S-BPM-Ont: An Ontology for Describing and Interchanging S-BPM Processes

Kai Michael Höver and Max Mühlhäuser

Technische Universität Darmstadt
Dept. of Computer Science
64289 Darmstadt, Germany
{kai,max}@tk.informatik.tu-darmstadt.de

Abstract. The description of business processes is important for an unambiguous definition and its reusability. However, due to different data models of business modeling tools, an exchange of process definitions is difficult. This demands for a formal model of business processes. An ontology is an explicit specification of a conceptualization and can therefore be used as an (inter)lingua to describe S-BPM processes. This paper presents a S-BPM ontology modeled with the Web Ontology Language.

Key words: S-BPM, Ontology, OWL, Modeling.

1 Introduction and Motivation

S-BPM is an emerging method for modeling business processes [1,2]. Its growing popularity and distribution comes along with a rise of tools for modeling S-BPM processes. Such tools, however, use different data models for representing S-BPM processes. This results in a low interoperability between S-BPM applications, be it modeling applications or applications for executing S-BPM processes. All of them need a representation of the processes, and these are as said stored in a various kind of data models and formats. To take an example, the two S-BPM modeling applications Metasonic Suite[1] and S-BPM Groupware [3] describe their process models differently. The first stores models in XML representations, the latter uses Scala data models [2], JSON (JavaScript Object Notation), and database models. Thus, an exchange of business models is not instantly possible.

One approach to facilitate the interoperability between applications is to write translators to convert from one S-BPM process data model to another. This solution demands for $O(n^2)$ translators for n different data formats. A metaformat helps to reduce the number of translators. One approach is to define S-BPM processes with the Extensible Markup Language[3] (XML), particularly with a definition of an XML schema[4] for S-BPM. However, this approach only

[1] `http://www.metasonic.de/en/metasonic-suite`
[2] `http://www.scala-lang.org`
[3] `http://www.w3.org/TR/xml/`
[4] `http://www.w3.org/XML/Schema`

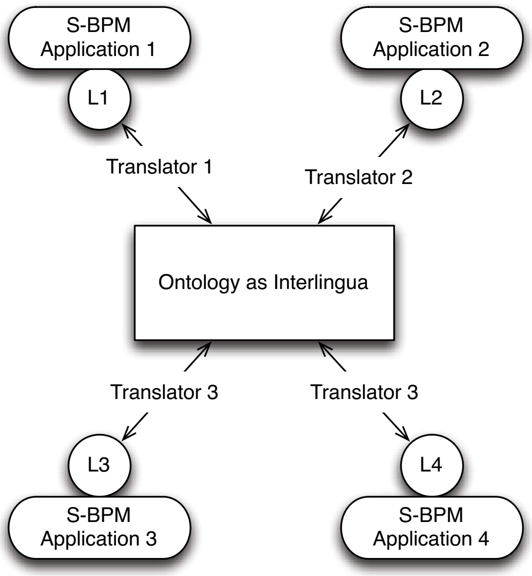

Fig. 1. Translation between different languages used by different applications

defines the syntax but lacks of semantics. In contrast, ontologies define a domain theory as a conceptual level on top of XML data.

Ontologies enable the formal definition of an inter-lingua meta-format. Using an ontology reduces the number of translators to only $O(n)$. "To assist interoperability, ontologies can be used to support translation between different languages and representations" [4]. Unlike data models, an ontology is independent of specific data model, it is "a representation of a world external to any particular system" [5], so that it "can be reused by different kinds of applications/ tasks" [6]. Figure 1 depicts the general approach and illustrates the use of an ontology as an interlingua between different languages (data representations) and corresponding translators to translate not directly between languages, but between a language and an ontology (interlingua).

The paper is structured as follows. First, we provide a short introduction to ontologies and the Web Ontology Language (OWL). Then we present an ontology for the representation of S-BPM processes. Here, we first explain the applied methodology and then present the ontology's concepts in more detail. We close with a summary and an outlook on future work.

