Softw Syst Model (2013) 12:5–9
DOI 10.1007/s10270-012-0306-4

EXPERT'S VOICE

Remarks on Egon Börger: "Approaches to model business processes: a critical analysis of BPMN, workflow patterns and YAWL, SOSYM 11:305–318"

Wolfgang Reisig

Received: 8 October 2012 / Accepted: 2 November 2012 / Published online: 31 January 2013

Abstract Egon Börger (SOSYM, 11, pp. 305–318, 2012) challenges the concepts of BPMN, workflow patterns and YAWL as useful contributions to the modeling of business processes. I show that he misjudges the role of BPMN, YAWL and similar techniques in the modeling of business processes. In particular he mistakes YAWL's formal basis, i.e. Petri nets. Börger furthermore suggests evaluation criteria for business process modeling tools. I argue that his criteria overemphasize some less important aspects, while ignoring some decisive ones.

Keywords Business process modelling · BPMN · YAWL · Petri nets · Evaluation criteria for tools

1 Introduction

Egon Börger [1] disputes the value of two business process modeling (BPM) languages, BPMN and YAWL, and the adequacy of the present version of workflow patterns.

I concentrate on four of his observations regarding BPMN. All of them are true at a superficial level of understanding. However, they miss the intent of BPM. For good reasons BPMN is made in a manner that Börger criticizes. Any "repair" in the way suggested by Börger would destroy the usability of BPMN.

Börger disputes YAWL and in particular its formal basis, Petri nets, to be of any value for BPM. By means of a toy examples I show that Börger makes a number of errors about Petri nets.

Communicated by Bernhard Rumpe.

W. Reisig
Computer Science Department, Humboldt-Universitaet zu Berlin, Berlin, Germany
e-mail: reisig@informatik.hu-berlin.de

Börger suggests to evaluate the existing BPM tools. I doubt the scientific relevance of such an enterprise.

2 Remarks on Sect. 2 "Problems of BPMN 2.0"

Börger deplores "…numerous ambiguities in the descriptions and underspecifications of semantically relevant concepts." The authors of BPMN certainly have been aware of those ambiguities and underspecifications. They are justified by the role of modeling languages in the course of system modeling: a model is supposed to express intuitive ideas, thus supporting communication among users, and to prepare implementation of some, but usually not all, components of the model. A BPM language with all semantically relevant concepts entirely formalized could not accomplish this.

BPM fundamentally differs from conventional software modeling in that not all components are intended to be implemented. Some describe behavior of institutions, human beings such as clients, or technical devices.

"A general notion of state is missing and, as a consequence, the specification of relevant data dependent conditions is only poorly supported." There are good reasons for this: The progression of a companies' business processes as a whole should not be modelled as steps between states. For example, one may identify a state where a secretary starts writing a letter and a computer starts performing a transaction. Would it be adequate to model these two activities in some order (in which one?), or as occurring coincidentally? (Global) states and totally ordered sequences of event occurrences adequately model the behavior of small, centralized, single-threaded systems. A large, distributed, multi-threaded system's behavior is better modelled as a set of activities that are *partially* ordered along individual threads. (I give an example in the "Appendix").

With locality of actions and separate threads of activities taken seriously, a global view on data is inadequate. Conditions and decisions depending on data from several different threads should be avoided anyway. Where it is really unavoidable, a model should show this very explicitly. Summing up, the deplored missing states and poorly supported data dependencies are most adequate features of BPMN.

Börger disapproves that "BPMN comes with a plethora of interdefinable constructs. Instead of defining a core of independent constructs in terms of which other constructs can be defined." True, by now there are more than 200 graphical BPMN elements. However, they indicate the success of BPMN in the software industry as well as the wide variety of application domains and of aspects (viewpoints) to be expressed in BPMN models. Companies are less interested in systematic buildup; they find it more useful to define their own shorthands and conventions instead. There is nothing wrong with this. Experience with BPMs furthermore indicates that the suggested "core of independent constructs" may cover barely more than a few principles. They may include, for example, to start modeling by separating passive components such as locally confined states, control and data, from active components that *update* local states and data. Specializations could then cope with objects, users, etc. Further experience and trial are needed before such a core of independent constructs can be established.

