Solving the Petri-Nets to Statecharts Transformation Case with UML-RSDS

K. Lano, S. Kolahdouz-Rahimi, K. Maroukian
Dept. of Informatics, King’s College London, Strand, London, UK

This paper provides a solution to the Petri-Nets to statecharts case using UML-RSDS. We show how a highly declarative solution which is confluent and invertible can be given using this approach.

Keywords: Petri-Nets; Statecharts; UML-RSDS.

1 Introduction

This case study [4] is an update-in-place transformation which simultaneously modifies (by deletion and simplification) an input Petri-Net model, and (by construction and elaboration) an output statechart model. We provide a specification of the transformation in the UML-RSDS language [5] and show that this is terminating, confluent and invertible.

UML-RSDS is a model-based development language and toolset, which specifies systems in a platform-independent manner, and provides automated code generation from these specifications to executable implementations (in Java, C# and C++). Tools for analysis and verification are also provided. Specifications are expressed using the UML 2 standard language: class diagrams define data, use cases define the top-level services or functions of the system, and operations can be used to define detailed functionality. Expressions, constraints, pre and postconditions and invariants all use the standard OCL notation of UML 2.

For model transformations, the class diagram expresses the metamodels of the source and target models, and auxiliary data can also be defined. Use cases define the main transformation phases of the transformation: each use case has a set of pre and postconditions which define its intended functionality.

The Petri Net to statecharts transformation can be sequentially decomposed into three subtransformations: an initialise transformation, which copies the essential structure of the Petri Net to an initial statechart, followed by the main pn2sc reduction/elaboration transformation. A final cleanup transformation removes elements which do not contribute to the target structure.

Figure 1 shows the source and target metamodels of the transformation, and the three use cases representing the sub-transformations.

We extend [4] by asserting that name is unique for HyperEdge, Basic and OR:

\[
\begin{align*}
\text{HyperEdge} & \rightarrow isUnique(name) \\
\text{Basic} & \rightarrow isUnique(name) \\
\text{OR} & \rightarrow isUnique(name)
\end{align*}
\]

This means that object indexing by name can be used for these entity types: \(OR[s]\) denotes the or-state with name \(s : String\), for example, if such a state exists.

*Research supported by the HoRTMoDA EPSRC project
2 Initialisation transformation

This has the precondition that the statechart is unpopulated: $State.size = 0$, $Statechart.size = 0$, and that name is unique for NamedElements. There are four postconditions, which define the intended state at termination of the transformation. These postconditions are also interpreted as definitions of the transformation steps.

Postcondition $I_1$ applies to elements of Place to map them to Basic and OR states:

$$\text{Basic} \rightarrow \exists ( b \mid b.name = \text{name} \land \text{OR} \rightarrow \exists ( o \mid o.name = \text{name} \land b : o.contains ) )$$

Logically this can be read as “for all $p$ in Place, there exists $b$ in Basic with $b.name = p.name$, and $o$ in OR with $o.name = p.name$ and $b$ in $o.contains$”. The inverse link $rcontains$ is set implicitly ($o : b.rcontains$).

Postcondition $I_2$ applies to Transitions to map them to HyperEdges:

$$\text{HyperEdge} \rightarrow \exists ( e \mid e.name = \text{name} )$$

$I_3$ sets up the next/rnext links between hyperedges and basic states based upon the corresponding post/prep links in the Petri Net:

$$t : \text{post} \Rightarrow \text{HyperEdge}[t.name] : \text{Basic}[\text{name}].next$$

applied to Place (“if $t$ is a post-transition of self, then the hyperedge corresponding to $t$ is in the next states of the basic state corresponding to self”).

$I_4$ sets up the next/rnext links between basic states and hyperedges based upon the corresponding post/prep links in the Petri Net:

$$p : \text{postp} \Rightarrow \text{Basic}[p.name] : \text{HyperEdge}[\text{name}].next$$

applied to Transition.

This transformation uses the ‘Map objects before links’ pattern [1] to separate mapping of elements and their links. It avoids the need for recursive processing: each of $I_1, ..., I_4$ can be implemented by a linear iteration over their source domains. This implementation is generated automatically by UML-RSDS as a Java program.

