A DSL and a SPIN-frontend for river-crossing problems defined with Xtext

Th. Baar*

Abstract. Bodin [1] has applied SPIN to solve puzzles like the Japanese river puzzle, an advanced version of the famous wolf-goat-cabbage puzzle. Defining a Promela model can become cumbersome and debugging can be very time-consuming, since SPIN does the syntax check (e.g. type checking) only at runtime and developers might have decided to use general-purpose editors, from which they do not get any support when editing Promela models.

The purpose of the work described in this paper is twofold. Firstly, it addresses the lack of rich editors for Promela models. Thus, an Eclipse-editor has been defined based on Xtext, which is able to assist the developer of Promela models and which comes with rich features such as auto-completion, syntax highlighting, and rename refactoring. Together with a shell script to start SPIN on the edited model, this editor enables the developer to work with Promela models efficiently.

Secondly, a domain-specific language (DSL) has been developed based on Xtext that allows us to formulate river-crossing puzzles in a very intuitive way. The DSL comes with a rich Eclipse-editor as well as with a code generator, which generates the corresponding Promela model as input for the model checker.

Keywords: SPIN, model checking, Promela editor, dsl, Xtext.

1. Motivation

The model checker SPIN [2][3] has been widely used both in academia and in industry to verify system properties automatically [4]. SPIN traverses the state space of a system and verifies whether some given safety or liveness properties hold as expected. The system itself is defined in the C-like language Promela.

Bodin [1] has applied SPIN to solve puzzles like the Japanese river puzzle, an advanced version of the famous wolf-goat-cabbage puzzle. The Japanese river puzzle is described as follows¹:

1. The boat can carry no more than two people.
2. Only the adults (Mom, Dad and Policeman) can operate the boat.
3. Dad can’t be left alone with the Daughters without Mom.
4. Mom can’t be left alone with the Boys without Dad.

*The results described in this paper have been worked out during my research stay at Novosibirsk State University and at A.P. Ershov Institute of Informatics Systems (Akademgorodok, Novosibirsk, Russia) during winter term 2015/16.

¹The description was taken from http://izismile.com/2010/12/07/oldie_of_the_day_famous_japanese_river_crossing.html and has been slightly changed.
5. The Thief can’t be left alone with any of the family without the Policeman.

After encoding the transportation process together with the above constraints in Promela, SPIN finds a path for moving all persons safely from one river side to the opposite one. The actual Promela model in the form of the input text for SPIN as well as an accurate explanation of the encoding can be obtained from [1] and is not discussed further here. For the purpose of this paper, it is sufficient to refer to the coarse structure of the Promela model:

1. Declaration of the domain-specific things to be moved (as mtype)
2. Description of the target state DONE (as a propositional logical formula formulated using #define)
3. Description of dangerous situations that have to be avoided (as propositional logical formulas formulated using #define)
4. Declaration of global variables
5. Inline-functions for defining certain steps in the process of crossing the river
6. Process init() to formulate the complete process of crossing the river

Changing and testing this Promela model can become very time-consuming and error-prone if only a general purpose text editor like vi is used and if SPIN is launched by means of a shell script.

In order to make the work with SPIN more fun as well as more efficient (both aspects are probably very interrelated), we decided to implement an Eclipse-editor for Promela models having the above structure. This implementation is based on the Xtext-framework [5], which allows text editors to be implemented with moderate effort (see the textbook from Bettini [6] to minimize the effort). Since the resulting editor is integrated into Eclipse, Eclipse has become an IDE for developing Promela models. Note that also the launch script for SPIN can be integrated into Eclipse.

An editor for Promela models does not help much if one wants to change the problem description a little bit, e.g. to switch from the Japanese river puzzle as described in [1] to the simpler but more well-known wolf-goat-cabbage puzzle. The developer has to manually encode both the entities and the constraints of the new puzzle into the Promela model (which prerequisits an in-depth understanding of the existing Promela model!).

Surely, the SPIN technology would be useful for many users but a lot of prospective users refrain its usage due to the considerable effort to understand SPIN’s input language and to acquire enough skills to write complex models such as the ones authored by Bodin in [1]. For users who wish to solve their own river-crossing problems without learning the odds and ends of Promela, the DSL rcrs has been developed. By this DSL, a user can formulate the settings of a specific river-crossing problem using an intuitive and simple syntax. The DSL comes with a generator, which translates automatically the problem formulated in rcrs into a Promela model.