Börger criticizes BPMN because "the standard document fails to provide a seamless systematic mechanism for refinement from conceptual to executable models, which is necessary to guarantee the reliability of the implementation." This criticism is superfluous because people have failed to solve this problem for decades, for any kind of modeling technique.

Summing up, a BPM is usually not fully formal; global control of states and data cannot be assumed; variants and domain specific notations are useful; and the desire for property preserving or property enforcing refinement cannot be fulfilled these days.

Any technique to model business processes must respect all this. BPMN does so. Ultimately, this is what distinguishes a specification language from a programming language. Börger's demands fail to see the purpose of BPMN.

3 Remarks on Sect. 4 "Problems of YAWL"

Petri nets have been chosen as the formal basis for YAWL. Börger criticizes Petri nets as unsuitable for BPM. This is the result of numerous misconceptions about Petri nets in his text. I show this in the most concrete manner by means of a simple example given in the "Appendix". Consequently, Börger's remarks on YAWL are, to a great extent, inadequate.

As for "the missing coding free support of data types": The vending machine's data types, as given in the "Appendix", are simple; the general case is obvious. Given a signature Σ (i.e. a finite collection of constant- and function symbols) and a set of variables, each variable-free term over Σ can serve as a token and each term with variables as an arc inscription. A transition then occurs w.r.t. a valuation of the variables (as visible at the vending machine's transition insert).

Börger claims that "[t]he problem with the Petri net model of computation becomes evident when one has to describe non-local or truly concurrent or recursive business process behavior." I discussed the issue of "non-local behavior" in the context of BPMN already. "Truly concurrent behavior" is captured by distributed runs (as defined by Petri [2]), as exemplified in the "Appendix". "Recursive behavior" is rare in the context of BP (and there do exist recursive Petri net extensions).

Furthermore, Börger claims that "the non-deterministic execution model of Petri nets hides an implicit 1-core interleaving assumption ('no 2 transitions fire simultaneously')." However, the execution model of distributed runs is not non-deterministic and employs no interleaving assumption. The vending machine's only nondeterminism is due to the unspecified order of the two occurrences of insert, purposely enforced by the loop with interleaving control. Without this loop, the net was deterministic and would thus have exactly one distributed run.

As for "the so-called 'open workflow net' extension (which uses strong fairness assumptions)": open workflow nets are special Petri nets and use the standard progress assumption.

Börger falsely states that "Petri-net tools usually assume (for fairness reasons) that loops terminate; this excludes possible execution paths in real-life BP behavior." Yet, not a single one of the more than 80 tools assumes this [4]. Simulation tools can, of course, hardly cope with the excluded, i.e. infinite, execution paths. Later, Börger himself contradicts his own complaint about the exclusion of infinite runs by stating that "fairness is mostly studied as a property of infinite runs (which, in practice, do not occur […])."

4 Remarks on Sect. 5 "Evaluation criteria for BPM systems and a challenge"

To improve BPM, Börger suggests criteria to evaluate existing BPM tools. He claims that, to this end, a BPM system has to support specific development and design methods. Furthermore, he suggests to transfer technology from software engineering to BPM, to "resolve the deplorable status of BPM standardization in three steps." I consider neither of those proposals particularly useful, for the following reasons:

1. The evaluation of existing BPM tools may be of commercial interest. Success or failure of such languages depends on many aspects, including support of the vendor for his

clients, etc. However, the *scientific* value of such an evaluation will remain limited. Of scientific interest would be an evaluation of the *methods* and *principles* behind BPM tools. Such methods and principles are far from being fully understood today. Few such principles are agreed upon generally. Models are required that can be understood by—and are valuable for—the various users. It is debatable whether a one-serves-all model (Börger's "ground model") would be useful. For example, a companies' clients, accountants and staff department each require different information. A joint model would either be too coarse or too refined for at least one of them.

2. The idea of property preserving *refinement* has been ubiquitous for many decades. Such principles are rarely generally applicable and if they are, remain superficial. Ontology-based, domain-specific principles, on the other hand, may turn out really useful.
3. Börger suggests *abstraction* as a core method. He requires "a rich enough set of abstract data types [...] without the detour of language-dependent encodings." This is a reasonable requirement. The most liberal of such methods allow the modeler to freely choose any signature Σ (i.e. any finite set of constant-and function symbols) together with an interpretation of those symbols. Together with a set X of variables, the *terms* constructed from Σ and X can then be interpreted in the real world. Terms provide the basis to write *specifications*. Two modeling techniques follow this most liberal approach: abstract State Machines and high-level Petri nets. Both, of course, have to pay a price: generally applicable analysis techniques are rare.

