Modeling Workflows, Interaction Patterns, Web Services and Business Processes: The ASM-Based Approach

Egon Börger[1] and Bernhard Thalheim[2]

[1] Università di Pisa, Dipartimento di Informatica, I-56125 Pisa, Italy
boerger@di.unipi.it
[2] Chair for Information Systems Engineering, Department of Computer Science, University of Kiel D-24098 Kiel
thalheim@is.informatik.uni-kiel.de

Abstract. We survey the use of the Abstract State Machines (ASM) method for a rigorous foundation of modeling and validating web services, workflows, interaction patterns and business processes. We show in particular that one can tailor business process definitions in application-domain yet rigorous terms in such a way that the resulting ASM models can be used as basis for binding contracts between domain experts and IT technologists. The method combines the expressive power and accuracy of rule-based modeling with the intuition provided by visual graph-based descriptions. We illustrate this by an ASM-based semantical framework for the OMG standard for BPMN (Business Process Modeling Notation). The framework supports *true concurrency*, *heterogeneous state* and *modularity* (compositional design and verification techniques). As validation example we report some experiments, carried out with a special-purpose ASM simulator, to evaluate various definitions proposed in the literature for the critical OR-join construct of BPMN.[1]

1 Introduction

Over the last five years the Abstract State Machines (ASM) method has been used successfully in various projects concerning modeling techniques for web services, workflow patterns, interaction patterns and business processes.

An execution semantics for (an early version of) the Business Process Execution Language for Web Services (BPEL) has been provided in terms of ASMs in [19,23] and has been reused in [17,18]. In [4] one finds a survey of recent applications of the ASM method to design, analyze and validate execution models for *service behavior mediation* [3], *service discovery* [2,20] and *service composition* techniques [22], three typical themes concerning Service Oriented Architectures

[1] The work of the first author is supported by a Research Award from the Alexander von Humboldt Foundation (*Humboldt Forschungspreis*), hosted by the Chair for Information Systems Engineering of the second author at the Computer Science Department of the University of Kiel/Germany.

E. Börger et al. (Eds.): ABZ 2008, LNCS 5238, pp. 24–38, 2008.

(SOAs). In [22] multi-party communication is viewed as an orchestration problem ("finding a mediator to steer the interactions"). A systematic analysis, in terms of ASMs, of complex communication structures built from basic service interaction patterns has been carried out in [5]. The workflow patterns collected in [27,24], which are widely considered in the literature as paradigms for business process control structures, have been shown in [10] to be instances of eight (four sequential and four parallel) basic ASM workflow schemes.

Recently, we have adopted the ASM method for a systematic study of business process modeling techniques [14,13,12]. As authoritative reference for basic concepts and definitions we have chosen the OMG standard for BPMN [15], which has been defined to reduce the fragmentation of business process modeling notations and tools. In the following we describe the salient methodological features of this work and report our experience in applying the ASM framework [11] to provide a transparent accurate high-level definition of the execution semantics of the current BPMN standard (version 1.0 of 2006).

The paper is organized as follows. In Sect. 2 we explain how ASM models for business processes can serve as *ground model* and thus as basis for a precise software contract, allowing one to address the *correctness question* for a business process description with respect to the part of the real-world it is supposed to capture. In Sect. 3 we list the main methodological principles which guided us in defining a succinct modularized ASM that constitutes an abstract interpreter for the entire BPMN standard. In Sect. 4 we formulate the ASM rule pattern that underlies our feature-based description of specific workflow behaviors. In Sect. 5 we show how this scheme can be instantiated, choosing as example BPMN gateways. In Sect. 6 we illustrate by a discussion of the critical BPMN OR-join construct how one can put to use an appropriate combination of local and global state components in ASMs. We report here some results of an experimental validation, performed with a special-purpose ASM interpreter [25] that integrates with current graphical visualization tools, of different definitions proposed for the OR-Join in the literature. In Sect. 7 we point to some directly related work and research problems.

2 Building ASM Ground Models for Business Processes

To guarantee that software does what the customer expects it to do involves first of all to accurately describe those expectations and then to transform their description in a controlled way to machine code. This is a general problem for any kind of software, but it is particularly pressing in the case of software-driven business process management, given the large conceptual and methodological gap between the business domain, where the informal requirements originate, and the software domain, where code for execution by machines is produced. The two methodologically different tasks involved in solving the problem have been identified in [9] as *construction of ground models*, to fully capture the informal requirements in an experimentally validatable form, and their mathematically verifiable *stepwise detailing* (technically called refinement) to compilable code.

