as accessing the documented origins behind the logical rule applied in a situation.

The next dotted arrow, "*triggering rule*", in the architecture diagram goes from explanations back up to logic program code. This is where our architecture focuses on human-accessible traceability instead of the automated generation of human-understandable text, as other research does. Instead of processing all of the triples of an explanation into text, we propose focusing on the information regarding the rule that is triggered. That rule can then, in turn, lead to the CNL text and original document text that people wrote earlier to describe that rule. The other triples in the explanation data describe mainly the data that triggers the rule, instead of the triggered rule itself. The challenge here is to find the code for the triggered rule from all the data for an explanation.

In Fig. 2's scenario, the rule triggered is in the first line of the explanation, especially where it says "exactly 1". This line refers to the code for an OWL restriction, which has Semantic Web-defined properties that define the restriction. One property assignment declares the type of cardinality as exactly one. Another assigns it to the property "has receiving branch". This restriction resource is then assigned as a superclass of car movements. This is how the restriction defines that all movements much have exactly one receiving branch.

The question is then: where does this OWL-code come from? The "*SBVR for logic*" dotted arrow leads from the machine-processed logic code back up to that rule's definition in layperson-readable SBVR text. A platform providing this function would need to support links from chunks of logic code, such as OWL, to fragments of SBVR text. HTML could encode these fragments of SBVR text as anchors, giving each a URL. Then in the au-

tomation layer, Semantic Web code can associate these SBVR fragment URLs with the Semantic Web-defined logical constructions that implement them. What the end user then sees in the interface is navigable web browser links from the display of the triggered rule in the explanation to the corresponding piece of SBVR. In our example, this would be an additional property linking the restriction to the fragment of SBVR code defining it. The value would simply be a URL with a fragment identifier linked to the relevant portion of SBVR code in its online document.

The next step, "*document for SBVR*", lets the user go from such a segment of SBVR code to the extract of documentation that is its source. Again, this could be a URL leading from that SBVR fragment to the fragment of the other document providing its original definition. If the SBVR is in HTML in order to link it to the Semantic Web, then this HTML can also give the user a hyperlink from SBVR to the corresponding portions of the HTML-defined source document. The end user can then browse from reasoned conclusions through the triggered rule in an explanation, back to the rule's SBVR source, and then to the document text. This enables the final step in rule traceability: discussing the document text with those who formed it, perhaps in order to improve it, which in turn improves how well system end users can understand and carry out those rules. With such a reference architecture for traceability, we aim to help developers of software for this architecture's processes craft exchange formats that let all components of the broader platform exchange all information needed to provide a traceable whole.

6 Acknowledgements

We thank Mariette Lokin for her helpful comments and suggestions about this architecture and paper.

References

AmpersandTarski GitBook project. Documentation of Ampersand. https://ampersandtarski.gitbook.io/documentation/. Accessed: 2021-07-05.

Anouschka Ausems, John Bulles, and Mariette Lokin. 2021. *Wetsanalyse voor een werkbare uitvoering van wetgeving met ICT*. Boom Juridische Uitgevers.

Brian Patrick Davis. 2013. *On Applying Controlled Natural Languages for Ontology Authoring and Semantic Annotation*. Ph.D. thesis, National University of Ireland Galway.

Norbert E Fuchs, Kaarel Kaljurand, and Tobias Kuhn. 2008. Attempto controlled english for knowledge representation. In *Reasoning Web*, pages 104–124. Springer.

Object Management Group. 2006. Semantics of Business Vocabulary and Business Rules (SBVR). *OMG Specification*.

Rudy Italiaander. 2019. AgentRole, TimeInstant en InverseOf Ontology Design Patterns voor efficiëntere afleidingen van beweerde data. Master's thesis, Open University of the Netherlands, Heerlen, The Netherlands.

Stef Joosten. 2017. Software Development in Relation Algebra with Ampersand. In *Relational and Algebraic Methods in Computer Science*, pages 177–192, Cham. Springer International Publishing.

Jaroslav Karpovic, Gintare Krisciuniene, Linas Ablonskis, and L. Nemuraite. 2014. The Comprehensive Mapping of Semantics of Business Vocabulary and Business Rules (SBVR) to OWL 2 Ontologies. *Inf. Technol. Control.*, 43:289–302.

Mark A. Musen and Team Protégé. 2015. The Protégé Project: A Look Back and a Look Forward. *AI matters*, 1(4):4–12.

Object Management Group. 2016. Semantics of Business Vocabulary and Business Rules (SBVR), Appendix G - EU-Rent Example.

Emmanuel Pietriga, Christian Bizer, David Karger, and Ryan Lee. 2006. Fresnel: A browser-independent presentation vocabulary for RDF. In *International Semantic Web Conference*, pages 158–171. Springer.

PNA Group. Cognitatie. http://www.cognitatie.nl/. Accessed: 2021-07-05.

RuleArts. RuleXpress. http://www.rulearts.com/RuleXpress. Accessed: 2021-07-05.

Lloyd Rutledge, Thomas Brenninkmeijer, Tim Zwanenberg, Joop van de Heijning, Alex Mekkering, J. N. Theunissen, and Rik Bos. 2016. From Ontology to Semantic Wiki – Designing Annotation and Browse Interfaces for Given Ontologies. In *Semantic Web Collaborative Spaces*, pages 53–72, Cham. Springer International Publishing.

Silvie Spreeuwenberg and Rik Gerrits. 2006. *Business Rules in the Semantic Web, Are There Any or Are They Different?*, pages 152–163. Springer Berlin Heidelberg, Berlin, Heidelberg.

Silvie Spreeuwenberg and Keri Anderson Healy. 2010. SBVR's Approach to Controlled Natural Language. In *Controlled Natural Language*, pages 155–169, Berlin, Heidelberg. Springer Berlin Heidelberg.