Abstract State Machine Nets. Closing the Gap between Business Process Models and their Implementation

Egon Börger
Università di Pisa
Dipartimento di Informatica
I-56125 Pisa, Italy
boerger@di.unipi.it

Albert Fleischmann
D-85276 Pfaffenhofen
Albert.Fleischmann@interaktiv.expert

ABSTRACT

The gap between on the one side the users' understanding of Business Process Models (BPMs) and on the other side the run behavior of model implementations is still with us. We introduce Abstract State Machine Nets (ASM Nets) whose component ASMs can be defined using a simple combination of textual (data-oriented) and intuitive graphical (control-flow oriented) yet semantically rigorously defined descriptive means. This allows the BP experts to design BPMs whose underlying ASM Nets constitute a reliable precise contract—a contract which guarantees the BP domain experts that the application-domain focussed understanding of the BPMs they design is also a correct understanding of the code behavior provided by the implementation of the models by software experts. This paves the way for the development of certifiably correct BPMs and their implementations. To illustrate one practical and one conceptual application of the concept we a) instantiate ASM Nets to model the behavioral meaning of the graphical and textual notations used in the commercial S-BPM tool suite with its focus on communication (service interaction) and b) show that applying the rigorous ASM refinement concept to ASM Nets supports IBM's Guard-Stage-Milestone approach to adaptive case management by an accurate conceptual foundation.[1]

1. INTRODUCTION

It has been shown in numerous occasions that the Abstract State Machines (ASM) Method [16] can help to close the gap between the detailed fine-grained execution behaviour of a software based system and the users' abstract understanding of it. For references and characteristic examples see [46, 16]. During the last two decades the problem has received an increasing attention for the special case of Business Process (BP) modeling and execution systems.

A peculiar feature of the development of BPs is a widespread tradition to let the BP expert define Business Process Models (BPMs) using graphical notations which often come with only loose, incomplete or even ambiguous descriptions of the behavior they are intended to suggest and to leave the definition of an executable model to the software developer—without mediating between the loose graphical and the (by the nature of software necessarily precise) textual software model to guarantee that the latter correctly implements the former. As a consequence of the early (unsatisfactory[2]) focus of BPMs on control flow still today many BPM approaches deal with data, communication, resources and conditions on the process environment (if at all) separately from process flow, as an add-on (which furthermore the BP domain expert may have difficulties to control)[3] and often (as for example in the widely process-centric OMG standard BPMN [2]) only in a cursory manner, outside the process specification and without precise connection to it.

We show that multi-agent ASMs can be defined as BP specific nets of components whose definitions are supported by a uniform notation which seamlessly combines textual and intuitive pictorial yet semantically rigorously defined descriptive means. The graphical part of the notation used for these so-called ASM Nets extends the classical flowchart notation for Finite State Machines (FSMs) and is integrated with the description of data, communication and resource aspects as well as of conditions on the process execution environment (e.g. coming through business rules), adaptable to the desired level of abstraction. This avoids the frequent mismatch between process and data layers in BPMs.[4]

In Sect. 2 we explain why it is important and what it

[1]The first author gratefully acknowledges support of this work by a reinvitation by the Alexander von Humboldt Foundation for a 2014 summer term research stay in Germany, sequel to the *Humboldt Forschungspreis* awarded in 2007/08.

S-BPM ONE '15, April 23-24, 2015, Kiel, Germany

http://dx.doi.org/10.1145/2723839.2723840.

[2]Traditional flow-driven assembly and orchestration of service components are 'too rigid and static to accomodate complex, real-world business processes' [47, p.415].

[3]For example in [50] to insert data into a workflow model one must use XML and XSD, languages most BP domain experts will not know so that only an IT expert can insert the data into the workflow model.

[4]This mismatch can be observed in mainly process-centric modeling approaches like BPMN where the data dependency of (or the data manipulation performed by) a BP activity is modeled mostly outside the BP model (if at all). This is the main motivation for so-called data-centric approaches to BPM, see [18].

needs to bridge the gap between high-level BP models and their executable versions. The section is of epistemological character and can be skipped by the reader who is interested only in technical matters. In Sect. 3.1 we define ASM Nets, in Sect. 3.2 we explain how applying the ASM refinement method [10] to ASM Nets can enhance IBM's Guard-Stage-Milestone approach proposed in [27]. In Sect. 4 we instantiate ASM Nets by behavioral diagrams as they are used in the Subject-Oriented BPM method [20]. In Sect. 5 we discuss some methodological consequences the use of ASM Nets and their refinements yield for certifiability of BPs.

2. REQUIREMENTS CAPTURE BY GROUND MODELS

It is a well-known problem that the behavior of implementations of BPMs often does not satisfy the expectations of the domain expert who defined the models. This mismatch between the users' understanding of a process and the behavior of the machine which executes the process is not even avoided by building the models using only standard notations, e.g. of the OMG standard for BPMN [2] and its 'standard' compilation to the OASIS standard BPEL [7], given the insufficient precision and the lack of completeness of these standards and the conceptual mismatch between them, which are amply documented in the literature (see for ex. [35, 36, 48] and [12] with further references there).

The situation is an instance of a problem many complex software intensive systems face. It has two methodologically different but complementary aspects, an epistemological one—how to provide accurate *models* of processes in the real world—and one concerning software engineering methods—how to correctly *code* such rigorous models by software which controls the execution of the intended processes by machines. Numerous methods and techniques exist to solve in more or less satisfactory ways the software engineering part of the problem, prominent among them various practical stepwise refinement approaches [49, 4, 10, 16, 5]. But the epistemological side of the coin, known as ground model problem, is often neglected and therefore briefly characterized below before discussing its specific BP characteristics and proposing a way to solve it satisfactorily.

2.0.1 *The ground model problem.*

The problem is about requirements capture, the first activity at the two ends of the development of software-based systems: *understanding* and formulation of real-world problems by humans to prepare the deployment of problem solutions by *code execution* using various machines and platforms. Requirements capture deals with descriptions of real-world problems written by domain experts, formulated in natural language, interspersed with diagrams, tables, formulae, etc. Frequently such descriptions suffer from lack of precision, ambiguity, incompleteness, inconsistency—yet they have to serve as basis for the software design experts who usually are not knowledgeable in the application domain but have to develop software representations of the required systems, i.e. compilable programs written for mechanical elaboration by machines and therefore in need of implementation details at a level of precision and completeness that is far away from the application domain view.