Ground models represent accurate "blueprints" of the piece of "real world" (here a business process) one wants to implement, a system reference documentation that binds all parties involved for the entire development and maintenance process. The need to check the accuracy of a ground model, which is about a not formalizable relation between a document and some part of the world, implies that the model is described in application domain terms one can reliably relate to the intuitive understanding by domain experts of the involved world phenomena. Ground models are vital to reach a firm understanding that is shared by the parties involved, so that a ground model has to serve as a solid basis for the communication between the (in our case three) parties: business analysts and operators, who work on the business process design and management side, information technology specialists, who are responsible for a faithful implementation of the designed processes, and users (suppliers and customers). We refer for a detailed dicussion of such issues to [9] and limit ourselves here to illustrate the idea by three examples of a direct (i.e. coding-free, abstract) mathematical representation of business process concepts in ASM ground models. The examples define some basic elements we adopted for the abstract BPMN interpreter in [13].

Example 1. Most business process model notations are based on flowcharting techniques, where business processes are represented by *diagrams* at whose nodes activities are executed and whose arcs are used to contain the information on the desired execution order (so-called control information). We therefore base our BPMN model on an underlying graph structure, at whose nodes ASM rules are executed which express the associated activity and the intended control flow.

Furthermore, usually the *control* flow is formulated using the so-called *token* concept, a program counter generalization known from the theory of Petri nets. The idea is that for an activity at a target node of incoming arcs to become executable, some (maybe all) arcs must be *Enabled* by a certain number of tokens being available at the arcs; when executing the activity, these tokens are CONSUMEd and possibly new tokens are PRODUCEd on the outgoing arcs. This can be directly expressed using an abstract dynamic function *token* associating (multiple occurrences of) tokens—elements of an abstract set *Token*—to arcs[2]:

$$token : Arc \to Multiset(Token)$$

The use of an abstract predicate *Enabled* and abstract token handling machines CONSUME and PRODUCE allows us to adapt the token model to different instantiations by a concrete token model. For example, a frequent understanding of *Enabled* is that of an atomic quantity formula, stating that the number of tokens currently associated to an arc *in*coming into a given node is at least a quantity $inQty(in)$ required at this arc.

$$Enabled(in) = (|\ token(in)\ | \geq inQty(in))$$

[2] In programming language terms one can understand $f(a_1, \ldots, a_n)$ for a dynamic function f as array variable.

With such a definition one can also specify further the abstract control related operations, namely to CONSUME ($inQty(in)$ many occurrences of) a token t on in and to PRODUCE ($outQty(out)$ many occurrences of) t on an arc outgoing from the given node.

$$\text{CONSUME}(t, in) = \text{DELETE}(t, inQty(in), token(in))$$
$$\text{PRODUCE}(t, out) = \text{INSERT}(t, outQty(out), token(out))$$

We express the *data* and *events*, which are relevant for an execution of the activity associated to a node and belong to the underlying database engine respectively to the environment, by appropriate ASM *locations*: so-called controlled (read-and-write) locations for the data and monitored (only read) locations for the events. Any kind of whatever complex value an application needs for data or events is allowed to be stored in an ASM location, directly, corresponding to the given level of abstraction, avoiding any encoding a later refinement to implementable data structures may require.

This approach allows us to combine the visual appeal of graph-based notations with the expressive power and simplicity of abstract-state and rule-based modeling: we can paraphrase the informal explanations in the BPMN standard document of "how the graphical elements will interact with each other, including conditional interactions based on attributes that create behavioral variations of the elements" [15, p.2] by corresponding ASM rules, which address issues the graphical notation does not clarify. More generally speaking, (asynchronous) ASMs can be operationally understood as extension of (locally synchronous and globally asynchronous [21]) Finite State Machines to FSMs working over abstract data. Therefore a domain expert, when using graphical design tools for FSM-like notations, can reason about the graphical elements in terms of ASMs whenever there is some need for an exact reference model to discuss semantically relevant issues.

Example 2. In business process design it is usual to distinguish between a *business process* (the static diagram) and its *instances* (with specific token marking and underlying data values). This distinction is directly reflected in the ASM model in terms of instantiations of the underlying parameters, which appear abstractly but explicitly in the ASM model. For example a token is usually characterized by the process ID of the process instance pi to which it belongs (via its creation at the start of the process instance), which allows one to distinguish tokens belonging to different instances of one process p. It suffices to write $token_{pi}$ to represent the current token marking in the process diagram instance of the process instance pi a token belongs to. In this way $token_{pi}(arc)$ denotes the *token* view of process instance pi at arc, namely the multiset of tokens currently residing on arc and belonging to process instance pi. BPEL uses this for a separation of each process instance by a separate XML document.

