A Unified Toolset for Business Process Model Formalization

Behnaz Changizi[1,2], Natallia Kokash[3], Farhad Arbab[4]

Science Park 123, 1098XG
Centrum voor Wiskunde en Informatica
Amsterdam, The Netherlands

Abstract

In this paper, we present a toolset to automate the transformation of Business Process Modeling Notation (BPMN), UML Sequence Diagrams, and Business Process Execution Language (BPEL), into their proposed formal semantics expressed in the channel-based coordination language Reo. Such transformations enable the animated execution and verification of the aforementioned notations with the help of verification and model checking tools available for Reo.

Keywords: Business process models, Reo, ECT, verification, model transformation.

1 Introduction

Business process modeling is a fundamental activity in system engineering to identify, refine, implement, monitor and manage enterprise processes. Multiple notations have been created to give business analysts and software developers the ability to communicate in a standard manner. Unified Modeling Language (UML) and Business Process Modeling Notation (BPMN) are among the notations that have established themselves as de facto standards.

BPMN is a graph based notation for modeling enterprise business processes which represents control flow using different types of events, gateways, exceptions and compensation associations.

UML is a graphical language used to specify, visualize and document the artifacts of a software-intensive system. It offers a standard way to visualize a system's architectural blueprints, including elements such as business processes, components and activities. UML2.x offers 14 types of diagrams divided into two categories:

[1] This work is part of the ICT COMPAS project, funded by the European Commission, FP7-ICT-2007-1 contract number 215175, http://www.compas-ict.eu/.

[2] Email:b.changizi@cwi.nl

[3] Email:natallia.kokash@cwi.nl

[4] Email:farhad.arbab@cwi.nl

7 diagram types provide means to describe structural information, while the rest allow designers to represent system behavior, including 4 diagrams for modeling actor interaction. Of these categories, it is the dynamic behavior diagrams that are often used for modeling business processes, such as the UML Activity Diagram and the Use Case Diagram. UML Sequence Diagrams (SDs) represent a useful model for visualizing message exchanges between involved entities.

To simplify the integration of various pieces of software into value-added applications, the idea of service-oriented computing has emerged. BPEL is a standard XML-based executable language for workflow-based composition of web services. It describes interactions between services in terms of communication actions.

Similar to many other industrial standards, these notations do not have commonly agreed upon formal semantics. Many efforts have been directed at disambiguating and formalizing these languages. Among the formalisms used for this purpose are Petri nets [14] and Reo [2]. Petri nets constitute a graph-based modeling language for the description of distributed systems. Reo is a channel-based language for exogenous coordination of software components and services. Both notations have graphical syntax and exact mathematical definitions of their execution semantics. We foster the use of Reo for business process modeling due to its compositional semantics and more advanced synchronization mechanism. Moreover, Reo better fits the paradigm of service-oriented computing which in the past few years has become the common trend in Web system development. Being compositional, Reo simplifies refinement in a step-wise manner, which enables reasoning about process properties without taking the whole model into consideration. Capability of Reo to represent and verify business processes is discussed in [4].

In this paper, we present a toolset to automate the transformation of BPMN, BPEL and UML into the Reo coordination language. After such a transformation, refined and annotated Reo process models can be visualized, verified and transformed into executable code with the help of the Eclipse Coordination Tools (ECT) [3]. In this paper, we solely consider UML SDs among UML diagrams. Transformation of the UML Activity Diagrams can be performed similarly to that of BPMN and is not discussed here. Our focus on these notations is primarily due to their exploitation in the EU COMPAS project [5] for the design of business process fragments. Process fragments in this context can be understood as design patterns for the implementation of service-based systems compliant to relevant legislation and organizational policies.

The rest of this paper is organized as follows. In Section 2, we describe the Reo coordination language. In Section 3, we present the tools for generating Reo process models from BPMN, UML SDs and BPEL specifications. In Section 4, we illustrate the application of one of the conversion tools, BPMN2Reo, in a case study. In Section 5, we overview related work. In Section 6, we outline our future work and conclude the paper.