Both the Promela editor described in Section 3 as well as the tool set for rcrs are open source software. Together with the examples discussed in this paper, they can be downloaded from [7].
2. Related work

Inventing and applying domain-specific languages became a simpler task over the last decade thanks to DSL frameworks and tools like Xtext [5], Spoofax [8], MetaEdit+ [9] and others. However, we are not aware of any DSL targeting river-crossing problems such as our rcrs (see Section 4). None of the numerous community DSLs defined with Xtext and published in [10] is designed as a frontend DSL to facilitate the interaction with a model-checker such as SPIN for a certain domain (in our case, the domain river-crossing puzzles).

For the domain-independent interaction with Spin, there is an Eclipse-plugin called EpiSpin [11], which is generated from the Spoofax workbench. EpiSpin has encoded the grammar of Promela and offers an editor for Promela files. One can configure and invoke SPIN directly from EpiSpin, which also offers a graphical view on the edited Promela files. EpiSpin can be freely downloaded from [12].

3. Frontend for SPIN

The Eclipse-framework Xtext allows one to define textual domain-specific languages (DSLs) easily [5]. At first, a DSL-developer has to define the grammar for the textual notation. In Xtext, grammar definition is done in an EBNF-inspired formalism. Based on this grammar, the Xtext-framework can generate already a fully functioning textual editor supporting advanced features like syntax highlighting, rename refactoring, auto-completion, etc. However, the grammar alone is not sufficient in most cases and the auto-generated tools like the editor or Outline pane need customization in order to function properly. Also well-formedness rules on the
DSL models have to be implemented separately (as validators), since they are not expressed within the grammar.

Defining the grammar which matches the Promela models having the structure described in Section 1 was almost straightforward (even if the current solution might not be perfect yet). The grammar has a length of appr. 200 lines. More important is the resulting editor, which can be seen in Figure 1.

The figure shows the original Promela file developed by Bodin in the Eclipse-editor generated by Xtext. As one can see, the keywords and comments are highlighted in different colors. The syntax error Bot (the correct element would be Boat) is marked by an error marker and a tooltip suggests two possible corrections, among which is the right one.

On the left side, there is the Outline pane showing the structure of the file. It starts with mtype, then there are the propositions defined via #define followed by declarations of global variables and inline functions. At the end, there is the definition of the process init. As usual in Eclipse, by clicking on one element in the Outline pane, the cursor in the editor moves to the corresponding location, which facilitates navigation within large model files a lot.

Using the Eclipse-editor for Promela models makes it much more fun to work on them. However, it does not take the burden to understand the Promela model deeply before one is able to adapt it to a different but similar situation. For this purpose, the second DSL rcrs has been developed, as described in the next section.

4. A DSL to formulate river-crossing puzzles

Wouldn’t it be nice if one could use SPIN without much effort in order to solve problems like the Japanese river puzzle? This is exactly the goal of the DSL rcrs (for river crossing) described in this section. It enables the user to describe the essentials of a river-crossing problem in an intuitive and concise way.

If one wants to have a concise notation for characterizing single instances among a class of problems, one should point out first, what all problems of the class have in common (the more they share, the less a single problem instance has to be configured by a DSL model). It is assumed that the following invariants hold for all river-crossing problems:

- There are different types of persons/items (e.g. Mom, Dad, Boy, Cop\(^2\)). Usually, there is only one instance per type, but a type can also be declared to have more than one instance (e.g. two Boys).
- At the beginning, all persons/items together with the boat are on the left side of the river. In the target state, all persons/items have been brought to the right side of the river.
- There is exactly one boat available to move persons/items from one river side to the other.

\(^2\)In difference to the introductory description of the Japanese river puzzle given in Section 1, we use in the following the term Cop instead of Policeman and Criminal instead of Thief. These renamings bring us closer to the original puzzle encoding for SPIN made by Bodin in [1].
Figure 2. The Japanese river puzzle formulated in DSL \textit{rcrs}

- The boat needs a driver and can take in addition one person/item as a passenger. The passenger is optional, i.e. the boat can also go with the driver alone.
- Not all persons might be eligible to be the boat’s driver. The eligible types of persons have to be selected.

4.1. The Japanese river puzzle formulated in \textit{rcrs}

Figure 2 shows the editor for \textit{rcrs} on the Japanese river puzzle, which illustrates this DSL in full. Let us go through the problem description of the Japanese river-crossing problem line by line:

The first line
\texttt{project japanese} \hspace{1cm} // opens a namespace

\texttt{// sorts of elements (boat is standard)}
\texttt{types Cop Dad Mom Criminal Boy(2) Girl(2)}

\texttt{// Boat - specification (always two places:)}
\texttt{boat_drivers Cop Dad Mom}

\texttt{// dangerous situations for river side + boat}
\texttt{dangerous CriminalUnsafe : Criminal \&\& !Cop \&\& (Dad \| Mom \| Boy \| Girl)};
\texttt{BoysUnsafe : Mom \&\& !Dad \&\& Boy;}
\texttt{GirlsUnsafe : Dad \&\& !Mom \&\& Girl;}
\texttt{<}

\text{The first line}
\texttt{project japanese}

gives the subsequent specification a unique name. Organizing different specifications under different names has some technical advantages since it narrows the default visibility of each symbol to that specification, in which the symbol has been declared.