In other words the question is how to link 'informal' requirements documents to necessarily formalized executable code, the latter written to satisfy the former, in a way to guarantee that the *code does what the requirements describe* and not something else. Furthermore such a link must be reliably *preserved during maintenance*, e.g. upon requirements change or process optimization when the process model and its implementation have to be adapted to the new situation. It is here that ground models and their stepwise refinements come into play.

A *ground model* is a blueprint of the to-be-implemented piece of real world—called 'golden model' in the semiconductor industry [44]. It defines what in [17] is called 'the conceptual construct' or 'the essence' of the software system. It does this prior to coding, abstractly and rigorously:

- at an application-problem-determined level of detailing (*minimality*), avoiding details that are only needed for an implementation or only belong to the used modeling language, but expressing domain knowledge as far as it is relevant for the intended system behavior,
- formulated in application domain terms with *technical precision* (sometimes called informal accuracy, as used in mathematics), without ambiguity, based upon the domain experts' know-how,
- *authoritatively* for the further development activities including maintenance, serving in particular as design contract for the software developer.

The name 'ground model' refers to the fact that these models have to ground the design in reality by justifying their definition as

- *correct*: the model elements reliably convey the original intentions of the requirements, reflecting the technical know-how of the experts in the given domain directly, precisely and understandably for the software designer,
- *complete*: every semantically relevant feature is present, for the software designer there is no gap in understanding or filling in of 'what to build',
- *consistent*: ambiguities and conflicting objectives in the requirements have been removed resp. resolved by the domain experts.

As a consequence ground models have to solve the three typical problems every development process for software intensive systems faces:

- the *communication problem* to offer a language in which software system designers, domain experts and customers can formulate prior to coding a common understanding of 'precisely what to build',
- the *verification method problem*: since there is no infinite regress, there can be no mathematical transition from informal to precise descriptions, but model inspection and domain-specific reasoning can provide evidence of the intended direct correspondence between ground model concepts and the reality the model has to capture (using empirical interpretations of extra-logical terms to guarantee model completeness and correctness),
- the *validation problem*: one has to validate the model behaviour by repeatable experiments which systematically try to (in the Popperian sense) falsify model execution by runtime verification and scenario-based testing.

A way to solve the problem is to define ground models

using the not formalized, but mathematically precise general purpose language of Abstract State Machines [16]. We cannot show this here and instead refer the interested reader to [16, 46, 9, 11] for numerous references to ground model ASMs for a large variety of complex software-based systems and industrial standards. In this paper we want to show instead how one can tailor ASMs specifically to support effective ground model construction techniques in the BP domain.

2.0.2 *The ground model concern for BPs.*

In the BP domain we did not find the term "ground model" but the concern is there. Under various names one finds in the literature the two fundamental ground model related requests for BPMs:

application correctness BPMs should come with a minimal "semantical distance to human understanding" [21, Sect.1, pg.1] and "manage a business-meaningful scope of a business" in a way which "enables business insights and improves communication among diverse stakeholders about the operations and processes of a business" [27, pg.2]. For the GSM (Guard-State-Milestone) approach in IBM's ArtiFact project "the core constructs of GSM were chosen ... to be very close to the way that the business stakeholders think—in terms of milestones and business rules" (ibid., pg.4). Much of the focus on pictorial definitional means in the BP domain is motivated by this concern to support building BP models which are close to the domain expert's knowledge so that he can control them to be epistemologically 'correct', from the point of view of the application.

implementation correctness The mapping of human-understandable BPMs to executable code "should allow propagating the information from a value chain perspective to a software-development perspective in a coherent and consistent way" providing the BP expert with an effective "end-to-end control ... to build process-managed enterprise" [21, Sect.2, pg.3-4]. The declared "second broad goal" of IBM's ArtiFact project is that BPMs "while reasonably intuitive, are *actionable*, in the sense that there is a relatively direct path from the specification to an implementation of running systems" [27, pg.4]. This concern is about means to reliably relate ground models to executable code using stepwise model refinement with various (experimental) validation and (mathematical) verification techniques.

The Subject-Oriented approach to BPM (S-BPM [20]) stands out among the ones which are concerned about the ground model problem by addressing it at its epistemological roots instead of only focussing on pictorial notational means for process representation. S-BPM achieves human understanding of BPMs above all by aligning BP descriptions to three fundamental constituents of elementary sentences in natural languages, namely "subjects" which perform actions (expressed by "predicates") on "objects" and in particular communicate by sending (output) or receiving (input) messages. As a result, using the S-BPM approach:

1. Stakeholders need only to be familiar with natural language ... to express their work behavior ...
2. Stakeholder specifications can be processed directly *without further transformations*, and thus, experienced as described'. [21, Sect.1, pg.2]

The close conceptual relation analysed in [13] between the S-BPM approach and the ASM method for the development of software intensive systems triggered the idea to tailor multi-agent ASMs for BPs to a domain-specific, user-friendly, graphically supported modeling language (Sect. 3) in which BP users can express their design using directly BP-knowledge-based (including graphically represented) terms and notations which can be supported in two directions:

- by the underlying ASM constructs which correctly (for the BP expert) and precisely (for the software expert) express their intuitive understanding and can be justified by inspection and validation to do so, thus solving the ground model problem,
- by implementations of these constructs the correctness of which is validatable by experiments and (in principle) provable, given the mathematical character of ASMs and their refinements (in the sense defined in [10]) to executable code.