[5] http://www.compas-ict.eu/

2 Reo

Reo is a channel-based language for exogenous coordination of software components and services. Reo supports modeling of complex behavioral protocols using a small number of predefined primitives, called *channels*. Each Reo channel has two ends, each of which is either of *source* type or *sink* type. A *source* end reads data into its channel, while a *sink* end writes data out. Both ends of a channel can have the same type. The open-ended set of channels in Reo exhibits a variety of behavioral policies on the data flow in terms of synchronization, lossiness, buffering, mutual exclusion, etc.

Figure 1 shows some basic Reo channels. The *Sync* channel, which has a *source* and a *sink* end, accepts data from its source end iff data can be simultaneously dispensed through its sink end. The lossy variant of *Sync*, the *LossySync* channel, differs from *Sync* in that it always accepts incoming data through its source end, but it loses this data if its sink end cannot dispense the data simultaneously. A *SyncDrain* channel has two source ends. It accepts data through its ends iff both ends are ready to interact at the same time and immediately loses this data. *SyncDrain* loses all accepted data. A *FIFO1* has a source and a sink end and a buffer of size one. If the buffer is empty, the *FIFO1* accepts incoming data through its source end and buffers it. If the buffer is full and its sink end is ready to dispense data, the channel takes data out of buffer and sends it out through its sink end (thus, the buffer becomes empty). The *FIFO1* channel ends do not act simultaneously.

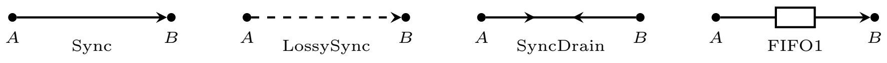


Fig. 1. Some basic Reo channels.

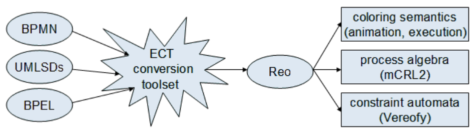
Reo Channels can be connected via nodes to form *connectors*, also called *circuits* or *networks*. Reo supports hierarchical modeling by allowing users to abstract from the internal structure of a connector by converting it to a *component*, a stand-alone entity with an observable behavioral interface.

The Eclipse Coordination Tools (ECT) [6] is a framework for developing component-based software using the Reo coordination language. This framework consists of a set of integrated tools that are implemented as Eclipse plug-ins. Currently, ECT provides support for the design, animation, analysis, verification, execution and transformation of Reo connectors based on their operational semantics, most notably : colorings and constraint automata.

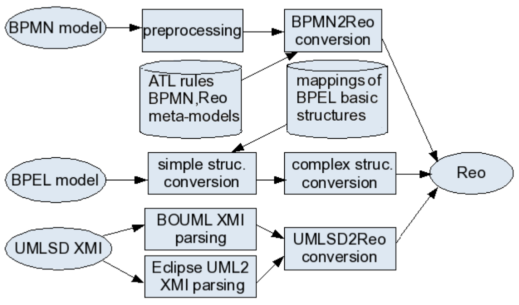
3 Mapping BPMN, UML SDs and BPEL to Reo

In this section, we discuss tools for converting BPMN, UMLSD and BPEL process models to Reo circuits. The theoretical basis of this work was presented in [5,6,15]. We provide the first implementation for the BPMN to Reo conversion, and a considerably refined implementation of the BPEL to Reo converter. The first version of the latter converter, developed at the University of Tehran, was erroneous and

[6] http://reo.project.cwi.nl/


BPMN
UMLSDs
BPEL
ECT conversion toolset
Reo
coloring semantics (animation, execution)
process algebra (mCRL2)
constraint automata (Vereofy)

(a) An overview of the ECT converters toolset



(b) ECT converters toolset architecture

Fig. 2. ECT converters toolset

inefficient due to its failure to compose different parts of Reo circuits and the absence of abstraction. Additionally, we improve the UML SDs to Reo conversion tool developed at the University of Leiden. In our version of this tool, users no longer need to specify UML SDs manually in a text file. The tool accepts graphical models created with UML design tools. All three tools are integrated into the ECT framework. Graphical representation of process models in Reo helps us to trace the errors discovered using model checking tools back to the source models. This would not be possible if we assumed direct conversion to automata-based models or process algebras. The architecture of the ECT converters toolset is shown in figures 2(a) and 2(b).