Correspondingly one has to further detail the above predicate *Enabled* by the stipulation that only tokens belonging to one same process instance have to be considered:

$$Enabled(in) = (|\ token_{pi}(in)\ | \geq inQty(in)\ \textbf{forsome}\ pi)$$

The reader will notice that the use of abstract INSERT and DELETE operations in defining the macros PRODUCE and CONSUME for tokens, instead of directly updating $token(a, t)$, comes handy: it makes the macros usable in a concurrent context, where multiple agents, belonging to multiple process instances, may want to simultaneously operate on the *token*s on an arc. Note that it is also consistent with the special case that in a transition with both DELETE(in, t) and INSERT(out, t) one may have $in = out$, so that the two operations are not considered as inconsistent, but their cumulative effect is considered.

Thus the ASM model of the given business process represents the scheme of the process, statically defined by its rules; its instances are the scheme instantiations obtained by substituting the parameters by concrete values belonging to the given process instance pi. In accordance with common practice, one can and usually does suppress notationally the process instance parameter pi, as we did when explaining the function *token* above, as long as it is clear from the context or does not play a particular role.

Example 3. This example is about the need for global data or control structures in various business process constructs, e.g. synchronization elements owned by cooperating process agents or more generally speaking data shared by local processes. They can be directly expressed in terms of global locations of possibly asynchronous ASMs, thus avoiding detours one has to invent in frameworks where transitions can express only local effects. For an illustration we refer again to the definition of the BPMN standard in [15]. It uses a predominantly local view for task execution and step control, although some constructs such as splits and joins, multi-instance processes, gotos (called links) , and sub-processes are bound by context integrity constraints.

For example splits can be intimately related by such integrity constraints to joins and thus their execution is not free of side (read: not local) effects. For an illustration see the discussion of the OR-join gateway construct of BPMN in Sect. 6.

Another example are data dependencies among different processes, whose description in [15] seems to be relegated to using associations, but really need global or shared locations to appropriately represent their role for the control flow.

3 Separation of Different Concerns

For design and analysis of business processes it turned out to be crucial that the ASM method supports to first explicitly separate and then smoothly combine the realization of different concerns, based upon appropriate abstractions supporting this form of modularization. We list some of the main separation principles, which we have used with advantage for the definition of the execution semantics for BPMN by ASMs.

Separation Principle 1. This principle is about the *separation of behavior from scheduling*. To cope with the distributed character of cooperating business processes, one needs descriptions that are compatible with various strategies to realize the described processes on different platforms for parallel and distributed

computing. This requires the underlying model of computation to support most general scheduling schemes, including *true concurrency*.

In general, in a given state of execution of a business process, more than one rule could be executable, even at one node. We call a node *Enabled* in a state (not to be confused with the omonymous *Enabled*ness predicate for arcs) if at least one of its associated rules is *Fireable* at this node in this state.[3]

We separate the description of workflow behavior from the description of the underlying scheduling strategy in the following way. We define specific business process transition rules, belonging to a set say *WorkflowTransition* of such rules, to describe the behavioral meaning of any workflow construct associated to a node. Separately, we define an abstract scheduling mechanism, to choose at each moment an enabled node and at the chosen node a fireable transition, by two not furthermore specified selection functions, say $select_{Node}$ and $select_{WorkflowTransition}$ defined over the sets *Node* of nodes respectively *WorkflowTransition*. These functions determine how to choose an enabled node and a fireable workflow transition at such a node for its execution. We then can combine behavior and scheduling by a rule scheme WorkflowTransitionInterpreter, which expresses how scheduling (together with the underlying control flow) determines when a particular node and rule (or an agent responsible for applying the rule) will be chosen for an execution step.

WorkflowTransitionInterpreter =
let $node = select_{Node}(\{n \mid n \in Node$ **and** $Enabled(n)\})$
let $rule = select_{WorkflowTransition}(\{r \mid r \in WorkflowTransition$ **and** $Fireable(r, node)\})$
rule

Separation Principle 2. The second principle is about the *separation of orthogonal constructs*. To make ASM workflow interpreters easily extensible and to pave the way for modular and possibly changing workflow specifications, we adopted a *feature-based* approach, where the meaning of workflow concepts is defined elementwise, construct by construct. For each control flow construct associated to a *node* we provide a dedicated rule (or set of rules) WorkflowTransition(*node*), belonging to the set *WorkflowTransition* in the WorkflowTransitionInterpreter scheme of the previous example, which abstractly describe the operational interpretation of the construct. We illustrate this in Sect. 5 by the ASM rules defining the execution behavior of BPMN gateways, which can be separated from the behavioral description of BPMN event and activity nodes.