3. TUNING ASMS AS BPM LANGUAGE

The definition of ASM Nets below uses as components a variant of so-called control state ASMs which were defined in [8] (see also [16, Ch.2.2.6]). There are two reasons for doing this in an attempt to align with the in the BP domain widespread graphical representation of processes:

- Control state ASMs offer the BP designer a *flexible high-level compositional structure* to split complex models by the introduction of modes or phases into components. This compositional structure is long-known in software engineering and conveniently taken from Finite State Machines (FSMs), but control state ASMs combine it with the full power of component abstraction and refinement ASMs offer, thus avoiding a) the restriction FSMs suffer from only being able to manipulate mere letters (or words, trees, etc. in various extensions of FSMs) and b) incomprehensible spaghetti flowcharts, a risk the use of BPMN faces due to its poor process structuring means (see [36]).
- Control state ASMs inherit from FSMs the *intuitive pictorial control flow representation* by flowcharts and allow the designer to directly express also the underlying data, resource and environmental process conditions, at the level of detail which corresponds to the level of abstraction of the model and with textual definitions where appropriate (read: simpler, direct and without the risk of different interpretations of complex pictorial representations by different users). This uniform way to combine graphical and textual descriptions avoids the introduction of an overwhelming number of graphical language elements—a defect for which the OMG standard BPMN (which comes with more than 100 by no means all intuitive graphical elements) has been rightly critized for [51, 52]. To find a good balance between graphical and textual representation is pragmatically important since each increase in graphical notation yields increased meta-model complexity and risks to make transparent faithful implementations more and more impractical.[5]

[5]The possibility to insert textual descriptions where a graphical format would complicate matters seems to be a dividing line between so-called process-centric and data-centric ap-

3.1 Definition of ASM Nets

An *ASM Net* N is a (finite) diagram built from ASM net transitions (also called ASM net rules with body M) of the form defined by Fig. 1 where it is allowed to connect any exit node of one transition (read: to identify it) with one and at most one entry node of another transition.

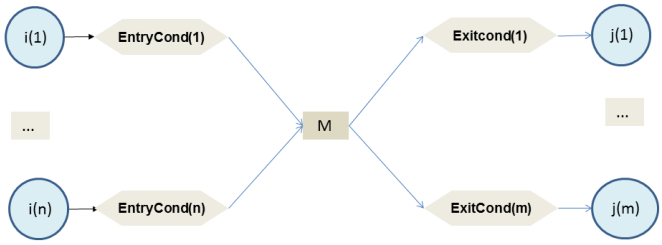

Figure 1: ASM Net Transition

The Behavior(*trans*) resulting from applying the *trans*ition is defined as execution of (an instance of) the body M which is

- Started if N is in the phase denoted by the entry node and the corresponding *EntryCond*ition is satisfied where *EntryCond* can be any state property or event (e.g. to trigger a service) or a combination of both,[6]
- continued if the body is already active and not yet *Completed*,
- Finalized when M's execution is *Completed* letting the net proceed in the phase denoted by the exit node whose *ExitCond*ition is satisfied.

Multiple *EntryCond*itions are required to be pairwise disjoint unless otherwise stated; similarly for the *ExitCond*itions for which it is also assumed that at least one of them is true when $Completed(M)$ holds.[7]

The behavior of the entire (sequential) ASM net N with given set *Transition* of *trans*itions is then simply the union of all single transition behaviors, formally:

$$\text{Behavior}(N) = \bigcup\{\text{Behavior}(trans) \mid trans \in Transition\}$$

The Behavior($\mathcal{C}$) of a concurrent system $\mathcal{C}$ of ASM Nets, where multiple agents execute asynchronously each one (sequential) ASM Net $N \in \mathcal{C}$, can be defined as the set of all Behavior(N) with $N \in \mathcal{C}$. For a general precise definition of the semantics of runs of concurrent ASMs see [14].

Fig. 1 is tailored as a scheme one can instantiate as needed for specific BPs and at the desired level of detail by stepwise refining the *Entry*/*ExitCond*itions, the body behavior (i.e. that of M, Start(M) and Finalize(M)) and the termination criterion (defining when the execution of the body is considered as *Completed*). We describe in Sect. 3.2 two concrete classes of ASM Nets and their refinements which are useful for BPM.

To achieve the mediation goal between BP and software experts mentioned in the previous section we provide for the intended intuitive understanding of Fig. 1 also a mathematically precise textual counterpart whose semantics is defined by the semantics of ASMs [16, Ch.2.4]. Here i denotes any entry mode [8] and $EntryCond_i$ its entry condition, j any exit mode and $ExitCond_j$ its exit condition, $Mode(M)$ the set of possible internal *mode* values of the body M (which are assumed to be different from any entry/exit *mode* value).

```
Behavior(transition) = forall i, j
  if mode = i and EntryCond_i then  // enter M
    Start(M)
    mode := start(M, i) // make M active
  if active(M) then
    if not Completed(M) then M // iterate M
    if Completed(M) then  // exit M
      Finalize(M)
      if ExitCond_j then mode := j
where active(M) = (mode ∈ Mode(M))
```

The graphical representation in Fig. 1 is borrowed from FSM flowcharts where

- each node, pictorially indicated as circle or oval usually carrying a name i, stands for a *mode* or *phase* of the ASM net, also called control state and corresponding to what traditionally is called an 'internal state' of an FSM,[9]
- rhombs name the *Cond*ition which must be true (or the event which has to happen, or both) for the transition to be applicable,
- rectangles name the to-be-taken transition action, the body behavior.

3.2 Instantiating the Definition of ASM Nets

The way we propose to use ASM Nets for BP modeling deliberately avoids the complex sets of actions offered in the (syntactically richer) activity diagrams of UML 2.0 [1][10] or similar concepts in BPMN [2]. We want to use only graphical notations which more or less directly and clearly suggest the intuitive meaning the underlying ASM formalization defines for them as basis for a semantically correct transformation to executable code—without paying for the major reliability by a loss of generality, given that the underlying ASMs are expressivity wise as general as a computational system can be.

proaches to BPM: "Conditions and [business] rules become essential when business policies are too intricate or cumbersome to express in a graphical format alone." [27, pg.4]

[6] *EntryCond*itions support loose coupling of system components by separating the generation of an event from triggering the to-be-executed process, a crucial feature for reactive BP systems.

[7] Alternatively, the first possible entry/exit mode is assumed to be taken in the order in which their guards appear in the diagram or a selection discipline is provided.

[8] Most BPM approaches require for every M a unique start mode $start(M)$. Using the same *mode* location for the ASM Net transition and its body M determines the sequential (mono-agent) nature of the defined ASM Net.

[9] 'Control' states represent the control flow part of the comprehensive *state* concept of ASMs which comprises also the data, resource and communication part.

[10] In this context we mention [37, 38, 29, 30] which provide a precise mathematical foundation for the major graphical UML 2 notations, including activity diagrams, by developing ASM ground models together with their implementation, thus offering a unified precise framework for the major UML diagram types.

