An SBVR Editor with Highlighting and Auto-completion

Alexandros Marinos[1], Pagan Gazzard[1], Paul Krause[2]

Rulemotion Ltd., Surrey Technology Centre,
30 Occam Rd., Surrey Research Park,
GU2 7YG, Guildford, Surrey, United Kingdom
{pagan, alexandros}@surrey.ac.uk

Department of Computing, FEPS, University of Surrey,
GU2 7XH, Guildford, Surrey, United Kingdom
p.krause@surrey.ac.uk

Abstract. This paper presents the implementation of an SBVR editor. Our editor supports automatic highlighting and offers auto-completion suggestions as the model is being typed. These capabilities have been designed to reduce the overhead in the writing of SBVR models as much as possible. The editor has been built with web technologies, and can run in any browser.

Keywords: SBVR, editor, OMeta, highlighting

1 Introduction

Semantics of Business Vocabulary and Rules (SBVR) [1] is a modeling language standardised by Object Management Group and is the result of many years of research by the Business Rules Group. SBVR holds a lot of promise due to its being completely declarative, having a solid logical foundation, and the possibility of representing its rules in a subset of English, readable by an untrained user. The standard has been in development for several years; however the tooling support seems to be lagging.

Anyone wanting to write rules as they are seen in the standard is expected to type them and highlight them by hand, possibly using Microsoft Word templates or something of that kind. This means that highlighting is left up to the human mind to determine, which becomes less and less reliable a method the more complex a vocabulary becomes. Even in the standard itself [1], highlighting inconsistencies can readily be noticed, for instance by searching for the string 'the set of'. The authors of this paper have been researching potential use cases for SBVR [2][3], but these use cases cannot be fully exploited without the proper environment for writing SBVR. For this reason we have invested effort in developing an editor that can infer the correct highlighting and offer auto-completion suggestions for models that are written in SBVR Structured English. These models can then feed into our parser which

generates SBVR Logical Formulation. The result is SBVR in its native representation and a whole range of possible use cases opens up, unhindered by the difficulties of attaining properly formatted SBVR-LF.

1.1 Related Work

Saying that tooling is lagging is not intended to mean that there are no tools whatsoever. But the range and capabilities they offer are limited. The earliest project that appeared in this space was an Eclipse IDE add-on called SBeaVeR [4]. Unfortunately SBeaVeR has not been updated since 2006 and the only release was marked as an alpha prototype. SBeaVeR did do some highlighting, but that was predicated upon adding an inelegant requirement for those writing SBVR models. Any term or verb that consisted of multiple nouns had to be joined by a dash. So a fact type that pertained to a student's registration for a study programme would have to be written as follows:

student *is-registered-for* **study-programme**

This adds unnecessary cognitive overhead for the modeler and the reader of the model, for the benefit of making the parsing significantly easier. We found it preferable to invest additional time in the one-off task of writing a parser rather than roll-over the difficulty to the modeler, which the tool was supposed to help. Additionally SBeaVeR did not offer a path to extract SBVR Logical Formulation from the rules. One can imagine that this was on the developers' roadmap, but the project has never been continued.

Another tool for writing SBVR models is RuleXpress by RuleArts [5]. RuleXpress offers impressive options in terms of vocabulary management, but only highlights terms, not verbs or keywords. Besides the reduced functionality, this simple string-matching approach may lead to errors if a word that can be used both as a verb and a noun is declared as a term (e.g. conduct, digest, escort, insult, produce, record, set).

2 Features

2.1 SBVR coverage

Our parser does not implement the full breadth of the SBVR specification yet, but rather a large and usable subset with a focus on expressing complex rules. The parser can be extended to include less common features of SBVR and indeed this is part of the future work planned. The features currently implemented are: declaration of terms and fact types, all modalities for rules, all quantifiers, and the keyword 'that' as a means of introducing atomic formulations that constrain variables.

With this subset of SBVR, even complex rules such as the following can be highlighted appropriately and parsed into their logical formulation:

It is necessary that each **student** that *is registered for* a **module** *is enrolled in* a **study programme** that the **module** *is available for*

We also support attributes for terms and fact types, although only definitions are highlighted at the moment. Finally, the editor can recognise fact types of any arity.