Then, one has to declare which types of persons/items actually exist in the current puzzle. If a type has more than one instance, the instance number is given in parenthesis after the type (e.g. Boy(2)).

\texttt{types Cop Dad Mom Criminal Boy(2) Girl(2)}
Then, one has to select the types of persons eligible to drive the boat:

**boat_drivers** Cop Dad Mom

At the end, one has to specify all constellations that can become dangerous. Each constellation is given a name (e.g. BoysUnsafe) and specified by a propositional formula (e.g. Mom && !Dad && Boy). The dangerous constellations must be avoided during the transportation process on both river sides as well as on the boat!

**dangerous**

CriminalUnsafe : Criminal && !Cop &&
(Dad || Mom || Boy || Girl);

BoysUnsafe : Mom && !Dad && Boy;

GirlsUnsafe : Dad && !Mom && Girl;

### 4.2. The grammar for rcrs

This simple structure is encoded by the following grammar, which is much simpler and shorter than for the language Promela described in Section 3. Note that the complexity of formulating the input model for SPIN is now mastered by the code generator for rcrs. The definition of the code generator is lengthy but straightforward.

```plaintext
grammar de.htwberlin.selabs.RcrsDsl with org.eclipse.xtext.
    common.Terminals

generate rcrsDsl "http://www.htwberlin.de/selabs/RcrsDsl"

RcrsModel:
    'project' name=ID
    td=TypeDecl
    bd=BoatDecl
    dd=DangerousDecl;

TypeDecl:
    'types' types+=Type+;

Type:
    name=ID ('{' num=INT '}')?
    ;

BoatDecl:
    'boat_drivers' {BoatDecl} drivers+=[Type]+;

DangerousDecl:
    'dangerous' constraints+=Constraint+;
```
Constraint :
    name=ID ' ; ' body=Fml ' ; ' ;

Fml:
    OrFml
    ;

OrFml returns Fml:
    AndFml ({ OrFml . left = current } ' || ' right=AndFml) *
    ;

AndFml returns Fml:
    AtomicFml ({ AndFml . left = current } ' && ' right=AtomicFml) *
    ;

AtomicFml returns Fml:
    { Compound } ' ( t = Fml ' ) ' |
    { Plain } v = [ Type ] |
    { Neg } ' ! ' nv = [ Type ];

5. Conclusions

The model checker SPIN has proved to be applicable for many different purposes and is also able to solve demanding problems. However, formulating input files in Promela is a challenging cognitive task and using a general purpose editor is less efficient than using dedicated editors. Both problems have been addressed in this paper by implementing two DSLs using the framework Xtext. The resulting editors are of great help when working with models. Model elements can be renamed flexibly and syntax errors are detected and partially resolved on-the-fly when typing the model. The language rcrs is used as a frontend language for the interaction with SPIN for problems from a certain domain (river-crossing puzzles). Since SPIN finds solutions for the Japanese river puzzle and similar problems (e.g. the more simpler wolf-goat-cabbage puzzle) within milliseconds, it is now very tempting to investigate different variants of the problem using SPIN. For example, just by changing very small portions in the rcrs input file, the user can immediately make statements on the solvability of variants of the Japanese puzzle, e.g. if the number of boys is changed from 2 to 1 or to 3. Another interesting dimension for variants is to add new constraints for dangerous situations or to change the existing ones. Note that the formulation of each problem variant in plain Promela would be a demanding, time-consuming and error-prone task, which is now fully automatized by the rcrs’s code generator.

Acknowledgements: I thank the German Academic Exchange Service (Deutscher Akademischer Austauschdienst, DAAD) for financial support of my research stay in Akademgorodok near Novosibirsk, Russian Federation, in winter term 2015/16. I am deeply indebted to Evgeny V. Bodin for drawing my attention to the model checker SPIN and to puzzle solving. Since I had no experience in using SPIN before
coming to Akademgorodok, Evgeny helped a lot to master technical problems and provided many useful example inputs and invocation scripts for SPIN. Without him, I would not have written this paper.

References