3.1 BPMN2Reo

The implementation of the BPMN2Reo converter essentially consists of a set of pattern matching and transformation rules, implemented in Atlas Transformation Language (ATL) [1], a general-purpose model transformation framework available as an Eclipse plug-in. The BPMN meta-model specification used for transformation is provided by the Eclipse BPMN modeler [7]. ATL supports both declarative and imperative forms of programming. An ATL transformer consists of several rules that match source model elements to generate output model elements. ATL also allows developers to define auxiliary methods or helpers. Using this framework, one can achieve a better separation of concerns and simplify transformation code maintenance.

The ATL rules for the BPMN to Reo conversion are defined based on the conceptual mapping of basic BPMN modeling primitives to Reo presented in [5]. BPMN modeling primitives split into two groups. The first group contains basic structures such as *task*, *parallel join* or *merge* that can be mapped to Reo circuits solely using ATL rules. The second group contains structures such as exceptions, compensations and subprocesses, that require a preprocessing step. The BMPN2Reo ATL transformation module consists of 16 ATL rules and 33 helpers. It is purely declarative and acts in a top-down manner, starting from the top-most element in the source model, *BpmnDiagram*, and transforms it into the Reo top-most element, *Module*, which may contain several Reo connectors and components. During the application

[7] http://www.eclipse.org/bpmn/

```
rule sequenceEdge2sync
{
    from
        q : bpmn!SequenceEdge(not(q.source.Fifoable or q.source.IsRuleEvent or q.source.IsXorDb
            or q.source.IsInclusiveOr_Open or q.source.IsTimerEvent or q.source.IsMessageEvent
            or q.IsSpecial))
    to
        y : reo!Sync(sourceEnds <- c, sinkEnds <- k),
        c : reo!SourceEnd(node <- q.source),
        k : reo!SinkEnd(node <- q.target)
}
```

Fig. 3. A sample of ATL rule used in BPMN2Reo

of each rule, the rules required to transform the nested elements are invoked recursively. Figure 3 depicts a sample BPMN2Reo ATL rule, which is used to transform a *SequenceEdge* element, or a BPMN control flow arrow, into a Reo synchronous channel. Each rule has a *from* segment which is augmented with a precondition that ensures its application takes place in a proper context. The *to* part describes the structures that must be generated by the rule. In our example, these consist of a synchronous channel connected to two Reo nodes. To complete the transformation, this rule then invokes the rules that perform the transformation of the source and the target of the input *SequenceEdge* in order to generate the output channel ends.