Termination, confluence and invertibility of such transformations follows by construction [1]. The computational complexity is linear in $\text{NamedElement.size}$. The transformation establishes $\text{Basic} \rightarrow \text{isUnique(name)}$, $\text{HyperEdge} \rightarrow \text{isUnique(name)}$ and $\text{OR} \rightarrow \text{isUnique(name)}$ because of the uniqueness of names of named elements. Indeed these properties are invariants of initialise.
3 Main transformation

This has as its preconditions 11, 12, 13, 14, together with the uniqueness properties of name for Basic, HyperEdge and OR, and that AND is empty. An invariant Inv asserts that for all places, there is a unique OR state with the same name:

\[ \text{Place} \rightarrow \forall p : \text{OR} \rightarrow \exists! o : o.\text{name} = p.\text{name} \]

This ensures that there is an injective function equiv : Place → OR. In our notation, \text{OR}[p.\text{name}] is equiv(p) for p : Place.

The uniqueness properties of name for Basic, HyperEdge and OR are also invariant. Inv is established by initialise because of postcondition I1 and the uniqueness of name on OR.

The highest priority rule (postcondition) is Post1, which performs the OR-reduction of \cite{4} on Transition instances:

\[
\begin{align*}
\text{prep}.\text{size} = 1 & \text{ & } \text{postp}.\text{size} = 1 & \\
q : \text{ prep} & \text{ & } r : \text{ postp} & \\
(q.\text{pret} \land r.\text{pret}) \rightarrow \text{size}(o) = 0 & \text{ & } \\
(q.\text{postt} \land r.\text{postt}) \rightarrow \text{size}(o) = 0 \Rightarrow \\
\text{OR} \rightarrow \exists! p : p.\text{name} = q.\text{name} + "\text{OR}" + r.\text{name} & \\
& \text{ & } p.\text{contains} = \text{OR}[q.\text{name}].\text{contains} \cup \text{OR}[r.\text{name}].\text{contains} & \\
& \text{ & } q.\text{name} = p.\text{name} & \\
& \text{ & } q.\text{pret} \rightarrow \text{includesAll}(r.\text{pret}) & \\
& \text{ & } q.\text{postt} \rightarrow \text{includesAll}(r.\text{postt}) & \\
& \text{ & } r \rightarrow \text{isDeleted} & \\
& \text{ & } self \rightarrow \text{isDeleted} & \\
\end{align*}
\]

This follows very closely the specification in \cite{4}, with self : Transition playing the role of t. The updates to the Petri-Net are the last five lines, q replaces the q → self → r structure and is renamed to match the new OR state, thus maintaining Inv.

For AND-reduction there are two postconditions/rules for the symmetric cases: Post2 merges pre-places with equivalent connectivities, and again is applied to each Transition:

\[
\begin{align*}
p1 : \text{ prep} & \text{ & } \text{prep}.\text{size} > 1 & \\
& \text{prep} \rightarrow \forall p2 : p2.\text{pret} = p1.\text{pret} & \text{ & } p1.\text{postt} = p2.\text{postt} \Rightarrow \\
& \text{AND} \rightarrow \exists! a : a.\text{contains} & \text{ & } \\
& \text{ & } a : \text{prep}.\text{contains} & \text{ & } a.\text{contains} = \text{OR}[\text{prep}.\text{name}] & \\
& \text{ & } p.\text{name} = "\text{AND}1_{-}\text{name} + \text{name} & \text{ & } a.\text{name} = "\text{a1}_{-}\text{name} & \\
& \text{ & } p1.\text{name} = "\text{AND}1_{-}\text{name}\) & \\
& \text{ & } (\text{prep} - \{p1\}) \rightarrow \text{isDeleted} & \\
& \text{ & } \text{prep} = \text{Set}\{p1\} & \\
\end{align*}
\]

The last three lines define the update to the Petri-Net: all prep places of self are deleted except for p1, which is renamed to match the newly created OR state (therefore maintaining Inv).

Post3 merges post-places with equivalent connectivities, for each applicable Transition:

\[
\begin{align*}
p1 : \text{ postp} & \text{ & } \text{postp}.\text{size} > 1 & \\
& \text{postp} \rightarrow \forall p2 : p2.\text{pret} = p1.\text{pret} & \text{ & } p1.\text{postt} = p2.\text{postt} \Rightarrow \\
& \text{AND} \rightarrow \exists! a : a.\text{contains} & \text{ & } \\
& \text{ & } a : \text{postp}.\text{contains} & \text{ & } a.\text{contains} = \text{OR}[\text{postp}.\text{name}] & \\
\end{align*}
\]
This maintains Inv for the same reason as Post2.

### 4 Cleanup transformation

This transformation deletes OR states with empty contents:

\[
\text{contains.size} = 0 \Rightarrow \text{self} \rightarrow \text{isDeleted()}
\]

on OR.

