Simulation and Formal Analysis of Workflow Models

Máté Kovács [1,3], László Gönczy [2,3]

Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary

Abstract

We present a framework for the simulation and formal analysis of workflow models. We discuss (i) how a workflow model, implemented in the BPEL language, can be transformed into a dataflow network model, (ii) how potentially incorrect execution paths can be incorporated, and (iii) how the properties of a workflow can be formally verified using the SPIN model checker. For the several model transformation steps from workflow to analysis models, we use graph transformations.

Key words: BPEL, Workflow, Verification, Fault simulation, Dataflow Networks

1 Introduction

In the past data was kept on paper. A piece of paper, containing information, could be interpreted as sort of a token flowing through the basic activities. This kind of a paper is called the *work item*. The colleagues carry out *activities* on work items. The *workflow* comprises all the activities. As such, the workflow defines the order in which the activities have to be carried out.

Today offices have significant IT infrastructure to enhance the efficiency and productivity often using computer aided business process coordination. There are several languages that allow a very high-level, executable description of workflows, for instance, BPEL (Business Process Execution Language) [11] or XPDL (XML Process Definition Language) [13]. The complexity of workflows is close to that of regular programming languages. Therefore, new problems arise with electronic business process execution.

[1] Email: km432@hszk.bme.hu
[2] Email: gonczy@mit.bme.hu
[3] This work was partially supported by the SENSORIA European project (IST-3-016004).

Computer-based workflow execution involves communication between loosely coupled information systems. This makes the testing of distributed workflows very difficult as data taken from several databases is often required to be manipulated. Moreover, the side effects of the transactions generated in a test phase need to be undone. Another choice is to establish the entire test environment with multiple servers and databases containing the test data. Both solutions are time consuming and expensive.

As a consequence, there are several semantic requirements that a workflow has to meet before enactment, such as:

- There must not be any deadlock. In case of a deadlock the workflow execution would come to a halt.
- All activities have to be reachable. In case of unreachable activities there might be unused resources in the company, which is far from desirable.
- Each variable has to be written before being read. Reading an uninitialized variable could lead to an unpredictable result.

The most important contribution of the framework is the ability to check the last requirement of those above mentioned. There are other solutions to verify the first two [1,7].

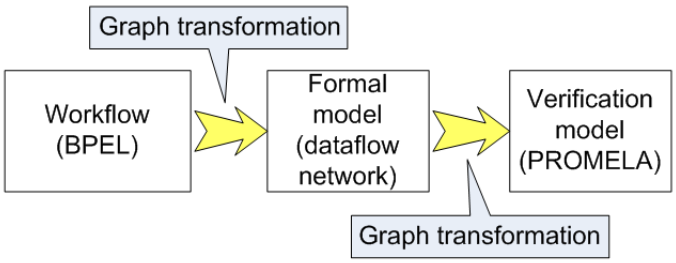

Fig. 1. The workflow analysis method

In this paper we discuss a method to formally verify the above mentioned properties of a BPEL model. As Fig. 1 illustrates the high level description of workflows, such as BPEL, needs to be transformed into a low level mathematical notation which can be verified automatically. We have chosen dataflow networks [2] for this purpose.

The formalism of dataflow networks is meant to model complex, distributed computing systems with well defined semantics. The abstraction level of this formalism is between the BPEL model and the verification model, the PROMELA implementation. It combines the state based description of finite state automata, and the data (token) flow of Petri nets. A further advantage of modeling with dataflow networks is that with additional rules and states of nodes, *fault simulation* can also be performed.

Workflows can be checked against requirements such as mentioned above. The properties of the process that are to be checked need to be formulated as linear temporal logical expressions (LTL), regarding the PROMELA (Process

Meta Language) [12] implementation of the dataflow network representation of the workflow. The SPIN model checker [12] will evaluate these LTL expressions.

2 From workflow models to dataflow networks

2.1 Workflows

Workflow description languages resemble very much to regular structured programming languages. The structural elements of workflows are the sequence, selection, iteration and parallel execution constructs.

Activities have input and output parameters. These data elements are called *messages* that are passed between different computers running in a distributed environment.

In BPEL data is maintained using variables. The value of variables may be sent and received as messages, and the control flow may be determined by them. The manipulation of the variables is called data handling.