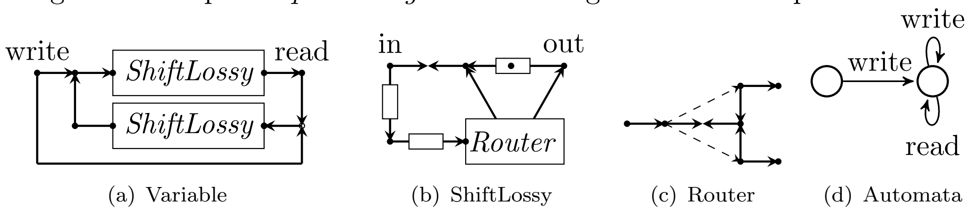


Fig. 4. Mapping of the BPEL variable to Reo: circuits and constraint automata

3.2 BPEL2Reo

The BPEL2Reo transformer is based on the approach proposed in [15]. The main motivation for our work is to resolve problems we discovered in the tool presented in [15]. One of the problems is that it generates a number of disjoint nodes in an output Reo model. This happens because the tool developers fail to properly connect separate Reo circuits generated for various BPEL structures. Another problem is the large size of the output models for relatively small BPEL structures. For example, the mapping of the widely used BPEL element *variable* leads to a Reo circuit shown in Figure 4(a), which consists of two *ShiftLossy* circuits shown in Figure 4(b). Each *ShiftLossy* circuit, in turn, contains a *Router* connector shown in Figure 4(c). Thus, the mapping of a single BPEL variable produces a circuit with more than 20 channels, including 6 channels with asynchronous FIFO1 buffers. For any middle-size BPEL specification containing dozens of variables, such a mapping already leads to a state explosion problem. We have resolved this issue by employing components. Thus, rather than dealing with complex connectors, we map some BPEL elements to components whose behavior is described by fairly small constraint automata. Figure 4(d) shows such an automaton for the BPEL variable element. Intuitively, the states of this automaton represent uninitialized and initialized states of a variable: once a value is written to the variable, it can be read infinitely many times, as well as reset to another value written to the source node of the circuit. In the aforementioned automaton we abstract from the data constraints. However, for enabling data analysis, different values of the variable will correspond to different states of the automaton. The components that correspond to basic BPEL structures such as *variable*, *receive* or *reply* operations are predefined at the design time, and then loaded and placed into a target Reo model at the runtime. Compound structures such as *sequence*, *while* and *pick* are transformed into a set of channels that glue the appropriate basic structures. In this way, we achieve a significant performance optimization compared to the previous version of the tool.

3.3 UMLSD2Reo

The last plug-in of the ECT converter toolset, UMLSD2Reo, accepts UML2.x SD models as input and generates a set of Reo circuits for representing message exchange between communicating actors. The communication protocols expressed in Reo then can be visualized and verified automatically. This is especially important in the context of case-based development as it allows designers to reason about global communication schemas given the interaction protocols of particular entities.

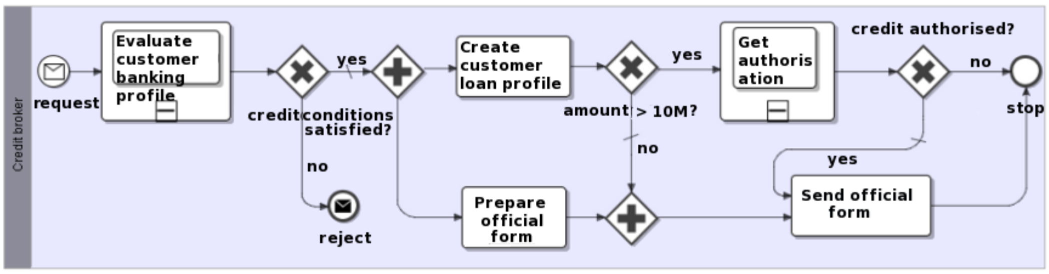
The initial version of this conversion tool was developed by Eccher Alessandro at the University of Leiden. It acts as a stand-alone program which accepts input files with message sequences in a proprietary text format. This hampers the tool's interoperability with other frameworks, in particular, with UML design toolkits, which makes the tool hard to use in practice. We modified the converter to enable the transformation of the models generated by a UML2 graphical designer called Bouml.[8] The choice of Bouml is justified by the fact that it is a free UML2 tool that runs under all major operating systems and supports the exchange of models via an eXchanging Model Information (XMI) format. XMI aims at providing a standard specification for exchanging model information between tools and repositories, but each UML2 tool has its own implementation of XMI. Therefore, the idea of exchanging UML2 models between tools is not yet practical. For this reason, in order to support each UML2 tool, we are required to have a corresponding XMI parser. Figure 2(b) shows XMI parsers of Bouml and Eclipse UML2 tool. The XMI parser parses the input and loads the model into objects that represent the elements of UML SDs (e.g., actors, lifelines, combined fragments, messages). After that, the loaded model is traversed and its elements are translated to Reo connectors.