Another example taken from BPMN is the separation of atomic tasks from non-atomic subprocesses and from activities with an iterative structure. BPMN distinguishes seven kinds of tasks:

$$TaskType = \{Service, User, Receive, Send, Script, Manual, Reference, None\}$$

[3] We treat the fireability of a rule (by an agent) as an abstract concept, because its exact interpretation may vary in different applications. For business process diagrams it clearly depends on the *Enabled*ness of the incoming arcs related to a rule at the given node, but typically also on further to be specified aspects, like certain events to happen, on the (degree of) availability of needed resources, etc.

These task types are based on whether a message has been sent or received or whether the task is executed or calls another process. The execution semantics for task nodes is given by one ASM rule (scheme) [13], which uses as interface abstract machines to SEND or RECEIVE messages and to CALL or EXECute processes.

A third example from the BPMN standard is the separation of cyclic from acyclic processes, whe we use for the discussion of the OR-join gateway in Sect. 6.

Separation Principle 3. The third principle is the *separation of different model dimensions* like control, events, data and resources. Such a separation is typical for business process notations, but the focus of most of these notations on control (read: execution order, possibly influenced also by events) results often in leaving the underlying data or resource features either completely undefined or only partly and incompletely specified. The notion of abstract state coming with ASMs supports to not simply neglect data or resources when speaking about control, but to tailor their specification to the needed degree of detail, hiding what is considered as irrelevant at the intended level of abstraction but showing explicitly what is needed. We illustrate this in Sect. 4 by the four components for data, control, events and resources in WORKFLOWTRANSITION, which constitute four model dimensions that come together in the ASM scheme for workflow interpreter rules. These four components are extensively used in the BPMN standard, although the focus is on the control flow, which is represented by the control flow arcs, relegating interprocess communication (via message flow arcs between processes) and data conditions and operations to minor concerns. For an enhanced specification of interprocess communication see the orchestration of processes in [28].

Separation Principle 4. The fourth principle, whose adoption helps to reduce the description size of abstract models, is the *separation of rule schemes and concrete rules*, where the concrete rules may also be specialized rule schemes. It exploits the powerful abstraction mechanisms ASMs offer for both data and operations, whether static or dynamic. We illustrate this by the COMPLEXGATETRANSITION in Sect. 5.1, a scheme from which one can easily define the behavior of the other BPMN gateways by instantiating some of the abstractions (see Sect. 5.2).

Separation Principle 5. The fifth example is about the *separation of design, experimental validation and mathematical verification* of models and their properties. In Sect. 6 we illustrate an application of this principle by an analysis of the OR-join gateway, where for a good understanding of the problem one better separates the definition of the construct from the computation or verification of its synchronization behavior. Once a ground model is defined, one can verify properties for diagrams, separating the cases with or without cyclces and in the former case showing which cycles in a diagram are alive and which ones may result in a deadlock. In addition, the specialized ASM workflow simulator [25] allows one to trace and to experimentally validate the behaviour of cyclic diagrams.

The principle goes together with the separation of different levels of detail at which the verification of properties of interest can take place, ranging from proof sketches over traditional or formalized mathematical proofs to tool supported

proof checking or interactive or automated theorem proving, all of which can and have been used for ASM models (see [11, Ch.8,9] for details).

Separation Principle 6. This pinciple is about the *separation of responsibilities, rights and roles of users* of BPMN diagrams. To represent different roles of users BPMN diagrams can be split into so-called pools, between which messages can be exchanged. Furthermore user actions can be separated by so-called swimlanes. Such a separation of user actions depending on the user's role within a diagram is supported in a natural way by the ASM concept of rule executing agents: one can associate different and even independent agents to sets of user rules; moreover these agents could be supervised by a user superagent coming with his own supervising rules, which leads to more general interaction patterns than what is foreseen by the BPMN standard (see [5]).

In the next section we show how from a combination of the separation principles formulated above one can derive an orthogonal high-level interpretation of the basic concepts of BPMN.

4 The Scheme for Workflow Interpreter Rules

For every workflow or BPMN construct associated to a *node*, its behavioral meaning can be expressed by a guarded transition rule WorkflowTransition(*node*) ∈ *WorkflowTransition* of the general form defined below. Every such rule states upon which events and under which further conditions—typically on the control flow, the underlying data and the availability of resources—the rule can fire to execute the following actions:

- perform specific operations on the underlying data ('how to change the internal state') and control ('where to proceed'),
- possibly trigger new events (besides consuming the triggering ones),
- operate on the resource space to handle (take possession of or release) resources.