We assume that the workflow to be checked is implemented in BPEL, using only a subset of the language. The transformation deals with the *basic* and *structured activities* of BPEL - like those in Fig. 2 - and data handling but all kinds of event handling is ignored.

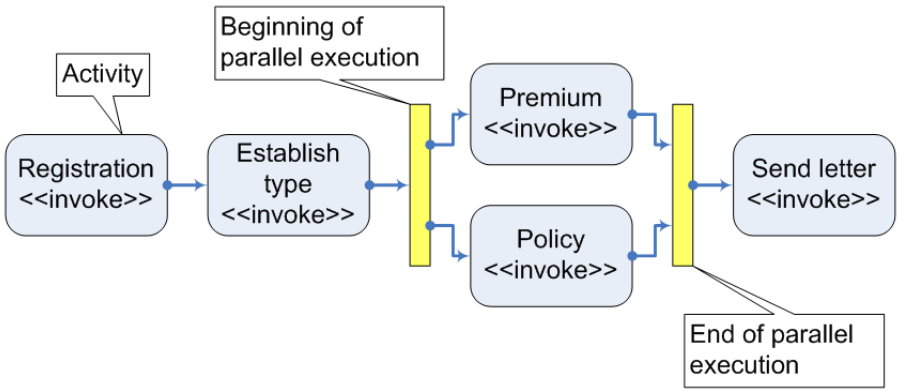

Fig. 2. Workflow concepts and corresponding BPEL keywords

A small fragment of the workflow in an insurance company is shown in Fig. 2. First, the client reports the damage which is recorded. Then the type of the damage is established. Next the insurance company has to decide whether to compensate the damage or not. This needs two independent activities that can be executed simultaneously. Finally a letter is sent to the client containing information about the decision.

2.2 Dataflow networks

Dataflow networks are designed to model distributed communicating systems. A dataflow network consists of data processing *nodes* interconnected with

channels. Channels transmit *tokens* between nodes. A channel does not contain the token queue. In contrast of the original [2] dataflow network formalism, we define *ports* which contain the token queue. A port is the connection between a node and a channel. A token remains in the input port until it is not removed during the application of a firing. A port can potentially contain an infinite number of tokens. A token is an atomic abstract data unit represented by its color. Each node is a finite state automaton that has *states*, and state transition *rules*. A rule consists of two parts. The first defines the *firing condition*, the second declares the *action* that should be performed in case of a firing. During the transition the node removes the tokens according to the condition, changes the state, and puts several tokens to the output ports.

2.3 Mapping workflows to dataflow networks

In the dataflow network model the control flow is represented by tokens of a special color: the *control tokens*. The number of control tokens in the network corresponds to the number of activities executed in parallel.

The execution order of activities can be defined by the structured activities: sequence, selection, iteration and parallel execution. In the dataflow network model they are represented by at least two nodes. The one at the beginning distributes the control tokens, the final one collects them representing the synchronization of the branches. The control constructs - that perform a selection - need additional nodes to represent the evaluation of the conditional expressions.

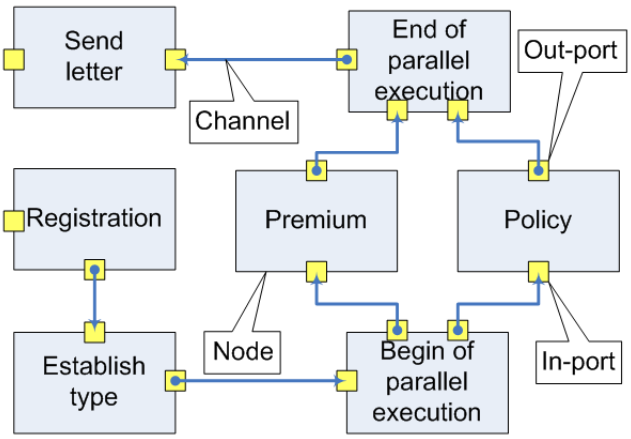

Fig. 3. Mapping the workflow in Fig. 2 to dataflow network

Applying the transformation rules on the workflow in Fig. 2 results in a dataflow network illustrated in Fig. 3. The rules are summarized in the table below.