Summing up, the key to improving BPM are not the tools, but methods and principles. Existing tools such as BPMN, YAWL and others help to gain experience with hitherto suggested methods and principles. Hence, the challenge is not to "identify a common kernel of major BPM tools," but to crystallize a kernel of BP modeling techniques, independent of their integration into actual tools.

Appendix: The Example of a Cookie Vending Machine

A small system is represented as a Petri net model that employs various features that are fundamental for Petri net models of real-world business processes. This small model exemplifies a number of Börger's false assumptions about Petri nets as stated in [1]. Furthermore, it may be taken as an exercise for other modeling techniques. For detailed information on Petri nets, refer to [3].

Figure 1 shows a Petri net model of a cookie vending machine together with activities performed by the vending machine's customers. The places purse and packet taken as well as the transitions insert and take specify the customer's behavior. The initial making M_0, shown in Fig. 1, presents two *tokens* in the customer's purse, 1 and 1. The machine will accept the euro coin 1 in exchange for a cookie packet, as initially present in the storage. The machine will reject the fake coin, 1. The interleaving control with its black dot token • will guarantee that the two coins are inserted in either order.

The marking M_0 of Fig. 1 enables the transition insert in the two *modes* x = 1 and x = 1 : a mode *validates* the variable x by a constant. With x = 1, each ingoing arc of insert (viz. the arcs (purse, insert) and (interleaving control, insert)) starts at a place that holds an item that the arc inscription specifies. M_0 enables no other transitions.

The occurrence of insert in mode x = 1 then yields the marking M_1 with the euro coin 1 moved from purse to coin slot. The marking M_1 enables two transitions: insert in mode x = 1 and sale. The transition sale has no modes, as no variables occur in its surrounding arc inscriptions. The occurrence of sale in M_1 yields M_2, with the coin 1 moved from the coin slot to the cash box and, coincidently, the cookie box moved from the storage to the compartment. Then the transition take moves from the compartment to packet-taken, yielding M_3. All four markings $M_0, \ldots, M_3$ also enable insert in mode x = 1. The occurrence of insert in this mode then moves 1 from purse to