In the scheme, the events and conditions in question remain abstract, the same as the operations that are performed. This allows one to instantiate them by further detailing the guards (expressions) respectively the submachines for the description of concrete workflow transitions.[4]

```
WorkflowTransition(node) =
  if EventCond(node) and CtlCond(node)
    and DataCond(node) and ResourceCond(node) then
      DataOp(node)
      CtlOp(node)
      EventOp(node)
      ResourceOp(node)
```

[4] We remind the reader that by the synchronous parallelism of single-agent ASMs, in each step all applicable rules are executed simultaneously, starting from the same state to produce together the next state.

WorkflowTransition(*node*) represents an abstract state machine, in fact a scheme (sometimes also called a pattern) for a set of concrete machines that can be obtained by further specifying the guards and the submachines for each given *node*. In the next section we illustrate such an instantiation process to define the behavior of BPMN gateways by ASM rules taken from the high-level BPMN interpreter defined in [13].

5 Instantiating WorkflowTransition for BPMN Gateways

In this section we instantiate WorkflowTransition for BPMN gateways, nodes standing for one of the three types of BPMN flow objects. The other two types are event and activity nodes, whose behavior can be described by similar instantiations, see [13] for the details. We start with the rule for so-called complex gateway nodes, from which the behavior of the other BPMN gateway constructs can be defined as special cases.

5.1 ComplexGateTransition

Gateways are used to describe the convergence (also called merging) and/or divergence (also called splitting) of control flow, in the sense that tokens can 'be merged together on input and/or split apart on output' [15, p.68]. For both control flow operations one has to determine the set of incoming respectively outgoing arcs they are applied to at the given node. The particular choices depend on the *node*, so that we represent them by two abstract selection functions, namely to

- $select_{Consume}$ the incoming arcs where tokens are consumed,
- $select_{Produce}$ the outgoing arcs where tokens are produced.

Both selection functions come with constraints: $select_{Consume}$ is required to select upon each invocation a non-empty set of enabled incoming arcs, whose *firingToken*s are to be consumed in one transition.[5] $select_{Produce}$ is constrained to select upon each invocation a non-empty subset of outgoing arcs o satisfying an associated $OutCond(o)$. On these arcs *complxGateToken*s are produced, whose particular form may depend on the *firingToken*s. We skip that in addition, as (part of) DataOp(*node*), multiple assignments may be 'performed when the Gate is selected' [15, Table 9.30 p.86] (read: when the associated rule is fired).

[5] A function $firingToken(A)$ is used to express a structural relation between the consumed incoming and the produced outgoing tokens, as described in [15, p.35]. It is assumed to select for each element a of an ordered set A of incoming arcs some of its $token(a)$ to be Consumed. For the sake of exposition we make the usual assumption that $inQty(in) = 1$, so that we can use the following sequence notation: $firingToken([a_1, \ldots, a_n]) = [t_1, \ldots, t_n]$ denotes that t_i is the token selected to be fired on arc a_i.

ComplexGateTransition($node$) =
  let
    $I = select_{Consume}(node)$
    $O = select_{Produce}(node)$
  in WorkflowTransition($node, I, O$)
where
  $CtlCond(node, I) = (I \neq \emptyset$ **and forall** $in \in I$ $Enabled(in))$
  CtlOp($node, I, O$) =
    if $O \neq \emptyset$ **and** $O \subseteq \{o \in outArc(node) \mid OutCond(o)\}$ **then**
      ProduceAll($\{(complxGateToken(firingToken(I), o), o) \mid o \in O\}$)
      ConsumeAll($\{(t_i, in_i) \mid 1 \leq i \leq n\}$) **where**
        $[t_1, \ldots, t_n] = firingToken(I), [in_1, \ldots, in_n] = I$

5.2 Instantiating ComplexGateTransition

The BPMN standard defines and names also special gateways, which can all be obtained by specializing the selection functions in ComplexGateTransition. To describe these instantiations here more clearly, we assume without loss of generality that these special gateways never have both multiple incoming and multiple outgoing arcs. Thus the so-called split gateways have one incoming and multiple outgoing arcs, whereas the so-called join gateways have multiple incoming and one outgoing arc.

For AND-split and AND-join gateway nodes, $select_{Produce}$ and $select_{Consume}$ are required to yield all outgoing resp. all incoming arcs.

For OR-split nodes two cases are distinguished: $select_{Produce}$ chooses exactly one (exclusive case, called XOR-split) or at least one outgoing arc (called inclusive OR or simply OR-split). For the exclusive case a further distinction is made depending on whether the decision is 'data-based' or 'event-based', meaning that $OutCond(o)$ is a $DataCond(o)$ or an $EventCond(o)$. For both cases it is required to select the first $out \in outArc(node)$, in the given order of gates, satisfying $GateCond(out)$.

