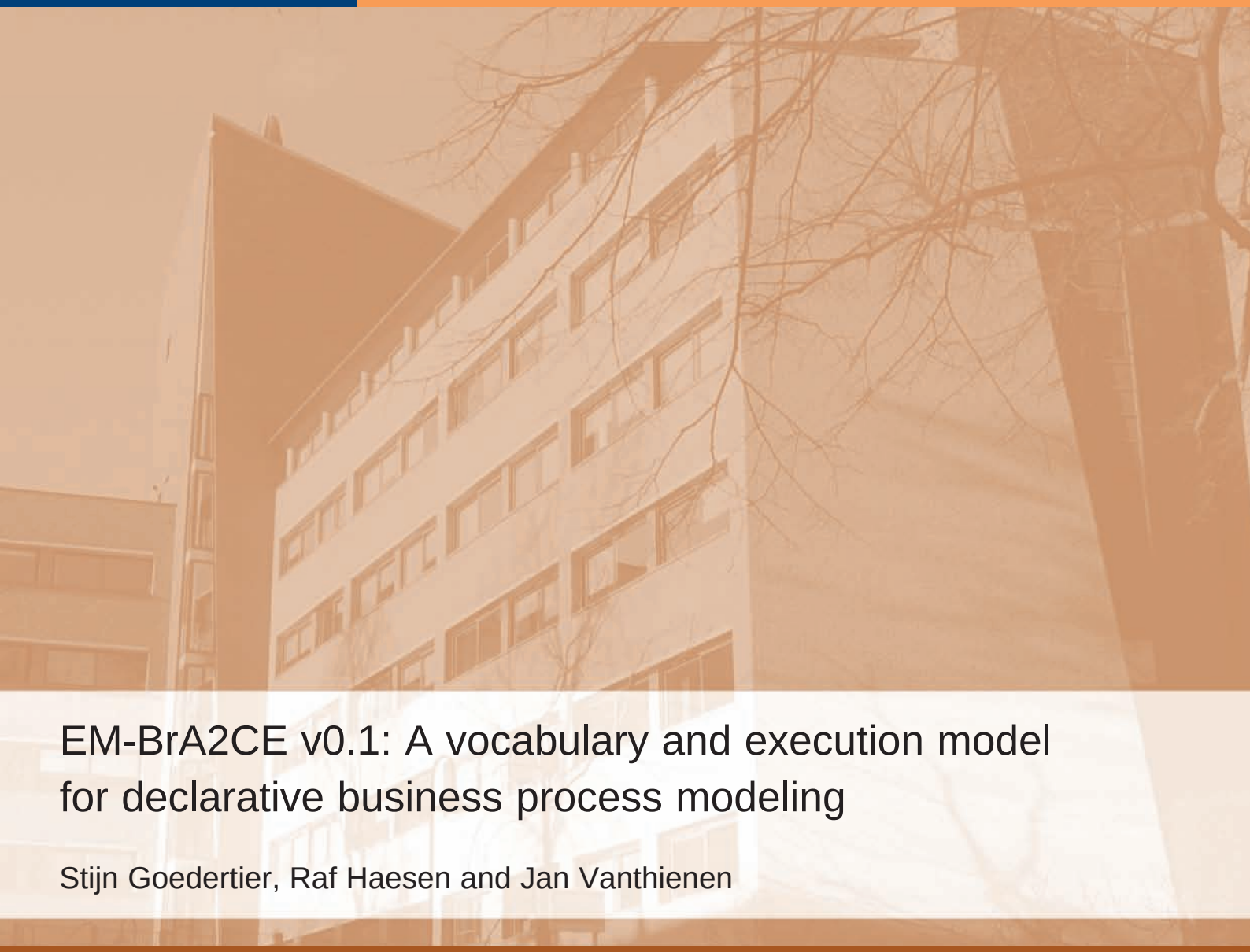
KATHOLIEKE UNIVERSITEIT
LEUVEN

Faculty of Economics and Applied Economics



EM-BrA2CE v0.1: A vocabulary and execution model for declarative business process modeling

Stijn Goedertier, Raf Haesen and Jan Vanthienen

DEPARTMENT OF DECISION SCIENCES AND INFORMATION MANAGEMENT (KBI)

KBI 0728

EM-BrA^{2}CE v0.1: A Vocabulary and Execution Model for Declarative Business Process Modeling

Stijn Goedertier[1], Raf Haesen[1,2] and Jan Vanthienen[1]

[1] Katholieke Universiteit Leuven,
Department of Decision Sciences & Information Management,
Naamsestraat 69, B-3000 Leuven, Belgium
{stijn.goedertier;raf.haesen;jan.vanthienen}@econ.kuleuven.be

[2] Vlekho Business School Brussels, Belgium
Koningsstraat 336, 1000 Brussels, Belgium
raf.haesen@vlekho.wenk.be

Abstract

In management theory of the last decades, much importance has been attached to a process-oriented perspective on organizational (re)structuring. Yet to date, organizations still experience difficulties in applying this process-oriented perspective to the design and maintenance information systems. The root of the problem lies with a procedural representation of business processes that contains inadequate information for computer systems to provide flexible automated business process support. The counterpart of a procedural representation is a declarative one that explicitly takes into account the business concerns that govern business processes. Recently, a number of process modeling languages have appeared that could be identified as declarative languages. These modeling languages have very distinct knowledge representation backgrounds, often lack a formal execution model and often only model one aspect of the many business concerns that exist in reality. What is needed are meaningful ways to combine several kinds of expressions, called business rule types, independently of the used methods for knowledge representation and reasoning. In this paper, we present the EM-BrA^{2}CE (Enterprise Modeling using Business Rules, Agents, Activities, Concepts and Events) Framework, a unifying vocabulary and execution model for declarative process modeling. The vocabulary is described in terms of the Semantics for Business Vocabulary and Rules (SBVR) standard and the execution model is presented as a Colored Petri Net (CP-Net). In addition, we show how declarative process models can contribute to the model-driven design of Service-Oriented Architectures.

keywords: Business Process Management, Business Modeling, Service Modeling, SBVR, Service-Oriented Architecture

Contents

1 Introduction **3**

2 Related work **5**

3 Procedural versus Declarative Process Modeling **6**
3.1 Business Concerns Made Explicit 7
3.2 Declarative Business Rule Enforcement 8
3.3 Declarative Communication Logic 8
3.4 Dynamic Execution Scenarios 9
3.5 Activity-level Granularity 9
3.6 Model Differentiation by Modality 10
3.7 Assumption bias 11
3.8 Runtime Alteration 11
3.9 No Human-Machine Distinction 11
3.10 Coordination Work is Work 12
3.11 Multi-state Activities 12
3.12 Third-person perspective 12
3.13 Meaning is Separate from Expression 12

4 An Introduction to Declarative Process Modeling **13**
4.1 Process Model = State Space + Transition Constraints 13
4.2 History-dependent behavior 14
4.3 Running example: payment-after-shipment 15

5 A Vocabulary for Declarative Process Modeling **16**
5.1 Candidate Ontology Language 18
5.2 An Introduction to SBVR 19
5.3 The EM-BrA^{2}CE Vocabulary 22
5.3.1 Business concept – business concept type 22
5.3.2 Activity – activity type 24
5.3.3 State – state space 26
5.3.4 Agent – Role 27
5.3.5 Event – event type 29
5.3.6 Deontic assignment 30
5.3.7 Non-functional, quality-of-service concerns 32
5.3.8 Cost and time concerns 33
5.4 Business Rules in the EM-BrA^{2}CE Framework 34
5.4.1 Providing Logical Foundations for Temporal Rules 35
5.4.2 Semantic Formulation of Temporal Rules 37
5.4.3 Control-flow: temporal deontic rule 37
5.4.4 Control-flow: activity precondition 38
5.4.5 Control-flow: activity postcondition 39
5.4.6 Control-flow: reaction rule 39
5.4.7 Control-flow aspect: dynamic integrity constraint 40
5.4.8 Control-flow aspect: activity cardinality constraint 40
5.4.9 Control-flow aspect: serial activity constraint 41
5.4.10 Control-flow aspect: activity order constraint 41
5.4.11 Control-flow aspect: activity exclusion constraint 41
5.4.12 Control-flow aspect: activity inclusion constraint 42

5.4.13 Data aspect: Static integrity constraint 42
5.4.14 Data aspect: derivation rule 42
5.4.15 Organization aspect: activity authorization constraint 43
5.4.16 Organization aspect: activity allocation rule 43
5.4.17 Organization aspect: visibility constraint 44
5.4.18 Organization aspect: event subscription constraint 44

6 An Execution Model for Declarative Process Modeling 44
6.1 Business Rules in the Activity Life Cycle 44
6.2 A CP-Net-based Execution Model 46
6.2.1 Places and Color Sets 48
6.2.2 The Create transition 50
6.2.3 The Schedule transition 53
6.2.4 The Assign and Revoke transition 54
6.2.5 The Start transition 55
6.2.6 The fact manipulation transitions 55
6.2.7 The Complete transition 57
6.2.8 The Abort, Skip and Redo transitions 59
6.3 Unspecified semantics 62
6.3.1 Reactive Behavior 62
6.3.2 Transaction Handling 62
6.3.3 Composite State Transitions 63

7 Towards a Declarative Service-Oriented Architecture 63

8 Evaluation of the EM-BrA^{2}CE Framework 66

9 Conclusion 67

1 Introduction

In management theory of the last decades, much importance has been attached to a process-oriented perspective on organizational (re)structuring. Porter (1985), for instance, introduced the idea of the *value chain* to better understand the activities through which companies gain a competitive advantage. Kaplan (1998) drew attention to the technique of Activity-Based Costing. Davenport (1993) and Hammer and Champy (1993) coined the terms *process innovation* and *business (process) reengineering* (BPR). Processes also have a prominent place within the movement of continuous quality improvement. Six Sigma, for instance, has a measurement-driven methodology both to reduce variability in existing process designs and to create new process designs (Motorola Inc., 1986; Ehrlich, 2002).

Yet to date, organizations still experience difficulties in applying this process-oriented perspective to the design and maintenance information systems. Process-aware information systems (PAIS) (Dumas et al., 2005) provide automated support for the business processes of an organization by partially automating the (coordination) work. According to zur Muehlen (2004), the first process-aware information systems started to appear in the 1980s out of document management systems, e-mail systems and database management systems. Nonetheless, data management, not process management, demanded most of the attention in the 1980s. Real interest for automated business process support arose in the 1990s with the advent of new communication standards and new IT infrastructures. In an attempt to integrate disparate modules, ERP vendors started to adopt capabilities for

automated process support. At the same time a comparable effort was undertaken by Enterprise Application Integration (EAI) tool vendors, developing process-oriented mechanisms for application interoperability. In spite of these evolutions, process-aware information systems are sometimes far from the business requirements of efficiency, effectiveness, flexibility and compliance. For instance, information systems have at first sight made the business processes of organizations more compliant. By automating the coordination of work, organizations have better control over the actions of involved actors. Moreover, automated processes can help organizations in demonstrating business process compliance to all stakeholders. However, the downside is that automated business processes are often inflexible. In particular, automated processes have proved to be difficult to adapt at design time where each changed requirement triggers a lengthy development cycle in which it is impossible to identify and include all control and correction steps a priori (Heinl et al., 1999). Moreover, at run time, automated processes are often found too rigid to deal with the contingencies of real-life situations (Sadiq et al., 2005).

The root of the problem lies with a procedural representation of business processes that provides computer systems with inadequate information to deal with the idiosyncracies of every-day situations. In general, one can think of, among other, the following business concerns to play a governing role in the organization of work:

- **Business regulations**: externally imposed directives such as among others legal requirements, standards and contracts.
- **Business policies**: internally defined directives involving among others business strategies, tactics and operational procedures.
- **Costs and benefits**: the incurred benefits and costs of an activity.
- **Lead time**: the overall time to enact a process.
- **Information prerequisites**: the information required to enact a process.
- **Technical and common-sense constraints**

Organizations often only implicitly think about these business concerns when they design business processes but pay little attention to documenting why specific design choices have been made. Instead of making these business concerns explicit, they are implicitly used to determine task control flows, information flows and work allocation schemes. In other words these aspects remain implicit but their effects are – so to speak – **hard-coded** directly in procedural process models.

The counterpart of a procedural representation is a declarative one. A process model is *declarative* when it explicitly takes into account the business concerns that govern a business process leaving as much freedom as is permissible at execution time for determining a valid and suitable execution plan. Recently, a number of process modeling languages have appeared that could be identified as declarative languages. These modeling languages have very distinct knowledge representation backgrounds, often lack a formal execution model and often only model one aspect of the many business concerns that exist in reality. What is needed are meaningful ways to combine several kinds of expressions, called business rule types, independently of the used methods for knowledge representation and reasoning. In this paper, we show how the business concerns that govern business processes can be modeled declaratively.

The paper is structured as follows. In section 2 we introduce some languages for declarative process modeling that exist in the literature. In section 3 we contrast declarative and

procedural process modeling and provide an informal introduction to declarative process modeling in the next section 4. In section 5 we give a vocabulary for declarative process modeling, called the EM-BrA^{2}CE Vocabulary. This vocabulary is an extension of the SBVR and allows to declaratively refer to the state of a business process. In this section we also identify a number of types of business rule that are formulated in this vocabulary. Several of these business rule types are underpinned by a form of temporal logic. In section 6 an execution model is provided by modeling the dynamics of an activity life cycle through the use of Colored Petri Nets. In section 7 we show how declarative process models can contribute to the model-driven design of Service-Oriented Architectures. Finally, in section 8, we give a brief evaluation of the proposed framework.

2 Related work

In the literature, languages such as the case handling paradigm (van der Aalst et al., 2005), OWL-S (The OWL Services Coalition, 2006), the constraint specification framework of Sadiq et al. (Sadiq et al., 2005), the Web Service Modeling Ontology (WSMO) (Roman et al., 2005), the ConDec language Pesic and van der Aalst (2006) and the PENELOPE language (Goedertier and Vanthienen, 2006b) can be categorized as declarative languages.

None of these languages for declarative process modeling are expressive enough to cover the many real-life business concerns that exist in reality. For instance, the ConDec language and the PENELOPE language only allow to express business rules about sequence and timing constraints, i.e. the control flow aspects (Jablonski and Bussler, 1996). Web Service Orchestration standards such as OWL-S (The OWL Services Coalition, 2006) and WSMO (Roman et al., 2005), on the other hand, include the organizational and data model aspects, but do not provide a temporal logic to express temporal relationships between concepts such as activities or events.

Moreover, these languages make use of very different knowledge representation paradigms. For instance, the ConDec language is expressed in Linear Temporal Logic (LTL) whereas the PENELOPE language is expressed in terms of the Event Calculus. These heterogenous knowledge representation paradigms raise the question how it will be possible to reason about such heterogeneously expressed knowledge.

Finally, these languages do not have an explicit execution model or have an execution model that explicitly assumes either human or machine-mediated service enactment. The WSMO, for instance, has a specific execution model (WSMX) (M. Zaremba, 2005) that is focused on Web service mediated service orchestration. The case handling paradigm, for instance, assumes humans to perform atomic tasks but has an orchestration engine to perform the orchestration (coordination) work.

A common idea of declarative business process modeling is that a process is seen as a *trajectory* in a *state space* and that declarative constraints are used to define the valid movements in that state space (Bider et al., 2000). The differences between declarative process languages can in part be brought back to a different perception of state.

- The state space of the case-handling paradigm (van der Aalst et al., 2005) comprises the state of business concepts and activities. Although there is still a *preferred* or *normal* control-flow defined between the activities, the user has the freedom of choice to execute, skip and redo activities within a number of constraints based on availability of case data and the executed activities.

- In the constraint specification framework of Sadiq et al. (2005), order and inclusion constraints can be specified in a state space composed of activities.

- The state space of the ConDec language (Pesic and van der Aalst, 2006) consists exclusively of activities. Using Linear Temporal Logic the authors construct a template language that declaratively describes the temporal relationships between activity types.
- Another kind of constraint are the so-called temporal deontic assignment rules of the PENELOPE language (Goedertier and Vanthienen, 2006b). Here the state space consists of the temporal obligations and permissions that rest upon the agents involved in a business process.

Sadiq et al. (2005) show how it can be advantageous to combine both declarative and procedural aspects in process models. The authors present a foundation set of constraints for *partial* process modeling. A process model can contain, in addition to pre-defined activities and control flow, several so-called *pockets of flexibility*. Such pockets consist of activities, sub-processes and so-called order and inclusion constraints. Each time during enactment when a pocket of flexibility is encountered, the elicitation of the work within the pocket is done by a human end-user through a so-called "build" activity. Although such a combined approach has advantages, it is not considered in the research proposal that focusses exclusively on declarative process modeling.

The idea of declarative business process modeling is related to the business rules approach (Ross, 2003; Kardasis and Loucopoulos, 2005). By documenting and formalizing business rules, it is hoped that changes to these rules will no longer result in an avalanche of required information system updates and will thus reduce the IT bottleneck when bringing about business changes. Consequently there is a vivid interest among practitioners (Debevoise, 2005) and commercial software vendors in the confluence of business rules and business process modeling. ILOG, for example offers ILOG JRules integration solutions for several existing BPM products and Microsoft has added business rules functionality to BizTalk. In general, commercial approaches integrate production rule specification with imperative business process modeling. The integration is realized by including explicit calls to a rule engine in the business process model. In BizTalk, for example, such a call is represented as a so-called *decision shape*. On the basis of the information that a process gets back from the rule engine, the process is carried further. Although this approach allows for specifying to some extent the logic of control flow, data flow and task allocation by means of rules, these approaches do not belong to the category of declarative process modeling languages defined in the next section.

3 Procedural versus Declarative Process Modeling

A business process model is called **procedural** when it contains explicit, prescriptive information about how processes should proceed, but only implicitly keeps track of why these design choices have been made. When modeling business processes procedurally, modelers inevitably make a number of modeling assumptions that are not present in the earlier specified requirements, this is called the **assumption bias** of a model. Therefore procedural models inherently risk to be **over-specified** as they are likely to impose more restrictions on the control flow, information flow and work allocation in business process models than is strictly required. Procedural process models are modeled with **procedural languages** such as WorkflowNets (van der Aalst, 1997), the Business Process Execution Language (BPEL) (Andrews and et al., 2003), the Business Process Modeling Notation (BPMN) (Object Management Group, 2006a) and UML Activity Diagrams (Object Management Group, 2005).

	Procedural modeling	Declarative modeling
Business concerns	implicit	explicit
Rule enforcement	what, when and how	what
Communication	what, how	what
Execution scenario	design-time	run-time
Execution mechanism	state-driven	goal-driven
Model granularity	process-centric	activity-centric
Modality	what *must*	what *must*, *ought* and *can*
Assumption bias	over-specified	under-specified
Alteration	design time	design and run time
Coordinator/Worker	human-machine	agent
Coordination/Activity	coordination $\neq$ activity	coordination = activity
Activity life cycle	single event	multiple life cycle events
Language	procedural	declarative

Table 1: *Procedural versus declarative process modeling*

The counterpart of a procedural process model is a declarative one. Process modeling is said to have a **declarative** nature, when it explicitly takes into account information about the benefits, costs, time characteristics, constraints and business goals of a process, leaving as much freedom as is permissible at execution time for determining a valid and suitable execution scenario. Declarative process modeling does not merely focus on *how* an end state *must* be reached, but rather considers what *is*, *must*, *ought* and *can* be done in order to achieve the business goals. Declarative process models are modeled with **declarative languages**. The case handling paradigm (van der Aalst et al., 2005), the constraint specification framework of Sadiq et al. (2005), the ConDec language (Pesic and van der Aalst, 2006) and the PENELOPE language (Goedertier and Vanthienen, 2006b) can be categorized as such declarative languages. Table 1 summarizes the differences between procedural and declarative process modeling. These differences are discussed in the subsequent paragraphs.

3.1 Business Concerns Made Explicit

Declarative process modeling makes the underlying **business concerns** explicit in the form of business vocabulary and business rules. **Business rules** are atomic, formal expressions of business policies, business regulations and common-sense constraints. Business rules make business concerns explicit and traceable. Business rules can be seen as a common language between the business-side and IT-side of organizations. Such a language allows the business-side to formally represent models of how it operates internally and how it can legally interact with business partners. At the same time, such a common language allows the IT-side to have Information Systems support business processes accordingly, with as little development effort as possible. Ideally, Information System Technology must support declarative business process models in such a way that they become human-understandable, yet machine-executable specifications. In this way, changes to policies and regulations can be traced back to the business processes were they are to be enforced.

3.2 Declarative Business Rule Enforcement

Procedural process languages predominantly focus on the control-flow perspective of business processes. In such process languages it might be possible to **enforce business rules** using a control-flow-based modeling construct. For instance, the enforcement of a derivation or integrity constraint can be directly modeled in BPEL as a calculation or input validation step. The left-hand side of Figure 1 represents an excerpt from a BPMN model that models the enforcement of a discount rule as a decision shape in BPMN. Another example involves the enforcement of authorization rules. The left-hand side of Figure 2 models an authorization rule as a decision shape in BPMN. The disadvantage of procedural process modeling is that business rules cannot be formulated independently from the process models in which they are to be enforced. Consequently, the same business rule is often duplicated in several procedural process models. When the business rule changes it is likely that all process models must be reexamined. Declarative process modeling separates business rule modeling from business rule enforcement. In particular, it does not make use of control flow to indicate when and how business rules are to be enforced. Instead, it is left to the execution semantics of the declarative process models to define an execution model in which different kinds of business rules are automatically enforced. The latter is indicated in the right-hand side of Figures 1 and 2. This separation of business rule specification and enforcement facilitates design-time flexibility.

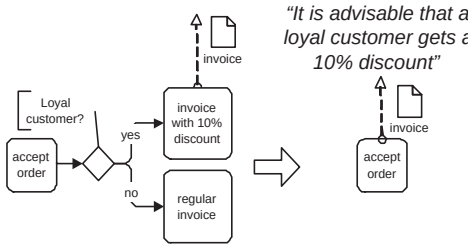


Figure 1: *Example: separating logic from enforcement (Goedertier and Vanthienen, 2006a)*

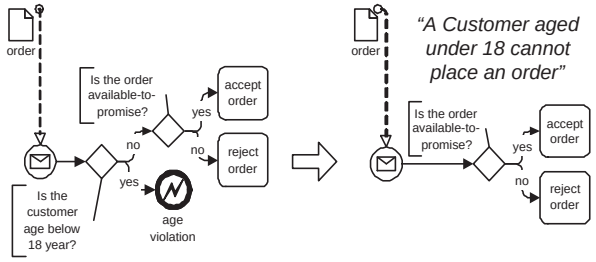


Figure 2: *Example: separating logic from enforcement (Goedertier and Vanthienen, 2006a)*

3.3 Declarative Communication Logic

Procedural process models are overburdened with **communication activities** intended to notify an external business partner about the occurrence of a relevant business event or to transmit information. Figure 3 represents an excerpt from the BPMN specification (Object Management Group, 2006a, p. 107) that contains the communication activities 'receive order' and 'send invoice'. Such communication activities depict communication logic in a procedural manner, because they specify how and when business events are communicated

and information is transmitted. Declarative process models are only concerned with the ability of business agents to perceive business events and business concepts. When an agent (for instance a business partner) can perceive a particular event, the event becomes non-repudiable to the agent, irrespective of how the agent is notified of the event. The execution semantics of a declarative process model determines how events are communicated. In particular, events can be communicated as messages that are sent by the producer (push model), retrieved by the consumer (pull model) or via a publish-subscribe mechanism.

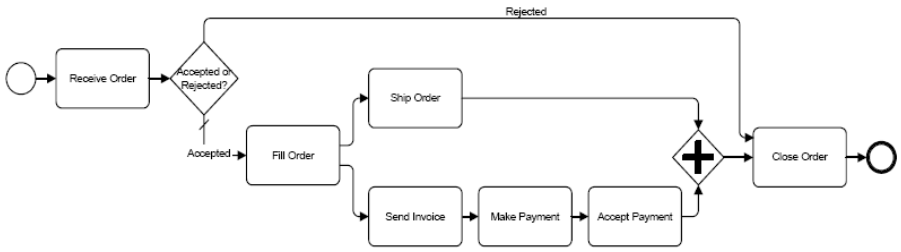


Figure 3: *Example: separating communication logic from process models (Object Management Group, 2006a, p. 107)*

3.4 Dynamic Execution Scenarios

Unlike procedural process modeling, declarative process modeling does not involve the pre-computation of task control flows, information flows and work allocation schemes. Whereas procedural process models explicitly enumerate all possible **execution scenarios**, these scenarios remain implicit in declarative process models. An explicit enumeration of all different execution scenarios is often not required – and often even difficult to obtain (Heinl et al., 1999) – for modeling purposes. However, for model checking (verification) purposes, execution trajectories can still be obtained from implicit process models. During the execution of a declarative process model, a suitable execution scenario is constructed (either by a human or machine coordinator) that realizes the business goals of the process model. The latter is called **goal-driven** execution and its automation is akin to planning in the domain of Artificial Intelligence (Fikes, 1971). In contrast, the execution mechanism of procedural process modeling languages is called **state-driven**.

