# Chapter 1
# Introduction

We hardly need to point out the importance of business process modelling and of respective automation in this place (see, e.g. [39, 45, 58, 110, 141]). Also the advantages and shortcomings of the various different methods and notations for Business Process Management (BPM) have been widely discussed in the above-mentioned literature and in further scientific work (see, e.g. [50, 114, 145, 146]). But while the diversity of methods and notations as well as their respective shortcomings are well known, from our point of view, the status quo has hardly changed so far, and we are not yet satisfied with the given alternatives.

Yet we do not deem it wise to introduce "yet another" completely different method and notation. After a time of experimentation, there must be a time of convergence and settlement so the industry can either adopt a single standard or at least be able to select from a few competing, mature options. People usually do not want to learn a new technique and notation for one and the same task every other year. Thus, we have decided to build on one existing, widely adopted method and notation, discuss it based on a new rigorous semantics and propose solutions for shortcomings.

We have chosen the Business Process Model and Notation (BPMN) 2.0 [95] as a basis for discussion and improvements, firstly because it is an international standard issued by a well-established group with a strong foundation in the industry, the Object Management Group (OMG); secondly because it has already gone through a practice-driven maturing process; and thirdly because it has already been widely adopted and is supported by various tools (see also [104]). While others might argue that similar criteria would hold for some other methods as well, we simply (and maybe subjectively) expect BPMN to have, and most importantly to continue to have, the greatest impact in industrial practice. This might also be confirmed by the emergence of further standards for business process modelling based on BPMN, such as the e-government standards in Switzerland (eCH) [40–42], and formally publishing BPMN 2.0 by the International Organization for Standardization (ISO) as the 2013 edition standard ISO/IEC 19510 [61].

F. Kossak et al., *A Rigorous Semantics for BPMN 2.0 Process Diagrams*,
DOI 10.1007/978-3-319-09931-6_1

To address the major categories of shortcomings of BPMN 2.0, we focus on the operational semantics of BPMN process diagrams. First, we present a complete formal semantics for the notation, in a way which is precise yet easily understandable. (The semantics of parts of BPMN, at least of previous versions (1.x), have been formalised before, typically using Petri nets; see Sect. 2. But those semantic models are not complete; they use languages (in particular, variants of Petri nets) with which not everyone feels comfortable, and no such formal model is included in the standard.)

During the process of formalisation, we identified various inconsistencies as well as ambiguities in the BPMN standard (though we were not the first to do so; see, e.g. [18, 20]). This meant that for a formal model, we sometimes had to choose between different possible interpretations of the standard or to choose which of two or more conflicting provisions to adopt and which to ignore or to add assumptions. Moreover, in some cases, we chose not to include certain constructs in our specification for pragmatic reasons (which are stated in their due places).

To our knowledge, this work constitutes the most complete as well as detailed formal model for BPMN 2.0 process diagrams. (Please note that we are always referring to version 2.0 of BPMN from here on.) We also provide vertical refinement of the core language, which leads the way to specific implementations. This kind of improvement allows for more consistency in the interpretation of comprehensive models as well as for real exchangeability of models between different tools. (Experiences in this respect are mixed. Our own experiences and those of another group were negative, but see also [48] and [115].)

Regarding horizontal refinement of the core language, in the form of different extensions which we find indispensable in many cases, we conclude with an outlook on future research. The extensions we propose address actor modelling (including an easily understandable way for denoting permissions and obligations), integration of user-centric views, a refined communication concept and integration of data. Although all of these layers together form an integrated whole, each of them can be of interest on its own.

Additional support for standardised tool support will be given in future research work through a reference architecture for a BPM system which includes, besides a workflow engine, actor management and user interaction, data management and different interfaces. Furthermore, we want to show how the abstract, platform-independent semantic model can be reliably refined towards a concrete implementation of a suitable workflow engine.

## 1.1 Motivation

BPMN is a widely used standard for business process modelling. The current major release of BPMN is quite comprehensive and spans more than 500 pages. However, major drawbacks of BPMN are the limited support for organisational modelling, the only implicit expression of modalities, as well as the lack of integrated user interaction and data modelling. In addition, the syntactical and, in particular, semantic definitions of the BPMN standard are in several cases inaccurate, incomplete or inconsistent. The current work addresses concrete issues concerning the execution semantics of business processes and provides a formal definition of BPMN process diagrams, which ought to be a solid basis for further extensions, i.e. in the form of horizontal refinements of the core language.