Finally, an instance \( sc : \text{Statechart} \) needs to be created, with \( sc.\text{topState} \) being the unique topmost AND state produced by the main transformation, if such a state exists:

\[
v = \text{OR} \rightarrow \text{select} (\text{rcontains.size} = 0) \land v.\text{size} = 1 \land ox : v \Rightarrow \text{AND} \rightarrow \text{exists} (a \mid a.\text{name} = \text{"TOPSTATE"} \land ox : a.\text{contains})
\]

and

\[
w = \text{AND} \rightarrow \text{select} (\text{rcontains.size} = 0) \land w.\text{size} = 1 \land ax : w \Rightarrow \text{Statechart} \rightarrow \text{exists} (sc \mid sc.\text{topState} = ax)
\]

This transformation is terminating and semantically correct by construction.

### 5 Results

Table 1 gives the test results for the performance tests for the Java 4 executable in the SHARE environment, and for the Java 6, C# and C++ executables on a standard Windows 7 laptop.

| Test    | Transformation execution time: Java 4 | Java 6  | C#    | C++ |
|---------|--------------------------------------|---------|-------|-----|
| sp200   | 100ms                                | 15ms    | 29ms  | 0s  |
| sp500   | 160ms                                | 31ms    | 63ms  | 2s  |
| sp1000  | 290ms                                | 94ms    | 198ms | 6s  |
| sp5000  | 3815ms                               | 1670ms  | 5069ms| 161s|
| sp10000 | 13713ms                              | 6614ms  | 21980ms| –   |
| sp20000 | 48s                                  | 35s     | 87s   | –   |
| sp40000 | 258s                                 | 177s    | 468s  | –   |
| sp80000 | 3142s                                | 5619s   | 10003s| –   |

Table 1: Performance test results for Java, C# and C++

The results for the Java 4, C# and Java 6 (which uses HashSet instead of Vector for sets) implementations were quite similar, which is in contrast to problems involving uni-directional associations, where the Java 6 translation is typically 100 times more efficient than the Java 4 version. C++ has efficiency
Table 2: Solution table

| Solution Name | Language (for all aspects) | Perform. optimisations | 5.2.1: verification | 5.2.2: simulation | 5.2.3: change-prop | 5.2.4: reverse | 5.2.5: debug. | 5.2.6: refactoring |
|---------------|-----------------------------|------------------------|---------------------|-------------------|-------------------|---------------|-------------|-------------------|
| UML-RSDS      | UML-RSDS                   | E                      | CT                  | N                 | N                 | Y             | N           | N                 |

problems for complex collection manipulations as used in this case study. All the versions may be found at [http://www.dcs.kcl.ac.uk/staff/kcl/uml2web/pn2sc/](http://www.dcs.kcl.ac.uk/staff/kcl/uml2web/pn2sc/).

Table 2 shows the summary table completed for our solution.

The optimisation provided (for rules Post1, Post2, Post3) is to omit tests for the truth of the succedent of the rule (i.e., the negative application condition of the rule) when applying the rule: the system can detect that a formula such as \( \text{self} \rightarrow \text{isDeleted}() \) is inconsistent with the positive application condition of the rule, and therefore that there is no need to evaluate the formulae before applying the rule.

The transformation can be reversed by reversing the initialisation.

References

[1] K. Lano, S. Kolahdouz-Rahimi, *Constraint-based specification of model transformations*, Journal of Systems and Software, to appear, 2012.

[2] K. Lano, S. Kolahdouz-Rahimi, T. Clark, *Comparison of model transformation verification approaches*, Modevva workshop, MODELS 2012.

[3] SHARE site for solution: [http://is.ieis.tue.nl/staff/pvgorp/share/?page=ConfigureNewSession&vdi=XP-TUe_TTC13::XP-NMF_Petri_Nets_State_C_pn2sc1.vdi](http://is.ieis.tue.nl/staff/pvgorp/share/?page=ConfigureNewSession&vdi=XP-TUe_TTC13::XP-NMF_Petri_Nets_State_C_pn2sc1.vdi), directory Desktop/pn2scLano/pn2sc/, 2013.

[4] Pieter Van Gorp, Louis Rose, *The Petri-Nets to Statecharts Transformation Case*, Sixth Transformation Tool Contest (TTC 2013), 2013, EPTCS, this volume.

[5] UML-RSDS toolset and manual, [http://www.dcs.kcl.ac.uk/staff/kcl/uml2web/](http://www.dcs.kcl.ac.uk/staff/kcl/uml2web/) 2013.