3.5 Activity-level Granularity

Declarative process models also have a more fine-grained **model granularity** than procedural process models. Whereas procedural process languages are **process-centric** in that they model business processes, declarative process languages are **activity-centric**, as they model the business concerns related to a set of activity types. Business process models are composed of activity types, but the same activity type can occur in multiple business process models. In addition, many business concerns range over activity types and are not specific to one business process model in particular. Therefore activity-centric models have the advantage that these governing aspects are not a-priori straitjacketed into a particular business process model. For instance, the regulation that a purchase order must never be paid prior to the reception of an invoice, can possibly be relevant in different business processes. To allow the reuse of this regulation, it must be specified across the boundaries of artificially delineated business process models. Although the process-oriented view on organizations has lead to a better understanding of the value chain (Porter, 1985; Davenport, 1993) and has improved business process redesign, there is little motivation in

letting this process-centricity set the granularity for process modeling. When required, a process-centric model can be obtained from an activity-centric model, the converse is not generally true.

3.6 Model Differentiation by Modality

Another point of differences is the **modality** that is attached to the information in process models. Procedural process models inherently have the necessity modality (what *must*) attached, whereas procedural process languages allow to differentiate by attaching different modalities like intention (what *ought*), advice (what *should*), possibility (what *can*) and certainty (what *is*) to parts of the process model. These modalities offer run-time flexibility. In particular, they allow to distinguish between what is strictly required (hard constraint) and what is merely desirable (soft constraint) behavior in a business process. This can help the coordinator of a business process to come up with a suitable yet valid execution plan.

The idea of different modalities is related to the research of Suchman (1995), Schmidt and Simone (1996) and Ross (2003). Suchman (1995) points out that business process models can never fully represent cooperative work in all its facets. In any organization, representations of work are required to create a common understanding of the work and thus facilitate coordination. However, workers may and should have conflicting views on the work. Suchman warns that a normative, *prescriptive account* of how the work gets done might severely differ from specific working practices. Although representations of work are a useful tool to reason about work and to steer activities, they risk to become useless when used outside the context of the work. According to the seminal work of Suchman (1987) representations of work need to be under-specified such that they are *plans for situated action*, in which the worker uses a plan as a guideline to go about but also determines the most suitable activity to undertake by himself from the context of the process *in situ*. To emphasize her point Suchman uses the metaphor of a map. "*Just as it would seem absurd to claim that a map in some strong sense controlled the travelers movements through the world, it is wrong to imagine plans as controlling actions. On the other hand, the question of how a map is produced for specific purposes, how in any actual instance it is interpreted vis-à-vis the world, and how its use is a resource for traversing the world, is a reasonable and productive one.*"

The situated action perspective on process models implies that there is little benefit in the automation of work coordination. However, this implication is not in congruence with the empirical evidence of many successful BPM automations found in contemporary organizations. Schmidt and Simone (1996) and Schmidt (1999) distinguish between two, according to them equally possible, accounts of process models by contrasting the metaphor of process models as *scripts* with Suchman's metaphor (1987) of process models as *maps*. A business process model can play the role of a *script* when it contains explicit, prescriptive information about how processes should proceed by making pre-computations of task dependencies. "*A script offers a limited selection of safe, secure, legal, valid, advisable, efficient or otherwise prescribed moves while excluding moves that generally would be considered unsafe, etc*" The application of a script can relieve the worker of computing "*a myriad of task interdependencies*" and optimization concerns. Conversely, a business process model can play the role of a *map* when it contains a codified set of functional requirements that provide a heuristic framework for distributed decision making. It is important that in the vision of Schmidt and Simone a same process model can be either a script or a map, depending on whether the context of the process conforms to relevant, pre-defined task interdependencies or not.

A more refined view of Schmidt and Simone's dichotomy is to acknowledge that a process

model can be used as a script and a map *within the same execution context*. Interestingly this combined view corresponds to the relationship between business process models and business rules depicted by (Ross, 2003). In Ross' view, process models consist of a set of pre-computed task dependencies, effectively called scripts, that can be supplemented with business rules that are either a number of **strict rules** that must be observed at all times or a number of heuristic **guidelines** that can just as easily be discarded. In terms of Suchman's traveler metaphor this would suggest the existence of a series of predefined sub-trajectories, scripts, that control a traveler's movements and a number of strict rules and guidelines that give direction to a travelers's movements, but leave the traveler with some freedom in choosing his or her own destination and trajectory.

3.7 Assumption bias

The business rules in declarative process models can be traced back to an original requirement, a regulation, a policy or a common sense constraint. Consequently, declarative process models are likely to only contain a minimum of constraints regarding a particular business process. This is not generally the case for procedural process models. Because procedural process models are the result of an implicit pre-computation of task dependencies, it is not generally guaranteed that procedural process models do not include a number of additional assumptions that **overly specify** the underlying business process.

3.8 Runtime Alteration

The declarative information about a business process allows a coordinator to reason about the effect of run-time **alteration** of the execution plan. Such adaption can be seen as deviating from the outlined soft constraints to better fit the idiosyncracies and contingencies of real-life situations. Procedural process models do not allow this form of reasoning. In procedural process models all control flows, information flows and work allocation policies have been pre-computed. Without information about the strict business rules (hard constraints) and guidelines (soft constraints) that have led to a particular process model, it is difficult to reason about the effect of a run-time alteration. For instance, Reichert and Dadam (1998) describe the rationale of the ADEPT($_{\text{flex}}$) workflow management system (WfMS) in which end-users can change the process model of the process instance at runtime. Although ADEPT($_{\text{flex}}$) provides extensive user support to prevent non-permissible structural changes, it provides little support in defining and distinguishing between non-permissible and non-advisable business changes. In particular ADEPT($_{\text{flex}}$) preserves control flow and data flow consistency regarding the addition, deletion and movement of tasks, but provides little support in identifying the business constraints and guidelines that restrict the modification of business process instances.

3.9 No Human-Machine Distinction

Information systems and machinery have lead to an extensive automation of both work and coordination work. But not all activities in every business process can be fully automated. Likewise, not every business process lends itself to the same degree of automated coordination. In many cases, some of the (coordination) work is performed by machines and some of it by humans. Ideally, declarative process models make abstraction from the differences between humans and machines in performing (coordination) work. Rather than making an ontological distinction between concepts like humans and machines, both concepts are unified through the use of the **agent** metaphor (Woolridge and Wooldridge, 2001). Agents can be entire organizations, organizational units or individual workers and machines. In

many cases, individual agents – whether humans, machines or a combination of both – act on behalf of the organization to which they pertain.

3.10 Coordination Work is Work

Business process management is about the **coordination of work** (Schmidt and Simone, 1996). Procedural process models are often an explicit specification of the coordination work. In contrast, declarative process models make no difference between coordination work and regular work. What may appear as work to an external agent, may very well be coordination work to another agent. For instance, a sales representative may instruct the expedition department to ship an order by a particular due date, but this activity may conceal the coordination of many other activities within the expedition department.

3.11 Multi-state Activities

Procedural process models do not explicitly consider the **life cycle** of the activities within a business process. The performance of an activity then most generally corresponds to its start or completion. Declarative process models, in contrast, consider other events in the life cycle of activities such as the creation, scheduling, assignment, start, fact manipulation, completion, skipping, cancelation and redoing of an activity.

3.12 Third-person perspective

The growing popularity of the Internet based on new IP-based communication protocols and technologies such as XML, has given rise to the requirement of automated coordination of business processes across the boundaries of individual organizations. As a consequence, it is not always technically or economically viable to have processes coordinated centrally. Another consequence of distribution is that it is unlikely that process designers can come up with only one representation of work. In many cases all business partners that participate in a cooperation might have different representations of the cooperative work. These representations are to be kept in part private from other process business partners. In conclusion a BPMS must try to reconciliate the disparate, public representations of the cooperative work that exist among business partners in an inter-organizational business process.

Business process management must reconcile disparate views of work. Business processes may include the concerns of many stake holders. When modeling behavior it is proposed to adopt a third-person perspective – what will an actor with a particular role do in response to what others do? – rather than a first-person perspective – what will I do in response to what others do? In a third-person perspective all roles, actors and organization structures are named without the modeler adopting a particular viewpoint. A third-person modeling perspective has the advantage that it is possible to distinguish multiple interacting actors within a single organization. Another advantage is that business rules can be more easily shared in a business community when they are expressed from a third-person perspective. This does not imply that all behavior and business facts are visible to all actors in a system.

3.13 Meaning is Separate from Expression

Procedural process models make no distinction between meaning and expression. In reality, the same meaning can be expressed using many different notations and expressions.

Declarative process modeling provides a general vocabulary to discuss the meaning of different kinds process expressions. For instance, it is possible to express a process model in part using the BPMN. However, as not all aspects of business process models can be modeled using the BPMN (Wohed et al., 2006), a BPMN diagram is unlikely to be fully interchangeable with a declarative process model.

4 An Introduction to Declarative Process Modeling

Each business process can be **modeled** by describing its state space and the set of business rules that constrain the possible transitions in this state space. The possible movements within a business process' state space, can be described by twelve generic activity state transitions: *create*, *schedule*, *assign*, *revoke*, *start*, *addFact*, *removeFact*, *updateFact*, *complete*, *abort*, *skip* and *redo*. Because they are generic, these twelve activity state transitions provide a means of defining an **execution model** that is described in section 6. Informally, it suffices to check prior to the occurrence of a state transition of a particular type whether a number of particular business rules will be violated or not. When no business rule is violated, the state transition can take place. When, on the other hand, the transition would lead to an intolerable violation of a business rule, the state transition is prevented from taking place.

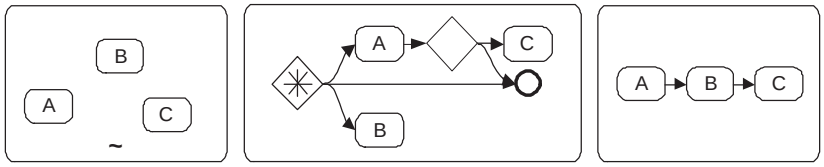
4.1 Process Model = State Space + Transition Constraints

Consider, for example, a process model called ABC that expresses that for each process instance the activities of type A,B and C can be performed at most once. Additionally, the constraint is imposed that activity C may only be started if activity A has already completed. In terms of the above defined terminology this process can be modeled as:

- **state space**: the state space of ABC is described by facts about
  - **composite activity types**: ABC
  - **atomic activity types**: A, B, C
  - **activity event types**: *created*, *assigned*, *started*, *completed*
- **rule**: "There is exactly one A activity that has parent an ABC activity"
- **rule**: "There is exactly one B activity that has parent an ABC activity."
- **rule**: "There is exactly one C activity that has parent an ABC activity."
- **rule**: "A C activity can only start after an A activity has completed."

The four business rules express a necessary abstract state (hard constraints). An execution model for these business rules can be defined in terms of the activity state transitions that are constrained by these rules. For instance, the first business rule must be enforced when the coordinator of an ABC activities creates a new A activity, when the execution plan already contains an A activity. The last business rule can be operationalized as a precondition on the *start* of an activity of type C, requiring the completion of an activity A.

It is instructive to compare the declarative process model ABC of the previous paragraph with its procedural counterpart. Figure 4 depicts three BPMN process models that to some extent model the intended business process. Figure 4 a) attempts to model the described process by means of an ad-hoc sub-process. Notice however that in this BPMN



(a) under specified (b) correct (c) over specified

Figure 4: *Expressing the ABC process model in BPMN*

solution it is not possible to express the activity order constraint that C requires A nor to express that each task can be performed only once. Figure 4 b) depicts the described process using a complex gateway, for which the OutgoingCondition (Object Management Group, 2006a, p. 261) must specify that either A, A and B or no activities can follow. Although this model portrays a correct semantics it is somewhat overburdened by decision shapes. In practice, modelers give away freedom of choice in exchange for a more simple process model. For instance, Figure 4 c) depicts a process model that adheres to the described semantics but that is overly restrictive with respect to the ordering of the activity B.

4.2 History-dependent behavior

Sometimes the behavior of a process instance is dependent on its own history. For instance, a worker is refused authorization to perform a certain task, when he or she has performed a related task in the past. Another example is the occurrence of history-based joins in the control flow of a business process (van der Aalst and ter Hofstede, 2002; van Hee et al., 2006b). This *non-local* behavior of business processes presents many challenges for process modeling (van Hee et al., 2006a). Consider as an example the process model ABCD in which each activity A,B, C and D can be performed at most once for each process instance. Moreover, the tasks A and C are mutually exclusive and D can only start when either A or C has been performed.

- **state space**:
  - **composite activity types**: $ABCD$
  - **atomic activity types**: A, B, C, D
  - **activity event types**: *created*, *assigned*, *started*, *completed*
- **rule**: "There is exactly one A activity that has parent an $ABCD$ activity"
- **rule**: "There is exactly one B activity that has parent an $ABCD$ activity."
- **rule**: "There is exactly one C activity that has parent an $ABCD$ activity."
- **rule**: "There is exactly one D activity that has parent an $ABCD$ activity."
- **rule**: "A and C activities are mutually exclusive."
- **rule**: "An activity D can only start after either an A activity or a C activity has completed."
- **rule**: "If an activity B has started, activity D can only start after the completion of B."

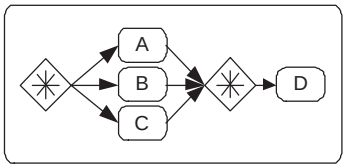


Figure 5: *History-based join*

The last rule makes the process model $ABCD$ history dependent. In particular, the start of an activity D depends on whether or not an activity B has been started in the history of a process instance. Because of the local nature of process languages like Petri nets or the BPMN this history-dependent join is difficult to model. The BPMN model, depicted in Figure 5, contains two complex gateways to model $ABCD$. Notice however that this visual representation does fully represent the intended semantics. In particular, due to the local nature of a BPMN model it cannot check whether an activity B has started or not. Consequently, this cannot be represented by an IncomingCondition on the second complex gateway. Even in this simple example the semantics of the history-based join can only be specified in BPMN using textual annotations.

4.3 Running example: payment-after-shipment

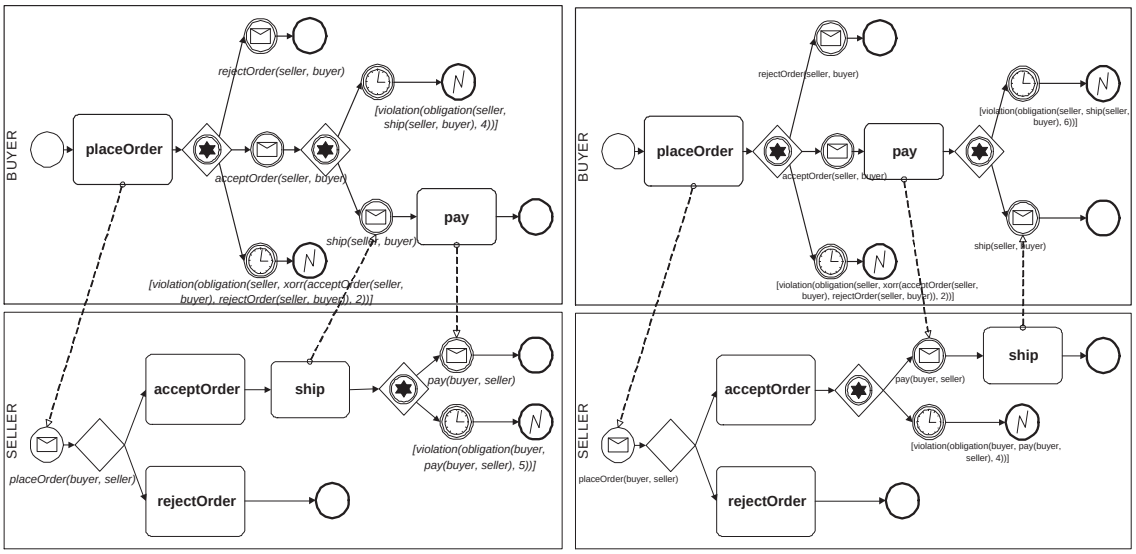
The above mentioned process models ABC and $ABCD$ are only concerned with the control-flow aspects of process modeling and make abstraction from data and organizational modeling aspects. To further illustrate declarative process modeling, an order-to-cash business process will be used as a running example throughout the text. The example has two versions: a payment-after-shipment and a shipment-after-payment version, both depicted using the Business Process Modeling Notation (BPMN) Object Management Group (2006a) in Figure 6. The payment-after-shipment process can be declaratively modeled as follows:

- **state space**: the state space of the order-to-cash process is described by facts about
  - **roles**: buyer and seller.
  - **composite activity types**: coordinate purchase order, coordinate sales order.
  - **atomic activity types**: Coordinate purchase order *can consist of* place order and pay activities. Coordinate sales order *can consist of* accept order, reject order, and ship activities.
  - **activity event types**: created, assigned, started, completed
  - **event types**: timeout, obligation violated
  - **business concepts**: order, order line
  - **business fact types**: order *has* order line, order *is critical*, order *has* due date, order *has* discount, order *has* customer, customer *is* loyal customer, customer *is* corporate customer,... Place order can manipulate the business fact types 'order has order line' and 'order has due date',...
- **rule**: "Initially a buyer has the permission to perform a place order activity."
- **rule**: "When a buyer completes a place order activity, the seller has the obligation to perform a accept order activity or a reject order activity within 2 time units."

- **rule**: "When the buyer completes a place order activity, the buyer has the obligation to perform a pay activity within 2 time units after the seller completes the ship activity."
- **rule**: "When the seller completes a accept order activity, the seller has the obligation to perform a ship activity within 2 time units."
- **rule**: " There exists exactly one place order activity that has parent a handle purchase order activity."
- **rule**: "There exists at most one accept order activity that has parent a handle sales order activity."
- **rule**: "Activities that have type place order, accept order, reject order and ship order must not be performed in parallel."
- **rule**: "Accept order and reject order activities are mutually exclusive."
- **rule**: "Accept order and ship activities are mutually inclusive."
- **rule**: "After the start of a ship activity, the order lines of the order can no longer be changed."
- **rule**: "Each order has at least one order line."
- **rule**: "The agreed price of a sales item is less or equal to the standard price of the sales item."
- **rule**: "A luxury product has a value-added-tax of 20 percent."
- **rule**: "An order has a 10 percent discount if the order is from a loyal customer."
- **rule**: "An agent that has age less than 18 years can not perform a place order activity."
- **rule**: "An agent that has function junior sales representative can not perform an accept order or reject order activity that is identified by an order that has an amount larger than 2000 euro."
- **rule**: "Coordinate purchase order can make visible the business fact type 'order *has* rejection notice." "It is necessary that a rejection notice is only visible to an agent that is a corporate customer."
- **rule**: "A buyer can subscribe to completed in the context of ship." "It is not possible that an agent that has role buyer perceives an event that is about a ship activity for an order that has a total amount of less than 2000 euro."

5 A Vocabulary for Declarative Process Modeling

In this section the vocabulary (or the metamodel) of the EM-BrA^{2}CE Framework is described that contains the fundamental building blocks for modeling business processes. EM-BrA^{2}CE stands for 'Enterprise Modeling using Business Rules, Agents, Activities, Concepts and Events'. Figure 7 depicts the relationship between the main building blocks of the framework.



(a) payment-after-shipment (b) shipment-after-payment

Figure 6: *Two order-to-cash processes*

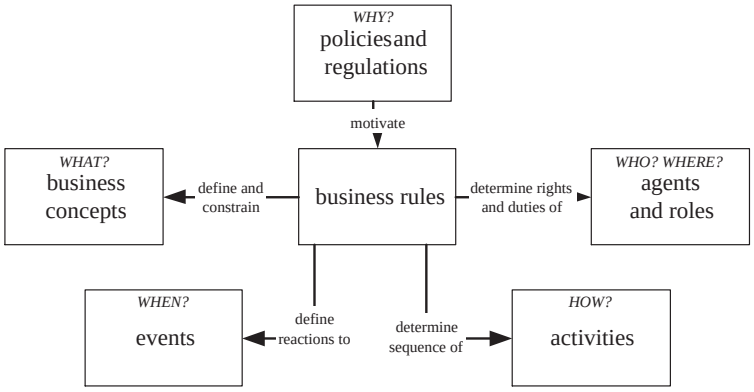


Figure 7: *The EM-BrA*^{2} *CE Framework*

This section is structured as follows. First it is motivated why the SBVR was chosen as ontology language. Subsequently, an introduction to the SBVR is provided. In the next section the EM-BrA^{2}CE Vocabulary is defined that extends the SBVR. In a subsequent section, some candidate logics and rule languages are briefly introduced that can be used to express temporal aspects of business processes. Finally, a classification is given of a number of candidate business rule types that are to be used for declarative process modeling.

5.1 Candidate Ontology Language

There exist several candidate ontology languages (or metamodeling languages) that can be used to define a metamodel for declarative process modeling. Among the most prominent candidates are the tandem Meta Object Facility(MOF) / Unified Modeling Language (UML) (Object Management Group, 2006c), the Web Ontology Language (OWL) and the Web Service Modeling Language (WSML) (Roman et al., 2005). Nonetheless it was chosen to model the metamodel in terms of the vocabularies provided by the Semantics of Business Rules and Vocabulary (SBVR) language (Chapin, 2005; Object Management Group, 2006b). The SBVR was chosen because it possesses many desired properties:

- **Model granularity.** Information models can use different levels of granularity to represent concepts in the world. In the last decade two paradigms have emerged: object-level and fact-level granularity. Fact-orientation perceives the world in terms of facts rather than in terms of objects, attributes and relationships. Fact types have a finer granularity compared to object types. This facilitates the expression of business rules (Halpin, 2000) and postpones implementation decisions about grouping attribute and relationship types into object types (Leung and Nijssen, 1988; Halpin, 1991).

- **Local Closure.** In knowledge representation one has to deal with incomplete knowledge of the world. In SBVR it is possible to indicate the predicates (fact types) over which the model has complete knowledge. Such a construct is called *local closure* and it is possible to indicate local closure in SBVR. In general, two assumptions are possible: an open-world and a closed-word assumption. Under an open-world assumption (OWA) it is accepted that a model incompletely represents the world. Under a closed-world assumption (CWA) it is assumed that the model completely represents the world. There is a difference between both assumptions when reasoning with negation (Wagner, 1991).

- **High-Order Classification.** In many conceptual information models it is most natural to be able to have instances of types that are types themselves (Halpin, 2004). This paradigm is known as higher-order typing. In UML, higher-order typing can be obtained using the UML stereotype mechanism (Atkinson and Kühne, 2003; Object Management Group, 2006c). In logic higher-order typing pertains to Higher-Order Logic. When restricted to Henkin semantics there exist a proof logic for higher-order logic that is sound and complete (Henkin, 1950). The SBVR has been given such a semantics.

- **Business Rules as natural language expressions.** Business rules are most often expressed in language. Consequently, the SBVR combines linguistics and formal logic. In particular, it has a vocabulary to express the meaning of (natural) language expressions in terms of formal logic. These are the fundamental building blocks for developing a natural language parser that allows to express the meaning of rules that have a textual notation. To date, two SBVR natural language parsers have been

developed: the Unisys Rules Modeler (Baisley, 2005; Unisys, 2005) and SBeaVeR (Digital Business Ecosystem (DBE), 2007).

- **Rule modality.** One of the characteristics of declarative process models is that they make a distinction between business rules that cannot be violated, that can be violated and guidelines. The current SBVR specification requires business rules to be either a necessity, an obligation, a prohibition or a possibility.

- **Reification.** The SBVR allows propositions to be treated as concepts in their own right. As such, propositions can be made about other propositions. This is called 'objectification' is the standard. Objectification is, for instance, useful to represent that a particular business fact has been asserted or retracted in the context of a given activity.

The SBVR has many features that make it an attractive languages for declarative process modeling. Nonetheless the SBVR does also have its shortcomings. For instance the SBVR does not incorporate any form of temporal logic. In a dynamic world of business processes, such knowledge representation and reasoning mechanisms are required to reason about properties qualified in terms of time or the effect of activities on the state of the world. Furthermore, the SBVR lacks the semantics to model business rules in terms of a number of general rules and exceptions. Such a means for representing and reasoning with default knowledge is, for instance, provided by defeasible logic Nute (1994); Antoniou et al. (2001). This way of knowledge representation is valuable, because it facilitates the incremental specification of business rules Grosof et al. (1999): new rules can be added without the conditions of previous rules need to be reconsidered. Ordinary rules, in contrast, require a complete, encyclopedic knowledge of all rules to be updated or decided upon.

5.2 An Introduction to SBVR

The Semantics of Business Vocabulary and Business Rules (SBVR) provides a number of conceptual vocabularies for modeling a business domain in the form of a vocabulary and a set of rules. As the EM-BrA^{2}CE vocabulary extends the fundamental vocabularies of the SBVR, these vocabularies will be discussed in the remainder of this section.

In SBVR, meaning is kept separate from expression. As a consequence, the same meaning can be expressed in different ways. In real-life, meaning is more often expressed in textual form than in diagrams as statements provide more flexibility in defining vocabulary and expressing rules. For these reasons, the SBVR specification defines a structured, English vocabulary for describing vocabularies and verbalizing rules, called SBVR Structured English (Object Management Group, 2006b, p. 133). One of the techniques used by SBVR structured English are font styles to designate statements with formal meaning. In particular,

- the term font is used to designate a noun concept.
- the name font designates an individual concept.
- the *verb* font is used for designation for a verb concept.
- the keyword font is used for linguistic particles that are used to construct statements.

The definitions and examples in the remainder of the text use these SBVR Structured English font styles.