To motivate this work on rigorous semantics for BPMN process diagrams, we discuss challenges in writing formal specifications of business process models resulting from the diversity of readers on the one hand, which requires general intelligibility, and a need for formality on the other hand.

In the following, we present some of the problems we have encountered, for example, concerning (instantiating) event-based gateways, compensation event sub-processes and process decomposition. We will further argue that the Abstract State Machine (ASM) method, developed from the mid-1980s on by Gurevich [14,15,53, 54] and practically applied in software engineering by Börger and others [17, 19, 20,23,24,132] from the 1990s on, offers considerable advantages over other formal methods with respect to the potential to bridge the gap between intelligibility and formality.

During our work on formalising the BPMN standard, we have faced ambiguities and inconsistencies, regarding both syntactical and semantic definitions in the standard.

Firstly, regarding syntax, the standard only provides a semiformal definition of the BPMN metamodel in the form of class diagrams, corresponding tables specifying the attributes and model associations, as well as XML schemas. However, the definition of an element in the class diagram is partly overlapping with the refined specification in the corresponding table and redundant to the XML schema. Due to this redundancy, the description of the metamodel is in several cases inconsistent and contradictory. Additionally, further syntactical rules are defined within natural text descriptions, also containing deviating information.

For example, considering the *transaction* element, the class diagram specifies two attributes, *protocol* and *method*, both of type "String", but in the corresponding table, only *method* is mentioned and defined to be of type "TransactionMethod". Moreover, the XML schema defines the default value "Compensate", which is missing in the attribute description of the table.

For a full definition of an element, it is further necessary to consider the specification of all superclasses. We have formally defined the syntax of BPMN by means of an ontology in [86], where we point out further contradictions in the BPMN standard concerning the class hierarchy. We have discussed contradictions regarding event triggers, amongst others, in [66] and regarding the class hierarchy and attributes of sub-processes in [89].

Secondly, regarding the semantic definitions, the definitions of elements are distributed across various sections and sometimes conflicting, i.e. the BPMN standard often specifies an element in a very general way in one place and then constrains this description in various other places. For example, start events are described several times within the chapter *Overview* [95, p. 27, pp. 31f]; in chapters *Process*

[95, pp. 238ff], *Choreography* [95, pp. 339f] and *BPMN Execution Semantics* [95, pp. 439f]; as well as in sections describing other elements that can comprise or connect to start events, e.g. event subprocesses [95, p. 177].

Consequently, an intense study of the BPMN standard revealed apparent inconsistencies or even contradictions between descriptions of the same element, while at the same time, the semantics of certain elements remains ambiguous. Studying further literature (see, e.g. [47]) often confirmed that certain parts of the BPMN standard can be interpreted in different ways, while certain constructs seem to be ignored by the literature and by tools. Sometimes additional literature even added to our puzzlement. Thus, by our formalisation of the semantics of BPMN models using ASMs, we aimed to address these ambiguities to gain a clear, well-defined modelling language.

In particular, event-related aspects and process decomposition cause misunderstandings and confusion about what the intended behaviour of some BPMN elements really is. We want to give some concrete examples.

***Example 1: Event-Based Gateways.*** First, we have identified several issues concerning event-based gateways as described in the BPMN standard, where the exact semantics seems ambiguous and sometimes even contradictory [66]. In particular, we faced problems with (a) triggering event-based gateways, (b) closely related, with determining the moment when an event-based gateway should be considered to have been triggered and, (c) in the case of an instantiating gateway, when a respective process instance should be created. All the interpretations of the BPMN standard which we could think of turned out to contradict some part of the standard. For example, it says that "The choice of the branch to be taken is deferred until one of the subsequent **Tasks** or **Events** completes" [95, p. 437]. This appears to suggest that no tokens are sent by the gateway to any of the events or receive tasks in its configuration. However, it is stated elsewhere in the standard that a receive task needs to be activated before it can start waiting for a message [95, p. 430], but also an intermediate event (the only event type possible here) is obviously supposed to get a token before it can start waiting for events: "Waiting starts when the **Intermediate Event** is reached" [95, p. 440].