workflow pattern	dataflow network pattern
variable	a node with appropriate state space (see Fig. 4)
basic activity	two nodes if the activity has in and out parameters, otherwise one node
sequence	two extra nodes, one at the beginning and one at the end
parallel execution	one start node multiplying the control flow, and one end node restoring the single control flow
selection (switch)	one node at the beginning and one at the end, the cases linked to each other according to the order of condition evaluation
a *case* in a selection	a node for each variable in the expression of the condition
iteration (while)	a modified selection with two branches: one leading to the activity being iterated and then back to the condition, the other leading to the next activity in the workflow

The transformation does not preserve the concrete values of variables. We only use an abstract state space to check the order of the variables being initialized, read, and written.

3 Fault simulation

A workflow execution engine can coordinate the work of many people, and it is usually connected to multiple computers of independent organizations. These computers are loosely coupled with no guarantee on their availability. Thus failures have to be considered. It is reasonable to add some redundancy to the workflow and then to check whether the planned fault tolerance was reached.

Error propagation [8] in the control flow is modeled with tokens of a specific color: the *faulty control tokens*. The abstract state of variables (i.e. "Written", "Written and read", etc.) is represented by the states of nodes.

3.1 Simulation of dataflow errors

In this case the error is only spreading across the variables but it does not have an effect on the control flow.

Fault injector activities are needed which write faulty data to their output

regardless of the input and the color of the control token they received. We assume that healthy nodes always write healthy data to their output unless the control or the input is erroneous. This way the error confinement region of a fault injector variable can be determined.

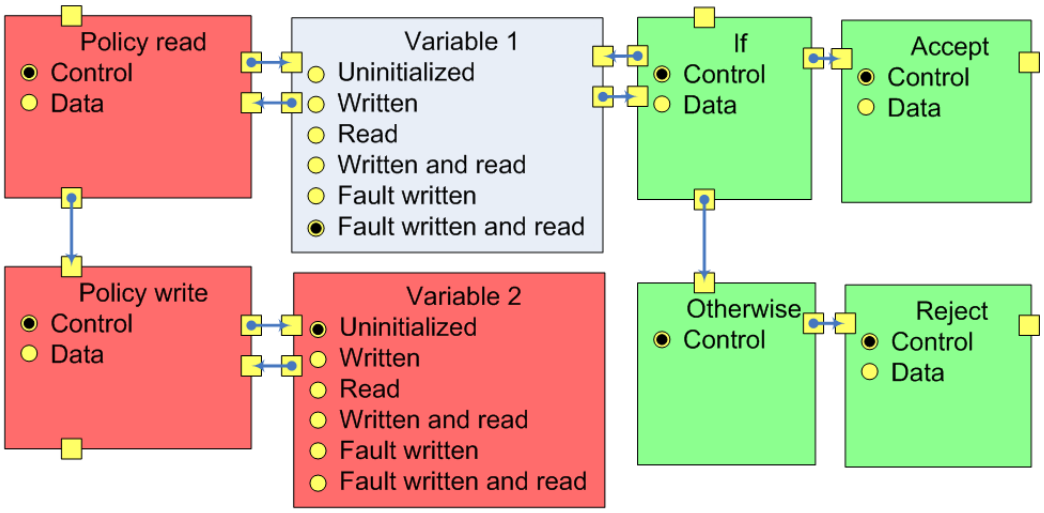

Fig. 4. Fault spreading

In Fig. 4 a small fragment of the dataflow network representation of the insurance company's workflow is shown. Variable 1 contains faulty data. The red nodes are involved with spreading the error among variables, the green nodes deal with control flow infection.

1. In the first step "Policy read" receives a control token in its input port. Changes the state "Control" to "Data" and sends a token of color *reading* to its input variable.
2. The reading token is received by the node representing the input variable (Variable 1). Now we assume that the variable contained faulty data i.e. the input variable's state was either "Fault written" or "Fault written and read". The state is switched to "Fault written and read" and a faulty control token is placed on the output port.
3. A faulty control token is received from Variable 1. The state is switched from "Data" to "Control" and a token of color *faulty write* is put on the output port towards the second node of the activity.
4. "Policy write" receives the faulty write token from its control port, switches the state from "Control" to "Data" and sends a faulty write token to its output variable.
5. "Variable 2" switches to "Fault written" state and sends back a control token to the second node of the activity.
6. The second activity node receives the control token from the output variable, switches its state to "Control" and sends a control token to its output port.