In SBVR a vocabulary and a set of rules make up a so called conceptual schema. A conceptual schema with an additional set of facts that adheres to the schema is called a conceptual model. Figure 8 depicts the relationship of a conceptual schema and a conceptual model to some of the core building blocks in SBVR. These core building blocks are part of the SBVR Meaning and Representation Vocabulary. This vocabulary contains among others the following definitions (Object Management Group, 2006b, p. 13):

> A conceptual schema *is* a combination of concepts and facts (with semantic formulations that define them) of what is possible, necessary, permissible, and obligatory in each possible world.
> A conceptual model or fact model *is* a combination of a conceptual schema and, for one possible world, a set of facts (defined by semantic formulations using only the concepts of the conceptual schema).

The facts in a conceptual model may cover any period of time. Changing the facts in a conceptual model creates a new and different conceptual model. In this way the SBVR gives conceptual models a monotonic semantics.

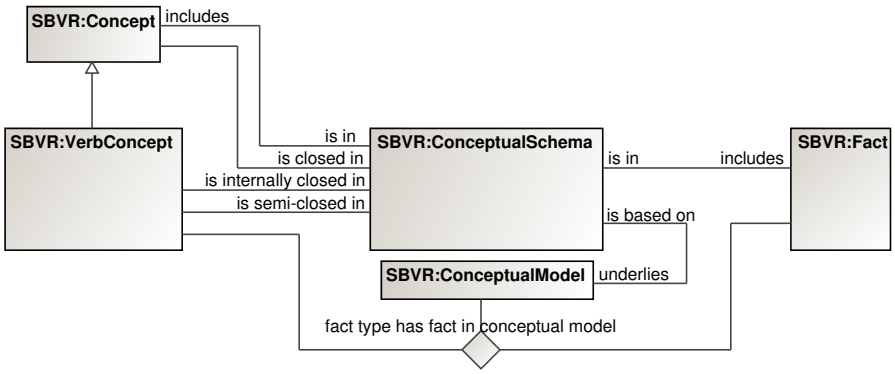


Figure 8: *A MOF/UML representation of SBVR conceptual schema and model (Object Management Group, 2006b)*

Informally speaking, the nouns and verbs that occur within a particular vocabulary can be related to noun concepts (or object types) and verb concepts (or fact types). In natural language, the grammar of a basic sentence can be seen as a subject-verb-object triple. Just as verbs can have the roles of subject and object in a sentence, verb concepts can have roles that refer to noun concepts playing a part, assuming a function or being used in some situation. In the SBVR Meaning and Representation Vocabulary, depicted in Figure 9, these concepts are formally defined as follows.

> A meaning represents what is meant by a word, sign, statement, or description; what someone intends to express or what someone understands.
> A concept *is* a meaning that represents a unit of knowledge created by a unique combination of characteristics.
> A verb concept or fact type *is* a concept whose instances are all actualities and that is a basis for atomic formulation, having at least one role. Concept type: concept type.
> A noun concept *is* a concept that is not a verb concept. Concept type: concept type.
> An individual concept *is* a concept that corresponds to only one thing. General concept: noun concept. Concept type: concept type.

A role *is* a noun concept that corresponds to things based on their playing a part, assuming a function or being used in some situation. Necessity: each role *is of* at most one fact type. 'Verb concept *has* role' *is* an abstraction of a thing playing a part in instances of the fact type. Concept type: concept type.
A concept type *is* a noun concept that *specializes* the concept 'concept'.
'$Concept_1$ *specializes* $concept_2$' the $concept_1$ incorporates each characteristic incorporated into the $concept_2$ plus at least one differentiator. This represents the specialization-generalization relationship.
An SBVR:proposition *is* a meaning that is asserted when a sentence is uttered or inscribed and which is true or false.
An SBVR:fact *is* a proposition that is taken as true.

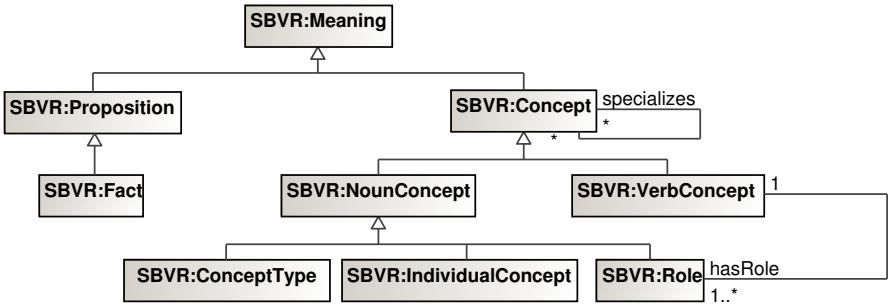


Figure 9: *A MOF/UML representation of the SBVR Meaning and Representation Vocabulary*

Although concepts have a particular meaning they by themselves do not constitute any statement about what is true, possible, necessary, permissible, and obligatory in a possible world. Such statements can be expressed by means of rules. The SBVR Vocabulary for Describing Business Rules, depicted in Figure 10, contains among others the following abbreviated definitions.

A business policy *is* a directive that *is* not *actionable* whose purpose is to guide an enterprise.
A rule *is* an *actionable* directive that introduces an obligation or a necessity.
A business rule *is* a rule that is under business jurisdiction. 'business rule *is derived from* business policy' represents the business policy from which a business rule originates.
A structural (business) rule *is* a (business) rule that is intended as a definitional criterion. A structural rule expresses a necessity that cannot be violated.
An operative business rule *is* a business rule that is intended to produce an appropriate or designed effect. An operative business rule expresses an obligation that can be violated.
A level of enforcement is something that represents a position in a graded or ordered scale of values that specifies the severity of action imposed in order to put or keep an operative business rule in force. 'operative business rule *has* level of enforcement' the level of enforcement that a particular operative business rules has.

The SBVR defines a business rule as a rule under business jurisdiction that is derived from a business policy. This definition can be seen as too limited because very often rules are imposed on organizations by a third party. On the other hand, imposed rules always have

to be internalized and in that regard the definition remains useful. A salient feature is to assign a level of enforcement to an operative business rule expressing an obligation or a prohibition. In this way a less crisp distinction can be made between strict business rules (hard constraints) and guidelines (soft constraints). It is possible that the final SBVR specification does make a distinction between advice statements and rule statements.

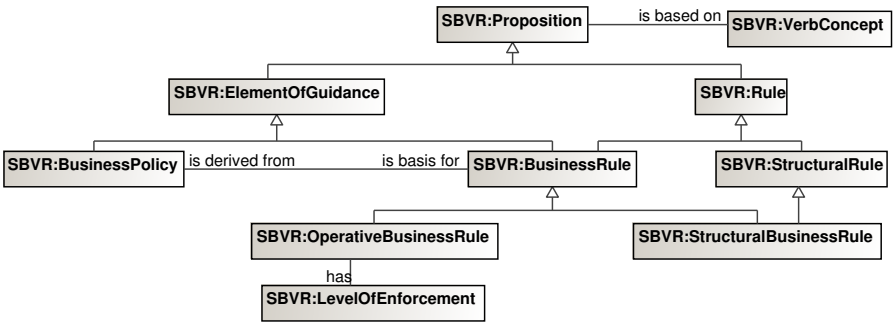


Figure 10: *A MOF/UML representation of the SBVR Vocabulary for Describing Business Rules*

In SBVR, meaning remains separate from expression. The SBVR provides a vocabulary called the Logical Formulation of Semantics Vocabulary to describe the structure and the meaning of vocabulary and business rules in terms of formalized statements about the meaning. Such formalized statements are semantic formulations (Baisley et al., 2005). Besides these fundamental vocabularies, the SBVR provides a discussion of its semantics in terms of existing, well-established formal logics such as First-Order logic, Deontic Logic and Higher-Order logic.

5.3 The EM-BrA^{2}CE Vocabulary

Although the SBVR provides extensive vocabularies for expressing business vocabularies and business rules, the current SBVR specification (Object Management Group, 2006b) does not have a built-in vocabulary for expressing process-related concepts such as agent, activity, event or deontic assignment. Such vocabularies and formal semantics for expressing dynamic constraints are deferred to a later version of the SBVR standard (Object Management Group, 2006b, p. 93).

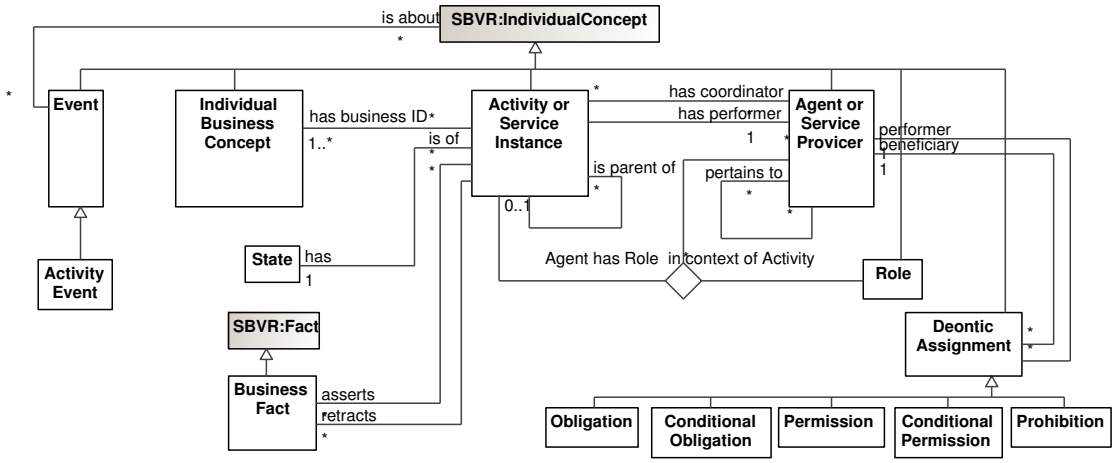
The EM-BrA^{2}CE Vocabulary has such characteristics. The vocabulary defines instance-level concepts that are meant for describing the state of a business process instance. In addition, it defines type-level concepts that are meant for describing the state space of a business process model. Figure 11(a) is a MOF/UML class diagram representation of the instance-level concepts in the vocabulary. Likewise Figure 11(b) represents the type-level concepts. Whereas all instance-level concepts extend SBVR:individual concept, all type-level concepts extend SBVR:concept type. To each instance-level individual concept a particular type-level concept type corresponds. In the following paragraphs these type-instance pairs are defined.

5.3.1 Business concept – business concept type

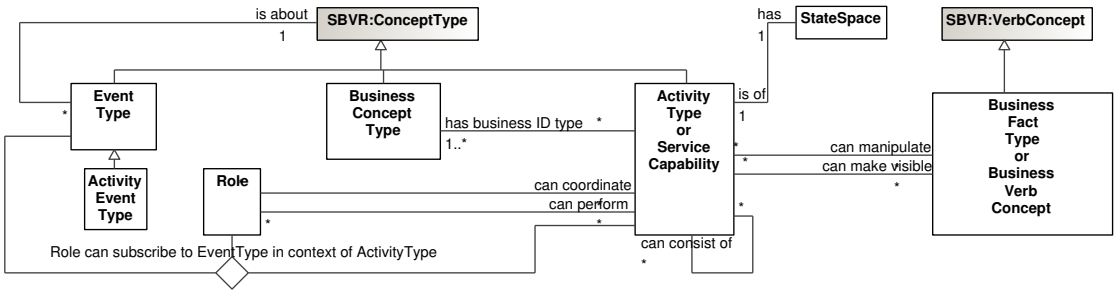
The flexibility of declarative business process modeling comes, among others, from the under-specification of process models and the use of guidelines (soft constraints). It does, however, not come from run-time adaptability of the process model. Therefore, the vocabulary distinguishes fact types that can be manipulated in the context of an activity, called

business fact typess. The following definitions apply.

A business concept type *is* an SBVR:concept type that *specializes* the individual concept 'individual business concept' and that *classifies* an individual business concept. Example: the business concept type 'purchase order'.
An individual business concept *is* an SBVR:individual concept of which the facts can be manipulated in the context of an activity. 'individual business concept *is a* business concept type' *is* an SBVR:assortment fact type that categories a business concept as being of a particular business concept type. Example: the individual business concept 'anOrderX', the assortment fact type 'anOrderX *is* a purchase order'.
A business fact type *is* an SBVR:fact type that has only business concept types as SBVR:role. Example: the business fact type 'purchase order *has due date* time point'.
A business fact *is* an SBVR:fact that is the basis for an atomic formulation of which every role binding *is bound to* a business concept. Example: the business fact 'anOrderX *has due date* July 2007'.



(a) instance-level



(b) type-level

Figure 11: *A MOF/UML representation of the EM-BrA^{2}CE Vocabulary*

5.3.2 Activity – activity type

The pair activity – activity type represents two of the most central concepts in the vocabulary. The following definitions apply:

> An activity type or service capability *is* an SBVR:concept type that *specializes* the individual concept 'activity' and that *classifies* an activity. Example: the activity type 'place order'.
> An activity or service instance *is* an SBVR:individual concept that represents a unit of (coordination) work to be performed by an agent. Example: the activity 'anActivityX'.
> 'activity *has type* activity type' *is* an SBVR:assortment fact type that categorizes an activity as being of a given activity type. Necessity: each activity *has type* exactly one activity type. Example: anActivityX *has type* coordinate purchase order.

A business process consists both of work and coordination work (Schmidt and Simone, 1996). This fundamental finding is recognized in the definitions of activity: an activity can either represent the act of performing an atomic unit of work or the act of coordinating a set of sub-activities. The former activity is called an atomic activity whereas the latter activity is called a composite activity. The fact types *can consist of* and *is parent of* indicate the activities a composite activity can consist of.

> 'Activity type *can consist of* activity type' *is* an SBVR:partitive fact type that represents that an activity of activity type involves the coordination of activities of activity type.
> 'Activity *is parent of* activity' *is* an SBVR:partitive fact type that represents an activity being composed of other activities. Example: the fact 'anActivityX *is parent of* anActivityY'.
> A composite activity type *is* an activity type that describes a category of composite activities. Example: the composite activity type 'coordinate purchase order'. Necessity: A composite activity type *can consist of* at least one activity type.
> An atomic activity type *is* an activity type that describes a category of atomic activities. Example: the atomic activity type 'place order', the fact type 'coordinate purchase order *can consist of* place order'.
> A composite activity *is* an activity that represents the coordination of a number of activities.
> An atomic activity *is* an activity that is not a composite activity and that represents an elementary unit of work. Necessity: an atomic activity *is* not *parent of* an activity.

When performing coordination work an agent can create an execution plan that consists of a number of sub-activities. In that case, the agent is identified as the coordinator of the created sub-activities. This is expressed by the *has coordinator* verb concept. The coordinator can schedule each activity in the execution plan for a particular due date, as expressed by the *has scheduled due date* fact type. The *has coordinator* fact type, is set by the coordinator when he assigns a given activity in the execution plan to a particular agent.

> 'Activity *has coordinator* agent' *is* an SBVR:associative fact type that represents an agent coordinating an activity.
> 'Activity *has scheduled due date* time point' is an SBVR:is-property-of fact type

that represents the scheduled due date of an activity.
'Activity *has performer* agent' *is* an SBVR:associative fact type that represents an agent performing an activity. Necessity: an activity *has performer* exactly one agent. Note: The latter constraint is not restrictive, since agents can form (ad-hoc) groups that are also agents.

An activity is uniquely **identified** by a set of business concepts. For example, a business concept of business concept type purchase order uniquely identifies an activity of type coordinate purchase order. This is expressed by the *has business ID* fact type. Another way of looking at business identifies is that they are the object on which an agent performs an activity. Consequently, the *has object* fact type is a synonym for the *has business ID* fact type and sets the business context of a given activity.

'Activity type *has business ID type* business concept type' *is* an SBVR:associative fact type that represents the business concept types that can identify an activity type. Example: coordinate purchase order *has business ID type* purchase order.
'Activity *has business ID* business concept' *is* an SBVR:associative fact type that represents an activity being (partially) identified by the business concept. Synonym: 'Activity *has object* business concept type' Example: anActivityX *has business ID* anOrderX or anActivityX *has object* anOrderX.

When performing an activity of a particular activity type, an agent can manipulate business facts of particular business fact types. This is expressed by the *can manipulate* fact type. Additionally, agents can retrieve information about particular business fact types when performing activities. The business fact types that are visible are indicated by the *can make visible* fact type.

'Activity type *can manipulate* business fact type' *is* an SBVR:associative fact type that represents that a business fact of type business fact type can be asserted or retracted during the performance of an activity of type activity type. Necessity: each business fact type that *can be manipulated by* an activity type *is in* the state space of the activity type. Example: place order *can manipulate* the business fact type 'purchase order *has due date* time point'.
'Activity type *can make visible* business fact type' *is* an SBVR:associative fact type that represents the business fact types that can be made visible in the context of activities of activity type. Note: visibility can be restricted by a visibility constraint. Necessity: each business fact type that *can be made visible by* an activity type *can be made visible by* the activity type. Example: coordinate purchase order *can make visible* the business fact type 'purchase order *has due date* time point'.

Within the context of an activity, a worker can perceive and manipulate only those business facts in which the business ID has a role. When an agent does business fact manipulations during the performance of an activity, the result of these manipulates is temporarily reflected by the *asserts* and *retracts* verb concepts. Only upon completion of the activity, the concept manipulations are committed to the entire system. Section 6 explains the execution model of EM-BrA^{2}CE process models and discusses this mechanism in detail.

'Activity *asserts* business fact' *is* an SBVR:associative fact type that represents a business fact has been asserted in the context of the activity. Example: anActivityY *asserts* the business fact 'anOrderX *has due date* Juli 2007'.
'Activity *retracts* business fact' *is* an SBVR:associative fact type that represents a business fact has been asserted in the context of the activity.

5.3.3 State – state space

An activity type (business process model) can be modeled by describing a state space and a set of business rules that constrain the possible transitions in this state space. Consequently, an activity (business process instance) has a particular state that corresponds to a specific set of facts that are true in this state.

> A state space *is* an SBVR:conceptual schema that *includes* the SBVR:concepts that describe a set of discrete states of an activity type. Necessity: a state space can only contain concepts that are instances of the concepts defined in the EM-BrA^{2}CE Vocabulary.
> 'activity type *has* state space' *is* an SBVR:associative fact type that represents the state space of an activity. Necessity: an activity type *has* exactly one state space.
> A state *is* an SBVR:conceptual model that *includes* facts about the concepts in the state space, that corresponds to a specific situation of an activity and that *is based on* the state space of an activity type.
> 'activity *has* state' *is* an SBVR:associative fact type that represents the state of an activity. Necessity: an activity *has* exactly one state.

State space is a specialization of an SBVR:conceptual schema, as depicted in Figure 12. Like a conceptual schema, a state space is described by the concepts, fact types and facts that adhere to the state space. As such, a state space describes a potentially infinite number of states. Likewise, state is a specialization of SBVR:conceptual model. Each state is based on a state space and contains a number of facts that adhere to the fact types in that state space.

In natural language, state is most often a relative notion that consists of a subgroup of states. For example, when defining the goal state of a business process, it is useful to consider the notion of an abstract state.

> An abstract state *is* a set of states that *conform to* the abstract state and that *is based on* the state space of an activity type.
> 'state *conforms to* abstract state' *is* an SBVR:associative fact type that a state corresponds to an abstract state.
> 'state space *has goal state* abstract state' *is* an SBVR:associative fact type that represents an abstract state being a goal or end state of a state space.
> 'state space *has start state* abstract state' is an SBVR:associative fact type that represents an abstract state being the start state of a state space.

An SBVR:business rule can be seen as a statement about an abstract state being either a necessity, an obligation, a prohibition or a possibility.

A *logic system* is called **monotonic** when the set of ground facts and logical formula in the system can produce a set of consequences that monotonically increases, even when new logical axioms are added. Logics with this property, namely that a derived fact cannot be invalidated by the addition of a logical formula that is consistent with this fact, are called **monotonic logics**. Conversely, a logic is non-monotonic when the addition of a logical formula can produce a reduction of the set of consequences that can be derived from it (Brachman and Levesque, 2004). A classical example of a non-monotonic system is Prolog, as its negation-as-failure entails that the addition of a fact might entail falsity of a previously derived fact.

In the EM-BrA^{2}CE framework (composite) activities represent the (coordination) work that occurs in the context of business processes. When an activity state transition occurs, a business process instance enters a new state and the transition is recorded by an

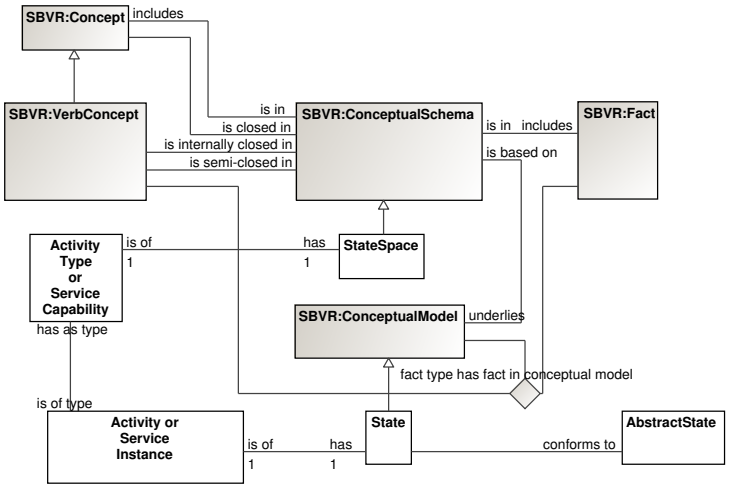


Figure 12: *A MOF/UML representation of state and state space in the EM-BrA^{2}CE Vocabulary*

activity event. Furthermore, agents can manipulate business facts in the context of an activity. Such a system that allows the manipulation of business facts could be interpreted as non-monotonic. However, the solution of the SBVR can be adopted that considers each conceptual model, consisting of a conceptual schema and collection of facts, as a logical system in its own right. Each time the facts in a conceptual model are changed, this creates a new and different conceptual model. In this way conceptual models are given a monotonic semantics (Object Management Group, 2006b, p. 77). This solution also conserves monotonicity when using negation-as-failure.

Monotonicity is a desired property for the EM-BrA^{2}CE framework. The contrary, having derived facts that become inconsistent trough the manipulation of facts, would not be in keeping with the intent of the EM-BrA^{2}CE Framework to provide a unifying execution model within which several knowledge representation paradigms can be used separately of one another. The reason for this is that non-monotonicity requires a logic system to revise earlier derived facts that might have become invalid; this is known as **belief revision**. In a setting of complex logical formulae of different kinds of logic, belief revision would be a complex operation. As a consequence backward chaining reasoning paradigms provide the reasoning of choice. Such a reasoning paradigm is, for instance, implemented in Prolog.

5.3.4 Agent – Role

Business Process Management Systems (BPMSs) must support business processes in which both humans and machines perform (coordination) work. To this end it is useful make abstraction from the differences between humans and machines through the use of the **agent** metaphor. This agent metaphor is present in many other ontologies for business modeling (Wagner, 2003; Guizzardi and Wagner, 2005). In the vocabulary, the agent concept in the vocabulary does not only represent individual workers or machines, but also ad-hoc groups of agents, such as for instance an entire department or company. This is expressed with the *pertains to* fact type.

> An agent or service provider *is* an SBVR:individual concept that represents an actor or a group of actors who can perform activities. Example: the agents 'workerX', 'purchase department', 'buyer inc.'.
> 'Agent *pertains to* agent' *is* an SBVR:partitive fact type that represents orga-

nizational structure and ad-hoc groups of agents. Example: the facts 'workerX *pertains to* purchase department','purchase department *pertains to* buyer inc.'. Note: the '*pertains to*' fact type is transitive.

In the context of a business process an agent can fulfill a particular role that represents an authorization to perform a number of activities. This conception of role is consistent with the Role Based Access Control (RBAC) standard (Sandhu et al., 1996; Ferraiolo et al., 2001; InterNational Committee for Information Technology Standards (INCITS), 2004). In the vocabulary the following definition applies:

A role *is* an SBVR:individual concept that represents a set of authorizations with regard to the performance of activities of given activity types.

Agents that have a particular role in the context of a business process have the authorization to perform a particular activity. This authorization is expressed by the *can perform* fact type. When performing an activity of a particular activity type, an agent can manipulate business facts of particular business fact types. This is expressed by the *can manipulate* fact type. Additionally, agents can retrieve information about particular business fact types when performing activities. The business fact types that are visible are indicated by the *can make visible* fact type.

'Role *can perform* activity type' *is* an SBVR:associative fact type that *represents* that an agent that has a given role can perform an activity of a particular activity type.
'Role *can coordinate* activity type' *is* an SBVR:associative fact type that *represents* the authorization that an agent of a particular role can coordinate an activity of a particular activity type.
'Agent *can have role* role' *is* an SBVR:associative fact type that represents that an agent can assume a particular role.
'Agent *has role* role *in the context of* activity' *is* an SBVR:associative fact type that represents that an agent assumes a particular role in the context of an activity. Note: These authorizations can be restricted by an activity authorization constraint.

The EM-BrA^{2}CE execution model distinguishes activity state transitions related to coordination (create, schedule, assign, revoke) and state transitions related to performing actual work (start, addFact, removeFact, updateFact, complete). Consequently, the vocabulary makes a distinction between the coordinator and the performer of an activity. The activity hierarchy determines whether an agent can coordinate an activity. In particular, when an agent has the authorization to perform a particular composite activity, he has the authorization to coordinate the activities of which the composite activity is parent. This is expressed by the following business rules.