2 Introduction to Ontologies

Several definitions for ontologies exist [7]. One of the most cited definitions is provided by Gruber in 1993. He defines an ontology as an "explicit specification of a conceptualization" [8]. In 1997, Borst extended this definition by the point that the conceptualization should express a shared view [9]. Studer et al. merge these definitions and define an ontology as a "formal, explicit specification of a shared conceptualization" [10]. The important elements of this definition are:

- *conceptualization*: an ontology describes the objects, concepts in a certain domain of interest and the relationships between them
- *formal* and *explicit*: the ontology should be specified in a language that comes with formal semantics
- *shared*: because if not, the benefits of having an ontology are limited.

One of the most popular ontology description languages is the Web Ontology Language (OWL), a W3C recommended standard since 2004[5]. OWL is based on description logics and enables the definition of classes, properties, and relations among conceptual objects [11].

After this short introduction we present a S-BPM ontology modeled with OWL.

3 A S-BPM Ontology

In this section, we present an ontology for describing S-BPM processes called S-BPM Ont. First, we present the applied methodology for engineering the ontology. Then we present the design of the ontology.

3.1 Methodology

We use the Ontology Development 101 methodology for the engineering of the S-BPM ontology [12]. It is an iterative process for creating ontologies. The methodology suggests to follow 7 steps:

1. Determine the domain and scope of the ontology
2. Consider reusing existing ontologies
3. Enumeration important terms
4. Define classes and the class hierarchy
5. Define the properties of the classes
6. Define constraints
7. Create instances

Subsequently, we orientate ourself by these steps for defining the S-BPM ontology.

3.2 Scope of the Ontology

The scope of the S-BPM ontology is the domain of subject-oriented business processes. The purpose is to be able to describe S-BPM processes in an application-independent, formal way.

In order to refine the scope we present some so called competency questions in the following as suggested in the methodology of Grüninger and Fox [13]. Competency questions have the purpose to ask questions the ontology should be able to answer.

[5] `http://www.w3.org/TR/owl2-overview/`

3.3 Reusing Existing Ontologies

To our best knowledge, there does not exist any ontology for describing S-BPM processes that could be reused. On et al. [14] define a behavior ontology for process algebra. As the ontology is not defined in any ontology language it cannot be directly reused, however partly its concepts. Related, but with another scope is the publication of Fleischmann et al. [15]. Here, the authors define a meta-model to support multi-agent business processes.

3.4 Important Terms

Albert Fleischmann defines the elements of the S-BPM modeling language in [1]. According to this publication we can identify several important terms. The basic modeling features include for the communication structure *subjects*, *messages*, and *business objects*. Further, subjects can perform actions which include *sending* and *receiving* messages as well as executing *internal actions*. The *behavior* is defined by these activities that are performed in a certain order. Depending on the outcome of activities, the execution of the process chooses one of the available *exit conditions* or *alternatives*. Subjects exchange messages via their *input pools*. Each input pool can define *size constraints* for the whole input pool, specific message types, or senders of messages.

3.5 Define Classes and the Class Hierarchy

In this section we define the classes of the ontology after identifying important terms. For this purpose and the definition of both properties and constraints we follow the definitions of S-BPM defined by Fleischmann [1,2] and Börger [16].

In the following, we define classes and the class hierarchy of the S-BPM ontology. It is depicted in figure 2.

Subject A subject is an acting element within a process. Subjects execute and synchronize their activities by exchanging messages. Each subject has an assigned behavior and an input pool.

Behavior The process behavior that is assigned to a subject and defined by its states and state transitions.

State A state is part of a behavior. A subject can be in only one state at a time. There are end states and initial states. A behavior diagram can have only one initial state but one or more end states. A state has further a specific type of action (send, receive, function).