3.2 Simulation of control flow errors

If a branching condition is evaluated using a faulty variable, the fault of the condition variable infects the control flow. This is illustrated by node "If" in Fig. 4.

1. The node called "If" is part of all the nodes of the switch construct. It receives a control token, then it changes the state from Control to Data and sends a reading token to the node called "Variable 1".
2. The state of the node representing a variable is changed from "Faulty written" to "Faulty written and read" and a faulty control token is sent back to the node of the selection construct.
3. The faulty control token is received by node called "If". It chooses non-deterministically whether to pass the token to the activity called Accept, or to the next condition.

   At this point the control token is changed to faulty control, all the activities that receive the faulty control token write faulty data and pass on faulty control token. Let us suppose that the faulty control token is sent to activity Accept.
4. The faulty control token is received by Accept, faulty data is written as it is shown in Section 3.1, and the faulty control token is passed on.

4 Verification of workflow models

As usually in structured programming languages, in PROMELA too we can use variables and subroutines called proctype. The types of variables are subsets of integer in order to guarantee that the program has finite state space.

The PROMELA provides a further type of variables, the FIFO channels. This way the channels of dataflow networks do not have to be implemented.

The SPIN is capable of exhaustive state space examination of a PROMELA program evaluating system requirements in the form of LTL expressions. This way the dynamic properties of a dataflow network can be verified.

The transformations are property preserving in a sence that every execution path of the BPEL process can be found in the PROMELA program too. If a property is valid, concerning the PROMELA model, then it holds for the BPEL process as well.

4.1 From dataflow network models to PROMELA implementation

Dataflow network constructs are mapped into PROMELA language patterns.

- Channels in the dataflow network are mapped into the channels of the PROMELA language. This can be done since there is always at most one token in a channel.

- Tokens are mapped into symbolic constants.
- State variables of a node are mapped into global integer type variables.
- Each node is mapped into a proctype construct. The initial state is set by the first instruction. The state transition rules are implemented by an infinite iteration containing the rules as conditional atomic sequence of instructions.
- A state transition rule of a node is mapped into an atomic sequence of instructions.

4.2 The verification

In Fig. 3 the activities called Premium and Policy are executed simultaneously. An interesting question is, whether the fault of one activity effects the other. The requirement needs to be formulated as LTL formula, the logical variables that take part in the formula have to be defined in form of C style define macros. Using the PROMELA source and the logical variable definitions the SPIN evaluates the LTL formula.

Example 4.1 Here we demonstrate how to formulate the requirement which states that the fault of activity "Policy out" should have no effect on "Policy in" regarding the model illustrated in Fig. 3.

```
#define Policy_out state_Policy_output!=Fault_written
#define Premium_in state_Premium_input!=Fault_written
G(!Policy_out -> Premium_in)
```

The first two lines define the meaning of the logical variables contributing to the value of the logical expression in the third line. Variable "Policy_out" is true, if and only if the value of the PROMELA variable "state_Policy_output" is "Fault_written". The meaning of "Premium_in" is defined analogously. The LTL formula of the requirement is shown in the third line. It states that at any state along the discrete time-line where "Policy_out" is false, "Premium_in" is true.

The evaluation of a formula, such as the one shown in Example 4.1, can result in positive and negative answers. The positive result guarantees that the PROMELA model meets the requirement. The negative may be a good test case of the BPEL process. However, because of the concrete values of BPEL variables, the process may not be able to run into the faulty execution path.

5 Related work

There is much research done about the formal verification of workflows implemented in several languages. In [9] a BPEL process is directly transformed into PROMELA code, the requirements are verified by SPIN. This approach

is similar to ours. The major difference is that we use an intermediate formal model of workflows, dataflow networks. This allows us to implement the workflow - PROMELA transformation in two steps, each a smaller step in abstraction level.

