An ASM Specification of the XPDL Language Semantics

Petr Matoušek
petr.matousek@vsb.cz

Dept. of Computer Science
FEEI, VŠB TU Ostrava
17.listopadu 15
708 33 Ostrava-Poruba
Czech Republic

1 Introduction and motivation

In [10] there is presented a method proposal for verification of business processes specified in the XPDL [1] language. The method is based on a translation of the XPDL process specifications to their equivalents in Promela – specification language of the SPIN [3] model checker. Using this tool then various system properties specified as LTL formulas can be verified.

In order to ensure validity of the method the translation must preserve semantics. But at the current stage of work it is specified only in informal way and its correctness is more expected than proved. To prove it a semantics specifications of both the XPDL and Promela are necessary as a basis for a formal specification of the translation and also for mentioned proof of its correctness.

In the case of Promela such specification was published in [7] and extended in [8]. As for the XPDL language its semantics was specified only informal way in [1]. Aim of this paper is to give a formal specification of the XPDL according to this informal one.

2 The XPDL language

The XPDL language is a XML [2] application designed as a standard interchange format supporting the transfer of workflow process definitions between separate workflow systems. It describes processes as a partially ordered sets of *activities* where this ordering determines the executing sequence of activities in time. As an underlying paradigm for the description of the ordering relation *state charts* [6] are used and extended with some features specific to workflow.

Activities can be *atomic* or *complex* that means they can be decomposed to a *loop* or a *nested subflow (process)*. Atomic activities then can be *automated* or not. In the automated case during their execution workflow engine automatically invokes some applications. In the non-automated case some *participant* is responsible for activity performing. Each activity can during its execution modify *workflow variables* and such way influence the control flow.

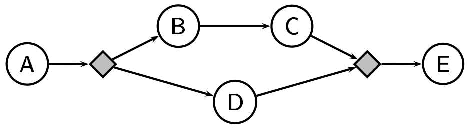

Fig. 1. Example of XPDL process

In the figure 1 there is a state chart of simple XPDL process depicted graphically. Circles represent activities and arrows represent so called *transitions* that define the partial ordering of activities. Simple transitions (e.g. B to C) represent trivial sequence (i.e. C is started just after finishing B). But more complex transition specifications are allowed. The XPDL implements both well known *split* and *join* constructs in the *and* and *xor* variants. (e.g. in the figure 1 there is an *and-split* from A to B and D and following *and-split* from C and D to E).

In fact this transition relation can be changed dynamically during workflow execution using boolean conditions assigned to transitions. Such transition is then involved in the ordering relation iff condition assigned to it is true. Note that these conditions can reference workflow variables that can be modified by activities.

3 Semantics definition

The definition of XPDL semantics given here is phrased in terms of Abstract State Machines [4, 5]. As XPDL enables more than one thread of control flow so called distributed (or also multi-agent) ASM with first order extensions is used and each its agent is responsible for execution of one thread.

3.1 Signature of XPDL ASM

There are universes *Processes* and *Activities* in the signature of the XPDL ASM as the XPDL specification can consist of one or more processes containing activities. The function $PartOf : Activities \rightarrow Processes$ assigns to each activity the process that contains it. The type of activity is determined by the function $ActType : Activities \rightarrow \{Atomic, SubFlow, Loop, XorJoin, AndJoin, XorSplit, AndSplit\}$. Initial activities of a process are described by the function $Initial : Processes \rightarrow 2^{Activities}$.

The transition relation defining the ordering of activity execution is given by the relation $Transition \subseteq Activities^2$. Note that this relation dynamically changes during the process execution with respect to evaluation of the transition guards. So the term $Transition(a_1, a_2)$ where $a_1, a_2 \in Activities$ should be seen in fact as a macro that is expanded to the guard assigned to this transition.

Parallelism is modeled using agents from the universe *Agents*. Each $agent \in Agents$ is responsible for execution of particular thread of control flow. It can be in several modes that are assigned to it by the function $Mode : Agent \rightarrow \{Running, Waiting, undef\}$. An agent in standard mode *Running* traverses the graph of process state chart and its actual position representing currently executed activity is given by the function $Active : Agents \rightarrow Activities$. The agents in mode $Waiting$ are those that wait for finishing of synchronously started subflow process or those that wait for other threads necessary for firing the AND split.

Finally workflow variables are identified by the elements of universe $Vars$ and take values from one common universe *Values* by the function $Value : Vars \rightarrow Values$. In order to model changes of their values during the execution of atomic activities an external function $ValueUpdate : Agents \times Activities \times Vars \rightarrow Values$ is used. For given thread represented by an agent an activity and a variable name it returns new value of this variable.