There exist in the literature mentions of ambiguities in SBVR-SE[10], but the paper mentions that using a lexicon should solve the problem for the example given. Our parser uses the vocabulary to inform the parsing of rules and therefore is not vulnerable to that kind of ambiguity presented. Using this setup, we have not come across any other ambiguous formulations, although we are open to the fact that they may appear. We take an engineering approach instead of a formal approach to this problem and have not attempted to prove that SBVR-SE as we parse it is completely impervious to contradictions.

2.2 Pluralisation

One of the more interesting aspects of our system is the automatic recognition of plurals. With a term such as **student** declared, any rule that uses the plural form **students** will be highlighted correctly. This also follows in the auto-complete suggestions. This recognition is accomplished with the help of an inflection component from the library Active Support for JavaScript [6]. This tool uses a number of well-known patterns of English for determining the plural of a given singular noun, and also includes a list of exceptions.

This of course does not mean all possible exceptions can be included. Even if it included every single irregular pluralisation in the largest available corpus of English nouns, new terms are coined continuously, and in the case of businesses, product names are often terms borrowed from other languages or coined de novo, which may have plurals that don't conform to obvious patterns. In these cases it would be useful for the modeler to have a way of declaring the plural of the relevant term, with this declaration overriding the judgment of the inflector.

The SBVR specification is the most authoritative document on SBVR Structured English, even though it does not claim to be a normative specification for it. While we have not reached a point where the guidance of the standard does not suffice for implementation, if we were to try plural parsing an exception as mentioned above, the best way would be to have available a 'Plural' attribute that can be defined for any term. This may not make sense for the original conception of SBVR which didn't necessarily anticipate support for tooling, however the ease of use that such tools offer may be worth accommodating in the standard. It is important to note that such an attribute would have no effect on the logical formulation. Its influence would be limited in assisting the automated parsing of models into logical formulation (which applies to highlighting and auto-completion as well) and stop there.

This extension of the SBVR attributes would not be something that would only be used in English. In fact, the grammar of English has in a way obscured this problem which would be much more obvious if another language was used as the basis for the

specification. While nouns in English can only be in singular and plural forms (and perhaps their possessive), nouns in other languages have many more cases, each of which dictates a different form for the noun, and is subject to much the same difficulties with regard to exception handling.

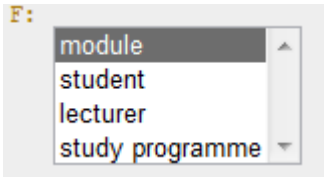
2.3 Highlighting

During the development of this editor, we considered the automatic highlighting of SBVR Structured English to be of great importance, as the writing of SBVR can be quite an ordeal otherwise, which can turn potential users away from SBVR. The editor highlights the SBVR features it implements as one would expect, recognising keywords, terms, and verbs according to the specification. One novel feature is that because we use the complete SBVR parser for the highlighting functionality, any input that cannot be highlighted, is input that cannot be parsed. This gives instant feedback to the modeler, which indicates that there is either some error in the rule, or the feature being used is not supported.

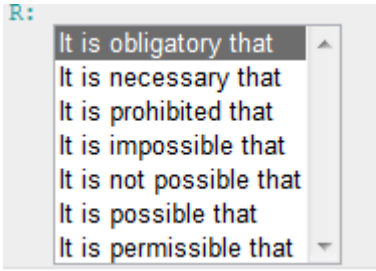
2.4 Auto-completion

At any point during the process of writing a fact type or rule, a user can press Ctrl+Space to get options for the next tokens.

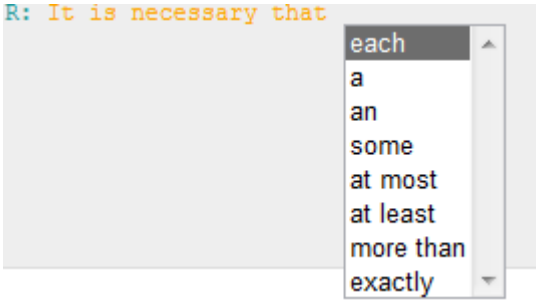
Requesting auto-completion at the start of a blank line gives the only 3 options which are to choose between a term, a fact and a rule.