Similarly also for OR-join nodes two versions are distinguished, an exclusive and data-based one—the event-based XOR is forbidden by the standard to act only as a Merge—and an event-based inclusive one. In the latter case $select_{Consume}$ is required to yield a subset of the incoming arcs with associated tokens 'that have been produced upstream' [15, p.80], but no indication is given how to determine this subset, which is a synchronization problem. We discuss this very much disputed issue further in the next section.

6 OR-Join Gateway: Global versus Local Description Elements

The OR-join concept is present in many workflow and business process modeling languages and is used with different understandings advocated in the literature, in different commercial workflow systems and by different users. Part of this

situation stems from the fact that in dealing with the OR-join concept, often two things are mixed up that should be kept separate, namely a) how the intended meaning of the concept is defined (question of semantics) and b) how properties of interest for the construct (most importantly its fireability in a given state) can be computed, validated (at run time) or verified (at design time) (question of computation, validation and verification methods).

It could be objected that an algorithm to compute the fireability of the OR-join rules defines the crucial synchronization property and thus the semantics of the OR-join. Speaking in general terms this is true, but then the question is whether there is agreement on which algorithm to use and whether the algorithm is understandable enough to serve as a behavioral specification the business process expert can work with. However, looking at the literature there seems to be no agreement on which algorithm should be used and the complexity of the proposed ones makes them unfit to serve as satisfactory semantical specification for the workflow practitioner.

The semantical issue disputed in the literature is the specification of the $select_{Consume}$ functions, which incorporate the critical synchronization conditions. $select_{Consume}(node)$ plays the role of an interface for triggering for a set of to-be-synchronized incoming arcs the execution of the rule at the given *node*. Unfortunately, most proposals for an OR-join semantics in one way or the other depend on the framework used for the definition. This is particularly evident in the case of Petri-net-based definitions, where, to circumvent the restrictions imposed by the local nature of what a Petri net transition can express, either the diagrams are restricted (to the possible dislike of a business process practitioner) or ad hoc extensions of Petri nets are introduced that are hard to motivate in application domain terms (see for example [33,31,32]). A side problem is that the BPMN standard document seems to foresee that the function is dynamic (run-time determined), since the following is required:

> Process flow SHALL continue when the signals (Tokens) arrive from all of the incoming Sequence Flow that are expecting a signal based on the upstream structure of the Process ... Some of the incoming Sequence Flow will not have signals and the pattern of which Sequence Flow will have signals may change for different instantiations of the Process. [15, p.80]

We refer to [12] for a detailed discussion of OR-join variations and ways to define and compute the underlying synchronization functions $select_{Consume}$. We restrict our attention here to report some experiments Ove Soerensen has made with various alternatives we considered to come up with a practically acceptable definition that could serve for the standard, in particular in connection with diagrams that may contain cycles. For this purpose Soerensen has built a specialized ASM workflow simulator [25] that is interfaced with standard graph representation tools, so that the token flow and the unfolding of a diagram cycle triggered by applying ASM OR-join rules can be visualized.

One alternative we considered is to a) pass at runtime every potential synchronization request from where it is originated (a split gateway node) to each

downstream arc that enters a join gateway node and to b) delete this request each time the synchronization possibility disappears due to branching. Assume for the moment that the given diagram contains no cycles and assume without loss of generality that there is a unique start node. Then it suffices to operate the following refinement on our BMPN model.

- **Split gate transition refinement.** When due to an incoming token t at a split *node* a new token $t.o$ is produced on an arc o outgoing *node*, a computation path starts at o that may need to be synchronized with other computation paths started simultaneously at this *node*, so that also a synchronization copy is produced and placed on each downstream arc that enters a join node, i.e. an arc entering a join node to which a path leads from o. We denote the set of these join *arc*s by $AllJoinArc(o)$. Simultaneously the synchronization copy of t is deleted from all such arcs that are reachable from *node*.
- **Join gate transition refinement.** We consume the synchronization tokens that, once the to-be-synchronized tokens have been fired, have served their purpose, and produce new synchronization tokens for the tokens the join produces. To $CtlCond(node, I)$ we add the synchronization condition that I is a synchronization family at *node*, which means a set of incoming arcs with non-empty *syncToken* sets such that all other incoming arcs (i.e. those not in I) have empty *syncToken* set (read: are arcs where no token is still announced for synchronization so that no token will arrive any more (from upstream) to enable such an arc).