It is necessary that a role *can coordinate* an activity type$_1$, if an activity type$_2$ *can consist of* the activity type$_1$ and role *can perform* activity type$_2$.
It is necessary that an activity$_1$ *has coordinator* an agent, if the activity$_1$ *has parent* an activity$_2$ and activity$_2$ *has performer* the agent.

The activities that are performed by a subsidiary agent, are performed by the agents to which the agent pertains.

5.3.5 Event – event type

In the last decades, events have been actively investigated in research communities such as the Knowledge Representation domain, Active Database domain, the architecture description domain. But even within these domains there exist quite distinct conceptions events. A substantial distinction is whether these events are considered volatile or non-volatile. **Volatile events** are perdurants that are immediately consumed (removed) after detection. In the Active Database community event definition languages and event detection prototypes such as for example SAMOS (Gatziu and Dittrich, 1993) and Snoop (Chakravarthy and Mishra, 1994) have this conception of event. **Non-volatile events**, on the other hand, are endurants that are never removed but are considered to persist. In the Event Calculus (Kowalski and Sergot, 1986), for instance, events are considered to persist. In active database systems, volatile events have been used to model reactive behavior. Each time when an event is detected, it is reacted upon and the event is removed from the model. The disadvantage of such an event removal policy, however, is that it does not allow for detecting so-called composite events. **Composite events** represent situations that correspond to the (non-)occurrence of several (atomic) events. To detect composite events, events need to non-volatile or they must at least be retained in the system during some time. Unlike atomic events, which occur at a particular point in time, composite events occur over a time interval that spans at least the occurrence times of each involved atomic event. Many event detection languages, among which SAMOS and Snoop, do not incorporate this interval logic and Galton and Augusto (2002) report on the unintended semantics of some composite event operators in these languages.

In the EM-BrA^{2}CE Vocabulary, the state of an activity (or service instance) includes the event history of the activity or its sub-activities. Consequently, events are given a non-volatile semantics. Although composite events are not considered explicitly by the vocabulary, composite events can still be included in business rules expressions.

> An event *is* an SBVR:individual concept that corresponds to an instantaneous, discrete state change of a concept in the world.
> 'Event *is about* SBVR:concept' *is* an SBVR:associative fact type that represents the concept whose state change is reported by the event.
> An event type *is* an SBVR:concept type that *specializes* the individual concept 'event' and that *classifies* an event.
> 'event type *is type of* event' *is* an SBVR:assortment fact type that categorizes an event as being of a particular event type. Necessity: it is necessary that an event *has type* exactly one event type.
> 'event *occurs at* time' *is* an SBVR:is-property-of fact type that represents the time at which an event occurs.

Although events occur instantaneously, they are asserted to the state space and are assumed not to be retracted. As such events make up the history of a business process. For the purpose of declarative process modeling such an event history allows for a greater expressiveness compared to procedural process languages. In such languages, the state of a process instance is represented by tokens that are local to the enabled activities. In order to allow historic events to influence the current behavior, the event history has to be reflected into the local state of the tokens. van Hee et al. have shown that such history-dependent behavior is in general difficult to model using Petri nets and propose to include event history into the state of a process instance.

Unlike many ontologies for business modeling, such as for instance the Agent-Object-Relationship (AOR) (Wagner, 2003) or Unified Foundational Ontology (UFO) (Guizzardi

and Wagner, 2005), a distinction is made between activities and events. Activities are performed by agents and have a particular duration whereas events occur instantaneously and represent a state change in the world. Changes to the life cycle of an activity are reflected by means of activity events. In section 6 twelve generic activity state transitions are described that correspond to twelve activity life cycle events.

> An activity event type *is* an event type that describes a category of activity state changes. Example: the activity event types 'created', 'scheduled', 'assigned', 'revoked', 'started', 'factAdded', 'factRemoved', 'factUpdated', 'aborted', 'skipped', 'completed' and 'redone'.
> An activity event *is* an event that corresponds to the state change of an activity. Necessity: it is necessary that an activity event *is about* exactly one activity. Necessity: it is necessary that an activity event *has* exactly one activity event type. Example: anEventX, anEventX *has type* scheduled, anEventX *is about* anActivityX.
> A business fact event *is* an event that involves the state change of a business fact. Necessity: a business fact event *is about* exactly one business fact.

The distinction between activity and event allows for reactive behavior. At each point during execution the history of a business process instance might be inspected through the use of an event query language. When an external event is added to the current state of an activity, that activity enters a new state. In this new state, the activity can undergo an additional transition as a reaction to the external event. Because this second transition is also recorded as an activity event, the system keeps track of its own state, reflecting the external (composite) events that have been reacted upon. The latter prevents the system from reacting twice to the same event.

The fact type 'role *can subscribe to* event type *in context of* activity type' expresses the visibility of events to agents in the context of an activity. It does not express how agents are notified of the event, which can generally occur using either a pull, a push or a publish-subscribe mechanism (Bailey et al., 2005). Furthermore, it is possible that the visibility is constrained by so-called event subscription constraint business rules.

> 'role *can subscribe to* event type *in context of* activity type' *is* an SBVR:associative fact type that expresses that an agent with a particular role can subscribe to an event of event type in the context of an activity of activity type. Example: seller *can subscribe to* completed *in the context of* ship.
> 'agent *perceives* event' *is* an SBVR:associative fact type that expresses that an event is non-repudiable to a particular agent. Example: anAgentX *perceives* anEventY.

5.3.6 Deontic assignment

Business regulations impose sequence and timing constraints on the activities in business processes. In a software-release process, for instance, a new version may only be put in production after it has been tested and approved. Similarly, in an order-to-cash process, an order may only be shipped by the dispatching office after it has been accepted by a salesperson. Designers often think implicitly about these kinds of permissions and obligations when modeling and hard-code their sequence and timing constraints in procedural, control-flow based process models. Such procedural languages define an explicit order relation between the activities in the process. What is lacking is a declarative approach that makes the partial order relations due to legal requirements more explicit. Bons et al. (1995) identify

this need to incorporate the legal state into the model of a trade procedure. To this end, the authors propose to annotate the states in Petri nets with a description of the deontic state. Regulations can be specified between the business partners in a business collaboration (between external agents). In this context regulations are called business protocols Bussler (2001) or business contracts. Several authors describe a language for intelligent agents to reason about contract state (Marín and Sartor, 1999; Yolum and Singh, 2004; Knottenbelt and Clark, 2004; Governatori, 2005; Paschke and Bichler, 2005; Goedertier and Vanthienen, 2006b).

In the vocabulary the legal permissions and obligations that originate from business regulations are called deontic assignments. A deontic assignment represents among others the obligation or permission of an agent to perform a particular activity by a particular due date.

> A deontic assignment *is* an individual concept that represents an obligation, prohibition, permission, conditional obligation or conditional permission of an agent towards another agent (beneficiary) regarding the performance of an activity with respect to a given due date.
> 'deontic assignment *has* due date'
> 'deontic assignment *involves* activity'
> 'deontic assignment *has performer* agent'
> 'deontic assignment *has beneficiary* agent'
> An obligation *is* a deontic assignment that represents the obligation of an agent to perform a particular activity by a particular due date.
> A permission *is* a deontic assignment that represents the permission of an agent to perform a particular activity before a particular due date.

A deontic assignment can also be expressed conditionally. When an agent performs a given activity a conditional deontic assignment may result from it. For instance, in the shipment-after-payment process model visualized in Figure 6(b) a buyer makes a conditional commitment when he places an order. In particular, a buyer has the conditional obligation to pay the seller if the seller accepts the order. If the seller rejects the order, no obligation results from it. The following definitions are included in the vocabulary:

> An conditional obligation *is* a conditional deontic assignment that represents the conditional obligation that rests on an agent to perform a particular activity before a given due date, after – and on the condition that – a particular agent has done a particular activity within a particular due date.
> An conditional permission *is* a deontic assignment that represents the conditional permission of an agent to perform a particular activity before a particular due date, after – and on the condition that – a particular agent has done a particular activity within a given due date.
> 'conditional deontic assignment *has conditional due date* date'
> 'conditional deontic assignment *involves conditional* activity'
> 'conditional deontic assignment *has conditional performer* agent'
> 'conditional deontic assignment *has conditional beneficiary* agent'

The existence of deontic assignments is entirely defined by temporal deontic rules and is dependent on the historic behavior of agents playing a particular role in the context of a composite activity. Deontic assignments should not be confused with the deontic propositions of the SBVR. The Deontic propositions in SBVR resemble those of Standard Deontic Logic (SDL) (Føllesdal and Hilpinen, 1971) and express that a particular state of affairs is permissible, necessary, obligatory or prohibited. Like SDL the SBVR expresses

the obligation to bring about a certain proposition in an impersonal way: it cannot express the agent to whom a particular obligation or permission applies. Another difference with deontic assignments is that deontic propositions are static; they cannot represent deontic properties that come into effect and cease to hold because of timeouts on deadlines or other events. Finally, the SBVR is not able to express so called contrary-to-duty obligations (Governatori and Rotolo, 2002), reparative obligations that come into existence as the result of the violation of an obligation. For instance, after a due date on an obligation to pay has passed, a violation event occurs.

> A violation event is an event that occurs when an agent does not perform an obligation within the due date of that obligation.
> Necessity: Each violation event *is about* exactly one obligation.

Many deontic logics are closed such that, for instance, prohibition can be derived from the lack of either an obligation or a permission deontic assignment. It would however be unfair to assume that a process modeler must specify deontic assignment rules for each activity type that occurs within a process model. Therefore it is useful to indicate the activities for which explicit deontic assignments must be derived in order to perform them. This is expressed by the is-property-of fact type 'activity type *is deontically closed in* state space' (Segerberg, 1982).

> 'activity type *is deontically closed in* state space' *is* a fact type that expresses that in each state based on the state space, the entire extension of every deontic assignment that involves an activity of the activity type is given in the facts included in the state.

When an activity type is deontically closed in a state space, prohibition is derived from the absence of permission or obligation. When, in contrast, this is not the case, no deontic assignment can be derived from the absence of information.

5.3.7 Non-functional, quality-of-service concerns

Given its origin in telecommunication, the term 'quality of Service' (QoS) at first sight has little ado with business modeling. However, in the academic research involving web services, the term quality of service refers to a number of non-functional quality requirements such as availability, robustness, scalability, security and trust information (Roman et al., 2005). QoS concerns are also business concerns that can be specified in a language that the business understands. The vocabulary considers the following QoS concerns.

> Spatial availability *is* a quality of service specification that determines the location from which activities of a given activity type can be performed or that business facts of a given business fact type can be accessed.
> Temporal availability *is* a quality of service specification that determines the amount of time during a time period that activities of a given activity type can be performed or that business facts of a given business fact type can be accessed.
> Response time *is* a quality of service specification that determines the maximum time period it may take to perform a state transition on an activity of given activity type or on a business fact of a given business fact type.
> Throughput *is* a quality of service specification that determines the ratio of activity state transitions or business fact accesses per unit of time.
> Historic window *is* a quality of service specification that determines the time period during which historic information about activity events or business concept

manipulations must be stored.
Latency *is* a quality of service specification that determines the maximum delay by which concept modifications are propagated.
Security *is* a quality of service specification that determines the identity, privacy, alteration and repudiation facets related to performing activities or consulting information.

Quality of service specifications can be imposed both on fact types (information) and activity types (processes). QoS concerns must be both information- and process-aware rather than exclusively information- or process-driven. This entails that Quality of Service (QoS) specifications on information access should contain information about the activity (or service) context in which information is retrieved. This is particularly important when the same information (or facts) is required in the context of different activities with different QoS requirements. For example, when verifying whether a customer is a high-volume customer, it is not so important to have zero latency on the historic sales records that are consulted. In contrast, when determining the total amount of outstanding debt with a customer, it is likely that sales records must be consulted without latency. Clearly the activity context in which information (facts) are retrieved is an important differentiator of QoS specifications. This is reflected in the vocabulary:

'Fact type *must have* temporal availability *in the context of* activity type'
'Fact type *must have* spatial availability *in the context of* activity type'
'Fact type *must have* response time *in the context of* activity type'
'Fact type *must have* throughput *in the context of* activity type'
'Fact type *must have* historic window *in the context of* activity type'
'Fact type *must have* latency *in the context of* activity type'
'Fact type *must have* security *in the context of* activity type'

QoS specifications on information (or fact types) must be process aware. This relation also holds in the opposite sense: QoS specification on processes (or activity types) must be information aware. The latter is particularly important when strict QoS specifications on business processes are disproportionate with less strict QoS specifications on information. The fact types 'activity type *can manipulate* business fact type' and 'activity type *can make visible* business fact type' keep track of the business fact types that are accessed by activities of a given activity type. It can be used to determine whether activity type QoS specifications are aligned with fact type QoS specifications.

'Activity type *must have* spatial availability'
'Activity type *must have* temporal availability'
'Activity type *must have* response time'
'Activity type *must have* throughput'
'Activity type *must have* historic window'
'Activity type *must have* latency'
'Activity type *must have* security'

5.3.8 Cost and time concerns

Cost and time concerns affect the coordination of activities (or services). For example, in an order acceptation process, a sales representative will not include an expensive review creditworthiness activity that is disproportionate with the insignificant amount of the order. Likewise, a sales representative would not schedule a slow, time-consuming shipment for a rush order of an important customer.

The performance of an activity (or service) inadvertently has financial implications. When activities are performed among agents of different organizations, the financial implication is called a price. When activities are performed among agents that pertain to the same organization, the financial implication is called a cost. O'Sullivan et al. (2002) discuss different techniques for agents (or service providers) to charge money for providing their services and to settle payment. Within organizations cost accounting techniques are usually put in place to determine the internally incurred cost of the activities (or services) that are performed and corresponding incentive-compatible cost allocation models. The EM-BrA^{2}CE Vocabulary does not provide a vocabulary to express charging styles, settlement models and allocation schemes in detail. Instead, it provides a single cost measure that informs the coordinator of an activity about the expected financial impact of having the activity performed.

> Cost of performance *is* the cost that is incurred when performing a given activity. 'Activity *has an expected cost of* cost of performance' *is* an SBVR:is-property-of fact type that represents the cost of performance that is expected to be incurred prior to the start of the activity. Example: anActivityY *has an expected cost of* 4.5 euro.

Derivation rules can specify the fact type 'activity *has an expected cost of* cost of performance' based on the properties of the activity such as the activity type, the agent assigned to perform the activity, the object (or business id) of the activity and the scheduled due date of the activity.

The performance of an activity (or service) inadvertently takes time. When performing a coordination activity, a coordinating agent must take into account the scheduled due date of the coordination activity. In particular, all required sub-activities in the execution plan of the coordination activity must be completed prior to the completion of the coordination activity. For instance, when a sales representative coordinates the processing of a sales order, the order acceptation and shipment sub-activities must be completed before the due date imposed on the coordination activity. Many non-functional properties influence the time required for a service provider to perform an activity: capacity, throughput, arrival rates. The EM-BrA^{2}CE Vocabulary does not provide a vocabulary to express these concerns. Instead, it provides a time measure that informs the coordinator of an activity about the expected duration of performing an activity.

> Duration of performance *is* the duration that is required to perform an activity. 'Activity *has an expected duration of* duration of performance' *is* an SBVR:is-property-of fact type that represents the expected time needed to complete a particular activity. Example: anActivityY *has an expected duration of* three working days.

Derivation rules can define the fact type 'activity *has an expected duration of* duration of performance' based on the properties of the activity such as the agent assigned to perform the activity and the object (or business id) of the activity .

5.4 Business Rules in the EM-BrA^{2}CE Framework

This section identifies sixteen business rule types. They refer to one of the three aspects of business process modeling that are generally considered (Jablonski and Bussler, 1996): the control-flow, the data and the organizational aspect. The control-flow aspect of business process models describes the activities and their execution order. The data aspect deals with business and processing data, such as events that flow between the agents that are internal or external to the process model. The organizational aspect provides information

Table 2: *Business rule types*

business rule type	aspect	modality
Temporal deontic rule	control flow	alethic
Activity precondition	control flow	alethic, deontic, guideline
Activity postcondition	control flow	alethic, deontic, guideline
Dynamic integrity	control flow	alethic, deontic, guideline
Activity cardinality	control flow	alethic, deontic, guideline
Serial activity constraint	control flow	alethic, deontic, guideline
Activity order	control flow	alethic, deontic, guideline
Activity exclusion	control flow	alethic, deontic, guideline
Activity inclusion	control flow	alethic, deontic, guideline
Reaction rule	control flow	alethic, deontic, guideline
Static integrity	data	alethic, deontic, guideline
Derivation rule	data	alethic, deontic, guideline
Activity authorization	organization	alethic
Activity allocation rule	organization	alethic, deontic, guideline
Visibility constraint	organization	alethic
Event subscription	organization	alethic

about the organizational structure in the form of human and machine roles responsible for executing tasks. Table 2 indicates the model aspect of each business rule type identified in the framework.

Ross (2003) advocates that business rules not always need to express strict necessities but also guidelines or possibilities. Correspondingly, the SBVR standard classifies business rules according to the intended modality as being either structural or operative (Object Management Group, 2006b). Structural business rules express a necessity or impossibility that cannot be violated without leaving the system in an inconsistent state. Operative business rules express an obligation or a prohibition that agents can violate. To each operative business rule a level of enforcement can be assigned that indicates the degree in which a business rule must be enforced and allows to distinguish advice - what ought to be true - from strict obligation - what should be true. In the remainder of this section sixteen business rules as defined as specializations of structural or operative business rules or guidelines. Table 2 indicates the possible modalities that can be attached to each business rule type identified in the framework.

5.4.1 Providing Logical Foundations for Temporal Rules

The semantics of SBVR expressions is underpinned by first-order logic, Simple Deontic Logic, restricted Higher-Order logic and reification. Although Structured English provides two linguistic techniques to express temporal relationships: objectification (Object Management Group, 2006b, p. 59, p. 198) and intensional roles, it lacks a temporal logic to represent and reason about temporal relationships. The inclusion of temporal logic is deferred to a later version of SBVR (Object Management Group, 2006b, p. 93). The latter is likely to be required for the purpose of declarative process modeling. For instance, looking at some existing languages for declarative process modeling, it can be observed that the ConDec language of Pesic and van der Aalst (2006) makes use of Linear Temporal Logic (LTL) to express business rules and that the PENELOPE language (Goedertier and Vanthienen, 2006b) makes use of the Event Calculus to model the effects of performing ac-

tivities with respect to the coming into existence (or ceasing to exist) of temporal deontic assignments. The following candidate temporal logics could be incorporated to define the semantics of declarative process models:

- An event query language. At this point it might be useful to define a number of event operators that are useful to query the event history of a particular activity. To this end the event operators of Snoop (Chakravarthy and Mishra, 1994) could be adopted that have been given interval semantics by Galton and Augusto (2002). The negation event operator requires closure the EM-BrA^{2}CE:**event** concept and of all related event fact types in the EM-BrA^{2}CE Vocabulary.

- **Linear Temporal Logic (LTL).** As demonstrated by Chomicki (1995) and Bacchus and Kabanza (2000) and Pesic and van der Aalst (2006) (Past) LTL expressions can be used to represent desirable or undesirable patterns within a history of events. LTL is a modal temporal logic that allows to express temporal constraints on infinite paths within a state space. LTL formula can be evaluated by obtaining the Büchi automaton that is equivalent to the formula and checking whether a path corresponds to the automaton. Unfortunately most LTL checking algorithms assume infinite paths and construct non-deterministic automata (Pesic and van der Aalst, 2006). Another disadvantage is that LTL does not allow to express the effect that results from a particular transition in a state space. For these reasons, it is not evident to express a goal state in LTL nor to construct automata for planning an execution scenario to obtain a goal state (Bacchus and Kabanza, 2000).

- **The Event Calculus.** In first-order logic there is a formalism that elegantly captures the time-varying nature of facts, the events that have taken place at given time points and the effect that these events reflect on the state of the system. This formalism is called the Event Calculus. The **Event Calculus**, introduced by Kowalski and Sergot (Kowalski and Sergot, 1986), is a logic programming formalism to represent and reason about the effect of *events* on the state of a system expressed in terms of *fluents*. The Event Calculus is appealing for several reasons. For instance, the Event Calculus builds on a first-order predicate logic framework, for which efficient reasoning algorithms exist. In addition the Event Calculus not only has the ability to deductively reason about the effects of the occurrence of events events (leading to the coming into existence of fluents or the ceasing to hold), most importantly, it also has the ability of reasoning **abductively**. Abductive reasoning over the event calculus has been shown to be equivalent to planning. In particular, abductive reasoning produces a sequence of transitions (denoted by events) that must happen for a particular fluent to hold in the future (Eshghi, 1988; Shanahan, 1997; Van Nuffelen and Kakas, 2001). For these reasons, the Event Calculus is a suitable language both for specifying the semantics of state transitions in the EM-BrA^{2}CE framework and as a planning mechanism.

The business rules types that are introduced later in this section can be given a plethora of logical foundations. Consequently, providing one particular logical foundations for these rules would not be in keeping with the intent of EM-BrA^{2}CE Framework to provide a unifying vocabulary and execution model within which several knowledge representation paradigms can be used separately of one another. Instead, the text relates the business rule types to existing work in the literature and indicates which temporal logic could be applied.

5.4.2 Semantic Formulation of Temporal Rules

As business rules are most often formulated as (natural) language statements, the SBVR contains an English vocabulary for describing vocabularies and stating rules. The processing of natural language pertains to the Artificial Intelligence domain of Natural Language Processing (NLP). It involves on the one hand the understanding of natural language statements in terms of semantic formulations (Baisley et al., 2005), and on the other hand the verbalization of semantic formulations into natural language statements. To date, two SBVR natural language parsers have been developed: the Unisys Rules Modeler (Baisley, 2005; Unisys, 2005) and SBeaVeR (Digital Business Ecosystem (DBE), 2007). These parsers analyze the meaning of natural language expressions in terms of semantic formulations.

The kinds of semantic formulations described by the SBVR do not allow to represent temporal knowledge (Object Management Group, 2006b, p. 39). In particular, it is not possible to describe the semantic structure of business rules that discuss the relationship between states and events or that define the effect of activities. The hereafter introduced business rule types of the EM-BrA^{2}CE Vocabulary are likely to require such semantic formulations. However, including temporal semantic formulations requires choosing for a particular logic to represent temporal knowledge. As this is not in keeping with the intent of provide a unifying framework, semantic formulations are left outside the framework.

5.4.3 Control-flow: temporal deontic rule

Business policy and regulations contain a lot of implicit order and timing information. In a trade community, for instance, different business protocols might exist for engaging in a business interaction. Such business protocols lay down the obligations and permissions of all business partners in an interaction and can be expressed in the form of temporal deontic rules.

> A temporal deontic rule *is* a structural business rule that defines when deontic assignments come into existence or cease to exist based on the (non-)occurrence of events.

The rules describe behavior from a third-person perspective and can, for instance, be expressed in the PENELOPE language (Goedertier and Vanthienen, 2006b).

The displayed temporal deontic rules categorize the external business regulation payment-after-shipment visualized in Figure 6(a). Assuming that the agents in a business interaction do not intend to violate these deontic assignment rules, the resulting permissions and obligations impose partial order constraints on the activities in a business process.

> It is necessary that initially a buyer *has* the permission to perform a place order activity.
> It is necessary that when a buyer completes a place order activity, the seller *has* the obligation to perform a accept order activity or a reject order activity within 2 time units.
> It is necessary that when the buyer completes a place order activity, the buyer *has* the obligation to perform a pay activity within 2 time units after the seller completes the ship activity.
> It is necessary that when the seller completes a accept order activity, the seller *has* the obligation to perform a ship activity within 2 time units.

In the activity life cycle of the EM-BrA^{2}CE Framework, a temporal deontic rule constrains *schedule*, *start* and *redo* activity state transitions. Temporal deontic rules indirectly

affect the sequence and timing of activities. An agent who coordinates an activity will try to observe the deontic assignments that result from performing the activities. In addition, the coordinator will take into account that other agents potentially could violate the deontic assignments that are imposed on them. In particular, a coordinator will schedule the activities such that he does not violate any permissions and that the deadlines on obligations are observed. Additionally, he will take appropriate action when other agents violate the deadlines that are imposed on them.

5.4.4 Control-flow: activity precondition

Although any activity state transition can be constrained using preconditions, we only consider preconditions imposed on the *start* and *complete* activity transitions to be business rules. A precondition on the *start* transition is called an activity precondition:

> An activity precondition *is* a business rule that defines the conditions that are required to start an activity of a given activity type.
> Example:
> To start an accept order activity, it is necessary that a place order activity has been completed and that no accept order or reject order activity has been started.
> To start a reject order activity, it is necessary that a place order activity has been completed and that no accept order or reject order activity has been started.
> To start a ship activity, it is necessary that a accept order activity has been completed and that no ship order activity has been started.
> To start a pay activity, it is necessary that a accept order activity has been completed, a ship order activity has been completed and that no pay activity has been started.