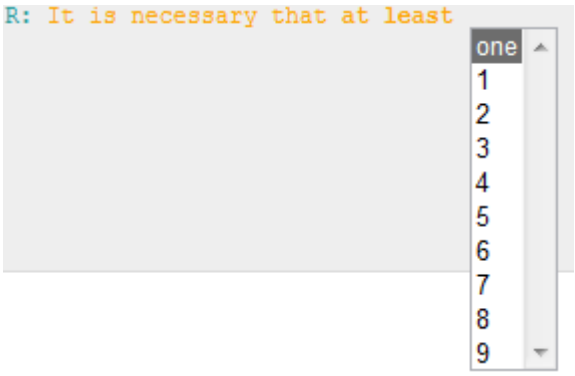
At the start of a fact, the only allowed options are terms so only terms are displayed, however these terms are all in their singular form as showing both singular and plural versions could make the number of options unwieldy.



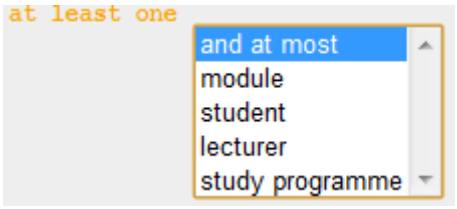
At the start of a rule, the only options available are the predefined modalities.



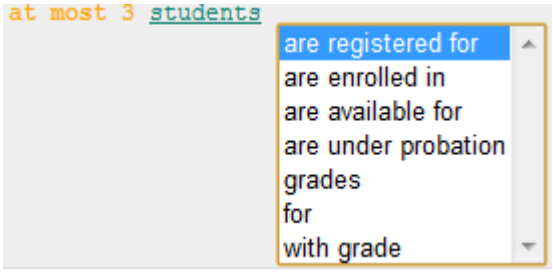
After a modifier there has to be a quantifier so a list of available quantifiers are given.



We are now offered a simple list of quantities as the ‘at least n’ quantifier requires. Although any number is allowed only the numbers from 1-9 are listed to keep it manageable and to give the user the idea that a number is necessary here. Any other number can simply be typed in and will be highlighted and accepted.



If a quantifier that can join into another is chosen, we are given a list of the available joining quantifiers as well as the terms.



After the term we are offered available verbs which are in their plural form due to following a plural term. Unfortunately all verbs are given even though only a subset of verbs applies to the chosen term.

3 Implementation

To accomplish editing SBVR in the browser, we needed to build on an editing component, intended for writing editors for programming languages. This came at the benefit of reusing mature code for complex functionality, even if the intended purposes were slightly different (modeling vs. programming), which led to a number of issues during the development process.

To choose the appropriate editing component, we reviewed a number of available ones, such as Ace [7], CodeTextArea [8], EditArea [9]. We ended up using CodeMirror2 [11], as it represents an optimal mix of features, simplicity, and project activity for our purposes.

3.1 Implementing in CodeMirror

The system needs to be able to highlight SBVR text and to provide auto-completion. To implement a syntax in highlighter in CodeMirror 2 you must provide a JavaScript closure which contains a member function called “token”, with 3

optional functions, "`startState`", "`indent`" and "`copyState`" as well as one optional variable "`electricChars`".

The token function takes two arguments, the first being a `StringStream` as defined in codemirror.js and the second being a state object, which starts as either true or whatever is returned by `startState` if it is implemented. The state object will stay consistent throughout the document and reflect changes made during previous token operations. The function returns the style that should be used for all text that has been read from the stream object during the execution of this function.

The `startState` function is available to return an initialised state variable if required, we will use this to return an object containing empty arrays for storing terms and verbs we have encountered during tokenising of the document.

The indent function takes two arguments, the state and the text of the line and returns the appropriate indentation level. As SBVR does not use indentation we can override this to always return a level of 0.

The `copyState` function takes one argument, the state, and returns a copy of the state, if this function is not implemented then the state object is just copied as-is, since we do not need any specific copying functionality we can safely ignore this function.