4 Case Study

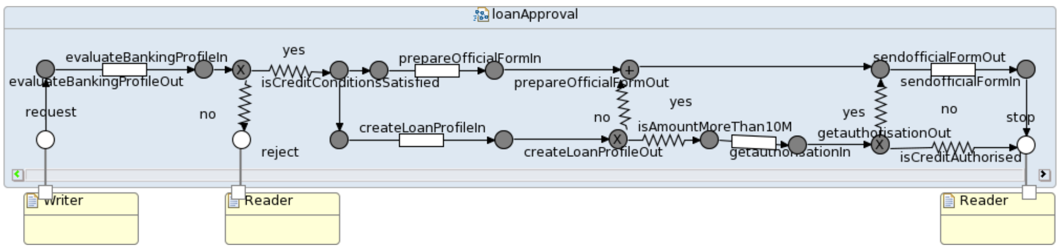
In this section, we show the application of the ECT conversion toolset in practice. Due to the space limit, here we consider only one of the transformers, BPMN2Reo.

Figure 5(a) shows a BPMN model for our case study, a Loan Approval process. In this process, a user request for a loan is not approved if either the customer's credit status is not satisfactory or the user request for a large amount is not authorized. The model contains several structural errors, namely, if the authorization for a large loan does not arrive, the credit broker will not be notified and may continue to prepare the official documents. Also, in the case of request rejection, the user will

[8] At the time of development, BOUML (http://bouml.free.fr/) was the only free UML2 tool we found that supported both Activity Diagrams and XMI format export. Recently Model Development Tools (MDT) project for Eclipse Galileo has provided support for UML2 Activity Diagrams. Now we are looking forward to a more stable version of MDT to integrate in our toolset.



(a) BPMN model



(b) Reo circuit

Fig. 5. Modeling a Loan Approval scenario

not receive any reply from the bank. The discovery of such errors and inconsistencies in more complex models is not an easy task. Our transformation tool provides the facilities to verify models automatically.

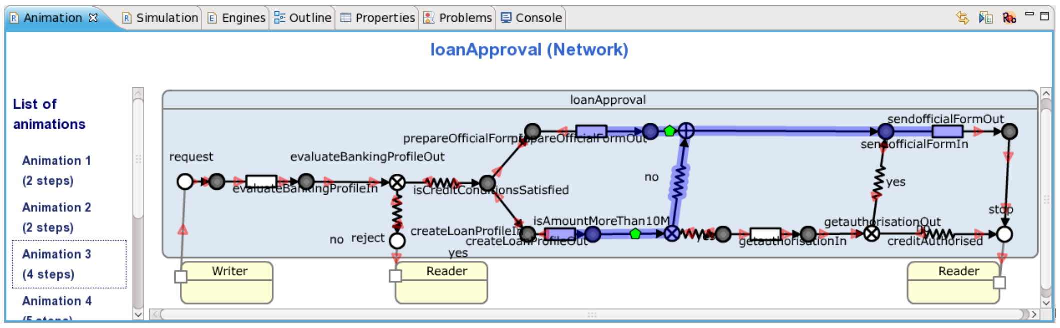
Some of the rules used for this conversion are:

- A BPMN *Task* is mapped to a Reo *FIFO1* channel with two *Replicate* nodes attached to its ends.
- A *Data based exclusive OR gateway* with one incoming and multiple outgoing *sequence edges* is converted to a Reo *Route* node, while the incoming and outgoing *sequence edges* are mapped to multiple *Filter* channels and a *Sync* channel respectively.
- A *Parallel gateway* with multiple incoming *sequence edges* and one outgoing *sequence edge* is transformed to a *Join* node and its incoming and outgoing *sequence edges* become *Sync* channels.
- *Message* and *Empty* events are mapped to *Replicate* nodes connected to *Writer* and *Reader* components depending on whether they are of type *Start* or *End* respectively.