Further problems concern the use of event-based gateways to instantiate subprocesses, where we discovered further discrepancies between different parts of the BPMN standard (see, e.g. [95, p. 299, p. 430, p. 440]). Mainly, the interpretation of the semantics of instantiation by parallel event-based gateways is ambiguous in a crucial way, namely, concerning the question whether it allows for asynchronous behaviour of different branches after the gateway or not [95, p. 299, p. 416, p. 437]. This question also affects the question whether such a gateway could be replaced by other constructs, e.g. special forms of start events. Hence, while the general idea of event-based gateways seems intuitive at first sight, the semantic details are far less so, most of all in the case of instantiating event-based gateways. Clearly, the standard needs to clarify many points regarding the semantics of event-based gateways, and especially Chaps. 10

and 13 of the standard need to be rendered consistent with each other (see [66] for more information).

***Example 2: Compensation Event Sub-process.*** A further open question regarding compensation is whether compensation event sub-processes really work. Event sub-processes (as well as boundary events) may be defined for starting compensation handlers [95, p. 248, p. 252]. Compensation is generally aligned with undoing actions that were already successfully completed [95, p. 302], or "*Compensation* of a successfully completed **Activity** triggers its *compensation handler*" [95, p. 235]. Hence, an activity that is still active cannot be compensated, or if "*compensation* is invoked for an **Activity** that has not yet completed, or has not completed successfully, nothing happens" [95, p. 235]. However, how can a compensation event sub-process be activated if its parent activity has already been completed? The description of the default behaviour of starting an event sub-process ("while the parent **Process** is active" [95, p. 177]) is inconsistent with the definitions above, and, therefore, we are led to suppose that the compensation event sub-process can never be started.

On the contrary, there are also definitions that define an execution semantics which seems to be consistent with the introductory statement, i.e. that an event sub-process may start compensational actions but conflict with the default behaviour of starting an event sub-process. For example, according to [95, p. 442], a compensation event sub-process will become enabled when its parent activity reaches the state "Completed", whereupon a snapshot of the parent activity's data is preserved for later usage by the compensation event sub-process. These specifications could give rise to the idea that a compensation event sub-process is considered as a particular case of an event sub-process. Yet no such provision is mentioned as a particular case in the description of an event sub-process [95, pp. 173ff], causing serious inconsistencies regarding the execution semantics of event sub-processes and compensation event sub-processes in particular (see [60] for more information).

***Example 3: Decomposition and Reusability.*** In contrast to some other business process modelling languages, BPMN provides explicit concepts for decomposition to cope with complexity and reusability in order to enhance the consistency of process models. In particular, BPMN specifies sub-processes and call activities for addressing decomposition and reusability. However, although BPMN provides such elements, the support for process decomposition is nevertheless limited and identified as a major drawback of BPMN (see, e.g. [104, 106]). In detail, the standard shows contradictions and limitations regarding the instantiation of sub-processes and call activities, due to several uncontrolled incoming and/or outgoing sequence flows, which may lead to implicitly created instances with neither the execution order nor the merging mechanism being defined. Similar problems arise on using several start events placed on the boundary or on defining activities without incoming/outgoing sequence flows, all causing confusion about the intended behaviour of a process.

Further contradictions and limitations concern the use of activities, in particular sub-processes, in combination with swimlanes and in unstructured diagrams.

For example, the class *FlowElementsContainer* (including sub-processes) can comprise 0..n lanes [95, p. 89], and each lane can comprise 0..n *FlowNodes* (including sub-processes) [95, p. 309]. This cyclic definition may be problematic, e.g. if someone wants to comprise two tasks which are located in different lanes. Furthermore, a limitation of BPMN, and most other business process modelling languages, is the missing support for asynchronous decomposition (see [89] for more information).

Considering this diversity of ambiguities, a serious issue is how to guarantee that the executable behaviour of a particular model is the same in different tools. Graphical notations like BPMN seem intuitive enough to be well understood almost at first sight. Unfortunately, they typically lack the precise mathematical basis that is required to render them really unambiguous. On the other hand, partial attempts on formalisation, e.g. based on Petri nets, are too difficult to understand even for most developers.

We chose the ASM method to formalise the semantics of BPMN. The ASM method is a formal technique that facilitates the formalisation of requirements at the level of abstraction determined by the given application domain while maintaining the correct-by-construction paradigm and also keeping specifications easy to understand. It comes with a small set of intuitive core constructs and is very flexible regarding notation. ASMs can be seen as "a rather intuitive form of abstract pseudo-code" [22, p. 2], though based on a precise yet minimal mathematical theory of algorithms, yet also as "Virtual Machine programs working on abstract data" [22, p. 5]. The ASM method consists of a notation for state-based models (automata) and a method for refinement. Models can be arbitrarily abstract, and abstract models can be stepwise refined towards programming code.