- **InitialState** The initial state of a subject behavior
- **EndState** An end state of a behavior. A subject behavior may have one or more end states

Edge An edge defines the transition between two states. A transition is true if it satisfies a certain exit condition.

- **Timeout** Timeouts are a subset of edges as well as
- **UserAbruption** that defines the interruption of a process by a user.

ExitCondition The class of exit conditions

Action In a state a certain type of action is defined. We distinguish between internal function and communication acts (send or receive). Formally, an *Action* is defined as $Action \equiv CommunicationAct \sqcup Function$.

- **CommunicationAct** A *Communication Act* is either an act of sending or receiving messages. It is defined as $CommunicationAct \equiv Send \sqcup Receive$. Further, the classes of *Send* and *Receive* are disjoint: $Send \sqcap Receive \equiv \emptyset$
  - **Send** A send communication act
  - **Receive** A receive communication act
- **Function** An internal function that is performed by a subject. The class of internal functions is disjoint with the class of communication acts and thus its complement: $CommunicationAct \sqcap Function \equiv \emptyset$

InputPool An input pool for exchanging messages between subjects

Restriction An input pool may be configured by size restrictions, i.e.,

- **TypeRestriction** A size restriction regarding a certain type of messages
- **SenderRestriction** A size restriction regarding a certain sender subject

Message A message is exchanged between subjects via their input pools

MessageType Each message is of some type like a business trip request.

3.6 Properties of the Classes and Their Constraints

After outlining the classes of the S-BPM ontology, we define the relations between them, technically also called *properties*. In OWL, properties are also called *roles*. Relations are always binary. OWL distinguishes between datatype properties and object properties. *Datatype properties* define relations between classes and concrete data types like string, integer, date, etc. *Object properties* define relations between classes.

Relations between Messages, Message Types, and Subjects

A message can be of a certain type. We therefore define a relation *hasType* between a *message* and a *message type* with

$$\exists hasType.\top \sqsubseteq Message \tag{1}$$

$$\top \sqsubseteq \forall hasType.MessageType \tag{2}$$

defining that the domain of *hasType* is a message and the range is the type of the message. Further, a message is of exactly one message type.

A message has exactly one sender and one receiver (this is also true for multi-subjects at the process structure level). This constraint is expressed by the relations *sender* and *receiver*, and the specification of the *Message* class (see equations (5)-(7)). The definition of *sender* is