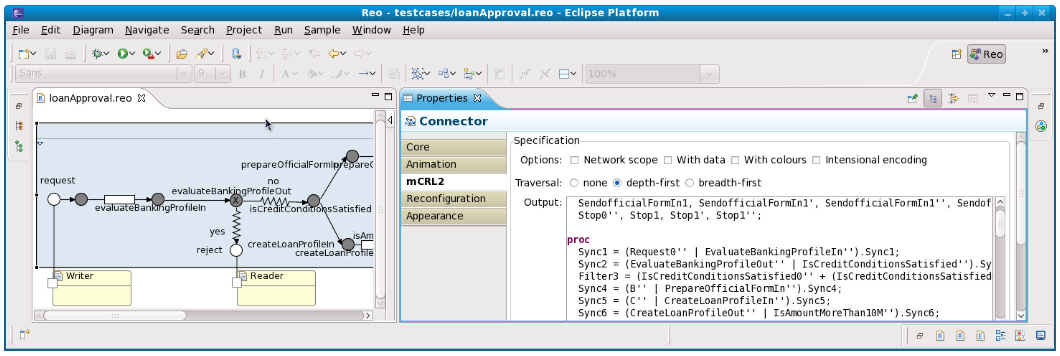
Figure 5(b) depicts the Reo circuit obtained from the presented BPMN model by our converter. Such a Reo circuit can then be converted into constraint automata or process algebra specifications suitable for automated analysis, or animated with the ECT animation engine. Figure 6(a) shows an animation view that allows designers to simulate process execution step by step prior to its implementation. Figure 6(b) depicts a snapshot of the ECT view with the process algebra specification of this model suitable for model checking with the mCRL2[9] toolset.

For example, We can analyze a structural properties of the presented

[9] http://www.mcrl2.org/



(a) Animation view



Specification

Options: ☐ Network scope ☐ With data ☐ With colours ☐ Intensional encoding

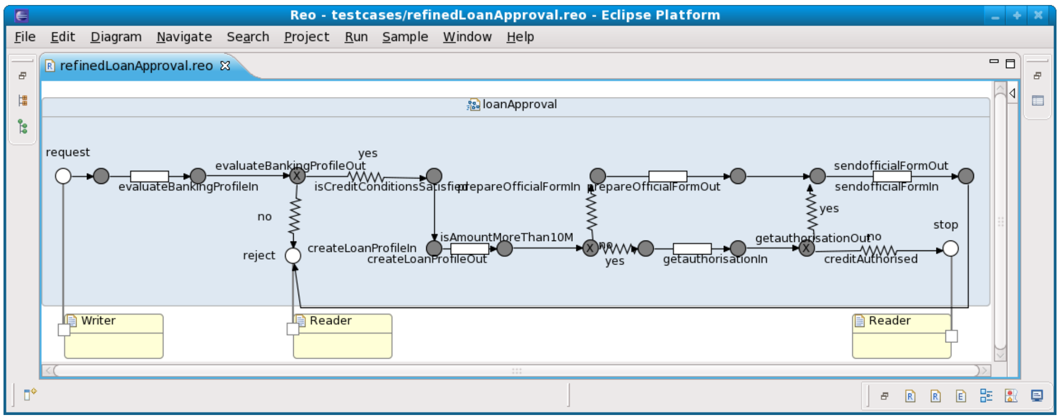
Traversal: ○ none [✓ depth-first] ○ breadth-first