ASMs have demonstrated their strength in various domains, e.g. specifying the semantics of programming languages [17] and modelling languages [19], verifying the specification of the Java Virtual Machine [132] or formalising the ITIL change management process [69]. In addition, some encouraging steps towards a formal business process model using ASMs have been achieved.

A particular class of ASMs, called *control state ASMs*, has been shown to represent a "normal form" for Unified Modeling Language (UML) activity diagrams (see, e.g. [22]). Control state ASMs are frequently used and thus well understood in practice. They build upon the concept of classical finite state machines. As BPMN process diagrams can be seen as an extended and specialised form of activity diagrams, we can employ control state ASMs for modelling the semantics of BPMN process diagrams as well. As a matter of fact, we can already build on existing work in which transformations of process diagrams into ASMs have already been suggested [20, 23] and refine the mapping of each construct of BPMN to ASM constructs.

Transforming process diagrams into ASMs will enable us to:

- highlight shortcomings in the BPMN standard regarding the semantics of process diagrams,

- prove desired properties of process diagrams in general as well as of particular process diagrams (or classes of process diagrams),
- better apply model-checking techniques to process diagrams, and
- provide a basis for building software tools for business process modelling.

We therefore base our rigorous semantics for BPMN process diagrams on the ASM method and also apply intuitive specification writing style guidelines that improve the understandability of formal specifications by combining rigour with a way of expression that is closer to natural language [67].

## 1.2 Intended Readership and Relevance

This book provides the most complete formal specification of the semantics of BPMN process diagrams available to date, in a style that is easily understandable for a wide range of people—not only by experts in formal methods. Thereby, as a side benefit, we also demonstrate the benefits of the ASM method combined with efforts to stay close to natural language as a style of rigorous specification which can be used much more widely.

Such kind of rigorous specification makes it possible to analyse the BPMN standard profoundly and to develop it further to modelling and simulation tools for BPMN, which is demonstrated in this book. It can also serve as a basis for further development of the BPMN standard and additional add-ons.

Of particular importance are the potential insights that can help to resolve a multitude of today's sources of faults in business process implementation projects, i.e. it is expected that the results will contribute to a more solid practice in the development of process-oriented applications. It can further be expected that the formal approach will have a considerable impact on a wide range of modelling approaches and tools. Having a comprehensive, sound, and formal model, tools and frameworks can rely on these findings and adjust the semantics of their modelling and refinement constructs according to the formal specification. In doing so, the semantics cannot be lost or changed by working with these tools and techniques. This definitively constitutes a big step towards semantically correct business process implementations. For example, modelling products could guarantee that a refinement step has been applied semantically correct. Furthermore, models created with different modelling tools could become comparable.

A summary of the complete ASM model, including the signature, can be found on the Web page:

http://h-bpm.scch.at

## 1.3 Outline

The rest of this book is structured as follows: Related work with a focus on established business process modelling techniques, formal descriptions of BPMN, and the suitability of BPMN for business process modelling is studied in Chap. 2. Chapter 3 provides an introduction to modelling the semantics of process diagrams with ASMs. The core part of this book is Chap. 4, where we define a rigorous semantics for BPMN process diagrams in detail.

Chapter 5 gives an overview over potential uses of such a formal model, with a special focus on validation and verification of process diagrams, and Chap. 6 provides a detailed discussion of the BPMN 2.0 standard. An approach to refine the semantic model towards a workflow engine by the method of stepwise refinement is given in Chap. 7. We conclude with a final discussion of the results and give an overview over future research in Chap. 8.

Appendix A contains an overview of auxiliary constructs which we use in addition to standard ASM notation, including set notation, and the complete signature of the ASM ground model.

Springer

http://www.springer.com/978-3-319-09930-9

A Rigorous Semantics for BPMN 2.0 Process Diagrams
Kossak, F.; Illibauer, C.; Geist, V.; Kubovy, J.;
Natschläger, C.; Ziebermayr, Th.; Kopetzky, T.;
Freudenthaler, B.; Schewe, K.-D.
2014, X, 235 p. 80 illus., 1 illus. in color., Hardcover
ISBN: 978-3-319-09930-9