In the Web Service Modeling Ontology (WSMO) (Roman et al., 2005), it is possible to assign a precondition to a service capability. However, preconditions can only be expressed in terms of (business) concepts. In particular, it is not possible to include event conditions or to query the properties of activities. This limitation potentially has the disadvantage that it is required to add artificial business concepts to a business vocabulary. For instance, instead of stating that an order has been accepted, it is required to refer to a potentially artificial business concept acceptation notice. The same activity preconditions, expressed in terms of business concepts only, would then be formulated as:

> To start an accept order activity, it is necessary that the order exists and does not *have* an acceptation notice or a rejection notice.
> To start a reject order activity, it is necessary that the order exists and does not *have* an acceptation notice or a rejection notice.
> To start a ship activity, it is necessary that the order the order *has* an acceptation notice and does not *have* a shipping order.
> To start a pay activity, it is necessary that the order the order *has* an acceptation notice and the order *has* a proof of delivery and the order has not yet been paid.

Unlike the EM-BrA^{2}CE Framework, WSMO makes a distinction between the state of the information space and the state of the world. Because the EM-BrA^{2}CE Framework is more situated on the conceptual modeling level, no such distinction between the world and the information system is made.

In the activity life cycle of the EM-BrA^{2}CE Framework, an activity precondition constrains *start* and *redo* activity state transitions.

5.4.5 Control-flow: activity postcondition

A precondition on the *complete* transition is called an activity postcondition:

> A business fact postcondition *is* a business rule that specifies the abstract state of an activity of a particular activity type upon its *completion*.
> Example:
> To complete an activity that *has type* place order, it is necessary that the order exists.
> To complete an activity that *has type* accept order, it is necessary that the order *has* an acceptation notice.
> To complete an activity that *has type* reject order, it is necessary that the order *has* a rejection notice.
> To complete an activity that *has type* ship, it is necessary that the order *has* a proof of delivery.
> To complete an activity that *has type* pay, it is necessary that there exists a proof of payment.

This constraint subsumes a *mandatory* constraint in the case handling paradigm and is similar to a *postcondition* in WSMO. The case handling paradigm allows to specify which case data types are free, mandatory or restricted with respect to performing an activity of a particular activity type (van der Aalst et al., 2005). A **free** business fact type can be manipulated in every sub-activity. A business fact type is **mandatory** for a particular activity type, when a fact of this fact type is required for the *completion* of the particular activity. A business fact type is **restricted** to a particular (or a number of) activity type, when a fact of this fact type can only be manipulated in the context of an activity of this type. van der Aalst et al. (2005) provide a means in which these constraints can be operationalized by considering them as post conditions on the completion of a particular activity. In WSMO, it is possible to assign a post condition to a service capability. However, the same restriction applies as with respect to preconditions.

In the activity life cycle of the EM-BrA^{2}CE Framework, an activity post condition constrains *complete* activity state transitions.

5.4.6 Control-flow: reaction rule

> A reaction rule or event-condition-action (ECA) rule *is* a business rule that expresses the activities that are to be undertaken, given the (non-)occurrence of certain events and a particular condition being fulfilled.

In spite of their apparent simplicity, business processes using only reaction rules cannot be classified as being declarative process models. The reason is that process models that are composed of reaction rules only constitute an explicit execution scenario that risks to be over-specified and can be regarded to be as procedural as control-flow based models. What is needed is a hybrid approach, in which the freedom of choice that is left by other business rules is filled in – when required – by a small set of reaction rules. Therefore, reaction rules are considered among other business rules to specify behavior, but constitute by themselves no means for declarative process modeling. Another problem is that ECA rules and reaction rules in general lack comprehensibility. It is difficult to understand even a small number of reaction rules. To tackle the comprehensibility problem, reaction rules need to be grouped in small sets of reaction rules that display mutually exclusive behavior for a given situation. This can be addressed by grouping reaction rules into so-called decision points (Goedertier and Vanthienen, 2005). The properties of relevant abstract

states in the execution model of a business process can be used to describe decision points. For example, the above discussed payment-after-shipment temporal deontic rules, state that a seller has the obligation to either accept or reject an order, when a buyer places an order. Although possible from a modeling perspective, the protocol does not stipulate what the buyer must do in case the seller, for instance, rejects the order. This freedom of choice can be represented as an abstract state or decision point that is described by the following abstract state expression:

> An agent of role seller has the obligation either to accept or reject an order.

From this abstract state a number of mutually exclusive abstract states can be derived: the seller has accepted the order, the seller has rejected the order or the order times out. Reaction rules can impose a suitable reaction to each of these states:

> When an order is rejected and if the order *is critical*, then notify a purchase representative.
> When an order is rejected and if the order *is* not *critical*, then reorder with a different seller.
> When an order times out and if the order *is critical*, then notify a purchase representative.
> When an order times out and if the order *is not critical*, then reorder with the same seller.

In the activity life cycle of the EM-BrA^{2}CE Framework, a reaction rule defines *start* activity state transitions.

5.4.7 Control-flow aspect: dynamic integrity constraint

Within the context of an activity, agents can manipulate business facts that are related to the activity. There are, however, conditions on the state change of business facts that could prevent an activity from taking place. Wagner (2003) calls such conditions dynamic integrity constraints.

> A dynamic integrity constraint *is* a business rule that defines the admissible state changes of a business fact.
> Example: After the start of a ship activity, the order lines of the order can no longer be changed.

Activities always remain implicit in these rules as the fact type 'activity type *can manipulate* business fact type' already relates activities to business fact types.

In the activity life cycle of the EM-BrA^{2}CE Framework, a dynamic integrity constraint constrains *start*, *redo*, *addFact*, *removeFact* and *updateFact* activity state transitions.

5.4.8 Control-flow aspect: activity cardinality constraint

An agent that performs a composite activity, actually constructs an execution plan. This coordination work involves, among others, the creation of a number of activities. Although the composites of a coordination plan are defined by the 'activity type *can consist of* activity type' fact type, the fact type does not impose any restrictions on the number of such activities that may be included in the execution plan. For instance, an execution plan in the context of a handle purchase order activity might involve two separate ship activities, but it may only contain one accept order activity. Such cardinality restrictions are imposed by activity cardinality constraints.

> A activity cardinality constraint *is* a business rule that limits the number of activities of a particular activity type that occurs within the context of a same parent (coordination) activity.
> Example:
> There exists exactly one place order activity$_1$ that *has parent* a handle purchase order activity$_2$.
> There exists at most one accept order activity$_1$ that *has parent* a handle sales order activity$_2$.

This business rule type can be expressed in the ConDec language with so-called existence constraints.

In the activity life cycle of the EM-BrA^{2}CE Framework, an activity cardinality constraint constrains *create*, *start* and *redo* activity state transitions.

5.4.9 Control-flow aspect: serial activity constraint

When coordinating a composite activity, it is also relevant to know whether two activities can be performed concurrently. This is expressed by a serial activity constraint.

> A serial activity constraint *is* a business rule that imposes that activities belonging to a particular set of activity types must not be performed in parallel.
> Example:
> Activities that *have type* place order, accept order, reject order and ship order must not be performed in parallel.

Sadiq et al. (2005), for instance, include serial activity constraints in their constraint specification framework.

A serial activity constraint constrains *schedule*, *start* and *redo* activity state transitions.

5.4.10 Control-flow aspect: activity order constraint

An order constraint on two activities is a stronger condition than a seriality constraint.

> An activity order constraint *is* a business rule that imposes that activities of particular activity types must be performed in a specified order.
> Example:
> An accept order activity$_1$ can only start after a place order activity$_2$ has completed.

This can, for instance, be expressed with an order constraint in the constraint specification framework of Sadiq et al. (2005) or with different types of relation constraints in the ConDec language. The ConDec language, in particular, allows for expressing a

In the activity life cycle of the EM-BrA^{2}CE Framework, an activity order constraint constrains *schedule*, *start* and *redo* activity state transitions.

5.4.11 Control-flow aspect: activity exclusion constraint

> An activity exclusion constraint *is* a business rule that imposes that two activities of a particular activity type are mutually exclusive.
> Example:
> It is necessary that accept order and reject order activities are mutually exclusive.

This can, for instance, be expressed with an exclusion constraint in the constraint specification framework of Sadiq et al. (2005) or with different types of negation constraints in the ConDec language.

In the activity life cycle of the EM-BrA^{2}CE Framework, an activity exclusion constraint constrains *create*, *start* and *redo* activity state transitions.

5.4.12 Control-flow aspect: activity inclusion constraint

> An activity inclusion constraint *is* a business rule that imposes that two activities of a particular activity type are mutually inclusive.
> Example: It is obligatory that accept order and ship activities are mutually inclusive.

This can, for instance, be expressed with an inclusion constraint in the constraint specification framework of Sadiq et al. (2005).

In the activity life cycle of the EM-BrA^{2}CE Framework, an activity inclusion constraint constrains *complete* activity state transitions.

5.4.13 Data aspect: Static integrity constraint

The performer of an activity can perform particular manipulations (addition, removal or update) of business facts. These state transitions are among others subject to particular integrity constraints. Integrity constraints involve cardinality constraints, domain constraints and the like.

> A static integrity constraint *is* a business rule that constrains the domain over which business facts can range by expressing a logical assertion that can, cannot, must or must not remain true (Wagner, 2003).
> Example:
> It is necessary that each order *has* at least one order line.
> It is advisable that the agreed price of a sales item is less or equal to the standard price of the sales item.

Integrity constraints can be operationalized by verifying whether the manipulation of a business fact (addition, removal and update) would lead to a violation of the integrity constraint. Not every integrity constraint needs to be evaluated. For instance, only those integrity constraints that range over a fact type that is currently being manipulated. In the database literature efficient algorithms have been proposed for static constraint verification (Gupta et al., 1994).

In the activity life cycle of the EM-BrA^{2}CE Framework, a static integrity constraint constrains *addFact*, *removeFact* and *updateFact* activity state transitions.

5.4.14 Data aspect: derivation rule

Almost any knowledge representation language allows to express so-called derivation or deduction rules.

> A derivation rule *is* a business rule that defines a business fact in terms of existing business facts (Wagner, 2003).
> Example:
> It is necessity that a luxury product *has a value-added-tax of* 20 percent.
> It is advisable that an order *has* a 10 percent discount if the order *is from* a loyal customer.

A derivation rule can have a deontic instead of an alethic nature. In other words, derivation rules can be SBVR:structural business rules or SBVR:operative business rules. These rule types require a different execution semantics. For instance, a business rule might recommend a particular price, but leave a salesperson with the freedom of choice of determining a custom-tailored prices.

In the above explained non-monotonic setting, each time facts of this fact type are required (for instance in the context of evaluating another business rule) the derivation rule can be consulted. Consequently, derivation rules augment the fact base (SBVR:conceptual model) during the processing of state transition requests. However, when a state transition occurs all derived facts are not automatically transferred to the new state. Reasoning paradigms such as backward chaining automatically support this execution semantics.

In the activity life cycle of the EM-BrA^{2}CE Framework, a derivation rule constrains *addFact* and *updateFact* activity state transitions.

5.4.15 Organization aspect: activity authorization constraint

An activity authorization constraint allows to constrain the agent-role assignments that can be granted to an agent. For instance, the fact 'sales representative *can perform* accept order' is constrained by the rule that sales orders larger than 2000 euro cannot be reviewed by junior sales representatives.

> An activity authorization constraint *is* a structural business rule that dynamically constrains the activities that can be assigned to an agent on the basis of the properties of the activity, the business facts in its state space and the properties of the agent.
> Example: It is necessary that an agent that *has age* less than 18 years can not perform a place order activity.
> It is necessary that an agent that *has function* junior sales representative can not perform an accept order or reject order activity that *is identified by* an order that *has* an amount larger than 2000 euro.

A similar kind of rule has been described by Strembeck and Neumann (2004) in the context of the Role-based access control (RBAC) standard Sandhu et al. (1996); Ferraiolo et al. (2001); InterNational Committee for Information Technology Standards (INCITS) (2004).

In the activity life cycle of the EM-BrA^{2}CE Framework, an activity authorization constraint constrains *assign* activity state transitions.

5.4.16 Organization aspect: activity allocation rule

It is possible that agents have the authorization to perform a particular activity, but that they are not the primary designated performers of a task. Activity allocation rules are guidelines that indicate to what extent assigning an activity to an agent is desirable.

> An activity allocation rule *is* an operative business rule that indicates the assigning of an activity to a particular agent as an obligation or a prohibition. In both cases a level of enforcement indicates the degree to which such an assignment is desirable.
> Example: It is not advisable that an agent that *has function* purchase department head is assigned to an activity that *has type* archive document.

Whereas activity authorization constraints deal with authorization, activity allocation rules deal with the fair distribution of work.

In the activity life cycle of the EM-BrA^{2}CE Framework, an activity authorization constraint constrains *assign* activity state transitions.

5.4.17 Organization aspect: visibility constraint

The fact type 'Activity type *can make visible* business fact type' indicates the business fact types that can be made visible in the context of an activity. It is possible to constrain the visibility of facts with visibility constraints.

> A visibility constraint *is* a structural business rule that dynamically constrains the visibility of business facts within an activity according to the properties of the activity, the business facts in its state space and the agent that has been assigned to the activity.
> Example: Coordinate purchase order *can make visible* the business fact type 'order *has* rejection notice'
> It is necessary that a rejection notice is only visible to an agent that *is* a corporate customer.

5.4.18 Organization aspect: event subscription constraint

The fact type 'role *can subscribe to* event type *in context of* activity type' expresses the visibility of events to agents in the context of an activity. With event subscription constraints is it possible to conditionally limit the visibility of events.

> A event subscription constraint *is* a structural business rule that constrains the conditions under which agents who have a particular role in the context of an activity can perceive the occurrence of an activity event.
> Example: A seller *can subscribe to* completed *in the context of* ship.
> A buyer *can subscribe to* completed *in the context of* ship.
> It is not possible that an agent that *has role* buyer *perceives* an event that *is about* a ship activity for an order that *has* a total amount of less than 2000 euro.

When an event occurs, each agent who has a particular role in the context of the activity and whose role is subscribed to the event type and for whom no subscription constraints apply, can perceive the event. Consequently, the event is non-repudiable to external agents such that any legal obligation that results from the event can be enforced.

6 An Execution Model for Declarative Process Modeling

Few process languages have a formal execution model. The Business Process Modeling Notation (BPMN) (Object Management Group, 2006a; Wohed et al., 2006) and UML Activity Diagrams Object Management Group (2005); Störrle and Hausmann (2005), for instance, lack such formal semantics. It is nonetheless difficult to unambiguously describe the meaning of a language in terms of informal, natural language. In addition a formal execution model allows to reason about language properties and to check whether a process model possesses much desired temporal properties. The latter is the case for Petri nets and WorkflowNets, for instance. These languages have a algebraical notation that is useful for defining and proving formal properties of the language and for state space analysis. In this section, an execution model is defined for declarative process models that have been modeled using the EM-BrA^{2}CE Vocabulary.

6.1 Business Rules in the Activity Life Cycle

A common idea of declarative business process modeling is that a process is seen as a *trajectory* in a *state space* and that declarative constraints are used to define the valid transitions

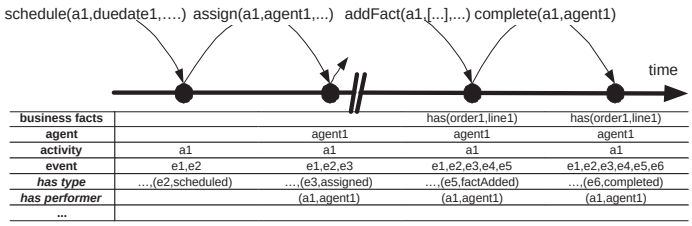
in that state space (Bider et al., 2000). Accordingly, in the EM-BrA^{2}CE Framework each business process can be **modeled** by describing the state space of each (composite) activity type and the set of business rules that constrain transitions in this state space. In terms of the EM-BrA^{2}CE Vocabulary the state space of an activity type can be described by a set of facts about the roles, sub-activity types (if any), business concept types, business fact types and event types that are relevant in describing the state of activities of the activity type.

The possible movements within a business process' state space, can be described by twelve generic activity state transitions that represent a change in the life cycle of an activity in a business process. Because these transitions are generic they provide a means of defining an **execution model** for the EM-BrA^{2}CE Framework. At each state a worker or coordinator might request a particular state transition to occur. This is modeled with the following state transition requests:

- $create(AId, AT, BId, PId, CoordinatorId)$: requests the creation of a new activity AId of type AT with business identifiers BId, parent activity PId by an agent $CoordinatorId$. Activity event types: *created*, *createRejected*.
- $schedule(AId, DueDate, CoordinatorId)$: requests the due date of activity AId to be set to $DueDate$ by an agent $CoordinatorId$. Activity event types: *scheduled*, *scheduleRejected*.
- $assign(AId, AgentId, CoordinatorId)$,$revoke(AId, AgentId, CoordinatorId)$: requests the assignment or revocation of the assignment of activity AId to an agent $AgentId$ by an agent $CoordinatorId$. Activity event types: *assigned*, *assignRejected*, *revoked*, *revokeRejected*.
- $start(AId, WorkerId)$: requests an activity AId to start by an agent $WorkerId$. Activity event types: *started*, *startRejected*.
- $addFact(AId, C, WorkerId)$, $removeFact(AId, C, WorkerId)$, $updateFact(AId, C_1, C_2, WorkerId)$: requests the addition or removal of business fact C or the update of a business fact C_1 by C_2 within the context of activity AId by an agent $WorkerId$. Activity event types: $factAdded$, $factUpdated$, $factRemoved$, $addFactRejected$, $updateFactRejected$, $re- moveFactRejected$.
- $complete(AId, WorkerId)$: requests the completion of activity AId by an agent $WorkerId$. Activity event types: *completed*, *completeRejected*. Upon completion of an activity, all business fact manipulations are committed to change to globally visible business facts.
- $skip(AId, CoordinatorId)$, $abort(AId, CoordinatorId)$, $redo(AId, CoordinatorId)$: requests to skip, abort or redo an activity AId by an agent $CoordinatorId$. Activity event types: *skipped*, *skipRejected*, *aborted*, *abortRejected*, *redone*, *redoRejected*.

Figure 13 illustrates a number of state transitions that occur to a place order activity a1. Each state transition results in a new set of concepts and ground facts, and thus a new state, that are partially represented in the columns of the figure. As each new activity state is considered to be a new SBVR:conceptual model, deductive reasoning can use a monotonic reasoning paradigm (Object Management Group, 2006b, p. 77). The current state of an activity determines which state transitions can occur. These state transitions might be subject to business and non-business concerns. The Petri net of Figure 14 models the allowable sequences of transitions in the activity life cycle as imposed by non-business

concerns. In the next section the semantics of the state transitions with regard to their effect on the state of an activity are explained in detail using a CP-Net model.



business facts			has(order1,line1)	has(order1,line1)
agent		agent1	agent1	agent1
activity	a1	a1	a1	a1
event	e1,e2	e1,e2,e3	e1,e2,e3,e4,e5	e1,e2,e3,e4,e5,e6
has type	...,(e2,scheduled)	...,(e3,assigned)	...,(e5,factAdded)	...,(e6,completed)
has performer		(a1,agent1)	(a1,agent1)	(a1,agent1)
...				

Figure 13: *An illustration of the state transitions of a place order activity a1*

These twelve transitions represent work coordination work, and exception handling. The *create*, *schedule*, *assign*, and *revoke* transitions represent coordination work that is to be executed by a coordinator agent as part of constructing an execution plan. The *start*, *addFact*, *removeFact* and *updateFact* transitions represent the actual work that is to be executed by a worker agent. The *skip*, *abort* and *redo* transitions represent the coordination work related to exception handling.

Business rules constrain the transitions in a state space. Informally, it suffices to check prior to the occurrence of a state transition whether relevant business rules will be violated or not. When no business rule is violated, the state transition can take place. When, on the other hand, the transition would lead to an intolerable violation of a business rule, the state transition is prevented from taking place. In each state an agent might request a particular state transition to occur. Table 3 indicates which business rule types constrain which state transition types.

6.2 A CP-Net-based Execution Model

In this section a formal execution semantics for the EM-BrA^{2}CE framework is provided in terms of timed Colored Petri Nets (CP-Net). There are several reasons for choosing CP-Nets. First of all, CP-Nets have a formal semantics (Jensen, 1993, 1996). Furthermore CP-Nets represent an expressive, high-level modeling language that portrays more modeling convenience compared to, for instance, classical Petri nets. Although each CP-net can be translated into a classical Petri net and vice versa, this does not guarantee the suitability of Petri Nets for modeling in practice (Jensen, 1993). In particular, it is difficult to model data manipulations with classical Petri nets, not allowing for token colors. Another reason for using CP-Nets is that CP-Net models can be simulated, making CP-Nets suitable for rapid prototyping process models and for generating artificial data sets of event logs that can later be used to evaluate the performance of process mining algorithms (Goedertier et al., 2007b). Additionally, CP-Nets allow for formal state space analysis that would, in theory, allow for directly analyzing the state space of individual declarative business process models. However, the inclusion of fact-oriented case data and event history into the state space of process models can be expected to result in too large a state space for analyzing realistic models. Consequently, reduction techniques would have to be put in place to reduce the state space into a state space of interest.

Jensen (1996) provides an extensive introduction to the semantics and analysis methods of CP-Nets. Throughout this section the semantics of CP-Nets in terms of their differences to classical Petri nets will be informally discussed whenever a new language construct is encountered.

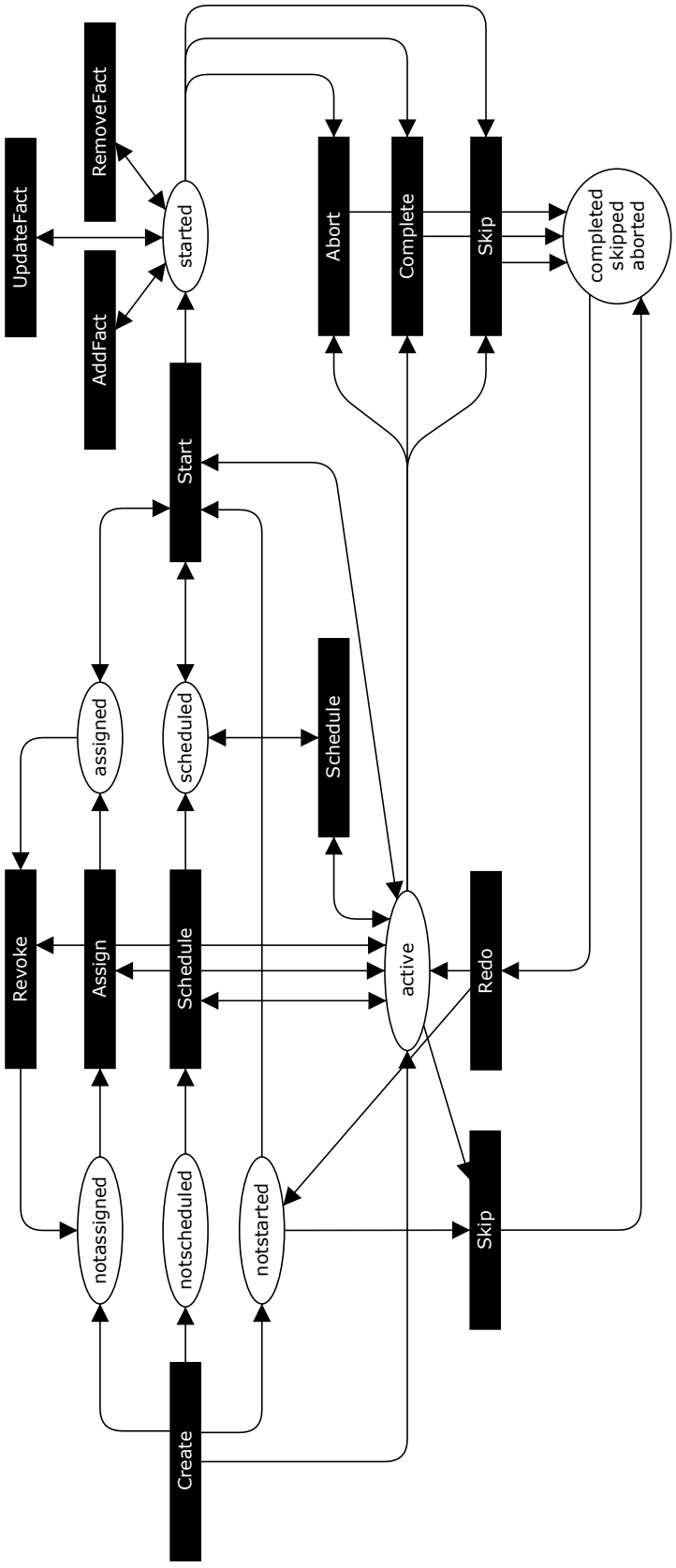


Figure 14: *A CP-Net model of the allowable activity state transitions in the EM-BrA^{2}CE execution model*

Table 3: *Relating transition types to business rule types*

	create	schedule	assign	revoke	start	addFact	removeFact	updateFact	complete	skip	abort	redo
Temporal deontic rule		x			x							x
Activity precondition					x							x
Activity postcondition									x			
Dynamic integrity					x	x	x	x				x
Activity cardinality	x				x					x		x
Serial activity constraint		x			x							x
Activity order		x			x							x
Activity exclusion	x				x							x
Activity inclusion									x			
Reaction rule	x				x							x
Static integrity						x	x	x				
Derivation rule						x		x				
Activity authorization			x									
Activity allocation rule			x									
Visibility constraint												
Event subscription												

6.2.1 Places and Color Sets

Just as in classical Petri nets the **state** or **marking** of a CP-Net is represented by the tokens that reside in each of the **places** of the CP-Net at a particular moment. Unlike classical Petri nets, however, CP-Nets allow to associate a data type, called a **token color**, to each place in the CP-Net such that only tokens of an indicated token color may reside in that place. It is possible for places to contain tokens prior to the occurrence of any state transition in the net. Such a start state or **initial marking** can be defined in terms of initialization expressions for a particular place. The state space of an EM-BrA^{2}CE process model can be modeled using four places, depicted in Figure 15:

- an `agent` place of color `AGENT` of which the tokens represent the agents that can coordinate or perform activities. A domain specific function `initAgents()` can be used to define the agents that are initially present.
- an `activity` place of color `ACTIVTY` of which the tokens represent the (composite) activities that are coordinated or performed by agents. The initial marking this place consists of a so-called `rootActivity`, that is parent to all other activity instances.
- a `businessfacts` place of color `FACTLIST` that holds one list token representing a list of business facts that can be manipulated by performing activities. A domain specific function `initFacts()` can be used to define the initial business facts, such as properties of agents, that are present in the system.
- an `eventhistory` place of color `EVENTLIST` that holds one list token representing an ordered list of historic events that have taken place throughout the life cycle of individual activity instances. This place contains the empty list `[]` as initial marking.