Output:

```
  SendofficialFormIn1, SendofficialFormIn1', SendofficialFormIn1'', Sendof
  Stop0'', Stop1, Stop1', Stop1'';

proc
  Sync1 = (Request0'' | EvaluateBankingProfileIn'').Sync1;
  Sync2 = (EvaluateBankingProfileOut'' | IsCreditConditionsSatisfied'').Sy
  Filter3 = (IsCreditConditionsSatisfied0'' + (IsCreditConditionsSatisfied
  Sync4 = (B'' | PrepareOfficialFormIn'').Sync4;
  Sync5 = (C'' | CreateLoanProfileIn'').Sync5;
  Sync6 = (CreateLoanProfileOut'' | IsAmountMoreThan10M'').Sync6;
```

(b) Process algebra mCRL2 specification generated from the formalized model



(c) A refined version of Loan Approval process

Fig. 6. Analyzing a Loan Approval scenario

Loan Approval process. As figure 5(b) shows, if the requested loan amount is larger than 10M, the prepared form will not be used. This problem can be discovered by checking the following formula in μ calculus, a property specification language accepted by mCRL2 model checker: $[true^{\star}.PrepareFormIn.true^{\star}]\,\langle true^{\star}.SendFormOut.true^{\star}\rangle\, true$. This property specifies that the occurrence of the *SendFormOut* action in any execution trace after the *PrepareFormIn* action. As expected, the verifier evaluates the formula on the model as *false*. Figure 6(c) demonstrates a refined version of the process, for which the discussed property holds.

5 Related Work

Despite the fact that many approaches to formalize BPMN, UML and BPEL specifications have been proposed, not many software tools are available to do this automatically. Among these approaches, those that translate into Petri nets have the best computer-aided tool support and remain among the most widely used frameworks for the analysis of industrial business processes. For example, the ability to convert BPMN models into Petri nets is provided by the BPMN2PetriNet[10] tool. The target Petri net model can then be analyzed using a process mining toolkit called ProM[11], which supports the verification of temporal constraints and performance analysis. Raedts et. al [9] present a tool for converting BPMN to Petri nets extended with inhibitor and reset arcs. Such models are then converted to the process specification language mCRL2 and verified against a wide variety of formally defined properties. In ECT, we follow a similar approach. In our recent work [11], we introduced support for the unified control and data flow analysis of the Reo process models with abstract data types using the mCRL2 toolset. Such kind of analysis is not available in the aforementioned Petri-net based toolkits. Together with this contribution, the conversion of process models to Reo offers an automated model-driven approach to verifiable development of business processes and service-based systems.

Hinz et al. [10] describe a tool called BPEL2PN that implements the transformation of BPEL specifications abstracted from data to Petri nets. The resulting Petri nets can be analyzed using the Low Level Analyzer (LoLA) tool, which supports verification of the standard properties of Petri nets, such as deadlock detection, and verification of properties expressed in the CTL temporal logic. A detailed overview of the approaches and tools for BPEL analysis is given in [13]. As mentioned above, due to its compositionality and expressiveness, Reo provides certain advantages over Petri nets. Moreover, being a coordination language, Reo is conceptually closer to BPEL than Petri nets in the sense that it aims at expressing "glue" code for composing external components or services. This makes its use to formalize BPEL more intuitive.

Bernardi et al. [7] present a tool for converting UML SDs and Statecharts to the generalized stochastic Petri nets. Among other works following this approach are [8,12]. The mapping from UML SDs to Reo appears to be more natural as there is a clear analogy between a binary message exchange and a channel. Reo distinguishes synchronous and asynchronous channels and supports propagation of synchrony, while the treatment of synchronous communication in Petri nets becomes problematic and non-compositional.

Because of the space limits, we do not discuss here all the tools for converting business process models directly to finite automata and process algebra specifications. Usually these approaches require very detailed input models which are difficult to obtain with high-level modeling notations like BPMN. Moreover, they make it difficult to relate the counterexamples with initial models and, thus, do not assist the designers with process refinement.

