OETCS/WP1/D02

openETCS

ITEA2 Project
Call 6 11025
2012 – 2015

Work-Package 7: “Toolchain”

Requirements Modeling for SSRS Activities with ProR

Results of ProR/SSRS Workshop in Braunschweig

Michael Jastram

October 8, 2013

Loading org.eclipse.emf.ecore
0.8.0.201308150733

This page is intentionally left blank

Work-Package 7: “Toolchain”

OETCS/WP1/D02
October 8, 2013

Requirements Modeling for SSRS Activities with ProR

Results of ProR/SSRS Workshop in Braunschweig

Michael Jastram
Formal Mind GmbH

Position Paper

Prepared for openETCS@ITEA2 Project

Abstract: During the Braunschweig workshop on October 8th, we had a small work session to start modeling the SSRS requirements in ProR. Participants were:

- Uwe Steinke
- David Mentre
- Jan Welvaarts
- Michael Jastram

The objective was to start a trial migration of the refined requirements created by Jan, in the context of the SSRS activities.

Modification History

Version	Section	Modification / Description	Author
0.1	all	Michael Jastram	

Table of Contents

Figures and Tables

Figures

Tables

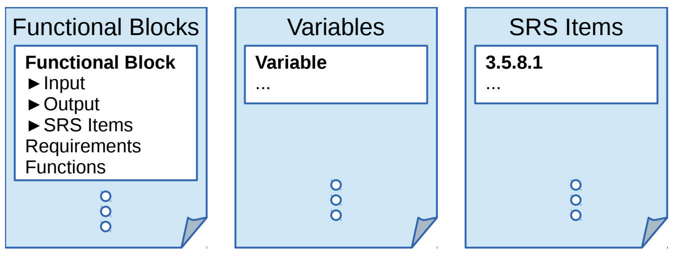


Figure 1. The major elements of the SSRS.

1 SRS Data Model

Jan Welvaarts already spent some time creating a document using OpenOffice, representing a subset of the SSRS. We captured this on a whiteboard (upper part of Figure 2. A first attempt to capture this with the data structures available by ProR is shown in the lower part. That part has been cleaned up and is shown in Figure 1.

In the following, we will describe the elements from these figures.

Functional Block. A *Functional Block* contains a set of *Functions*. Each *Function* can be referenced from multiple *Functional Blocks*. It contains references to *Imputs*, *Outputs* (which are *Variables*) and *SRS Items*. It also contains an arbitrary number of *Requirements*.

Function. An element of a *Functional Block* with *Imputs* and *Outputs*.

Variable. *Variables* are explicitly listed in the SRS (e.g. those extracted by the BitWalker Scripts). They can also appear implicitly in the requirements text of the SRS. The *Variables* referenced from *Functional Blocks* are typically compound structures consisting of multiple *Variables*.

Input. A *Variable* that is used as an input for a *Function* or *Function Block*.

Output. A *Variable* that is used as an input for a *Function* or *Function Block*.

SRS Item . The SRS consists of *SRS Items*. Most of the time, it is a paragraph, which has an ID (e.g. 3.5.7.5.3). Due to the heterogeneous nature of the SRS document, it can sometimes be something else (a table, a sentence, a group of items, etc.).

2 ProR Data Model

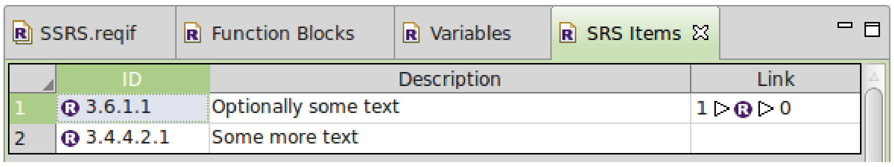
We prototypically built data structures in ProR, which can be used to represent the previously described data model. In the following, we will describe this data structure. It is also stored on gitHub at

`https://github.com/openETCS/SSRS/trunk/Requirements_Modeling`

In ProR, we organized the model into three top level *Specifications* (document-like structure that each contain a tree of elements): *Functional Blocks*, *Variables* and *SRS Items*.

2.1 SRS Items

In the *SRS Items*, one *SpecObject* per *SRS Item* is created. For now, this will just be an element with ID, and optionally some explanatory text. Later, this could be converted into a proxy element to provide an integrated traceability into the SRS. But this requires additional work. The figure below shows this, for two (dummy) entries:

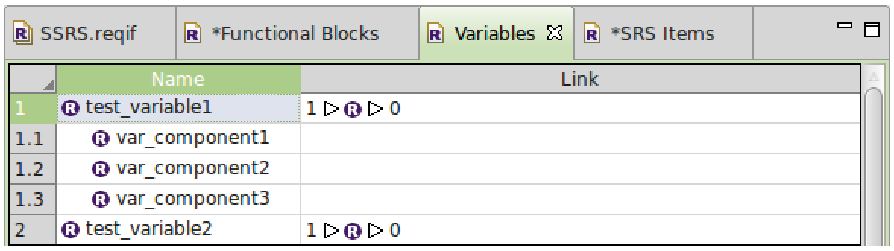


	ID	Description	Link
1	Ⓡ 3.6.1.1	Optionally some text	1 ▷ Ⓡ ▷ 0
2	Ⓡ 3.4.4.2.1	Some more text	

The *Link* columns indicates thast the item 3.6.1.1 has one incoming link, meaning that there is one *Function Block* referencing it.