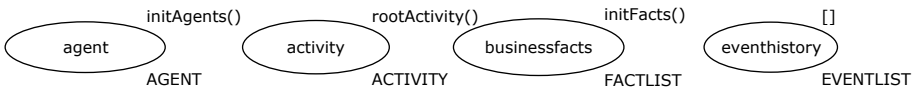


Figure 15: *The CPN places that span the state space of declarative business process models*

This representation depicts how state in the EM-BrA^{2}CE is constituted of a current state of affairs represented by the `agent`, `activity` and `businessfacts` places and a history of past business events, represented by the `eventhistory` place. There is some redundancy in this conception of business process state: the current state of affairs can always be obtained by "replaying" the history of past business events. However, this redundancy can be introduced without loss of generality. Moreover, not having to calculate the current state of the process model facilitates process modeling and improves efficiency of simulations.

In the source code below this paragraph, the above token colors are defined in terms of Standard ML (Milner et al., 1990), a functional programming language with compile-time type checking and type inference. The fact-oriented metamodel of the EM-BrA^{2}CE Vocabulary is translated into the color sets `FACT`, `CONCEPT` and `AGENT`, which are here treated as synonyms. These color sets represent a quadruple consisting of a statement identifier (for reification purposes), a subject identifier, a predicate representing one of the fact types and a value. For instance, the fact that a worker `workerX` belongs to department `departmentY` is now represented as a quadruple `(statementZ,workerX,fromDepartment,departmentY)`. To express the existence of an individual concept `workerX` of concept type `agent` the following quadruple can be constructed: `(statementZZ,workerX,has as type,agent)`. This form of knowledge representations corresponds to RDF with reification (W3C, 2004), one of the foundation languages of the Semantic Web. Notice treating `FACT` and `CONCEPT` as synonyms is a simplification of the SBVR ontology language. However, the simplification is only a limitation in the context of higher-order typing. The idea of representing agents as tokens in a CP-Net is, among other, present in van der Aalst's (1998) representation of workflow.

```
colset AGENTid = int with agentL..agentU;
colset CONCEPTid = int with cL..cU;
colset VALUE = union nb:INT + st:STRING + id:CONCEPTid;
colset NOUNCONCEPTTYPE = subset STRING with [...];
colset VERBCONCEPTTYPE = subset STRING with [...];
colset FACTTYPE = union nount:NOUNCONCEPTTYPE + verbt:VERBCONCEPTTYPE;
colset FACT = product       (*the statement id*)        CONCEPTid *
                            (*the subject id*)          CONCEPTid *
                            (*the predicate*)           FACTTYPE *
                            (*the object/value*)        VALUE;
colset FACTLIST = list FACT;
colset CONCEPT = FACT;
colset AGENT = CONCEPT;
```

The `ACTIVITY` color set is a septuple composed of an integer that denotes the non-business activity identifier, an activity type, a business id that is a list of business concepts that uniquely identify the activity, an activity identifier that denotes the immediate parent activity, an agent identifier that denotes agent that is currently assigned as the coordinator or performer of the activity, a time indication that denotes the due date by which the activity is to be performed, an event list that keeps track of the concept manipulation events that have occurred within the context of the activity. In addition activity is defined as a timed token. This allows to model time evolution. The idea of representing individual activities

as tokens in a CP-Net is based on Guenther and van der Aalst's (2005) representation of Case Handling in CP-Nets.

```
colset ACTIVITYid = int with aL..aU;
colset ACTIVITYTYPE = subset STRING with [...];
colset ACTIVITY = product   (*the activity id*)         ACTIVITYid *
                            (*the activity type*)       ACTIVITYTYPE *
                            (*the business id*)         FACTLIST *
                            (*the parent id*)           ACTIVITYid *
                            (*the coordinator/worker*)  AGENTid *
                            (*the due date*)            TIMESTAMP *
                            (*transaction events*)      EVENTLIST
                            (*timed token*)             timed;
```

The EVENT color represents the activity events that have occurred. It is a septuple represented as the cardinal product of an activity identifier, an activity type, a business id that is a list of business concepts that uniquely identify an activity, an event type denoting the nature of the activity state transition, an agent identifier that denotes the agent who has brought about the state transition, a list of facts that specify the event and a time indication that denotes the time at which the state transition has occurred. The idea of incorporating history into an event history token stems from van Hee et al. (2006b), Although the authors do not propose to incorporate activity events but rather propose to incorporate the consumption and production of tokens in each input and output place as events.

```
colset EVENTTYPE = subset STRING with [ "created","createRejected",
                                        "scheduled","scheduleRejected",
                                        "assigned","assignRejected",
                                        ...,
                                        "completed","completeRejected"];
colset EVENT = product  (*the activity id*)             ACTIVITYid *
                        (*the activity type*)           ACTIVITYTYPE *
                        (*the business id*)             FACTLIST *
                        (*the event type*)              EVENTTYPE *
                        (*the coordinator/worker*)      AGENTid *
                        (*event parameters*)            FACTLIST *
                        (*the time of occurrence*)      TIMESTAMP;
colset EVENTLIST =list EVENT;
```

Because each place in a CP-Net can contain multiple tokens, possibly of the same token color, the content of a place can be represented as a **multi-set**. This might raise the question why both the businessfacts and the eventhistory place consist of exactly one **list token** containing an ordered list of tokens. The reason for this modeling feature, is that it cannot be foreseen at design-time how many tokens will actually be required in order to fire a transition. For instance, it is not possible to foresee which and how many tokens representing business concepts, business facts and events actually need to be inspected when deciding upon assigning an agent to a particular activity. In addition, to trigger a transition upon the non-presence of a token without using inhibitor arcs also requires this modeling feature (Mulyar and van der Aalst, 2005). To overcome this problem, it is required to take all business concept, business fact and event tokens into consideration by removing a list with all tokens from both places, querying this list and returning a potentially updated list of tokens upon termination of a transition.

6.2.2 The Create transition

As in classical Petri nets, the dynamics of CP-Nets come from **transitions** (represented as rectangles). Places in CP-Nets are linked to transitions via input and output arcs. **Input arcs** indicate that a particular transition may remove (consume) tokens from a particular

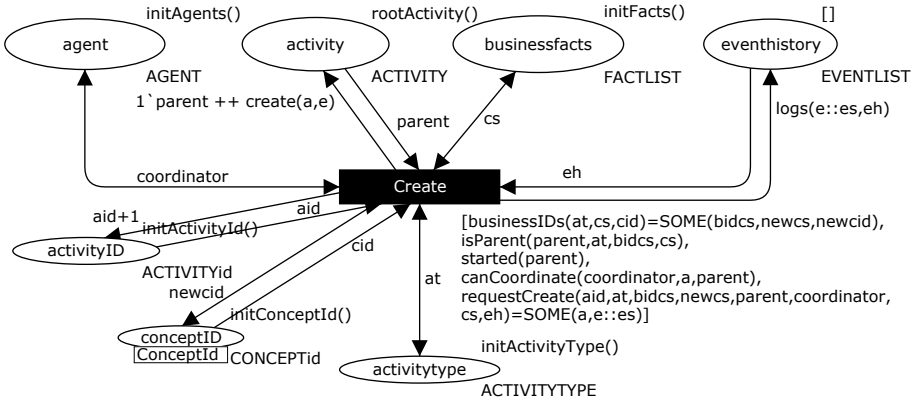


Figure 16: *A CP-Net model of the Create transition*

place, whereas a **output arcs** indicate that a transition may add (produce) tokens to a particular place. Informally, a CP-Net transition may fire (**fire rule**) when on each input place a required number of tokens can be found that together satisfy the **guard condition** of that transition. As a result of firing, a transition consumes a number of tokens on each input place and produces a number of tokens on each output place. The value and amount of tokens consumed and produced in place can be manipulated through the use of **arc expressions**.

The `Create` transition, depicted in Figure 16, is one of the most complex transitions in an activity life cycle as it involves many aspects:

- the determination of the activity type
- the determination of the activity identifiers
- the determination of the parent activity
- the determination of the agent who coordinates the activity
- establishing whether the activity can be created
- logging the creation of an activity as an event in the event history

In the remainder of this section these aspects are discussed consecutively.

The determination of the **activity type** of the activity that is about to be created can be modeled by consuming and producing an `ACTIVITYTYPE` token from the `activitytype` place and binding it to the variable `at`. As the same token is consumed and produced by the `create` transition, the incoming and outgoing arc are replaced by a **bidirectional arc**, that is both an input and an output arc. Because the `ACTIVITYTYPE` token is not timed, it can be consumed at the same modeling time for the creation of other activities. This pattern applies to the whole CP-Net, such that it allows for the desired behavior that activity state transitions can occur concurrently.

In the EM-BrA^2CE Vocabulary activities have two identifiers: a non-business identifier and a business identifier. The determination of a **non-business identifier** is modeled using the "ID Manager" Pattern (Mulyar and van der Aalst, 2005). Initially the `activityID` place stores one token with an integer value determined by `initActivityID()`. Each time the `Create` transition fires, the token is consumed and used as an identifier for the activity to be created. In addition, the incremented integer value is produced on the output arc. By

incrementing and memorizing the last identifier value, the uniqueness of the non-business identifier can be guaranteed.

The determination of an activity **business identifier** is more complex as it involves real-world business concepts. In some cases, the creation of a activity will (in part) involve the creation of a new business concept whilst in other cases the creation of an activity only involves the identification of existing business concepts. For instance, when a customer applies for credit, the activity `applyForCredit` might be identified by a new business concept of type `creditApplication` representing the new credit application. In contrast, when a customer wants to modify the requested duration of the credit, the activity `requestChange` might be identified by the already existing `creditApplication` business concept. This complexity is encapsulated within the guard condition `businessIDs(at,cs,cid)= SOME(bidcs,newcs,newcid)`. The function `businessIDs(at,cs,cid)` takes as input the variable `at`, a list of existing business concepts, bound to the variable `cs` and a unique identifier for new business concepts to be created, bound to variable `cid`. On its output the `businessIDs` either returns `NONE` when it fails to determine suitable business identifiers or a triple `SOME(bidcs,newcs,newcid)` with respectively the existing and newly generated business concepts and `newcid` with a properly incremented value of `cid`.

Some activities can exist only within the life cycle of a (composite) **parent activity**, whereas other activities can also be created independent from a parent activity. The latter activities have `rootActivity` as their immediate parent. The logic of determining a proper parent for an activity to be created is encapsulated in the `isParent(parent,at,bidcs,cs)` guard condition. The input variable `parent` is bound to an `ACTIVITY` token from the `activity` place. Additionally, for a child activity to be created, it required that the coordinating parent activity has already started. This is expressed with the `started(parent)` guard condition.

Upon creation of an activity is is unclear who will eventually perform the activity until an activity is assigned to a particular agent. Until that period, the accountability for an activity can be attributed to the **coordinating agent** (human or system) who has created the activity. In order to determine the coordinating agent an `AGENT` token is consumed and produced from the `agent` place and bound to the `coordinator` variable. When activities are created within the context of a parent activity, the agent assigned to the parent activity is also the coordinator of the new child activity. For instance, when `agentX` has been assigned to the parent activity `handleCreditApplication` he is also identified as the coordinator of a newly created child activity `reviewCredit`. When, in contrast, an activity has the `rootActivity` as parent, an agent can only create an activity when it has to role of an agent who can coordinate the particular activity. This logic is concealed in the `canCoordinate(coordinator,a,parent)` guard condition.

Additionally, a set of domain-specific business rules might determine whether the specified activity can be **created** or not. The guard condition

```
requestCreate(aid,at,bidcs,newcs,parent,coordinator,cs,eh)=SOME(a,e::es)
```

is a hook that allows for the evaluation of business rules to check whether an activity with the specified features can be created. The creation of an activity might depend on the evaluation of strict business rules (hard constraints) and upon the freedom of choice of agents not to follow general guidelines (soft constraints) with regard to the creation of a particular activity. When no activity is to be created the function returns either `NONE` resulting in a failure of the guard condition or `SOME(a,e::[])` for which `e` is bound to an event of type `createRejected`. When on the contrary an event must be created the function returns `SOME(a,e::es)`. In this case `a` is bound to a new activity token that is produced in place `activity`. In addition, `e` is bound to an event of type `created` and `es` represents a list

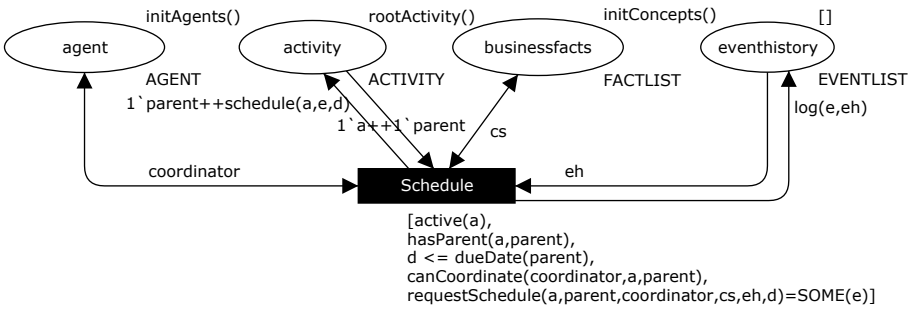


Figure 17: *A CP-Net model of the schedule transition*

of events that represents the newly created business concept identifiers (if any). The arc expression `1‘parent ++ create(a,e)` returns the parent token and, in the case of a successful `created` event, a new activity token `a`. This activity token also receives a token time that indicates the earliest time at which another activity state transition can occur. In this way, time can be incorporated into the model. The events `es` are added as transaction events to the newly created activity. Only when the activity completes these transaction events are committed and affect the list of business concepts and facts `cs`.

Via the arc expression `logs(e::es,eh)` multiple events related to the `Create` transition are incorporated in the **event history**, bound to the variable `eh`. The event `e` is of the event type `created` or `createRejected`. Additionally, the events `es`, related to the newly created identifying business concepts also need to be added to the event history.

6.2.3 The Schedule transition

Time aspects are often an important aspect in the coordination of activities. An important timing aspect is the time point at which an activity is expected to be completed. In the EM-BrA^{2}CE Vocabulary this moment is called the **due date** of an activity. Due dates on activities originate from (informal) service-level agreements, legal requirements or in-house timing policies and strategic plans. When it becomes clear that an activity is not going to be fulfilled before due date, coordinating agents must make alternative arrangements to speed up the processing of the activity or the minimize the consequences of tardiness. Making these arrangements is called **deadline-based escalation** by van der Aalst et al. (2007). The authors classify a number of escalation mechanisms that can be put in place and evaluate these mechanisms by means of simulation.

Figure 17 depicts the `Schedule` transition. In this transition the following aspects are contained:

- the identification of the parent activity
- the identification of an agent who can coordinate the activity
- establishing whether the specified schedule does not violate any business constraints or guidelines
- logging the scheduling of an activity as an event in the event history

In the remainder of this section these aspects are discussed consecutively.

In general, the life cycle of child activities is contained within the life cycle of a **parent** activity. This means that activities are created only when their immediate parent activity

has already started. Furthermore, a child activity must be completed prior to the completion of its parent activity. This conception of composite activities has an important consequence for scheduling activities: an activity cannot be scheduled with a due date later than the due date of its parent. Both the activity to be scheduled `a` and its parent activity `parent` are consumed from the `activity` place. Guard condition `hasParent(a,parent)` ensures that `parent` is bound to the parent activity of `a`. Furthermore, guard condition `d <= dueDate(parent)` ensures that the chosen due date `d` occurs before the due date of the parent activity.

The agent who will schedule the activity, the **coordinator**, is identified via the guard condition `canCoordinate(coordinator,a,parent)`.

To check whether the chosen due date `d` does not violate any hard **business constraints** and to model the freedom of choice with respect to observing soft **business guidelines** regarding deadlines, the `Schedule` transition has a `requestSchedule(a,parent,coordinator,cs,eh,d) = SOME(e)` guard condition. Only when the function `requestSchedule(a,parent,agent,cs,eh,d)` returns a `scheduled` event, the arc expression `1`parent++schedule(a,e,d)` will produce a parent token and an activity token with updated due date. When the function returns a `scheduleRejected` event, the activity token `a` will not be updated. When the function returns `NONE`, the transition will not take place.

When the state transition occurs, either a `scheduled` or a `scheduleRejected` event is to be incorporated within the event history. For optimization purposes, it can still be possible to leave out some of the activities in the event history. These considerations are however, not relevant to the discussion of the semantics of EM-BrA^{2}CE.

6.2.4 The Assign and Revoke transition

The coordination of activities also requires a coordinating agent to assign the activity to an agent who will perform the activity. Conversely, when an agent can no longer perform an activity as planned, the assignment must be withdrawn. This is modeled in the CP-Net by means of the `Assign` and, its counterpart, the `Revoke` transition depicted in Figure 18. The following aspects are contained:

- the identification of the coordinating agent and the agent who will be assigned to the activity
- checking whether the assignment or revocation does not violate any business constraints or guidelines
- logging the assignment of an activity as an event in the event history

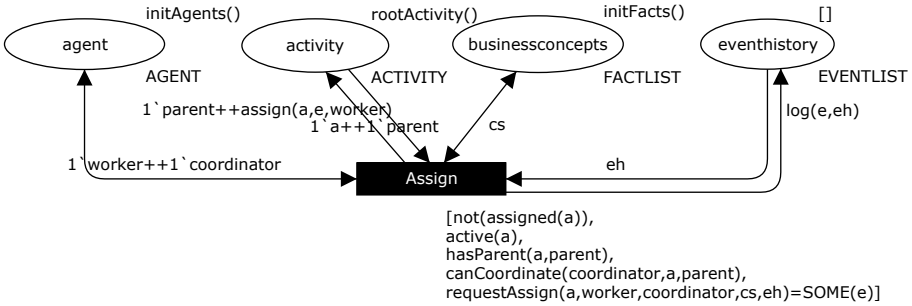
In the remainder of this section these aspects are discussed consecutively.

For an agent to be assigned to an activity, the activity must not yet been assigned. Conversely, for an assignment to be revoked, the activity must have been assigned to an agent. This is respectively expressed by the `not(assigned(a))` and `assigned(a)` guard conditions.

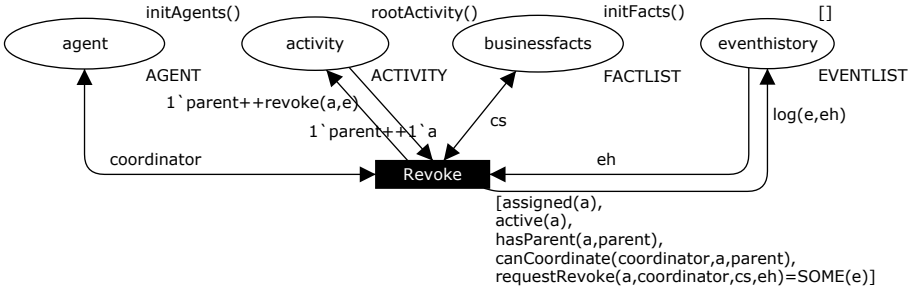
To identify an agent that can coordinate the assignment or revocation of a worker to a activity, an `AGENT` token must be bound to the `coordinator` variable that satisfies the guard condition `canCoordinate(coordinator,a,parent)`. To identify a worker agent, an `AGENT` token must be bound to the `worker` variable in the assign transition.

To check whether an assignment or revocation does not violate any business constraints or guidelines, respectively the

`requestAssign(a,worker,coordinator,cs,eh)=SOME(e)`



(a) the Assign transition



(b) the Revoke transition

Figure 18: *A CP-Net model of the Assign and Revoke transition*

and

```
requestRevoke(a,coordinator,cs,eh)=SOME(e)
```

have been conceived. The variable `e` is bound to a event of type `assign`, `assignRejected` or a `revoked`, `revokeRejected` respectively. In function of the event type, the `assign(a,e,worker)` and `revoke(a,e)` arc expressions update the state of the activity bound to `a` or make no activity state update.

6.2.5 The Start transition

Once an activity has been assigned to a worker agent, the activity can finally start. This start of an activity marks the moment from which a worker can actually perform operations that affect the environment or that collect information about the environment. Figure 19 represents the logic of the `Start` transition. As expressed by the `not(started(a))` guard condition, an activity can only start if it has not yet started. Moreover, the worker who triggers the start transition must be a worker that has been previously assigned to to the activity, as expressed by the `assignedTo(worker,a)` guard condition. The guard condition `requestStart(a,worker,cs,eh)=SOME(e)` is once again used to check whether the starting of an activity violates any existing business rules.

6.2.6 The fact manipulation transitions

During the performance of an activity a worker can bring about some changes in the environment (the physical world) or retrieve some information from the environment. These changes are reflected in the manipulation (the **addition, removal or update**) of business concepts and facts that pertain to the state space of an activity. Business concepts and facts, however, can be shared among multiple activities that reside within the (information)

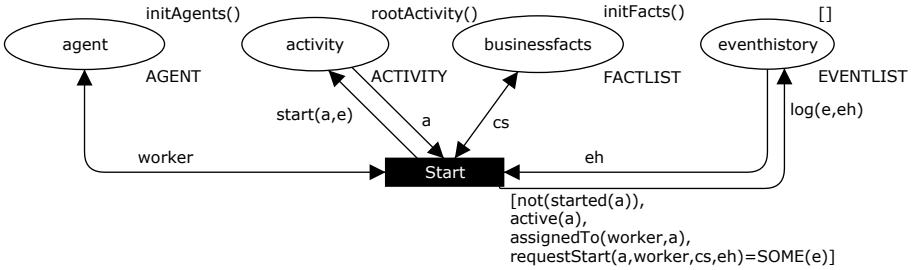


Figure 19: *A CP-Net model of the Start transition*

system. Therefore such manipulations can be treated in two different ways: a stateless and a stateful approach. In a **stateless approach** each manipulation is immediately proliferated to the entire system to alter the state of all other concurrent activities. Because the state of activities does not differ from the state of the information system, this is called a stateless approach. In a **stateful approach** manipulations immediately affect the state of the activity, but manipulations are only carried through (committed) once the activity has completed. Case handling, for instance, uses a stateless approach (van der Aalst et al., 2005), whereas BPEL4WS uses a stateful approach (Curbera et al., 2003). Although both approaches are equally meaningful, only the more complex stateful approach is modeled in the CP-Net model of the data manipulation transitions.

Many different transaction handling mechanisms can be put in place to guarantee the integrity of data manipulations, for example expressed in terms of "ACID properties" (however it is not always clear whether each ACID property is meaningful in a collaborative environment (Dumas et al., 2005)). A solution based on locking, for instance, is the distributed **Two-Phase Commit** (2PC) protocol (Bernstein and Goodman, 1981). Case handling tools such a FLOWer implement a 2PC transaction handling protocol. However, the required locking of business facts during the long-running active part of an activity life cycle is often seen as too restricting, inhibiting the concurrency of activities within business processes. A solution to the concurrency problem is, for instance, offered by the **Tentative Hold Protocol** (Roberts and et al., 2001). This protocol allows for tentative, non-blocking holds or reservations to be requested when starting an activity. When a worker has manipulated a business concept or fact in the contexts of performing an activity, other workers that have taken reservations on this business concept are signaled that their reservations do no longer hold and arisen conflicts are solved. Although transaction handling is a necessary requirement, for reasons of clarity it has been left out of the CP-Net model of the EM-BrA^{2}CE activity life cycle.

Figure 20 represents a CP-Net model of the `AddFact`, `RemoveFact` and `UpdateFact` state transitions. The following aspects are contained in the model and are consecutively discussed in the remaining paragraphs of this section:

- determining a unique concept identifier (statement id) in the case of addition
- identifying the business concept whose property is being manipulated (subject id)
- identifying the concept type of the fact to be manipulated (predicate)
- identifying the new value of the concept to be manipulated (concept value)
- identifying the worker as the worker that is assigned to the activity
- checking whether the manipulation does not violate any business rules

- logging the manipulation event and supplementing it to the activity transaction list (stateful approach)

When a new concept or fact is added during the performance of an activity, this concept or fact must be asserted to the system using a globally **unique identifier** identifying the concept or fact as a reified statement (a statement id). The determination of such a unique concept identifier is modeled using the "ID Manager" Pattern (Mulyar and van der Aalst, 2005) and involves place `conceptid`, that forms a **fusion place** that shares the same tokens with the previously discussed `businessID` place. In principle another globally unique identifier is required when adding a noun concept. However, without loss of generality, it can be assumed that a noun concept identifier is equivalent to the identifier of the statement that asserts its existence.