$$\exists sender.\top \sqsubseteq Message \tag{3}$$

$$\top \sqsubseteq \forall sender.Subject \tag{4}$$

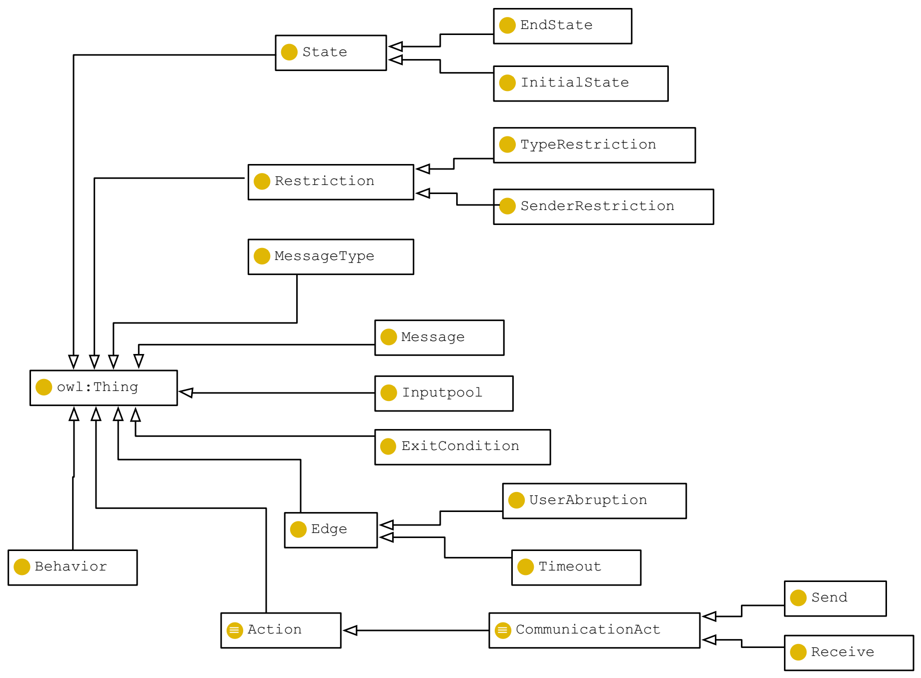

Fig. 2. The class hierarchy of the S-BPM ontology

The definition of *receiver* is analogous.

This leads us to the definition of the class *Message* with:

$$Message \sqsubseteq (\leq 1\ hasType) \sqcap (\geq 1\ hasType) \tag{5}$$
$$Message \sqsubseteq (\leq 1\ sender) \sqcap (\geq 1\ sender) \tag{6}$$
$$Message \sqsubseteq (\leq 1\ receiver) \sqcap (\geq 1\ receiver) \tag{7}$$

Relations between States, Edges, Behavior, and Subject

States and their transitions define the behavior of a subject. As introduced in section 3.5, subclasses of states are initial states and end states. Transitions are directed edges that have both a source and a target. For this we define two relations *source* and *target* for edges.

$$\exists source.\top \sqsubseteq Edge \tag{8}$$
$$\top \sqsubseteq \forall source.State \tag{9}$$
$$\exists target.\top \sqsubseteq Edge \tag{10}$$
$$\top \sqsubseteq \forall target.State \sqcap \neg InitialState \tag{11}$$

Both source and target relations are unambiguous, that means that they refer to exactly one state each. This constraint can be expressed by defining these relations as functional. This means that $source(s_1, e_1)$ and $source(s_1, e_2)$ implies $e_1 = e_2$. The same is true for the *target* relation.

States have one ore more *outgoing edges.* A special type of an edge is a time out edge. Each state may have at most one timeout edge but many user abruptions. An end state may have outgoing edges but an initial state has only outgoing edges. Doing so, the class of states is specified as

$$State \sqsubseteq \geq 1\ outgoingEdge.Edge \sqcap \leq 1\ outgoingEdge.Timeout \tag{12}$$
$$State \sqsubseteq (\leq 1\ hasServiceType) \sqcap (\geq 1\ hasServiceType) \tag{13}$$

Further, a state performs a certain type of action (send, receive, function), except of an end state. The relation hasServiceType between a state and an action class is defined as:

$$\exists hasServiceType.\top \sqsubseteq (State \sqcap \neg EndState) \tag{14}$$
$$\top \sqsubseteq \forall hasServiceType.Action \tag{15}$$

The behavior of a subject is described as a set of states and their direct edges between them:

$$Behavior \sqsubseteq (\geq 0\ Edge) \tag{16}$$
$$Behavior \sqsubseteq (\leq 1\ hasState.EndState) \sqcup (\geq 2\ hasState.State) \tag{17}$$

A subject is assigned to at most one behavior and exactly one input pool. The relation *hasInputpool* is inverse functional, because the owner of an input pool is unambiguous. The definition of these two relations are

$$\exists hasBehavior.\top \sqsubseteq Subject \tag{18}$$
$$\top \sqsubseteq \forall hasBehavior.Behavior \tag{19}$$
$$\exists hasInputpool.\top \sqsubseteq Subject \tag{20}$$
$$\top \sqsubseteq \forall hasInputpool.InputPool \tag{21}$$
$$\tag{22}$$