3.2.1 Control state ASMs.

A conceptually simple but general instance of ASM Nets results from interpreting the body M, START(M), FINALIZE(M) in an ASM Net transition as control state ASMs. The latter were defined in [8],[16, Ch.2.2.6] as ASMs all of whose rules have the form of Fig. 2 with pairwise disjoint $cond_i$.

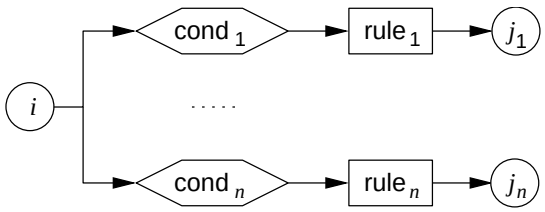

Figure 2: Control state ASM instructions

In Sect. 4 we will use only such control state ASMs.

3.2.2 GSM Approach.

A specific management instance of ASM Nets, formulated at an abstraction level which is appropriate for 'guiding the operations of a business' [26, pg.3], is IBM's recent Guard-State-Milestone (GSM) approach [27]. According to [23] it forms 'the conceptual basis of the forthcoming OMG Case Management Model and Notation (CMMN) standard' [3]. It is not difficult to define a GSM ground model by an ASM Net once all intended system features which are only sketched in [27] are fully clarified, given the close correspondence between the basic GSM concepts and their ASM net counterpart.

In GSM the word *stage* is used instead of *mode*, *phase* or *control state*; an *EntryCond* is called a *guard* which 'enables entry into the stage body' by launching (read: STARTing) an occurrence of it; a stage body is 'intended to achieve a milestone (or one of several related milestones)' [27, pg.3] which can be expressed by *ExitCond*itions (instantiating *Completed*(M) to their disjunction).

GSM permits nesting of stages where 'all occurrences of substages are terminated if the parent closes' (op.cit., pg.12); this is a specific instance of the ASM Net submachine concept. Also 'stages at the same level of nesting may execute in parallel' (ibid., pg.7), which is an instance of the synchronous parallel resp. truly concurrent concept of computation of ASM Nets. A "transactional consistency discipline based on read and write locks" is used, see [15] for an ASM model of such a discipline. Atomic stage bodies (i.e. without substages) are allowed to contain a sequence of several tasks, namely assignments, an invocation of external services, a request to create a new instance or to send a response to a call or to send an event (message). Such tasks can be described by simple control state ASMs (see for example [6]).

The GSM approach shares with the ASM method the concern to describe process behavior—via the GSM *lifecycle model* which corresponds to (read: can be described by) the rules of an ASM model—not separated from but together with the underlying process relevant data. This is the role of the GSM *information model*—which corresponds to (read: can be described by) the states of an ASM—where "all business-relevant information about an entity instance", namely data/event attributes, milestones and state information [27, pg.6-7], is held. Interestingly enough the often critized 'global' state concept ASMs support is used here in the sense that "the ... information model is shared among the multiple stakeholders involved ... a design feature that fosters communication across different groups and sub-organizations. This is a deliberate and significant departure from traditional SOA and object-oriented programming, where the internal data structures of a service or object are hidden."(ibid., pg.12)

A framework for or a rigorous foundation of what is called to 'drill-down' into a stage seems to be missing in op.cit. The ASM refinement concept defined in [10] provides both a) a practical general framework that supports systematic detailing of model elements down to executable code and b) a rigorous mathematical foundation (which has been further developed and implemented in a well-known interactive theorem prover in [39, 40, 42, 41, 43]). Enhancing 'drilling-down' by ASM refinements could help to implement what in [27, pg.4] is called "a central vision of Project ArtiFact", namely to develop GSM models stepwise, starting from "intuitive, inprecise, and/or incomplete" descriptions and piecewise adding more and more details provided by the stakeholders, eventually leading to a GSM specification. We refer to the literature [46, 16, 11] for the variety of real-life applications of ASM refinements to piecemeal system development and illustrate the method by some simple BPM examples in Sect. 4.2.

4. S-BPM INTERPRETER SPECIFICATION

In this section we instantiate ASM nets to the Subject Behavior Diagrams (SBDs) of the S-BPM method [20], thereby building a ground model for the behavioral meaning of the core concepts of its commercial tool suite [34]. This simplifies considerably the definition of an interpreter of graphically designed S-BPMs in [20, Appendix][11].

In an S-BPM of a (distributed) process each involved party (called subject) in each process phase performs one action which is either *internal* or a communication with another party (*Send* or *Receive* via synchronous or asynchronous message exchange). The *Subject Interaction Diagram* (SID) of a process defines what in [31, p.1162] is called the network topology, i.e. the *communication links* through which the subjects interact (read: *Send* and *Receive* messages) in the process. The *Subject Behavior Diagrams* (SBDs) of a process on whose interpretation our attention is focussed here define for each subject the sequence of the actions it performs using the relevant data of the involved business objects.[12] In the next Sect. 4.1 we use ASM Nets to define the BEHAVIOR$_{subj}$(*SBD*) of a *subj*ect when it walks through an *SBD*, exploiting that the mathematical graph structure underlying ASM Nets and SBDs is the same.[13]

4.1 Subject behavior diagrams

[11]The appendix contains some misleading text editing errors. A correct version can be downloaded from http://www.hanser.de/buch.asp?isbn=978-3-446-42707-5&area=Wirtschaft and http://www.di.unipi.it/~boerger/Papers/Bpmn/SbpmBookAppendix.pdf.

[12]The granularity of S-BPMs depends on the decision about which subjects should explicitly appear as actors of the to-be-defined BP. This decision reflects the particular business needs of the process and precedes the definition of the SBDs.

[13]SBDs are restricted to nets with exactly one *initial node* and one or more *end nodes* and such that each path leads to at least one end node.

By treating SBDs as ASM Nets the Behavior$_{subj}$(*SBD*) results from the local Behavior(*subj*, *node*) of the *subj*ect at any SBD *node*. In fact in an SBD each *node* represents a process phase or mode in which the executing subject is when (typically triggered by an *EntryCond*) it Performs the associated local action *A*—which constitutes the body (here called *service*(*node*)) of an ASM Net transition. Once the subject has *Completed* to Perform the *service*(*node*) it proceeds to one of possibly multiple successive modes each of which can be reached under a specific termination condition of the Performed service, its *ExitCond*ition. This explains the definition of the ASM net transition in Fig. 3.