The addition, removal or update of a business concept or fact requires the identification of the **concept type** of the fact that is being manipulated. This is modeled with the `concepttype` place, from which a `CONCEPTTYPE` token is taken and bound to the `ct` variable. In the EM-BrA^{2}CE Vocabulary concept or facts can only be manipulated by an activity of a particular activity type in two cases: when either their concept type pertains to a concept type that has been modeled as an input concept type or when the internal functioning of the activity can manipulate a derived concept, without it being explicitly provided as an input by a worker. These concerns are captured by the `hasInputType(activityType(a),ct)` and `canModifyFactType(activityType(a),ct)` guard conditions.

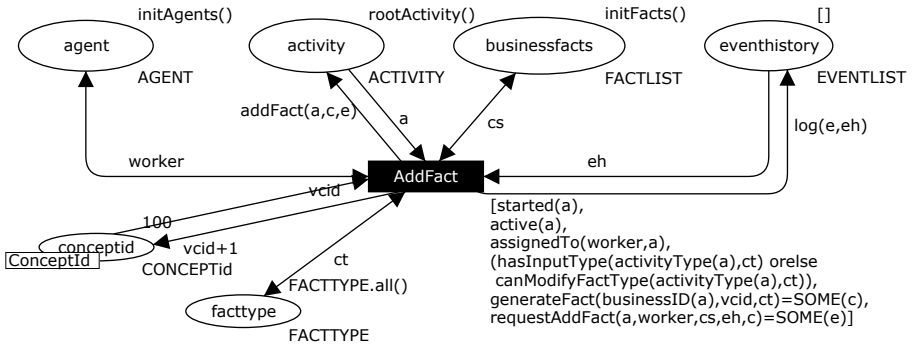
When adding or updating a concept or fact a new **value** is to be generated. Such a value of type `VALUE` can either be a number, a string or an identifier. Two non-business requirements can be formulated for concept values. First, both string and number values must pertain to the domain of the concept type. In addition, values that refer to the business id (subject id) of other business concepts must guarantee referential integrity. These concerns are present within the `generateFact(businessID:CONCEPTLIST,vcid:CONCEPTid,ct:CONCEPTTYPE)` function.

The manipulation of business concept facts might lead to the violation of hard and soft business constraints that are specified on these business concept facts. For instance, during a `requestChange` activity, a registered customer might remove earlier provided collateral information involving an open `creditApplication`. This might trigger a warning message (soft or hard business constraint) that a `creditApplication` business concept is incomplete without collateral information. The `requestAddFact(a,worker,cs,eh,c)`, `requestRemoveFact(a,worker,cs,eh,c)` and `requestUpdateFact(a,worker,cs,eh,c1,c2)` allow for checking compliance to such business rules.

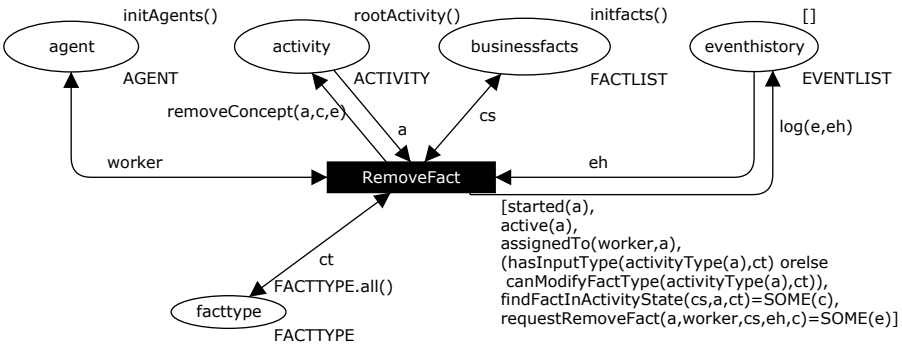
The arc expression `log(e,eh)` **logs** the data manipulations as activity events in the event history. Nonetheless, data manipulations are not yet forwarded to affect the business facts and concepts visible in the context of other activities. Instead data manipulations are only carried through once the involved activity has properly completed (stateful approach). Meanwhile every data manipulation event `e` is added to the transaction list of an activity `a` by the functions `addFact(a,c,e)`, `removeFact(a,c,e)` and `updateFact(a,c1,c2,e)`. This transaction list is modeled as a list of data manipulation events and is included in the color set of the `ACTIVITY` token. Upon completion of the activity, the transaction list is committed to affect the globally visible business concepts and facts.

6.2.7 The Complete transition

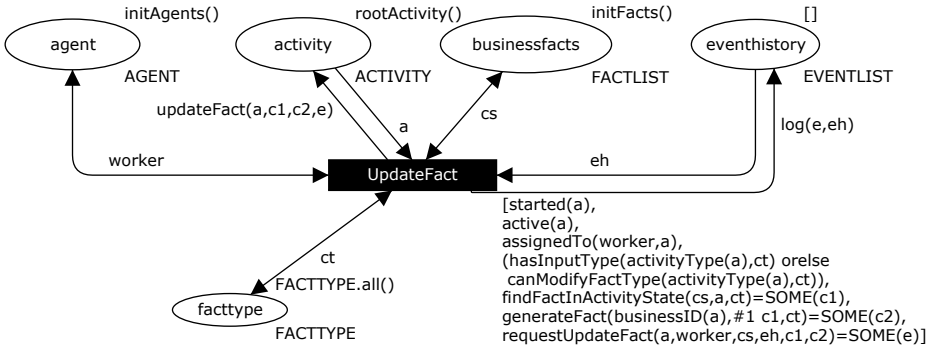
When a worker completes an activity, the work represented by the activity is considered to be completed and all data manipulations are proliferated to the entire system. After



(a) the AddFact transition



(b) the RemoveFact transition



(c) the UpdateFact transition

Figure 20: *A CP-Net model of the fact manipulation transitions*

completion, it is no longer possible for a worker to do supplemental business fact or concept manipulations without reopening the activity instance (modeled as a `Redo` transition). Figure 21 represents a CP-Net model of the `Complete` transition. The following aspects are addressed:

- identifying the worker as the worker that is assigned to the activity
- verifying that the activity has no active child activities
- verifying whether the completion of the activity does not violate any hard or soft business constraints
- committing the business fact manipulation transaction list to update the system's state accordingly
- logging the completion event and removing any activity events from the event history that are not able to affect the life cycle of other activities (event garbage collection).

An activity can only complete if it has been previously started. This is expressed with the `started(a)` guard condition. The worker who decides upon completing an activity must be a worker that has been previously assigned to to the activity, as expressed by the `assignedTo(worker,a)` guard condition. The guard condition `requestComplete(a,worker,cs,eh)=SOME(e)` is once again used to check whether the starting of an activity violates any existing business rules.

The work of coordinating a number of activities within a business process is modeled as a (composite) **parent** activity. Consequently, for a child activity to be created, it required that the coordinating parent activity has already started. Conversely, for an activity to complete it is required that the coordinating parent activity is still active. The latter requirement is verified with the `noActiveChildren(a,eh)` guard condition.

As discussed in the previous section, the CP-Net model contained in this text models the stateful approach with respect to business fact manipulation. During the execution of an activity each business fact manipulation is added to a transaction list, modeled as a list of data manipulation events included in the color set of the `ACTIVITY` token. Only upon completion of an activity the transaction list is **committed** to affect the globally visible business facts. This operation is modeled with the `commit(cs,a,e)` arc inscription.

The completion of an activity can trigger the creation or start of subsequent activities that are temporally dependent upon the activity. In that case the completion of the activity is a valuable event that needs to be retained in the event history. Other activity events (such as events of type `created` or `scheduled`) are perhaps less valuable, because it is detected that there exist no business rules that involve these activity events in relationship to life cycle events of other activity types. As such events are not able to affect the life cycle of other activities it is possible to remove them from the event history without side effects. This is called **event garbage collection**. These concerns are concealed in the `log(e,eh)` arc inscription.

6.2.8 The Abort, Skip and Redo transitions

In addition to creating, scheduling and assigning an activity, coordinating an activity could also involve the canceling, the imperfect (incomplete) termination and reopening of an activity. The latter activity life cycle events are respectively modeled as `Abort`, `Skip` and `Redo` transitions, depicted in Figure 22.

Sometimes it is necessary for a coordinating agent to cancel or **abort** an active activity or even an entire group of activities. Such cancelation might be part of natural behavior of

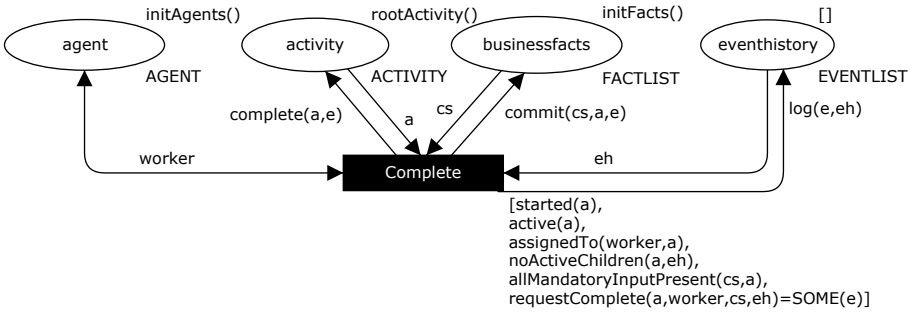
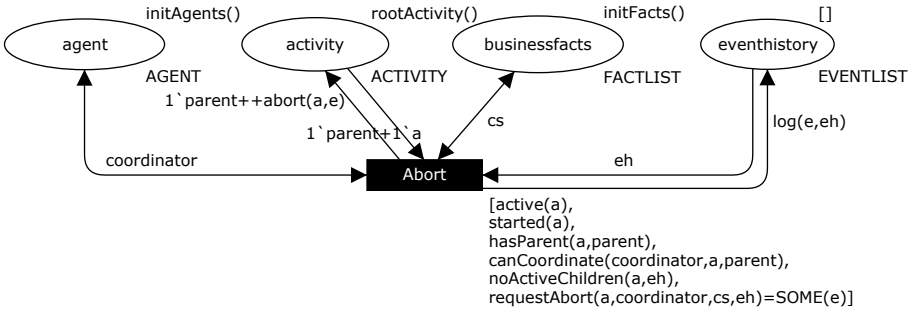
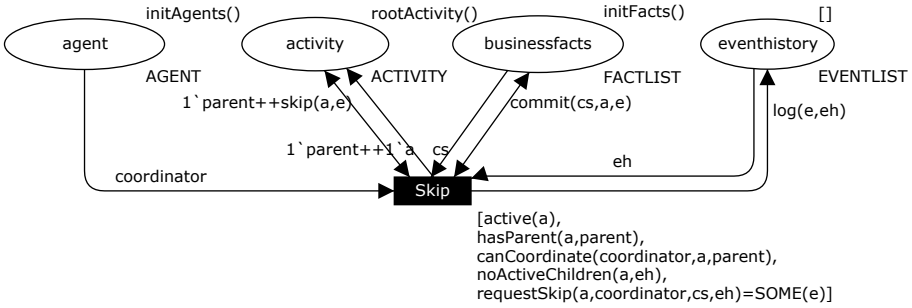


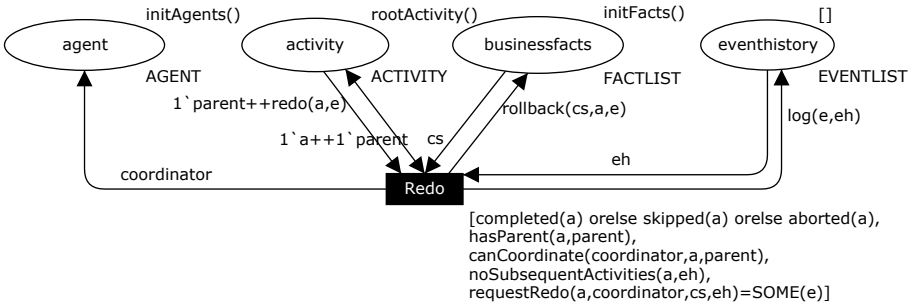
Figure 21: *A CP-Net model of the Complete transition*



(a) the Abort transition



(b) the Skip transition



(c) the Redo transition

Figure 22: *A CP-Net model of the Abort, Skip and Redo transition*

activities in other cases it might be required to abort an activity when an unforeseen exception occurs (Russell et al., 2006). For instance, in a purchase process it might be required to request a price quote with a minimum number of suppliers. When a required number of proposals are received from suppliers a purchase decision is made. At that moment any remaining active request for quote activities can naturally be canceled out. When, on the other hand, an external event would render the ongoing purchase unwanted, an exception has occurred and all activities within the purchase process must be aborted. An activity can only be canceled when it is in an active state, as modeled with the `active(a)` guard condition. Furthermore, a parent (composite) activity cannot be canceled when it still has active children, as modeled with the `noActiveChildren(a,eh)` guard condition. The semantics of a "cascading abort", in which all active children of a composite activity are aborted, cannot be directly modeled within the current CP-Net, as it cannot be foreseen how many activity tokens (children) need to be consumed from the `activity` place. However, such a "cascading abort" can be seen as the occurrence of multiple abort transitions within the CP-Net. Likewise, the CP-Net does not model so-called compensating activities – Although present in the EM-BrA^{2}CE Vocabulary. Compensation can be obtained by a combination of an `Abort` and `Create` transition. Unlike the `Complete` transition, the `Abort` transition does not commit the business fact manipulation events that pertain to the activity.

Skipping an activity is a form of imperfect completion of an activity, as it is not required that all mandatory business facts have been provided when skipping an activity. Unlike the `Abort` transition, the `Skip` transition does commit every business fact manipulation event that has occurred during the life cycle of an activity. This is modeled with the `commit(cs,a,e)` arc inscription. The possibility of skipping an activity stems from the case handling paradigm (van der Aalst et al., 2005) and allows a coordinator to by-pass an activity when it is no longer deemed required. In the case handling case, however, skipping means stepping over the entire activity without any case data being manipulated. Here skipping an activity can still involve the partial manipulation of business facts. Skipping provides a lot of flexibility as it enables a business process to bypass standard behavior as required without such a possibility being explicitly defined in the process model. For instance, when a customer requests information and preliminary, non-binding about a credit rate, it might be useful to perform a `makeProposal` activity without the customer even having identified himself. Such functionality could be provided, when a bank clerk can skip over a number of activities in the credit approval process.

After completing, aborting of skipping an activity, an activity can under some conditions be reactivated by performing a `Redo` operation. This transition has the effect that any previously committed business events are undone, this is modeled by the `rollback(cs,a,e)` arc expression. All previously committed business fact manipulation event that pertains to an activity's transaction list are retained. In this way, a worker can decide which data manipulation can be retained from previous executions and which require alteration. After alteration, a new list of business fact manipulation events can be committed upon completion of the activity. In the case handling paradigm redoing an activity changes the state of added case data from 'defined' to 'unconfirmed'. Although differently encoded, the semantics of the `Redo` transition closely resembles redoing an activity within the case handling paradigm. In addition to business rules constraining the redoing of an activity, an activity cannot be redone when any subsequent related activities have been using information that is being retracted from the system via the rollback operation. This is expressed in the `noSubsequentActivities(a,eh)` guard condition. When such subsequent activities are present, all of them need to be redone before the activity in question can be redone.

6.3 Unspecified semantics

Although the proposed CP-Net models a large proportion of the intended semantics of EM-BrA^{2}CE, not all aspects have been fully specified. For reasons of clarity or due to inherent limitations of CP-Nets a number of aspects have been consciously omitted. In this section these omitted aspects are briefly discussed.

6.3.1 Reactive Behavior

A disadvantage of CP-Nets is that it lacks the **reactive behavior** of an open system (Eshuis and Dehnert, 2003). In a CP-Net every state transition occurs within the model, whereas in reality state transitions occur on the initiative of an agent that resides in the environment of the modeled system. To synchronize state information both system and environment communicate by means of exchanging events. In reality, it is possible for reactive systems not being able to respond to their environment in a timely fashion. Since a Workflow Management Systems (WfMS) is also a reactive system that runs in parallel with its environment, it also risks to loose synchrony with its environment. Eshuis and Dehnert comment on the disability of Petri nets to model reactive behavior and contrasts Petri nets with the semantics of STATEMATE (Harel and Naamad, 1996). To make Petri nets reactive would require to model the environment in the Petri net as well and to distinguish between external transitions that represent state changes in the environment and internal transitions. Enabled internal transitions should always fire (must-fire rule) before enabled external transitions (may-fire rule). In this way, modeling checking would allow to test for compliance to the perfect synchrony assumption (Harel and Naamad, 1996), namely that the modeled system is always able to stay synchronized with the state of its environment. The consequences of the lacking reactive behavior of CP-Nets should however be put into the right perspective. First of all the possibility that a WfMS can go out of sync with its environment is a possibility that can be left out of consideration in specifying a semantics for EM-BrA^{2}CE. In addition Eshuis and Dehnert have shown that when modeling a system as a closed, active system soundness properties are preserved. Consequently, it is acceptable to include a number of events that would normally take place in the environment (such as the actions of external business process participants) within the boundaries of the CP-Net. Nonetheless, **timeout events**, an important category of events, cannot be incorporated within the closed system of a timed CP-Net. The reason is that model time is exogenous to the CP-Net and it cannot be used to conditionally fire a transition without provoking undesired side effects. Time-triggered activity life cycle operations such as deadline escalation can therefore not be explicitly included in the proposed model.

6.3.2 Transaction Handling

Although it is possible to include a **transaction handling** mechanism such as a Two-Phase Commit or Tentative Hold protocol within the CP-Net, such a mechanism was not included in the CP-Net. Transaction handling can be seen as a separate concern that can be added by keeping track of the locks or reservations that have been requested on particular business facts. Moreover, additional guard conditions have to be added that control the `Complete` and `Skip` transitions and model the resolution of potential integrity conflicts. This modifications are rather complex. Moreover choosing for one particular transaction handling mechanism is incompatible with the intent of the EM-BrA^{2}CE framework of integrating and developing new forms of declarative business process modeling.

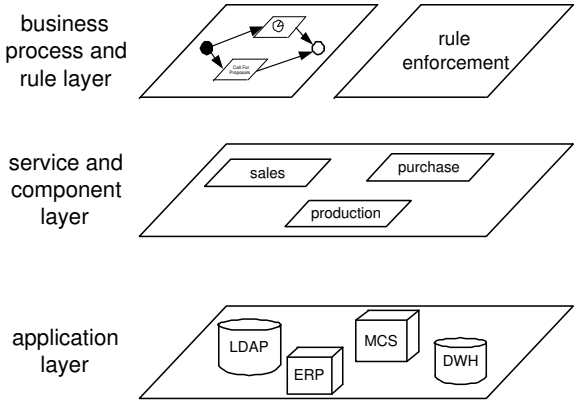


Figure 23: *Activity Coordination Service and Concept Service*

6.3.3 Composite State Transitions

Another part of intended semantics that has been left unspecified in the CP-Net is the presence of **composite state transitions**. Composite state transitions are state transitions that occur for a particular group of activities at the same time. "cascaded abort" and "cascaded redo" transition, for instance, are composite state transitions that represent the automatic aborting or redoing of all child activities when a composite activity is is being aborted or redone. A "delegate" transition, for instance, is the combination of a revoke and assign transition and represent the coordinating activity of a worker assigning an activity to another worker (Li et al., 2003; Zhang et al., 2003; Wainer et al., 2007). These transitions, however, can be omitted from the CP-Net without loss of meaning as they can be simulated by performing a number of subsequent atomic state transitions at the same model time.

7 Towards a Declarative Service-Oriented Architecture

Data, processes and logic are essential resources of any information system and can consequently be identified at any level of abstraction. For the purpose of declarative based process enactment, it is useful to consider these resources at the highest level of a service-oriented enterprise architecture (SOA) stack. Such an architecture stack, as displayed in figure 23, commonly consists of a number of layers. Applications and databases of one layer are concealed by the components and services of a higher layer. Services can be combined forming long-running business processes, which make up the highest layer.

'Service' is an overloaded concept that has several meanings. For the purpose of rule-based process enactment it is instructive to make a distinction between business services and software services. Business services belong to an Enterprise Model and can be modelled in terms of activity types (or service capability), agents (or service providers) and activities (or service instances). Software services are software artefacts that to some extent automate or support business services. It is useful to make a distinction between three kinds of software services that are represented as a UML component diagram in Figure 24:

- Service providers are software services that can perform activities or support the performance of activities. To each service provider an EM-BrA^{2}CE agent can be mapped. A service provider is a recursive structure: it can represent an entire organization as well as a particular department or individual worker. Service providers are notified of events to which they have event subscriptions. For instance, when an

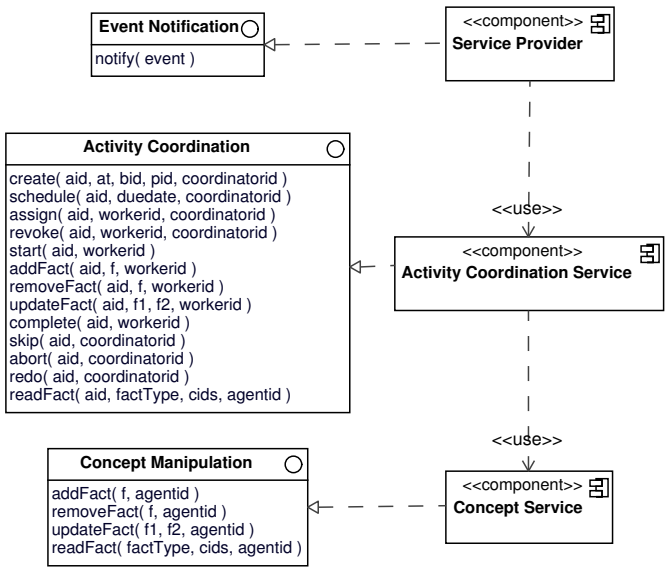


Figure 24: *Activity Coordination Service and Concept Service*

activity is assigned to an agent, the agent is notified of this event. Another example is the occurrence of an event to which an external business partner is subscribed. In addition, service providers can perform activities or can directly manipulate facts about (business) concepts. For instance, when an agent perceives or is notified of a relevant event that is not yet included in the fact base, the agent might add information about this event as facts to the fact base.

- Activity coordination services are stateless services that manage the state transitions of activities of given activity types. To each activity coordination service an EM-BrA^{2}CE activity type can be mapped. Each activity service has twelve generic operations that implement the twelve above-described state transitions. Activity coordination services manage the state and state transitions for the activities of a particular activity type and enforce the applicable business rules indicated in Table 1. Service providers can invoke activity coordination services to bring about an activity state change. Likewise, activity coordination services can invoke service providers to notify them of events to which they are subscribed.

- Concept services manage the state of the information system by providing access to facts about concepts (agents, activities, business concepts and events) and derive facts from concepts. To each concept service a (group of) EM-BrA^{2}CE concepts can be mapped. Concept services are used by service providers to request information about business concepts or to assert external events. For instance, a service provider might consult a concept service to learn about an activity to which it has been assigned. Concept services are also used by activity coordination services to query and to update the state of the activities that they manage. Concepts services enforce data visibility constraints and event subscription constraints that range over the concepts (agents, activities, business concepts and historic events) that it manages.

Figure 25 is a UML communication diagram that represents a simplified version of the interaction between several services that enact the payment-after-shipment business process of Figure 6(a). There are two service providers, aBuyer and aSeller, that represent two

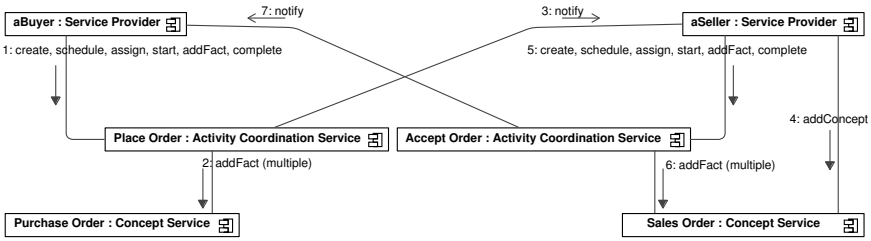


Figure 25: *Example: a communication diagram*

organizations. The place order and accept order activity coordination services manage the state transitions of the activities for the buyer and the seller organizations respectively. The purchase order and sales order concept services manage the fact data about the concepts (agents, activities, business concepts and events) that are relevant to the order process for the buyer and seller organization respectively. When the aBuyer service provider places an order (1) an activity of type place order is created, scheduled, assigned, started and completed. Synchronously, the purchase order concept service adds facts about related concepts (such as a place order activity, a purchase order business concept and related activity events) to the fact base (2). Because aSeller is subscribed to completed events in the context of the place order activity, aSeller is notified of the event (3). Facts about the place order completed event are added to the event history of aSeller (4). In reaction to the event, aSeller has de permission to accept the order. A new activity accept order is coordinated and enacted (5). Synchronously, the sales order concept service adds facts about related concepts (such as an accept order activity, a sales order business concept and related activity events) to the fact base (6). Because aBuyer is subscribed to the completed event in the context of the accept order activity, aBuyer is notified of the event (7). From this point, the process continues in a similar fashion.