Furthermore, the class of subjects is formally defined as

$$Subject \sqsubseteq (\leq 1\ hasInputpool) \sqcap (\geq 1\ hasInputpool) \tag{23}$$
$$Subject \sqsubseteq (\leq 1\ hasBehavior) \sqcap (\geq 1\ hasBehavior) \tag{24}$$

Relations between Messages, Input Pools and Their Constraints
Input pools can be configured by several size constraints:

– the overall capacity of an input pool
– size restriction regarding a special type of message
– size restriction regarding a special sender

We specify the overall capacity of an input pool with a datatype property *hasCapacity* which enables to set the maximum number of messages of an input pool with a positive number. For *message type restrictions* and *sender restrictions* we define a relation *hasRestriction* with its two sub-properties *hasTypeRestriction* and *hasSenderRestriction*. The class for message *type restrictions* is defined as

$$TypeRestriction \sqcap (\leq 1\ hasCapacity.posInt) \sqcap (\geq 1\ hasCapacity.posInt) \tag{25}$$

$$TypeRestriction \sqcap (\leq 1\ regarding.MessageType) \sqcap (\geq 1\ regarding.MessageType) \tag{26}$$

$$\tag{27}$$

Analogously, the class of *sender restrictions* is specified as

$$SenderRestriction \sqcap (\leq 1\ hasCapacity.posInt) \sqcap (\geq 1\ hasCapacity.posInt) \tag{28}$$

$$SenderRestriction \sqcap (\leq 1\ regarding.Subject) \sqcap (\geq 1\ regarding.Subject) \tag{29}$$

$$\tag{30}$$

An input pool then has an arbitrary number of restrictions regarding message types or senders (or both), but exactly one overall capacity restriction. This is necessary to be able to distinguish between synchronous and asynchronous communication.

$$InputPool \sqcap (\leq 1\ hasCapacity.posInt) \sqcap (\geq 1\ hasCapacity.posInt) \tag{31}$$

$$InputPool \sqcap (\geq 0\ hasSenderRestriction.SenderRestriction) \tag{32}$$

$$InputPool \sqcap (\geq 0\ hasTypeRestriction.TypeRestriction) \tag{33}$$

$$\tag{34}$$

Figure 3 illustrates the constraints model of input pools.

Create Instances. In our case, instances are not part of the S-BPM ontology until a concrete process is defined. Therefore, the proposed ontology only contains concept definitions also called TBox (terminology knowledge) and no assertional knowledge (ABox). However, in the following section we present an example business process and its instances that use the defined ontology.

3.7 Ontology Example

In order to illustrate the use of the ontology, we present the employee subject behavior known from the business trip application example (see [2, p. 20]). Figure 4 depicts the employee's behavior.

In figure 5 an ontological representation of the employee behavior diagram is presented. It depicts the employee with its input pool and its behavior. The behavior consists of the states (fill out request, send request, receive answer, do BT, and End) as well as the edges between them. Each edge defines a source and a target edge.