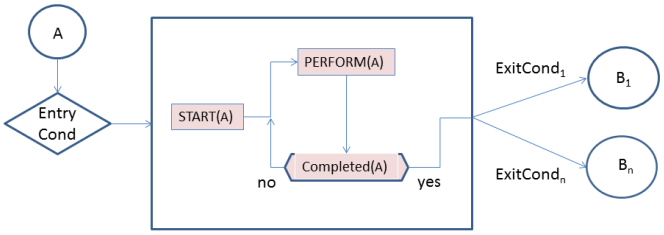

Figure 3: Behavior(*subj*, *node*) **(of a** *subj***ect in a** *node***)**

This intuitive understanding of the Behavior of a node is accurate (due to its mathematical foundation by the semantics of ASMs [16, Ch.2.4]) and complete *relative* to the understanding of what it means to Start and then to Perform a service until it is *Completed*—three notions which for internal actions are defined by the typically domain-specific meaning of these actions the BP expert understands and knows well. The ASM framework supports that one might use existing library functions for their implementation or, where needed, specify the relevant functional behavior separately. The semantical meaning of Performing a *ComAct* (*Com*munication *Act*ion, *ComAct* $\in$ {*Send*, *Receive*}) is defined by stepwise ASM refinement in the next Sect. 4.2.

4.2 Refining the Communication Actions

In this section we define by four ASM refinement steps the detailed conceptual meaning of Performing an S-BPM communication, avoiding however any reference to implementation details. First we define *single communication*, a *ComAct* of sending or receiving one message and refine it to *multiple communication* by which a given multitude of messages can be sent or received as a bundle. The latter is refined to *alternative communication* which allows one to select among a set of alternatives until for one choice (if there is any) the multiple send or receive action succeeds. The last refinement yields the detailed definition of Perform$_{subj}$(*ComAct*) for any S-BPM *ComAct*ion.

Each of these refinements happens to constitute a purely incremental (in logic also called conservative) extension of the given model, meaning that the extension performs previous behavior without semantical change and adds some new behavior. This special case of ASM refinements is methodologically important by the strong support conservative extensions provide for both modular design and compositional verification techniques.

In S-BPM each subject is equipped with an input pool where messages sent to this subject are placed by other subjects and where the receiver looks for a message when it is ready to take it. We abstract in this exposition from the way these input pools can be configured, accessed and blocked (for a message of a specific type and/or from a specific sender) to uniformly handle synchronous and asynchronous communication (see [20, pg. 321-324] for details).

4.2.1 Single Communication Action.

For a single send or receive action the sender first has to Prepare the *msgToBeHandled*, a message of a determined type to be sent or received. As second step the sender will Try the *ComAct* for the prepared *msgToBeHandled* and then Terminate it with either success or failure. This explains the refinement defined by the control state ASM in Fig. 4.

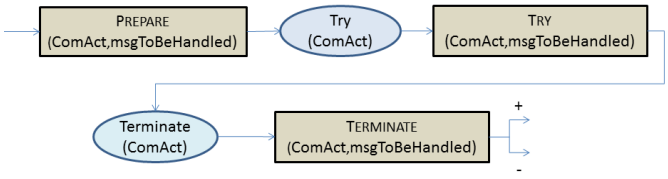

Figure 4: Single(*ComAct*)

By further refinements one can detail the behavior of the rather abstract components Prepare, Try and Terminate. We illustrate here the next refinement step for the latter two because they define the basic use S-BPM subjects make of their input pool. For Prepare see below the definition of Multi(*ComAct*).

The refinement of Terminate(*ComAct*, *msgToBeHandled*) states that after a successful communication the subject can CompleteNormally, otherwise it has to HandleFail*ure* of sending resp. receiving the *msgToBeHandled*. This explains the refinement defined in Fig. 5.

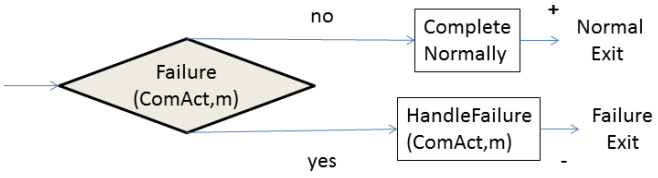

Figure 5: Refinement of Terminate(*ComAct*, *m*)

If the sender *CanAccess* the input pool of the receiver and the pool is not blocked (neither for the sender nor for the type of the *msgToBeHandled*), then the sender can PassMsg with *msgToBeHandled* for an asynchronous insertion into the pool[14] and Terminate with success. Otherwise, in case the sender can have a *Rendezvous* for a synchronous communication with the receiver, the message is directly taken by the receiver bypassing the input pool (synchronous Receive) and the sender can Terminate with suc-

[14] A refinement of PassMsg (skipped here) shows how depending on the configuration and current state of the pool the message is either dropped or inserted into the pool (possibly dropping some other message from it).

cess. In the other cases the send action will TERMINATE with failure. This explains the refinement defined in Fig. 6.

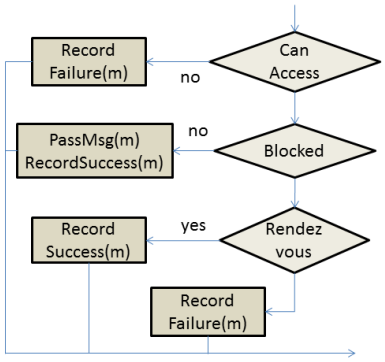

Figure 6: Refinement of TRY$(Send, m)$

Analogously a receiver, depending on whether it wants to asynchronously or synchronously receive an instance of the (kind of) *msgToBeHandled*, checks whether such a message is *Present* in its input pool or available via a *Rendezvous* with the sender and in case it is the receiver ACCEPTS the instance of the *msgToBeHandled* from its input pool resp. makes a local copy of what the sender offers. This explains the refinement defined in Fig. 7.