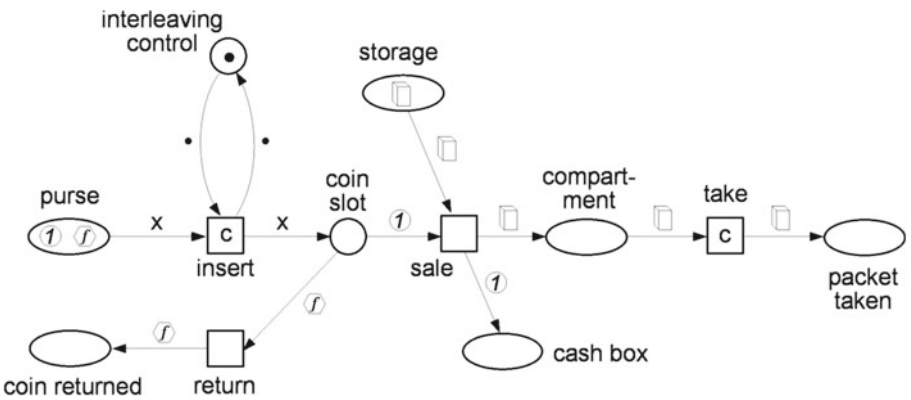


Fig. 1 A cookie vending machine

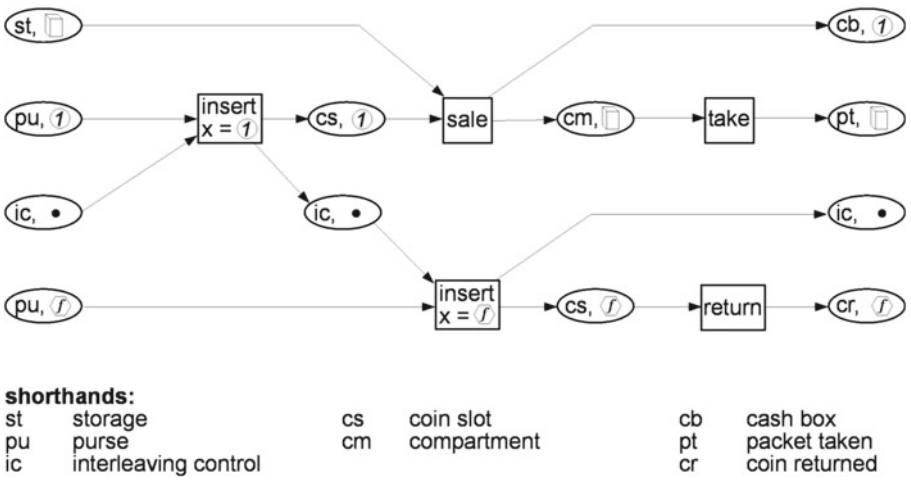


Fig. 2 A distributed run of the vending machine

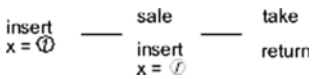
coin-slot, which in turn enables the transition return to move f to coin returned.

The above describes occurrences of the net's transitions which can be composed of different runs (behaviors). There are essentially three versions of behavior definitions around. They order the transition occurrences in three different ways: first, a conventional run is totally ordered, e.g.

(1)

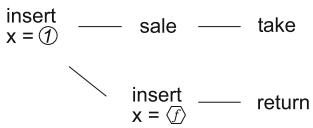
The potential of concurrent execution is not represented in (1).

Secondly, transitions may occur in lock step, such as, e.g. in



(2)

Thirdly, the most faithful kind of run orders the transition occurrences partially according to their causal dependencies, such as in



(3)

Technically, this partial order shows the transition occurrences of the *distributed run* in Fig. 2. A distributed run also protocols the tokens as they occur in the net's places.

Figure 2 clearly shows the role of the interleaving control: its token is *first* involved in (insert, $x = 1$) and then in (insert, $x = f$). No other transitions are linked by such a control. If not causally ordered by token flow, they occur *concurrently*. For example, return occurs concurrently to sale as well as to take. In contrast to this purely *causal* order, the lockstep execution model (2) still assumes a global time scale (and hence, execution of non-ordered transitions "at the same time").

The non-order of (3) just means causal independence. This is the essence of two concurrent transition occurrences.

The quest of *termination* of runs is a crucial aspect, particularly in the case of unbounded many-transition occurrences. Faithful models distinguish *hot* and *cold* transitions with the requirement that a run terminates without enabled hot transitions. For example, sale and return are hot in Fig. 1. In fact, one would consider the machine *broken* if it terminated with a coin in its coin slot. However, a customer may insert no coin or may not take the bought packet out of the compartment. Hence, the transitions insert and take may remain enabled forever. The inscription "c" in Fig. 1 denotes the cold transitions.

References

1. Börger, E.: Approaches to model business processess: a critical analysis of BPMN, work ow patterns and YAWL. SOSYM **11**, 305–318 (2012)
2. Petri, C.A.: Non-Sequential Processes GMD Report. In: ISF-77-5 (1977)
3. Reisig, W.: Understanding Petri Nets. Springer, Berlin (2013)
4. Universität Hamburg. Petri Net World. Last retrieved on 29 Sept 2012. URL: http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html

Author Biography

Wolfgang Reisig is a full professor at the Computer Science Department of Humboldt-Universitaet zu Berlin, Germany. He served as a research assistant and assistant professor at the University of Bonn and at RWTH Aachen, a visiting professor at Hamburg University, a project manager at Gesellschaft fuer Mathematik und Datenverarbeitung (GMD), and a professor at Technical University of Munich. Prof. Reisig was a senior researcher at the International Computer Science Institute (ICSI) in Berkeley, California in 1997, got the "Lady Davis Visiting Professorship" at the Technion, Haifa (Israel), the Beta Chair of Technical University of Eindhoven, and twice received an IBM Faculty Award for his contribution to cross-organizational business processes and the analysis of service models. Presently, he is the speaker of a PhD school on service-oriented architectures. Prof. Reisig is a member of the European Academy of Sciences, Academia Europaea. He has published and edited numerous books and articles on Petri net theory and applications.