It is not difficult to formulate this idea as a precise refinement (in the sense of [8]) of our ASMs for BPMN split and join rules (see [12]). To extend this approach to the case of diagrams with cycles (more generally subprocesses), one can refine the *AllJoinArc* function to yield only arcs of join nodes up to and including the next subprocess entry node; inside a subprocess *AllJoinArc* is further restricted to only yield join nodes that are inside the subprocess.[6] The production of synchronization tokens by the transition rule for join gate nodes that enter a subprocess is postponed to the exit node rule(s) of the subprocess.

There are obviously various other possibilities, with all of which one can experiment using the work that will be reported in [25].

7 Related and Future Work

There are two specific papers we know on the definition of a formal semantics of a subset of BPMN. In [16] a Petri net model is developed for a core subset of BPMN which however; it is stated there that due to the well-known lack of high-level concepts in Petri nets, this Petri net model "does not fully deal with: (i) parallel multi-instance activities; (ii) exception handling in the context of subprocesses that are executed multiple times concurrently; and (iii) OR-join gateways. "

[6] In Soerensen's tool this is realized by spanning a new diagram copy of the subprocess.

In [30] it is shown "how a subset of the BPMN can be given a process semantics in Communicating Sequential Processes", starting with a formalization of the BPMN syntax using the Z notation and offering the possibility to use the CSP-based model checker for an analysis of model-checkable properties of business processes written in the formalized subset of BPMN. The execution semantics for BPMN defined in [13] covers every standard construct and is defined in the form of **if** *Event* **and** *Condition* **then** *Action* rules of Event-Condition-Action systems, which are familiar to most analysts and professionals trained in process-oriented thinking. Since ASMs assign a precise mathematical meaning to abstract (pseudo) code, for the verification and validation of properties of ASMs one can adopt every appropriate accurate method, without being restricted to, but allowing one to use, appropriate mechanical (theorem proving or model checking) techniques.

In [29] an inclusion of process interaction and resource usage concerns is advocated for the forthcoming extension BPMN 2.0 of BPMN. It could be worth to investigate how the ASM models defined in [5] for the interaction patterns in [1] can be included into the current ASM model for BPMN, extending the current communication means in BPMN—event handling, message exchange between pools and data exchange between processes—to richer forms of interaction between multiple processes. Also a rigorous analysis of scheduling and concurrency mechanisms would be interesting, in particular in connection with concerns about resources and workload balancing that play a crucial role for efficient implementations.

The feature-based definition of workflow concepts in this paper is an adaptation of the method used in a similar fashion in [26] for an instructionwise definition, verification and validation of interpreters for Java and the JVM. This method has been developed independently for the definition and validation of software product lines [7], see [6] for the relation between the two methods.

References

1. Barros, A., Dumas, M., Hofstede, A.: Service interaction patterns. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 302–318. Springer, Heidelberg (2005)
2. Altenhofen, M., Börger, E., Friesen, A., Lemcke, J.: A high-level specification for virtual providers. International Journal of Business Process Integration and Management 1(4), 267–278 (2006)
3. Altenhofen, M., Börger, E., Lemcke, J.: A high-level specification for mediators (virtual providers). In: Bussler, C.J., Haller, A. (eds.) BPM 2005. LNCS, vol. 3812, pp. 116–129. Springer, Heidelberg (2006)
4. Altenhofen, M., Friesen, A., Lemcke, J.: ASMs in service oriented architectures. Journal of Universal Computer Science (2008)
5. Barros, A., Börger, E.: A compositional framework for service interaction patterns and communication flows. In: Lau, K.-K., Banach, R. (eds.) ICFEM 2005. LNCS, vol. 3785, pp. 5–35. Springer, Heidelberg (2005)