2.2 Variables

At least for the time being, *Variables* will be handled like *SRS Items*. But note that they are usually compound structures, which is handled through nesting, as shown below. This kind of nesting would be possible for the *SRS Items* as well.

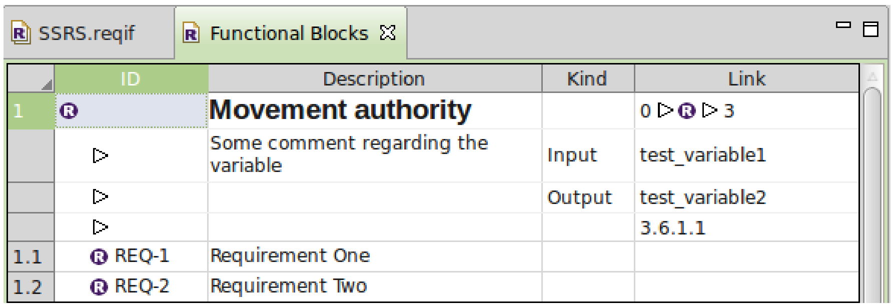


	Name	Link
1	Ⓡ test_variable1	1 ▷ Ⓡ ▷ 0
1.1	Ⓡ var_component1	
1.2	Ⓡ var_component2	
1.3	Ⓡ var_component3	
2	Ⓡ test_variable2	1 ▷ Ⓡ ▷ 0

Eventually, the work done by Siemens with the BitWalker tool could be used here by directly referencing the variables from the generated XML document.

2.3 Functional Blocks

The *Functional Blocks* essentially pull the other information together and augment it. This is shown below.



	ID	Description	Kind	Link
1	Ⓡ	**Movement authority**		0 ▷ Ⓡ ▷ 3
	▷	Some comment regarding the variable	Input	test_variable1
	▷		Output	test_variable2
	▷			3.6.1.1
1.1	Ⓡ REQ-1	Requirement One		
1.2	Ⓡ REQ-2	Requirement Two		

The name of the *Functional Block* is shown in a bigger font (realized with the ProR Headline plugin). This element has three outgoing traces: one *Input*, one *Output* and one *SRS Item*. These traces have attributes, and the links to the *Variables* contain an attribute called *Kind*, which classifies it (this is an enumeration attribute).

Realized as child elements (therefore indented), two additional *Requirements* have been added. The human-readable IDs have been generated by another ProR plugin.

Missing from the picture are the actual *Functions*, which will be discussed in the next section.

2.4 Functions

We didn't decide yet on how to model the *Functions*. From a usability point of view, simly adding them to the *Function Blocks* would be a good solution (indenting them by taking advantage of the tree structure). However, as *Functions* can be referenced by multiple *Functional Blocks*, a link may be better (but less readable). The readability issue could be addressed by generating reports that inline the *Functions*. ProR also allows referencing elements. Thus, multiple *Function Blocks* could simply point to the same *SpecObjects* that represent the *Functions*. This can still create inconsistencies. For instance, if a *Function* is extended by a *SpecObject* (to add some additional information, for instance), a reference to that SpecObject would have to be added to all places where this *Function* is referenced.

2.5 A Note about Text Formatting

So far, only plain text has been used. We suggest the usage of plain text for two reasons: First, using plain text will make model integration later on much easier, as color highlighting, parsing, etc. are much easier on plain text. Second, while ProR has rich text support, it is not very good and has limitations on some platforms (e.g. on Mac).

The team generally considered plain text sufficient, except that logical expressions are much more readable when formatted.

3 SysML Integration

So far, SysML integration has not been discussed at all, but should be eventually be covered in this document as well.

Integrating ProR and Papyrus will be relatively easy, and will be made available by Michael, as a contribution to this project. The integration will work as follows: ProR will be installed into Papyurus (or the other way around). The Papyurs Model Browser must be opened, which allows access to all SysML elements. Linking will be performed by dragging from the Model Browser onto the corresponding *SpecObject*.

The first version of the integration plugin will show a textual representation of the corresponding SysML element in ProR (e.g. the Block Name or Port Name).