3.2 Rules – XPDL ASM behavior

The behavior of the XPDL ASM is described by rules that define the changes of XPDL ASM state represented by an algebra with the signature described before. All of them has the same basis – according to the type of current agent's activity appropriate changes are made and the agent moves along the transition to the next activity.

if $Mode(Self) = Running \wedge ActType(Active(Self)) = Atomic$ **then**
  $UpdateVars$
  $MoveToOneNext$
endif

Fig. 2. Atomic activity rule

In the figure 2 there is a rule for execution of an atomic activity. Whenever an agent is in running mode and it is executing an atomic activity it updates workflow variables by the external function $ValsUpdate$ (here hidden in the macro $UpdateVars$) and then moves to the next activity (macro $MoveToOneNext$).

In general there are two macros for agent traversing through the graph of process state chart. Already mentioned $MoveToOneNext$ chooses one successing activity of the current one with respect to the transition relation and moves the agent to it. This macro is used in most cases of activity types. An exception is the activity type *AndSplit* where new threads must be created. For this particular case the macro $MoveToAllNext$ creates new agent for each enabled branch of the *AndSplit* and starts it and the original agent that has reached the *AndSplit* activity is killed.

Another activity type that needs special treatment is the *AndJoin*. The requested synchronization is provided by an agent that waits here for the completion of all enabled incomming branches. Such agents are started in all *AndJoin* activities of a process during its starting. Whenever this agent recognizes that the split is enabled – i.e. there is a waiting agent in the last activity of all branches incoming to the split – it kills all these agents and then start a new agent at the activity following the split.

In the case of *Loop* activity agent checks the control condition of the loop and according to the result it moves inside the loop body or continues to the following activity. Special care must be taken of the loop type that can be *Repeat* or *While*.

In the *SubFlow* activities new agents responsible for the execution of the nested process are started. Then according to the type of the subflow execution (*Synchronous* or *Asynchronous*) the original agent waits for its completion or simply continues to the next activity.

4 Conclusion and future directions

A rigorous formal definition of the XPDL language semantics based on the informal one published in the [1] was specified using the means of Abstract State Machines. But the model presented here does not completly cover the whole language and some of its aspects like e.g. time considerations or resource allocation problem (which participant is responsible for manual activity performing) were left behind.

Even this simplifications were done this semantics specification can be used as a basis for the goal mentioned at the begining – to prove that some translation from XPDL to Promela is correct i.e. it preserves the semantics of the original process specified in the XPDL. The next step that remains to be done is a formal specification of the translation and then finally the proof itself.

References

1. Workflow Management Coalition Workflow Standard: *Workflow Process Definition Interface – XML Process Definition Language*, Document Number WFMC-TC-1025, May 22, 2001
2. W3C Recommendation, *Extensible Markup Language (XML) 1.0 (Second Edition)*, `http://www.w3c.org/XML/`
3. SPIN Homepage, *On-the-Fly, LTL Model Checking with SPIN*, `http://netlib.bell-labs.com/netlib/spin/whatispin.html`
4. Y. Gurevich, *Evolving Algebras 1993: Lipari Guide*, Specification and Validation Methods, ed. E. Brger, Oxford University Press, pp. 9–36, 1995.
5. Y. Gurevich, *May 1997 Draft of the ASM Guide*, University of Michigan EECS Department Technical Report CSE-TR-336-97.
6. D. Harel, M. Politi, *Modeling Reactive Systems With Statecharts: The Statemate Approach*, McGraw Hill, October 1998
7. V. Natarajan, G. Holzmann, *Outline for an Operational Semantics of PROMELA*, `ftp://netlib.bell-labs.com/netlib/spin/ws96/Na.ps.Z`

8. C. Weise, *An incremental formal semantics for PROMELA*, In Proceedings of SPIN97 the 3rd SPIN Workshop (Twente University, Enschede, The Netherlands), April 1997 `ftp://netlib.bell-labs.com/netlib/spin/ws97/weise.ps.Z`
9. E. Brger, A. Cavarra, E. Riccobene, *An ASM Semantics for UML Activity Diagrams*, in Teodor Rus, ed., Algebraic Methodology and Software Technology, 8th International Conference, AMAST 2000, Iowa City, Iowa, USA, May 20-27, 2000, Proceedings, Springer LNCS 1816, pp. 293–308, 2000.
10. Petr Matoušek, *Verification of Business Process Models*, to appear at ISM 2002