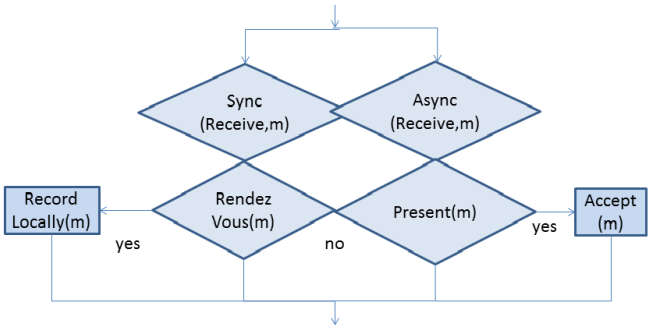

Figure 7: Refinement of TRY$(Receive, m)$

4.2.2 *Refinement of Single to* MULTI$(ComAct)$.

S-BPM permits to send or receive multiple messages in one communication action. The understanding is that all messages have first to be prepared together and then to be sent or received one by one, without pursueing any other action in between. In case of failure for at least one message the MULTI$(ComAct)$ion fails and leads to failure handling, otherwise it succeeds.

To capture this requirement by extending SINGLE$(ComAct)$ incrementally it suffices to a) refine the PREPARE component to prepare not one but a set *MsgToBeHandled* of messages and b) to apply the iteration pattern to the core component TRY of SINGLE$(ComAct)$.[15] Since the order in which

[15]Had the requirement not asked to first prepare all to be handled messages and only then send them one by one, one could have applied the iterator pattern directly to SINGLE$(ComAct)$ instead of applying it to its components.

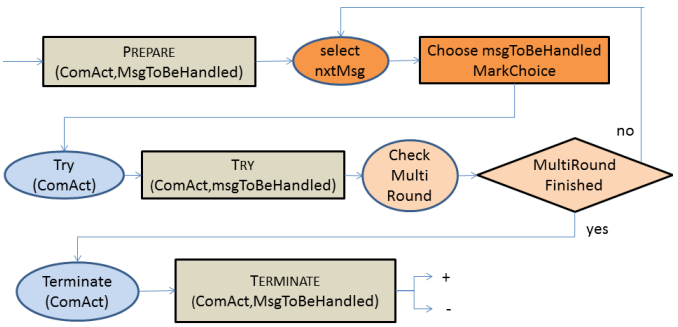

Figure 8: Refinement MULTI$(ComAct)$

the prepared messages are sent out or received is considered to be behaviorally irrelevant we use the non-deterministic CHOOSE operator to select for each iteration step the next *msgToBeHandled* out of *MsgToBeHandled* (together with marking the selected element to exclude double selections). In the definition of the refinement by Fig. 8 the two new components are placed on the right hand side of the reused SINGLE$(ComAct)$ components.

The definition of the (typically human-performed) PREPARE component provides the interface to the handling of the data to define (say for *Send*ing) the content of *msgToBeHandled*: a function *msgData* the *subj*ect uses to retrieve in the current *state* the needed data from the database as input for a function *composeMsg* to define the *msgToBeHandled*.

$$\begin{aligned}&\text{PREPARE}(Send, msgToBeHandled)(subj, state) =\\ &\quad msgToBeHandled(subj, state) :=\\ &\qquad composeMsg(subj, msgData(subj, state))\end{aligned}$$

The two auxiliary functions are platform specific and therefore left abstract here. But they serve to illustrate how textual descriptions can be seamlessly combined with graphical ASM Net elements. In fact the above PREPARE component is lifted for MULTI$(Send)$ as follows to compute a set *MsgToBeHandled* (where we introduce as further parameter *alt*ernatives of given *mult*itude which will be used below for the further TRYALT$(ComAct)$ refinement):

$$\begin{aligned}&\text{PREPARE}(Send, MsgToBeHandled)(subj, state, alt) =\\ &\quad \textbf{forall } 1 \leq i \leq mult(alt)\\ &\quad \textbf{let } m_i = composeMsg(subj, msgData(subj, state, alt), i)\\ &\qquad MsgToBeHandled(subj, state) := \{m_1, \ldots, m_{mult(alt)}\}\end{aligned}$$

Remark. The extension is conservative: if *MsgToBeHandled* is a singleton set $\{m_0\}$, then the new updates are trivial and the communication step of MULTI$(ComAct)(\{m_0\})$ boils down to execute SINGLE$(ComAct)(m_0)$.

4.2.3 *Refinement for communication alternatives.*

S-BPM forsees also alternative communication actions which allow subjects to repeatedly choose one *alt*ernative out of a set of *Alternatives* and try to communicate the message(s) prepared for the chosen alternative until the communication succeeds, in which case the alternative communication step succeeds.

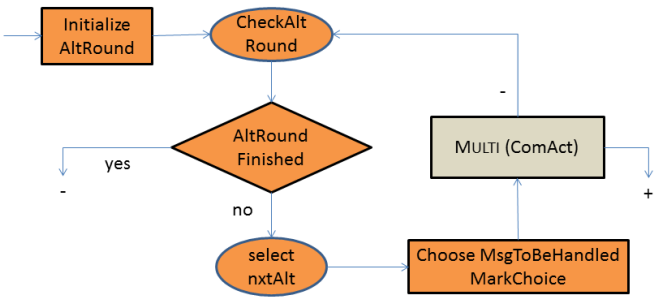

Figure 9: Refinement TRYALT(*ComAct*)

This extension consists in simply iterating MULTI(*ComAct*) through the given set of *Alternative*s until one succeeds. Assuming an appropriate initialization (which is part of the definition of the START machine for an alternative communication node and which we skip here) the meaning of a single step of the TRYALT(*ComAct*) refinement of MULTI(*ComAct*) is defined by Fig. 9, essentially parameterizing the set *MsgToBeHandled* by the *alt*ernative.

For the iteration of TRYALT(*ComAct*) the S-BPM framework stipulates that in a first (called 'non blocking') round each alternative is tried out once; if none of them succeeds further alternative communication attempts are performed in additional ('blocking') rounds which are interruptable by timeouts or user interactions. This requirement is captured by the definition in Fig. 10 which refines for *Com*munication *Act*ion nodes the PERFORM component of Fig. 3.