4 Backlog

In this section, we collect tasks that need to be done in order to make work with ProR practical. These are listed here in no particular order. Eventually, they should make their way into the WP7 backlog, to be implemented:

Reporting. While rudimentary reporting is available (via the print functionality, which generates HTML), we almost certainly will need tailored reporting for acceptance.

SRS Traceability. The traceability to the SRS suggested here is just a crutch. We either need a traceability into the Word documents, or we need to import the SRS into ProR. Both approaches have their respective advantages and disadvantages.

SysML/Papyrus Traceability. This has been discussed earlier.

Formatting Logical Expressions. As mentioned earlier, logical expressions would benefit from using formatted text. This either requires improving rich text support, or the creation of a dedicated representation for logical expressions (e.g. by using a DSL).

5 Appendix: Whiteboard

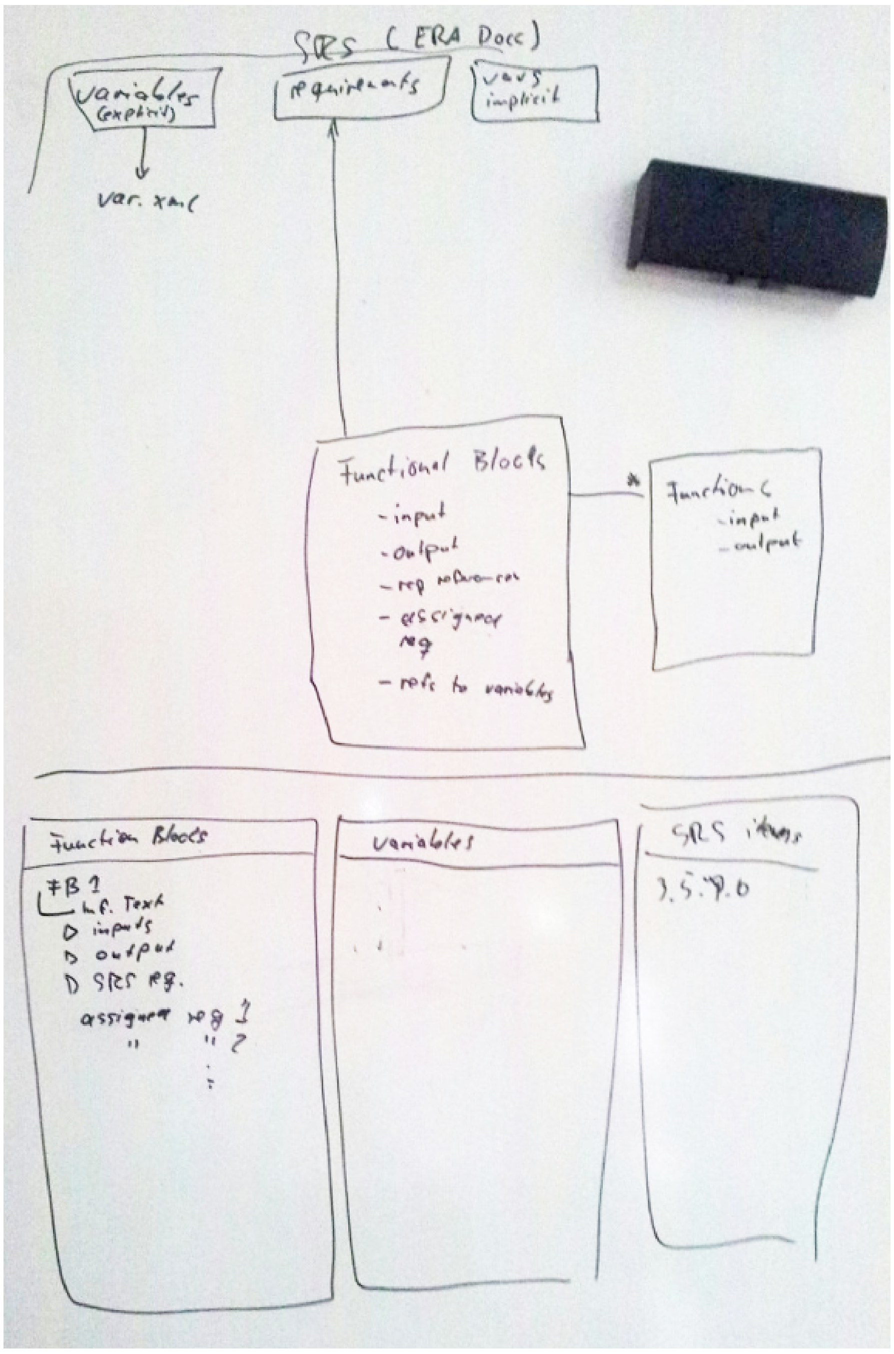


Figure 2. The whiteboard content that outlines the ProR data model.