6. Batory, D., Börger, E.: Modularizing theorems for software product lines: The Jbook case study. In: Hartmann, S., Kern-Isberner, G. (eds.) FoIKS 2008. LNCS, vol. 4932, pp. 1–4. Springer, Heidelberg (2008)
7. Batory, D., O'Malley, S.: The design and implementation of hierarchical software systems with reusable components. In: ACM TOSEM. ASM (October 1992)
8. Börger, E.: The ASM refinement method. Formal Aspects of Computing 15, 237–257 (2003)
9. Börger, E.: Construction and analysis of ground models and their refinements as a foundation for validating computer based systems. Formal Aspects of Computing 19, 225–241 (2007)
10. Börger, E.: Modeling workflow patterns from first principles. In: Storey, V.C., Parent, C., Schewe, K.-D., Thalheim, B. (eds.) ER 2007. LNCS, vol. 4801, pp. 1–20. Springer, Heidelberg (2007)
11. Börger, E., Stärk, R.F.: Abstract State Machines. A Method for High-Level System Design and Analysis. Springer, Heidelberg (2003)
12. Börger, E., Thalheim, B.: Experiments with the behavior of or-joins in business process models. J. Universal Computer Science (submitted, 2008)
13. Börger, E., Thalheim, B.: A high-level BPMN interpreter (submitted)
14. Börger, E., Thalheim, B.: A method for verifiable and validatable business process modeling. In: Advances in Software Engineering. LNCS, Springer, Heidelberg (2008)
15. BPMI.org. Business Process Modeling Notation Specification. dtc/2006-02-01 (2006), http://www.omg.org/technology/documents/spec_catalog.htm
16. Dijkman, R.M., Dumas, M., Ouyang, C.: Formal semantics and analysis of BPMN process models using Petri nets. Technical Report 7115, Queensland University of Technology, Brisbane (2007)
17. Fahland, D.: Ein Ansatz einer Formalen Semantik der Business Process Execution Language for Web Services mit Abstract State Machines. Master's thesis, Humboldt-Universität zu Berlin (June 2004)
18. Fahland, D., Reisig, W.: ASM semantics for BPEL: the negative control flow. In: Beauquier, D., Börger, E., Slissenko, A. (eds.) Proc. ASM 2005, Université de Paris, vol. 12, pp. 131–152 (2005)
19. Farahbod, R., Glässer, U., Vajihollahi, M.: Specification and validation of the Business Process Execution Language for web services. In: Zimmermann, W., Thalheim, B. (eds.) ASM 2004. LNCS, vol. 3052, pp. 78–94. Springer, Heidelberg (2004)
20. Friesen, A., Börger, E.: A high-level specification for semantic web service discovery services. In: ICWE 2006: Workshop Proceedings of the Sixth International Conference on Web Engineering (2006)
21. Lavagno, L., Sangiovanni-Vincentelli, A., Sentovitch, E.M.: Models of computation for system design. In: Börger, E. (ed.) Architecture Design and Validation Methods, pp. 243–295. Springer, Heidelberg (2000)
22. Lemcke, J., Friesen, A.: Composing web-service-like Abstract State Machines (ASMs). In: Workshop on Web Service Composition and Adaptation (WSCA 2007); IEEE International Conference on Web Service (ICWS 2007) (2007)
23. Farahbod, U.G.R., Vajihollahi, M.: An Abstract Machine Architecture for Web Service Based Business Process Management. Int. J. Business Process Integration and Management 1(4), 279–291 (2006)
24. Russel, N., ter Hofstede, A., van der Aalst, W.M.P., Mulyar, N.: Workflow control-flow patterns: A revised view. BPM-06-22 (July 2006), http://is.tm.tue.nl/staff/wvdaalst/BPMcenter/

25. Sörensen, O.: Diplomarbeit. Master's thesis, University of Kiel, forthcoming (2008), www.is.informatik.uni-kiel/~thalheim/ASM/MetaProgrammingASM
26. Stärk, R.F., Schmid, J., Börger, E.: Java and the Java Virtual Machine: Definition, Verification, Validation. Springer, Heidelberg (2001)
27. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow patterns. Distributed and Parallel Databases 14(3), 5–51 (2003)
28. Weske, M.: Business Process Management. Springer, Heidelberg (2007)
29. Wohed, P., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A., Russel, N.: On the suitability of BPMN for business process modelling. In: The 4th Int. Conf. on Business Process Management (2006)
30. Wong, P.Y.H., Gibbons, J.: A process semantics fo BPMN. Oxford University Computing Lab (preprint, July 2007), http://web.comlab.ox.ac.uk/oucl/work/peter.wong/pub/bpmn_extended.pdf
31. Wynn, M., van der Aalst, W., ter Hofstede, A., Edmond, D.: Verifying workflows with cancellation regions and OR-joins: an approach based on reset nets and reachability analysis. In: Dustdar, S., Fiadeiro, J.L., Seth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 389–394. Springer, Heidelberg (2006); Previous versions edited as BPM-06-16 and BPM-06-12
32. Wynn, M., Verbeek, H.M.W., van der Aalst, W., ter Hofstede, A., Edmond, D.: Reduction rules for reset workflow nets. Technical Report BPM-06-25, BPMcenter.org (2006)
33. Wynn, M., Verbeek, H.M.W., van der Aalst, W., ter Hofstede, A., Edmond, D.: Reduction rules for YAWL workflow nets with cancellation regions and OR-joins. Technical Report BPM-06-24, BPMcenter.org (2006)