[10] http://www.bpm.fit.qut.edu.au/research/projects/babel/tools/
[11] http://prom.win.tue.nl/tools/prom/

6 Conclusion and Future Work

In this paper, we presented a toolset for converting specifications in the high-level business process modeling languages BPMN, BPEL and UML SDs, into circuits in the Reo coordination language. This enables process verification with subsequent automated code generation from these models. Currently, we are working on a transformation tool to convert the UML Activity Diagrams into Reo and on its integration within the presented environment. Moreover, we plan to extend the BPEL2Reo converter with the ability to deal with the BPEL4People [12] specification. Finally, we are considering other model transformation tools targeted at giving Reo-based semantics to various process modeling notations.

References

[1] Allilaire, F., J. Bézivin, F. Jouault, I. Kurtev and P. Valduriez, *ATL: a QVT-like Transformation Language*, in: *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications* (2006), pp. 719–720.

[2] Arbab, F., *Reo: A Channel-based Coordination Model for Component Composition*, Mathematical Structures in Computer Science **14** (2004), pp. 329–366.

[3] Arbab, F., C. Koehler, Z. Maraikar, Y.-J. Moon and J. Proença, *Modeling, Testing and Executing Reo Connectors with the Eclipse Coordination Tools* (2008), tool session.

[4] Arbab, F. and N. Kokash, *Formal Behavioral Modeling and Compliance Analysis for Service-Oriented Systems*, in: *Formal Methods for Components and Objects*, 7th International Symposium **5751** (2009), pp. 21–41.

[5] Arbab, F., N. Kokash and M. Sun, *Towards using Reo for Compliance-aware Business Process Modelling*, in: *Proceedings of the International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, LNCS **17** (2008), pp. 108–123.

[6] Arbab, F. and M. Sun, *Synthesis of Connectors from Scenario-based Interaction Specifications*, in: *Proceedings of the International Symposium on Component Based Software Engineering (CBSE'08)*, LNCS **5282** (2008), pp. 114–129.

[7] Bernardi, S., S. Donatelli and J. Merseguer, *From UML Sequence Diagrams and Statecharts to Analysable Petri Net Models*, in: *Proceedings of the 3rd International Workshop on Software and Performance* (2002), pp. 35–45.

[8] Fernandes, J., J. Jorgensen, O. Ribeiro and S. Tjell, *Designing Tool Support for Translating Use Cases and UML 2.0 Sequence Diagrams into a Coloured Petri Net*, in: *Proceedings of the Sixth International Workshop on Scenarios and State Machines* (2007), p. 2.

[9] Groote, J., M. Petkovic, I. Raedts, Y. Usenko, L. Somers and J. van der Werf, *Transformation of BPMN Models for Behaviour Analysis*, in: *Proceedings of the 5th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems* (2007), pp. 126–137.

[10] Hinz, S., K. Schmidt and C. Stahl, *Transforming BPEL to Petri Nets*, in: *Proceedings of the International Conference on Business Process Management*, LNCS **2649** (2005), pp. 220–235.

[11] Kokash, N., C. Krause and E. de Vink, *Data-aware Design and Verification of Service Compositions with Reo and mCRL2*, in: *ACM Symposium on Applied Computing* (2010), to appear.

[12] Staines, T., *Supporting UML Sequence Diagrams using A Processor Net Model*, IEEE International Conference on the Engineering of Computer-Based Systems **0** (2007), pp. 279–286.

[13] van Breugel, F. and M. Koshkina, *Models and Verification of BPEL* (2006), extensive survey.

[14] van der Aalst, W., *The Application of Petri Nets to Workflow Management*, The Journal of Circuits, Systems and Computers **8** (1998), pp. 21–66.

[15] Zilouchian Moghaddam, R., M. Sirjani, S. Tasharofi and M. Vakilian, *Modeling Web Service Interactions using the Coordination Language Reo*, in: *Proceedings of the 4th International Workshop on Web Services and Formal Methods*, LNCS **4937** (2007), pp. 108–123.

[12] http://www-128.ibm.com/developerworks/webservices/library/ specification/ws-bpel4people/