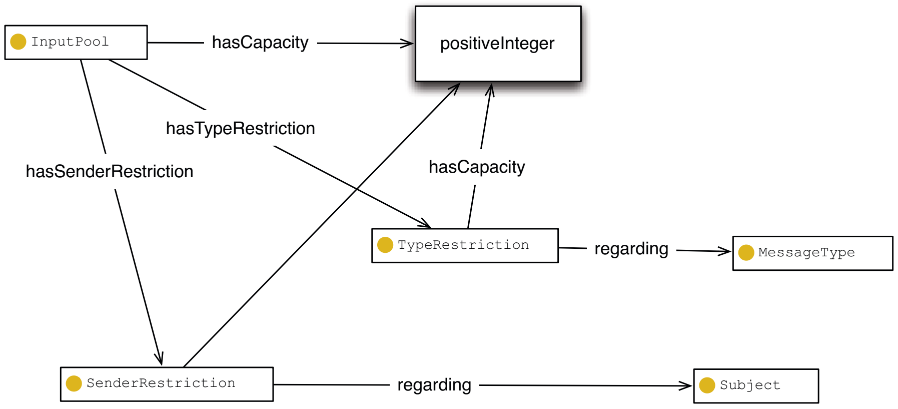

Fig. 3. The model of input pool size constraints

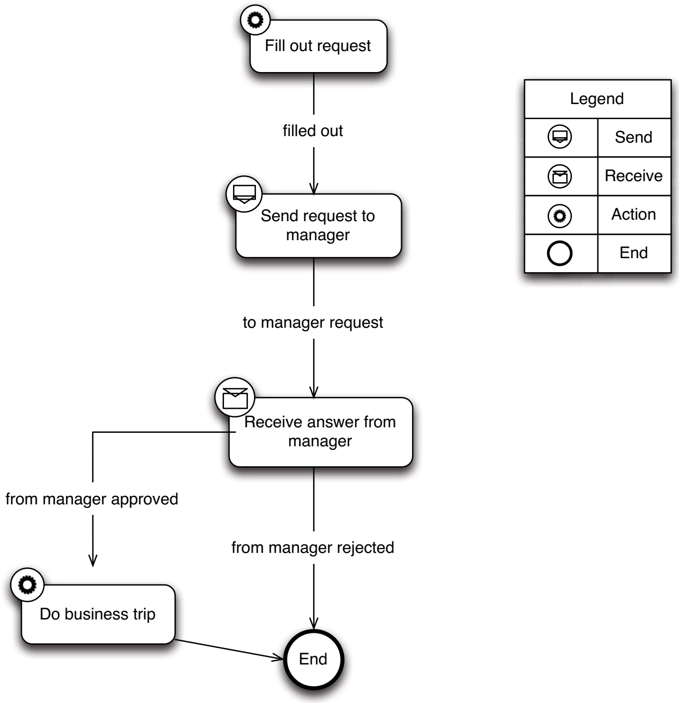

Fig. 4. The behavior diagram of the subject employee in the business application process

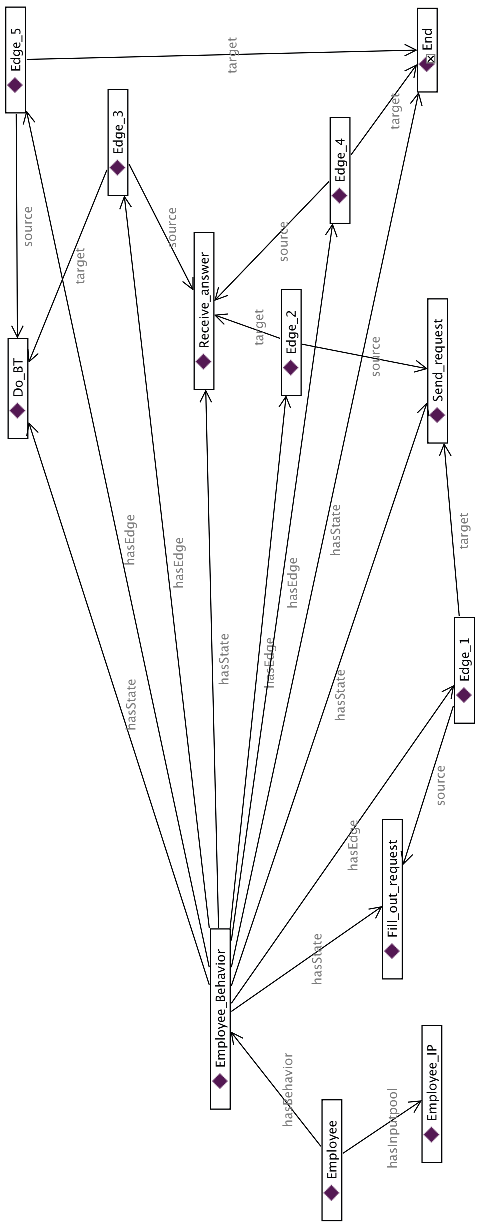

Fig. 5. An ontological representation of the employee's behavior

4 Summary and Future Work

The exchange of models between applications is crucial for interoperability and their reuse. This is especially true for business process models, as they often involve several distributed acting objects distributed over different IT infrastructures. Therefore, we have proposed an ontology for S-BPM processes. It provides a formal specification using the Web Ontology Language.