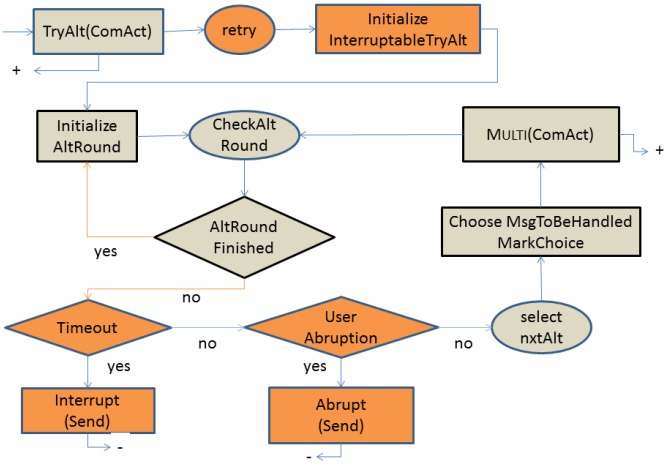

Figure 10: Refinement of $\text{PERFORM}_{subj}(ComAct)$

5. CERTIFIABILITY CONSEQUENCES

We have defined ASM Nets and illustrated that they can serve to mediate between BP views of different stakeholders, e.g. a domain expert and a software system designer. The mediation is possible because the language of ASMs uses only the (semantically well defined) fundamental *description* as well as *reasoning* scheme of both natural and scientific languages:

if *Condition* **then** *Statement*

where *Condition* describes an arbitrary state property or event (e.g. the arrival of a message or signal) which triggers the *action* to-be-performed in the current state resp. implies the to-be-proved *logical expression* describing a to-be-verified property (e.g. a system invariant).

To see the consequences for certifiability let us recapitulate:

- ASM Nets provide the BP expert with an intuitive but accurate and reliable behavioral definition of basic BP constructs in application-domain terms and thus enable to build BP ground models which can be systematically inspected and validated to be correct,[16]
- Ground model ASM Nets constitute a precise and complete specification for the software implementation of the BP they define; if the coding process is correct—a software design problem whose mathematical underpinning can remain hidden to the BP expert, the same as to an experienced car driver the technical details of the acceleration and brake mechanisms can remain hidden!—then the implementation is guaranteed to correctly reflect the meaning the BP expert intended when designing the high-level BPM.

As a consequence BPs developed this way can be certified. The quality (degree of reliability) of a correctness certificate is proportional to the quality of the ground model validation (e.g. by model inspection, model checking, model-based testing) and the verification of the stepwise refinements used to develop the executable BP—whether by compiling the ground model ASM Net using a compiler verified à la [22, 45, 46, 28, 33] or by providing a qualified design documentation which shows the refinement correctness in the form of proof sketches or mathematical or machine supported (interactive or fully automated) proofs.

Using ASM nets offers the ingredients to produce certifiably correct industrial BPs, a challenge which is a BP-specific version of Hoare's 'verified software grand challenge'[24, 25]. In the BP domain version of the challenge, ground models and their validation play an even more important role than in the general case and constitute a challenge in their own for the modeling community.

Acknowledgement. The idea for this work came during the Dagstuhl seminar on *Integration of Tools for Rigorous Software Construction and Analysis* (September 8-13, 2013). We thank the organizers U. Glässer, S. Hallerstede, M. Leuschel and E. Riccobene for the inspiring and constructive seminar.

6. REFERENCES

[1] UML 2.0 superstructure specification. Object Management Group, see http://www.omg.org/cgi-bin/doc?formal/05-07-04.

[2] Business Process Model and Notation (BPMN). Version 2.0. http://www.omg.org/spec/BPMN/2.0, January 2011. formal/2011-01-03.

[16] An open-source implementation [32] of the S-BPM interpreter on top of CoreAsm [19] can be used by the BP expert to experiment with executions of the process, without knowledge of the coding which translates the graphical model into executable code.

[3] Case Management Model and Notation (CMMN). Version 1.0 Beta 1. http://www.omg.org/spec/CMMN/1.0/Beta1, January 2013.

[4] J.-R. Abrial. *The B-Book*. Cambridge University Press, Cambridge, 1996.

[5] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, Cambridge, 2010.

[6] M. Altenhofen, E. Börger, A. Friesen, and J. Lemcke. A high-level specification for virtual providers. *Int.J.BP Integration and Management*, 1:267–278, 2006.

[7] Web Services Business Process Execution Language version 2.0. OASIS Standard, 2007. URL http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[8] E. Börger. High-level system design and analysis using Abstract State Machines. In D. Hutter et al., editors, *Current Trends in Applied Formal Methods (FM-Trends 98)*, volume 1641 of *LNCS*, pages 1–43. Springer, 1999.

[9] E. Börger. The ASM ground model method as a foundation of requirements engineering. In N.Dershowitz, editor, *Verification: Theory and Practice*, volume 2772 of *LNCS*, pages 145–160. Springer, 2003.

[10] E. Börger. The ASM refinement method. *Formal Aspects of Computing*, 15:237–257, 2003.

[11] E. Börger. 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.

[12] E. Börger. Approaches to modeling business processes. A critical analysis of BPMN, workflow patterns and YAWL. *J. Software and Systems Modeling*, 2011.

[13] E. Börger. The subject-oriented approach to software design and the Abstract State Machines method. volume 7260 of *LNCS*, pages 52–72, 2012. Reprinted in Vol. 104 of *LN in Business Information Processing*, pp.1-21, Springer, 2012.

[14] E. Börger and K.-D. Schewe. Concurrent Abstract State Machines, June 2014. submitted.

[15] E. Börger and K.-D. Schewe. Specifying transaction control to serialize concurrent program executions. In Y. Ait-Ameur and K.-D. Schewe, editors, *Proc. ABZ 2014*, volume 8477 of *LNCS*, pages 142–157. Springer, 2014.

[16] E. Börger and R. F. Stärk. *Abstract State Machines. A Method for High-Level System Design and Analysis*. Springer, 2003.

[17] F. P. J. Brooks. No silver bullet. *Computer*, 20(4):10–19, 1987.

[18] D. Cohn and R. Hull. Business artifacts: a data-centric approach to modeling business operations and processes. *IEEE Data Engineering Bulletin*, 32:3–9, 2009.

[19] R. Farahbod et al. *The CoreASM Project*. http://www.coreasm.org.

[20] A. Fleischmann, W. Schmidt, C. Stary, S. Obermeier, and E. Börger. *Subject-Oriented Business Process Management*. Springer Open Access Book, Heidelberg, 2012. www.springer.com/978-3-642-32391-1.