There are several advantages to this service-based execution model:

1. The execution maintains the advantages of declarative modeling as it allows to enforce business rules without requiring a specification of when and how to check for rule violations.

2. Because activity services retrieve their state from concept services, the same state information can be used in the context of different process instances (activities). This is advantageous as it renders state synchronization between process instances obsolete (Haesen et al., 2007). Nonetheless such concurrency would still require transaction handling mechanisms to guarantee transaction consistency. For instance, when the same business concepts are manipulated in the context of two different activities, transaction-handling mechanisms are still required.

3. Process models expressed using the EM-BrA^{2}CE Vocabulary can be the basis for model-driven design of software services. Notice however that SOA design can never occur in an entirely top-down fashion, because non-functional business concerns, organizational politics, legacy applications, quality-of-service requirements and the like must be taken into account. These concerns are outside the scope of this paper.

4. The execution model allows for a trade-off between expressiveness and performance. For instance, by including the event history into the state space of an activity, the state space becomes very large, threatening performance of high-volume processes.

However, should the event history be irrelevant to the process model, it can be omitted from consideration.

8 Evaluation of the EM-BrA^{2}CE Framework

In section 2 a number of declarative process modeling languages have identified from the literature. The EM-BrA^{2}CE Framework unifies the aspects of declarative process modeling that are present in these languages. It has the comparative advantage of expressiveness and formality. The framework is expressive in that it addresses control flow, data and resource aspects in business process modeling. This is realized by the vocabulary and by sixteen business rule types. As a consequence, the framework incorporates business concerns such as time, costs and security into business process modeling.

By defining its vocabulary in SBVR and by providing an execution semantics in terms of CP-Nets, the framework benefits from formality. Few languages for process modeling have a formal semantics.

Few languages for declarative process modeling include the event history into its state space. For instance, WMSO (Roman et al., 2005) does not consider events to indicate state. The inclusion of events nonetheless allows for expressing many new types of business rule.

OWL-S (The OWL Services Coalition, 2006) and WSMO (Roman et al., 2005) implicitly make assumptions regarding the role of humans. In general Semantic Web Services allow human users to formulate a goal and let a web service agent (orchestration service) realize that goal in a fully automated way. Current Semantic Web Service standards do not recognize the need for a far going human-machine interaction. In reality, every-day processes often require human intervention in the coordination work (service orchestration). For instance, for reasons outside the information space a particular process instance might require an extra activity. Another example, is the skipping of certain planned activities because they are not relevant to the current process instance.

To make a point, this paper has contrasted procedural and declarative modeling. The difference between procedural and declarative process modeling is out there, but no dichotomy is implied. Many of the design-time advantages of declarative process modeling can already be realized by a careful methodology of documenting the underlying business concerns. The run-time advantages of increased flexibility and user-involvement nonetheless require a formal declarative modeling language and execution model. The choice for modeling language then depends on the application domain. Dynamic, human-centric, non-standardized business process are most likely to require the run-time flexibility offered by of declarative process modeling. Examples are for instance order processing, calling center-mediated handling of distress calls, claim handling or the coordination of the medical process involving surgical procedures. At runtime static, machine-centric, standardized business processes are most likely only to require a procedural representation of the coordination work. Examples are for instance the online booking of flight tickets or the automatic scheduling of production orders.

Declarative process modeling allows to include a lot of functional and non-function aspects of business processes that is missing in procedural process models. However, a declarative process model in the form of a state description and a set of business rules can also be more difficult to understand than the graphically appealing process notations of UML Activity Diagrams and the BPMN. However, by enumerating (a relevant part) of the state space of a declarative process model it is possible to generate a graphical representation. This has, for instance, been shown for the PENELOPE language (Goedertier and Vanthienen, 2006b). The opposite direction is not true in general. Procedural process

models contain a pre-computation of activity control flows that are an explicit enumeration of all possible execution scenarios. It is not generally possible to automatically extract the underlying business concerns from a procedural business process model.

The vocabulary and execution model of the EM-BrA^{2}CE Framework have been validated experimentally. In a design-enact-analyze business process management life cycle three automated reasoning tasks are likely to be required: deduction, abduction and induction. **Deduction** is required for determining whether a particular state transition can occur at runtime. **Abduction** is needed when an execution plan is to be automatically constructed to achieve a particular goal state using a backward planning strategy. Finally, **induction** is required to analyze the execution logs of services to detect whether all business directives have been observed and whether the portrayed behavior in a business process corresponds to the modeled behavior. In (Goedertier et al., 2007a,b) we have used the vocabulary and execution semantics of the EM-BrA^{2}CE framework to generate simulation event logs from process models (deduction) and to learn the business rules that constrain the state transitions in the process model from these event logs supplemented with noise by applying rule-induction techniques. These simulation and model learning experiments indicate the suitability of the EM-BrA^{2}CE Framework throughout the design-enactment-analyze phases of the BPM life cycle.

9 Conclusion

A declarative process model represents a business process as a description of its state space and a set of business rules that constrain the valid movements in that state space. Declarative process modeling allows including a lot of useful information that otherwise would remain implicit in procedural process models. The advantages manifest themselves during the design, enactment and analyze phases of the BPM life cycle. By documenting the governing business concerns, organizations can more easily keep track of changes in the BPM design phase. Moreover, declarative process models are not overburdened with procedural information about how and when business rules are to be enforced or how and when events are communicated to external agents. Declarative process models can be used in the model-driven design of service-oriented architectures (SOAs). Because the state of a business process has real business meaning and is not maintained within the context of an explicit process engine, services can be deployed, maintained and removed as required by the business. Moreover, the proposed way of representing business processes as a history of activity life cycle events has been shown to be suitable for business process mining during the analysis phase.

Recently, a number of declarative process modeling languages have appeared that all deal with a particular aspect in business process modeling. In this paper we have indicated how declarative business process modeling can benefit from the upcoming SBVR standard. In particular, an SBVR vocabulary for process modeling was defined that allows to declaratively refer to the state of a business process when specifying process-related business rules. The sixteen business rule types identified by the framework allow to consider a broad range of control flow, data and organizational modeling aspects. In addition a formal execution model has been provided that covers a broad range of human-machine-mediated coordination, performance and exception handling activity life cycle events.

The EM-BrA^{2}CE Framework is intended to be a foundation in integrating and developing existing and new forms of declarative business process modeling. However, with the framework do not claim to have solved all the important issues. In particular, much research is still required regarding the parsing, visualization, verification of declarative process mod-

els and the model-driven design of service-oriented architectures; hence the attributed 0.1 version number.

References

Andrews, T. and et al. (2003). Business process execution language for web services (bpel4ws) version 1.1. `http://www.ibm.com/developerworks/library/ws-bpel/`.

Antoniou, G., Billington, D., Governatori, G., and Maher, M. J. (2001). Representation results for defeasible logic. *ACM Trans. Comput. Log.*, 2(2):255–287.

Atkinson, C. and Kühne, T. (2003). Model-driven development: A metamodeling foundation. *IEEE Software*, 20(5):36–41.

Bacchus, F. and Kabanza, F. (2000). Using temporal logics to express search control knowledge for planning. *Artif. Intell.*, 116(1-2):123–191.

Bailey, J., Bry, F., and Patranjan, P.-L. (2005). Composite event queries for reactivity on the web. In Ellis, A. and Hagino, T., editors, *WWW (Special interest tracks and posters)*, pages 1082–1083. ACM.

Baisley, D. (2005). OMG and Business Rules. Presentation available at `http://www.omg.org/docs/omg/05-04-09.pdf`.

Baisley, D. E., Hall, J., and Chapin, D. (2005). Semantic Formulations in SBVR. In Hawke et al. (2005).

Bernstein, P. A. and Goodman, N. (1981). Concurrency control in distributed database systems. *ACM Comput. Surv.*, 13(2):185–221.

Bider, I., Khomyakov, M., and Pushchinsky, E. (2000). Logic of change: Semantics of object systems with active relations. *Autom. Softw. Eng.*, 7(1):9–37.

Bons, R. W. H., Lee, R. M., Wagenaar, R. W., and Wrigley, C. D. (1995). Modelling inter-organizational trade using documentary petri nets. In *HICSS (3)*, pages 189–198.

Brachman, R. and Levesque, H. (2004). *Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Bussler, C. (2001). The role of b2b protocols in inter-enterprise process execution. In *TES '01: Proceedings of the Second International Workshop on Technologies for E-Services*, pages 16–29, London, UK. Springer-Verlag.

Chakravarthy, S. and Mishra, D. (1994). Snoop: An expressive event specification language for active databases. *Data Knowledge Engineering*, 14(1):1–26.

Chapin, D. (2005). Semantics of Business Vocabulary & Business Rules (SBVR). In Hawke et al. (2005).

Chomicki, J. (1995). Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Trans. Database Syst.*, 20(2):149–186.

Curbera, F., Khalaf, R., Mukhi, N., Tai, S., and Weerawarana, S. (2003). The next step in web services. *Commun. ACM*, 46(10):29–34.

Davenport, T. H. (1993). *Process innovation: reengineering work through information technology*. Harvard Business School Press, Boston, MA, USA.

Debevoise, T. (2005). *Business Process Management With a Business Rules Approach: Implementing the Service Oriented Architecture*. Business Knowledge Architects.

Digital Business Ecosystem (DBE) (2007). Sbeaver. `http://sbeaver.sourceforge.net`.

Dumas, M., van der Aalst, W. M., and ter Hofstede, A. H. (2005). *Process-aware information systems: bridging people and software through process technology*. John Wiley & Sons, Inc., New York, NY, USA.

Dustdar, S., Fiadeiro, J. L., and Sheth, A. P., editors (2006). *Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings*, volume 4102 of *Lecture Notes in Computer Science*. Springer.

Eder, J. and Dustdar, S., editors (2006). *BPM 2006 International Workshops*, volume 4103 of *Lecture Notes in Computer Science*. Springer.

Ehrlich, B. H. (2002). *Transactional Six Sigma and Lean Servicing: Leveraging Manufacturing Concepts to Achieve World-Class Service*. CRC.

Eshghi, K. (1988). Abductive planning with event calculus. In *ICLP/SLP*, pages 562–579.

Eshuis, R. and Dehnert, J. (2003). Reactive Petri Nets for Workflow Modeling. In van der Aalst, W. M. P. and Best, E., editors, *ICATPN*, volume 2679 of *Lecture Notes in Computer Science*, pages 296–315. Springer.

Ferraiolo, D. F., Sandhu, R. S., Gavrila, S. I., Kuhn, D. R., and Chandramouli, R. (2001). Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274.

Fikes, R. (1971). Monitored execution of robot plans producted by *trips*. In *IFIP Congress (1)*, pages 189–194.

Føllesdal, D. and Hilpinen, R. (1971). Deontic logic: An introduction. In Hilpinen, R., editor, *Deontic Logic: Introductory and Systematic Readings*, pages 1–35. D. Reidel Publishing Company, Dordrecht.

Galton, A. and Augusto, J. C. (2002). Two approaches to event definition. In Hameurlain, A., Cicchetti, R., and Traunmüller, R., editors, *DEXA*, volume 2453 of *Lecture Notes in Computer Science*, pages 547–556. Springer.

Gatziu, S. and Dittrich, K. (1993). Events in an Active Object-Oriented Database System. In Paton, N. and Williams, H., editors, *Proc. 1st Intl. Workshop on Rules in Database Systems (RIDS)*, Edinburgh, UK. Springer-Verlag, Workshops in Computing.

Goedertier, S., Martens, D., Baesens, B., Haesen, R., and Vanthienen, J. (2007a). A new approach for discovering business process models from event logs. FETEW Research Report KBI_0716, K.U.Leuven.

Goedertier, S., Martens, D., Baesens, B., Haesen, R., and Vanthienen, J. (2007b). Process Mining as First-Order Classification Learning on Logs with Negative Events, 3rd Workshop on Business Processes Intelligence (BPI'07), Proceedings. (forthcoming).

Goedertier, S. and Vanthienen, J. (2005). Rule-based business process modeling and execution. In *Proceedings of the IEEE EDOC Workshop on Vocabularies Ontologies and Rules for The Enterprise (VORTE 2005). CTIT Workshop Proceeding Series (ISSN 0929-0672)*, Enschede.

Goedertier, S. and Vanthienen, J. (2006a). Compliant and Flexible Business Processes with Business Rules. In Regev, G., Soffer, P., and Schmidt, R., editors, *7th Workshop on Business Process Modeling, Development and Support (BPMDS'06) at CAiSE'06*, pages 94–104. Presses Universitaires de Namur.

Goedertier, S. and Vanthienen, J. (2006b). Designing compliant business processes with obligations and permissions. In Eder and Dustdar (2006), pages 5–14.

Governatori, G. (2005). Representing business contracts in RuleML. *Int. J. Cooperative Inf. Syst.*, 14(2-3):181–216.

Governatori, G. and Rotolo, A. (2002). A gentzen system for reasoning with contrary-to-duty obligations. a preliminary study. In Jones, A. J. and Horty, J., editors, Δ*eon'02*, pages 97–116, London. Imperial College.

Grosof, B. N., Labrou, Y., and Chan, H. Y. (1999). A declarative approach to business rules in contracts: courteous logic programs in XML. In *ACM Conference on Electronic Commerce*, pages 68–77.

Guenther, C. W. and van der Aalst, W. M. P. (2005). Modeling the case handling principles with colored petri nets. In Jensen, K., editor, *Proceedings of the Sixth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2005)*, volume 576 of *DAIMI*, pages 211–230, Aarhus, Denmark.

Guizzardi, G. and Wagner, G. (2005). *Some Applications of a Unified Foundational Ontology in Business Modeling*, chapter in: Ontologies and Business Systems Analysis, ed. M. Rosemann and P. Green, pages 345–367. IDEA Publisher.

Gupta, A., Sagiv, Y., Ullman, J. D., and Widom, J. (1994). Efficient and complete tests for database integrity constraint checking. In Borning, A., editor, *PPCP*, volume 874 of *Lecture Notes in Computer Science*, pages 173–180. Springer.

Haesen, R., De Rore, L., Goedertier, S., Snoeck, M., Lemahieu, W., and Poelmans, S. (2007). Stateless process enactment. Accepted for Pattern Languages of Programming (PLoP 2007).

Halpin, T. A. (1991). A fact-oriented approach to schema transformation. In Thalheim et al. (1991), pages 342–356.

Halpin, T. A. (2000). A fact-oriented approach to business rules. In *ER*, pages 582–583.

Halpin, T. A. (2004). Information modeling and higher-order types. In Grundspenkis, J. and Kirikova, M., editors, *CAiSE Workshops (1)*, pages 233–248. Faculty of Computer Science and Information Technology, Riga Technical University, Riga, Latvia.

Hammer, M. and Champy, J. (1993). *Reengineering the corporation.* Harper Collins, New York, NY.

Harel, D. and Naamad, A. (1996). The STATEMATE Semantics of Statecharts. *ACM Trans. Softw. Eng. Methodol.*, 5(4):293–333.

Hawke, S., de Sainte Marie, C., and Tabet, S., editors (2005). *W3C Workshop on Rule Languages for Interoperability, 27-28 April 2005, Washington, DC, USA*. W3C.

Heinl, P., Horn, S., Jablonski, S., Neeb, J., Stein, K., and Teschke, M. (1999). A comprehensive approach to flexibility in workflow management systems. *SIGSOFT Softw. Eng. Notes*, 24(2):79–88.

Henkin, L. (1950). Completeness in the theory of types. *Journal of Symbolic Logic*, (15):8191.

InterNational Committee for Information Technology Standards (INCITS) (2004). Role-Based Access Control. `http://csrc.nist.gov/rbac`. American National Standard ANSI/INCITS 359-2004.

Jablonski, S. and Bussler, C. (1996). *Workflow Management. Modeling Concepts, Architecture and Implementation*. International Thomson Computer Press, London.

Jensen, K. (1993). An introduction to the theoretical aspects of coloured petri nets. In de Bakker, J. W., de Roever, W. P., and Rozenberg, G., editors, *REX School/Symposium*, volume 803 of *Lecture Notes in Computer Science*, pages 230–272. Springer.

Jensen, K. (1996). *Coloured Petri nets (2nd ed.): basic concepts, analysis methods and practical use: volume 1*. Springer-Verlag, London, UK.

Kaplan, R. (1998). One cost system isnt enough. *Harvard Business Review*, January-February:61–66.

Kardasis, P. and Loucopoulos, P. (2005). A roadmap for the elicitation of business rules in information systems projects. *Business Process Management Journal*, 11(4):316–348.

Knottenbelt, J. and Clark, K. (2004). An architecture for contract-based communicating agents. In *Proceedings of the Second European Workshop on Multi-Agent Systems*.

Kowalski, R. and Sergot, M. (1986). A logic-based calculus of events. *New Gen. Comput.*, 4(1):67–95.

Leung, C. M. R. and Nijssen, G. M. (1988). Relational database design using the niam conceptual schema. *Inf. Syst.*, 13(2):219–227.

Li, N., Grosof, B. N., and Feigenbaum, J. (2003). Delegation logic: A logic-based approach to distributed authorization. *ACM Trans. Inf. Syst. Secur.*, 6(1):128–171.

M. Zaremba, C. B. (2005). Towards dynamic execution semantics in semantic web services. In *In Proceedings of the Workshop on Web Service Semantics: Towards Dynamic Business Integration, International Conference on the World Wide Web (WWW2005)*, Chiba, Japan.

Marín, R. H. and Sartor, G. (1999). Time and norms: a formalisation in the event-calculus. In *ICAIL '99: Proceedings of the 7th international conference on Artificial intelligence and law*, pages 90–99, New York, NY, USA. ACM Press.

Milner, R., Tofte, M., and Harper, R. (1990). *The definition of Standard ML*. MIT Press, Cambridge, MA, USA.

Motorola Inc. (1986). Motorola University, Six Sigma in Action. `http://www.motorola.com/motorolauniversity`, consulted on Januari 18, 2007.

Mulyar, N. and van der Aalst, W. M. (2005). Patterns in Colored Petri Nets. BETA Working Paper Series WP 139, Eindhoven University of Technology, Eindhoven.

Nute, D. (1994). *Handbook of Logic in Artificial Intelligence and Logic Programming*, chapter Defeasible Logic, page 353395. Oxford University Press.

Object Management Group (2005). UML 2.0 Superstructure Specification. OMG Document – formal/05-07-04.

Object Management Group (2006a). Business Process Modeling Notation (BPMN) – final adopted specification. OMG Document – dtc/06-02-01.

Object Management Group (2006b). Semantics of Business Vocabulary and Business Rules (SBVR) – Interim Specification. OMG Document – dtc/06-03-02.

Object Management Group (2006c). UML 2.0 Infrastructure Specification. OMG Document – formal/05-07-05.

O'Sullivan, J., Edmond, D., and ter Hofstede, A. H. M. (2002). What's in a service? towards accurate description of non-functional service properties. *Distributed and Parallel Databases*, 12(2/3):117–133.

Paschke, A. and Bichler, M. (2005). SLA Representation, Management and Enforcement. In *EEE '05: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)*, pages 158–163, Washington, DC, USA. IEEE Computer Society.

Pesic, M. and van der Aalst, W. M. P. (2006). A declarative approach for flexible business processes management. In Eder and Dustdar (2006), pages 169–180.

Porter, M. (1985). *Competitive Advantage*. Free Press, New York.

Reichert, M. and Dadam, P. (1998). $\text{Adept}_{\text{flex}}$-supporting dynamic changes of workflows without losing control. *J. Intell. Inf. Syst.*, 10(2):93–129.

Roberts, J. and et al. (2001). Tentative hold protocol part 2: Technical specification. W3c note, World Wide Web Consortium. `http://www.w3.org/TR/tenthold-2/`.

Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., and Fensel, D. (2005). Web service modeling ontology. *Applied Ontology*, 1(1):77–106.

Ross, R. G. (2003). *Principles of the Business Rule Approach*. Addison-Wesley Professional.

Russell, N., van der Aalst, W. M. P., and ter Hofstede, A. H. M. (2006). Workflow exception patterns. In Dubois, E. and Pohl, K., editors, *CAiSE*, volume 4001 of *Lecture Notes in Computer Science*, pages 288–302. Springer.

Sadiq, S. W., Orlowska, M. E., and Sadiq, W. (2005). Specification and validation of process constraints for flexible workflows. *Inf. Syst.*, 30(5):349–378.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E. (1996). Role-based access control models. *IEEE Computer*, 29(2):38–47.

Schmidt, K. (1999). Of maps and scripts: The status of formal constructs in cooperative work. *Information & Software Technology*, 41(6):319–329.

Schmidt, K. and Simone, C. (1996). Coordination mechanisms: Towards a conceptual foundation of cscw systems design. *Computer Supported Cooperative Work*, 5(2/3):155–200.

Segerberg, K. (1982). A deontic logic of action. *Studia Logica*, 10:269–282.

Shanahan, M. (1997). *Solving the frame problem: a mathematical investigation of the common sense law of inertia*. MIT Press, Cambridge, MA, USA.

Störrle, H. and Hausmann, J. H. (2005). Towards a formal semantics of uml 2.0 activities. In Liggesmeyer, P., Pohl, K., and Goedicke, M., editors, *Software Engineering*, volume 64 of *LNI*, pages 117–128. GI.

Strembeck, M. and Neumann, G. (2004). An integrated approach to engineer and enforce context constraints in rbac environments. *ACM Trans. Inf. Syst. Secur.*, 7(3):392–427.

Suchman, L. A. (1987). *Plans and situated actions: the problem of human-machine communication*. Cambridge University Press, New York, NY, USA.

Suchman, L. A. (1995). Making work visible. *Commun. ACM*, 38(9):56–64.

Thalheim, B., Demetrovics, J., and Gerhardt, H.-D., editors (1991). *MFDBS 91, 3rd Symposium on Mathematical Fundamentals of Database and Knowledge Bases Systems, Rostock, Germany, May 6-9, 1991, Proceedings*, volume 495 of *Lecture Notes in Computer Science*. Springer.

The OWL Services Coalition (2006). OWL-S 1.2 Pre-Release. Available from `http://www.ai.sri.com/daml/services/owl-s/1.2/`.

Unisys (2005). Unisys rules modeler. `www.unisys.com/eprise/main/admin/corporate/doc/Unisys_Rules_Modeler_Insert.pdf` [10-11-2005].

van der Aalst, W. M., Rosemann, M., and Dumas, M. (2007). Deadline-based escalation in process-aware information systems. *Decision Support Systems*, 43(2):492–511.

van der Aalst, W. M. P. (1997). Verification of workflow nets. In Azéma, P. and Balbo, G., editors, *ICATPN*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer.

van der Aalst, W. M. P. (1998). The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66.

van der Aalst, W. M. P. and ter Hofstede, A. H. M. (2002). Workflow patterns: On the expressive power of (petri-net-based) workflow languages. In *Proc. of the Fourth International Workshop on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark, August 28-30, 2002 / Kurt Jensen (Ed.)*, pages 1–20. Technical Report DAIMI PB-560.

van der Aalst, W. M. P., Weske, M., and Grünbauer, D. (2005). Case handling: a new paradigm for business process support. *Data Knowl. Eng.*, 53(2):129–162.

van Hee, K., Oanea, O., Serebrenik, A., Sidorova, N., and Voorhoeve, M. (2006a). Modelling history-dependent business processes. In *MSVVEIS*, pages 76–85.

van Hee, K. M., Oanea, O., Serebrenik, A., Sidorova, N., and Voorhoeve, M. (2006b). History-based joins: Semantics, soundness and implementation. In Dustdar et al. (2006), pages 225–240.

Van Nuffelen, B. and Kakas, A. C. (2001). A-system: Declarative programming with abduction. In Eiter, T., Faber, W., and Truszczynski, M., editors, *LPNMR*, volume 2173 of *Lecture Notes in Computer Science*, pages 393–396. Springer.

W3C (2004). Resource Description Framework (RDF): Concepts and Abstract Syntax. Technical report, W3C Recommendation 10 February 2004.

Wagner, G. (1991). A database needs two kinds of negation. In Thalheim et al. (1991), pages 357–371.

Wagner, G. (2003). The agent-object-relationship metamodel: towards a unified view of state and behavior. *Inf. Syst.*, 28(5):475–504.

Wainer, J., Kumar, A., and Barthelmess, P. (2007). DW-RBAC: A formal security model of delegation and revocation in workflow systems. *Inf. Syst.*, 32(3):365–384.

Wohed, P., van der Aalst, W. M. P., Dumas, M., ter Hofstede, A. H. M., and Russell, N. (2006). On the suitability of bpmn for business process modelling. In Dustdar et al. (2006), pages 161–176.

Woolridge, M. and Wooldridge, M. J. (2001). *Introduction to Multiagent Systems.* John Wiley & Sons, Inc., New York, NY, USA.

Yolum, P. and Singh, M. P. (2004). Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. *Annals of Mathematics and Artificial Intelligence*, 42(1-3):227–253.

Zhang, L., Ahn, G.-J., and tseng Chu, B. (2003). A rule-based framework for role-based delegation and revocation. *ACM Trans. Inf. Syst. Secur.*, 6(3):404–441.

zur Muehlen, M. (2004). *Workflow-based Process Controlling. Foundation, Design, and Implementation of Workflow-driven Process Information Systems.*, volume 6 of *Advances in Information Systems and Management Science.* Logos, Berlin.