The `electricChars` string contains characters which when found in the string will trigger indentation to be performed; as we have no need for indentation we can safely ignore this.

3.2 Patching OMeta

Initially to enable highlighting we modified OMeta, the language the grammar is written in, to store a rule token which included the rule name, starting index, and length for each OMeta rule that was successfully matched. Within the highlighting wrapper we then picked out the rule names we were interested in highlighting and were able to generate a list of highlighting tokens using the starting indices and lengths.

Whilst this solution worked, it meant storing an absolute minimum of one rule token per character of the string (and generally a lot more, e.g. char, exactly, seq, token, etc.), most of which we were never interested in. So to get around this we modified OMeta to accept a list of rule names we are interested in tokens for and for OMeta to only store tokens for rules that match this list, this reduced the number of rule tokens stored dramatically and also meant that the highlighting wrapper did not need to check through for only the rules it was interested in and so could have its complexity reduced.

Due to the nature of the highlighting being a one to one mapping with the rules that we store it becomes necessary for the parser to use separate rules for parsing each token that needs to be highlighted differently, so some modification may be necessary. However the result of these modifications being required seems to be one of enforcing a good code style rather than one of creating an annoyance, similar to the use of significant white-space for block indentation like in python.

For auto-completion we modified OMeta to store the rule name and starting index for every attempt to match a rule, this way we can find all possible branches that

OMeta attempted to take at a given point. This is only guaranteed to be a complete list of branches at the point the match fails, so for our purposes we take the point in the line at which the user requested auto-completion hints and tell OMeta to parse up until that point, as such we know that the parsing will fail so we will get all possible branches that can be taken. The auto-completer then looks at the map of rule names to possibilities provided by the OMeta based parser and offers those possibilities.

4 Conclusion & Future Work

We have found using the SBVR editor useful and intend to release it as a commercial application soon. However, there still remain a number of potential improvements that can be made, and we will keep improving the codebase.

The obvious direction for improvements is in extending the amount of SBVR that our grammar can handle, improving both the highlighter and the parser at the same time. Also, adding support for multiple vocabularies and inclusion of vocabularies in others will make the environment more suitable for larger projects.

Another intriguing possibility, which may help in making SBVR more popular, is to add the possibility for publishing models on the Web, with an easily shareable URL. This will hopefully address the dearth of SBVR examples online currently, another barrier for newcomers.

We also expect to receive a lot of feedback as we make the tool available for use to wider audiences, and have reserved significant resources in our roadmap so we can be responsive to users' suggestions.

References

1. Object Management Group, "Semantics of Business Vocabulary and Rules Formal Specification v1.0", 2008, URL: http://www.omg.org/spec/SBVR/1.0/, Accessed: 16/9/2011.
2. Marinos, A., Krause, P., "An SBVR Framework for RESTful Web Applications" In Semantic Web Rules - International Symposium, RuleML 2010, Washington, DC, USA, October 21-23, 2010. Proceedings. LNCS 6403, Springer-Verlag Berlin, Heidelberg, 2010, pp. 144-158.
3. Marinos, P. Krause. "What, not How: A generative approach to service composition", IEEE Conference on Digital Ecosystems Technologies 2009 (DEST 2009)
4. SBeaVeR, URL: http://sbeaver.sourceforge.net, Accessed: 16/9/2011.
5. RuleXpress, URL: http://www.rulearts.com/RuleXpress, Accessed: 16/9/2011.
6. Inflection-js, URL: http://code.google.com/p/inflection-js/, Accessed: 16/9/2011.
7. Ace, URL: http://ace.ajax.org, Accessed: 16/9/2011.
8. CodeTextArea, URL: http://code.google.com/p/codetextarea/, Accessed: 16/9/2011.
9. EditArea, URL: http://www.cdolivet.com/editarea/, Accessed: 16/9/2011.
10. Kleiner M., Albert, P., Bézivin, J., "Parsing SBVR-Based Controlled Languages", In Model Driven Engineering Languages and Systems, LNCS, 2009, Volume 5795/2009, pp. 122-136
11. CodeMirror2, URL: http://codemirror.net/, Accessed: 16/9/2011.