[21] A. Fleischmann and C. Stary. Whom to talk to? A stakeholder perspective on business process development. *Universal Access in the Information Society*, pages 1–26, June 2011. DOI 10.1007/s10209-011-0236-x.

[22] W. Goerigk, A. Dold, T. Gaul, G. Goos, A. Heberle, F. W. von Henke, U. Hoffmann, H. Langmaack, H. Pfeifer, H. Ruess, and W. Zimmermann. Compiler correctness and implementation verification: The verifix approach. In P. Fritzson, editor, *Int. Conf. on Compiler Construction, Proc. Poster Session of CC'96*, Linköping, Sweden, 1996. IDA Technical Report LiTH-IDA-R-96-12.

[23] F. Heath et al. Barcelona: A design and runtime environment for declarative artifact-centric BPM. In *ICSOC 2013*, volume 8274 of *LNCS*, pages 705–709, 2013.

[24] C. A. R. Hoare. The verifying compiler: A grand challenge for computing research. *J. ACM*, 50(1):63–69, 2003.

[25] T. Hoare and J. Misra. Verified software: theories, tools, experiments. Vision of a Grand Challenge project. In B. Meyer, editor, *Proc.IFIP WG Conference on Verified Software: Tools, Techniques, and Experiments*, http://vstte.ethz.ch/papers.html, Zürich (Switzerland), October 2005. Chair of Software Engineering at ETH Zürich.

[26] R. Hull et al. Business artifacts with Guard-State-Milestone lifecycles: Managing artifact interactions with conditions and events. In *5th ACM Int.Conf. on Distributed Event-Based Systems (DEBS 2011)*. ACM, 2011.

[27] R. Hull et al. Introducing the Guard-State-Milestone approach for specifying business entity lifecycles. In M. Bravetti and T. Bultan, editors, *Web Services and Formal Methods*, volume 6551 of *LNCS*, pages 1–24. Springer, 2011.

[28] G. Klein and T. Nipkow. A machine-checked model for a Java-like language, virtual machine and compiler. *ACM Transactions on Programming Languages and Systems*, 2004.

[29] J. Kohlmeyer. *Eine formale Semantik für die Verknüpfung von Verhaltensbeschreibungen in der UML 2*. PhD thesis, Universität Ulm (Germany), 2009.

[30] J. Kohlmeyer and W. Guttmann. Unifying the semantics of UML 2 state, activity and interaction diagrams. LNCS, pages 206–217. Springer, 2009.

[31] L. Lamport and N. Lynch. *Handbook of Theoretical Computer Science*, chapter Distributed Computing: Models and Methods, pages 1157–1199. Elsevier, 1990.

[32] H. Lerchner and C. Stary. An open S-BPM runtime environment based on Abstract State Machines. In *Proc.IEEE 16th Confererence on Business Informatics*, pages 54–61, 2014. http://doi.ieeecomputersociety.org/10.1109/CBI.2014.24. See http://www.i2pm.net/interest-groups/open-s-bpm/sub-projects/open-s-bpm-workflow-engine.

[33] X. Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.

[34] Metasonic. Metasonic-suite. www.metasonic.de/metasonic-suite.

[35] J. Recker and J. Mendling. On the translation between BPMN and BPEL: Conceptual mismatch between process modeling languages. In *Proc.11th EMMSAD*, June 2006.

[36] J. Recker and J. Mendling. Lost in business process model translations: How a structured approach helps to identify conceptual mismatch. In K. Siau, editor, *Research Issues in Systems Analysis and Design, Databases and Software Development*, pages 227–259. IGI Publishing, Hershey, Pennsylvania, 2007.

[37] S. Sarstedt. *Semantic Foundation and Tool Support for Model-Driven Development with UML 2 Activity Diagrams*. PhD thesis, Universität Ulm, 2006.

[38] S. Sarstedt and W. Guttmann. An ASM semantics of token flow in UML 2 activity diagrams. volume 4378 of *LNCS*, pages 349–362. Springer, 2007.

[39] G. Schellhorn. Verification of ASM refinements using generalized forward simulation. *J.Universal Computer Science*, 7(11):952–979, 2001.

[40] G. Schellhorn. ASM refinement and generalizations of forward simulation in data refinement: A comparison. *Theoretical Computer Science*, 336(2-3):403–436, 2005.

[41] G. Schellhorn. ASM refinement preserving invariants. *J.UCS*, 14(12), 2008.

[42] G. Schellhorn. Completeness of ASM refinement. *Electr. Notes TCS*, 214, 2008.

[43] G. Schellhorn. Completeness of fair ASM refinement. *SCP*, 76(9):756–773, 2011.

[44] Semiconductor Industry Assoc. International technologoy roadmap for semiconductors. Design. http://www.itrs.net/Links/2005ITRS/Design2005.pdf, 2005.

[45] R. F. Stärk and J. Schmid. Completeness of a bytecode verifier and a certifying Java-to-JVM compiler. *J. of Automated Reasoning*, 30:323–361, 2003.

[46] R. F. Stärk, J. Schmid, and E. Börger. *Java and the Java Virtual Machine: Definition, Verification, Validation*. Springer-Verlag, 2001.

[47] J. K. Strosnider, P. Nandi, S. Kumaran, S. Gosh, and A. Arsanjani. Model-driven synthesis of SOA solutions. *IBM Syst.J.*, 41(5):415–432, 2008.

[48] M. Weidlich, G. Decker, A. Grosskopf, and M. Weske. BPEL to BPMN: The myth of a straight-forward mapping. In *On the Move to Meaningful Internet Systems: OTM 2008, Part I*, volume 5331 of *Springer LNCS*, pages 265–282, 2008.

[49] N. Wirth. Program development by stepwise refinement. *Comm. ACM*, 14, 1971.

[50] YAWL: Yet Another Workflow Language. http://www.yawlfoundation.org/.

[51] M. zur Muehlen and J. Recker. How much BPMN do you need? Posted at http://www.bpm-research.com/2008/03/03/how-much-bpmn-do-you-need/.

[52] M. zur Muehlen and J. Recker. How much language is enough? Theoretical and practical use of the Business Process Modeling Notation. In Z. Bellahsène and M. Léonard, editors, *Advanced Information Systems Engineering (CAiSE 2008)*, volume 5074 of *LNCS*, pages 465–479. Springer, 2008.