For future work we plan to continue and to add more S-BPM elements like macros. Specifications for multi-subjects and multi-send and -receive actions are missing in the current ontology version. We intend to integrate these concepts in the ontology and to refine the communication acts as special cases of multi-send and multi-receive, respectively. Furthermore, we plan to define logical rules and further constraints to improve the S-BPM ontology's semantics.

References

1. Fleischmann, A.: What Is S-BPM? In: Buchwald, H., Fleischmann, A., Seese, D., Stary, C. (eds.) S-BPM ONE 2009. CCIS, vol. 85, pp. 85–106. Springer, Heidelberg (2010)
2. Fleischmann, A., Schmidt, W., Stary, C., Obermeier, S., Brger, E.: Subject-Oriented Business Process Management. Springer, Heidelberg (2012)
3. Borgert, S., Mühlhäuser, M.: A S-BPM Suite for the Execution of Cross Company Subject Oriented Business Processes. In: Nanopoulos, A., Schmidt, W. (eds.) S-BPM ONE 2014. LNBIP, vol. 170, pp. 161–170. Springer, Switzerland (2014)
4. Uschold, M., Grninger, M.: Ontologies: Principles, methods and applications. The Knowledge Engineering Review 11(02), 93–136 (1996)
5. Colomb, R.M.: Ontology and the Semantic Web. Frontiers in Artificial Intelligence and Applications, vol. 156. IOS Press (2007)
6. Spyns, P., Meersman, R., Jarrar, M.: Data modelling versus ontology engineering. SIGMOD Record 31(4), 12–17 (2002)
7. Guarino, N., Oberle, D., Staab, S.: What Is an Ontology? In: Staab, S., Rudi Studer, D. (eds.) Handbook on Ontologies, International Handbooks on Information Systems, pp. 1–17. Springer, Heidelberg (2009)
8. Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. 5(2), 199–220 (1993)
9. Borst, W.N.: Construction of Engineering Ontologies for Knowledge Sharing and Reuse. PhD thesis, Universiteit Twente, Enschede (September 1997)
10. Studer, R., Benjamins, V.R., Fensel, D.: Knowledge engineering: Principles and methods. Data & Knowledge Engineering 25(1-2), 161–197 (1998)
11. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From {SHIQ} and {RDF} to OWL: The making of a Web Ontology Language. Web Semantics: Science, Services and Agents on the World Wide Web 1(1), 7–26 (2003)
12. Noy, N.F., Mcguinness, D.L.: Ontology Development 101: A Guide to Creating Your First Ontology. Technical report (2001)
13. Grninger, M., Fox, M.S.: Methodology for the Design and Evaluation of Ontologies. In: IJCAI 1995 Workshop on Basic Ontological Issues in Knowledge Sharing (1995)
14. On, J., Choe, Y., Lee, M.: An Abstraction Method of Behaviors for Process Algebra. In: 2013 IEEE 37th Annual Computer Software and Applications Conference Workshops (COMPSACW), pp. 133–138 (2013)

15. Fleischmann, A., Kannengiesser, U., Schmidt, W., Stary, C.: Subject-Oriented Modeling and Execution of Multi-agent Business Processes. In: 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), vol. 2, pp. 138–145 (2013)
16. Fleischmann, A., Schmidt, W., Stary, C., Obermeier, S., Brger, E.: A Precise Description of the S-BPM Modeling Method. In: Subject-Oriented Business Process Management, pp. 227–240. Springer, Heidelberg (2012)