In [1] the formal semantics of workflow models are defined by Petri nets. This approach focuses on the syntactic properties of workflows but the insertion of structural flaws, such as hanging paths resulting in unnecessary tasks is prevented by the XML validation of the above mentioned languages. The authors of [7] discuss a design procedure to transform a business model of the workflow into the IT model that requires the decisions of engineers. The IT model is represented by Communicating Nondeterministic Automata that can be analyzed with NuSMV [5] model checker. As it is shown in [10] colored Petri nets can also be used as a formal model of workflows. When modeling with dataflow networks, we also have the advantage of the formalism's compositionality.

Another approach is presented by [3] to enhance the quality of a workflow by runtime monitoring. This technique could be successfully applied in a business environment with services and processes changing frequently. Our verification method is meant to be used at design time but could also be helpful to verify the implementation of a small change in an already enacted business process.

6 Conclusion

In this paper we proposed a method to check correctness properties of workflows implemented in BPEL. Dataflow networks are used to define the formal semantics of the workflow. The BPEL model is mapped into dataflow network, the dataflow network is mapped into a PROMELA model.

The model transformations, creating the dataflow network model of the workflow and generating the PROMELA code, are implemented as graph transformations executed within the VIATRA2 (Visual Automated Transformations) framework [14]. The VIATRA2 combines the procedural and declarative programming paradigms, enabling the efficient formulation of the implementation of model transformations.

The source of the transformation illustrated in Fig. 3 contains 42 graph patterns and several ASM rules. There is a graph transformation rule for each important construct of the BPEL language. The dataflow network - PROMELA transformation consists of 32 graph patterns.

In the future we plan to support the automated generation of LTL expressions from the requirements formulated in the BPEL domain. Furthermore, the automatic back annotation of the counterexample, presented by the SPIN in case of a negative result, is also a future goal.

References

[1] Wil van der Aalst, Kees van Hee, "Workflow Management", The MIT Press 2002.

[2] A.J. Anderson, Data Flow Systems, In *Multiple Processing: A System's Overview*, ch. 10. pp. 441-488. Prentice Hall, UK, 1989.

[3] L. Baresi, S. Guinea. *Towards Dynamic Monitoring of WS-BPEL Processes*, Proceedings of the 3rd International Conference of Service-oriented Computing (ICSOC'05). Amsterdam, The Nederlands, Lecture Notes in Computer Science, volume **3826** (2005), pages 269 - 282.

[4] E. Börger and R. Stärk, "Abstract State Machines, A method for High-Level System Design and Analysis", Springer-Verlag, 2003.

[5] A. Cimatti, E. Clarke, F. Giunchiglia, M. Roveri, *NuSMV: a new Symbolic Model Verifier*, In Proceedings Eleventh Conference on Computer-Aided Verification (CAV'99), number **1633** (1999) in LNCS, pages 495-499. Springer

[6] H. Foster, S. Uchitel, J. Magee, and J. Kramer, *Model-based verification of web service compositions* IEEE ASE 2003, Montreal, Canada

[7] J. Koehler, G. Tirenni, S. Kumaran, *From Business Process Model to Consistent Implementation: A Case for Formal Verification Methods*, EDOC 2002, pages 96-106.

[8] J. Laprie, B. Randell, C. Landwehr, *Basic Concepts and Taxonomy of Dependable and Secure Computing*, IEEE Transactions on Dependable and Secure Computing, (Vol.**1**, No.**1**) (2004) pp. 11-33

[9] S. Nakajima, *Verification of Web service flows with model-checking techniques*, presented at First International Symposium on Cyber Worlds, 2002.

[10] Y. Yang, Q. Tan, Y. Xiao, *Verifying Web Services Composition Based on Hierarchical Colored Petri Nets*, Proceedings of the first international workshop on Interoperability of heterogeneous information systems Bremen, Germany Pages: 47 - 54

[11] Specification of the Business Process Execution Language Version 1.1. 2003: ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf

[12] Simple Promela Interpreter: http://www.spinroot.com

[13] XML Process Definition Language Version 2.0.: http://www.wfmc.org/standards/XPDL.htm

[14] VIATRA2 Eclipse GMT subproject: http://www.eclipse.org/gmt

[15] XSD Schema of the BPEL language Version 1.1.: http://schemas.xmlsoap.org/ws/2003/03/business-process/