
Rule-based business process modelling and enactment

Stijn Goedertier*, Raf Haesen and Jan Vanthienen

Department of Decision Sciences and Information Management,
Katholieke Universiteit Leuven,
Leuven, Belgium

E-mail: stijn.goedertier@econ.kuleuven.be

E-mail: raf.haesen@econ.kuleuven.be

E-mail: jan.vanthienen@econ.kuleuven.be

*Corresponding author

Abstract: A business process model is called *rule-based* if the logic of its control flow, data flow and resource allocation is declaratively expressed by means of business rules. Business rules are recognised as powerful representation forms that can potentially define the semantics of business process models and business vocabulary. To date, however, there is little consensus and fragmentary knowledge about the precise relationship between these elements of business modelling. In this article, we develop a first-version metamodel that is to be used as a foundation in integrating and developing existing and new forms of rule-based business process modelling. In addition, we show how rule-based process models can be brought to execution in the context of service-oriented architecture.

Keywords: process modelling; business rules; ontology; process enactment; service-oriented architecture.

Reference to this paper should be made as follows: Goedertier, S., Haesen, R. and Vanthienen, J. (2008) 'Rule-based business process modelling and enactment', *Int. J. Business Process Integration and Management*, Vol. 3, No. 3, pp.194–207.

Biographical notes: Stijn Goedertier has obtained his PhD at K.U. Leuven, Faculty of Business and Economics. His research interests are in the area of business process management and enterprise architecture.

Raf Haesen is a PhD candidate in the Faculty of Economic and Applied Economic Sciences. His research interests include model-driven development and service and component-based architectures.

Jan Vanthienen is a Full Professor within the Faculty of Economic and Applied Economic Sciences. His research interests include information systems analysis and design, verification and validation of knowledge-based systems and business intelligence.

1 Introduction

A major business requirement for business process management is to reconcile the goals of flexibility and compliance. Socio-economic factors like globalisation, outsourcing, mergers and acquisitions have made business environments more complex and prone to change. In such a setting companies must flexibly adapt their business policies and business processes to accommodate new market situations. But control over business processes is undoubtedly as important as flexibility. Companies are increasingly confronted by regulations that potentially affect every process within their organisation. The Sarbanes-Oxley Act, for instance, has a substantial impact both on business processes and IT processes (O'Connor, 2005). In general, compliance to internal policies and external regulations can be an important driver for automating business process

support. By automating the coordination of work organisations have better control over the activities that are conducted by its actors. Moreover, automated processes can help organisations in demonstrating regulatory compliance. The downside, however, is that automated business processes are often inflexible. The reason is that organisations often only *implicitly* think about the underlying business concerns when they model business processes and pay little attention to avoid hard-coding business logic directly in control-flow-based, procedural process models. Such procedural process models have proven to be difficult to adapt at design time where each change triggers a lengthy development cycle in which it is impossible to a priori identify and include all control and correction steps (Heinl et al., 1999). Moreover, at run time automated processes are often too rigid to deal with the contingencies of real-life situations (Sadiq et al., 2005).

Languages like BPEL4WS (Andrews et al., 2003), UML Activity Diagrams (Object Management Group, 2005) and the Business Process Modelling Notation (BPMN) (Object Management Group, 2006b) can be classified as languages for procedural process modelling.

In many applications there are ample examples of the lack of information provided by procedural process modelling. For example, in purchase order processing, an organisation might have the procedure to first receive a proof of delivery and then to pay for the order. This procedure might originate from the business concern only to pay for honoured purchase orders. However, such a concern might be in conflict with the concern of the seller never to ship orders without payment in advance. In general, when such business concerns remain undocumented, altering a procedure results in a lengthy development cycle because the underlying business concerns have to be re-identified prior to alteration. Problem solving in calling centres is another example. Very often, the predefined, so to say hard-coded, procedures that have been installed to assist customers in solving their problems are inadequate for the problem at hand, or they are suboptimal from a cost or time perspective. What is lacking is declarative information about the activities that are necessary, advisable and possible to undertake and information about the expected cost and time such activities will take. This information is useful to assist a calling centre operator in finding a suitable solution for the customer.

To cater for flexibility and provable compliance, enterprise architectures must adhere to truly declarative process models. Process models are *declarative* when they contain explicit information about the business concerns – i.e., the benefits, costs, time characteristics, constraints and business goals – of a business process, leaving as much freedom as is permissible for determining a valid and suitable execution plan. Conversely, process models are *procedural* when they contain explicit execution plans but only implicitly keep track of the business concerns that motivate why particular design choices have been made.

The idea of separating the business concerns from process models is related to the business rules approach (Ross, 2003; Kardasis and Loucopoulos, 2005). Business rules are atomic, formal expressions of business policies or business regulations. By documenting the business rules that govern business processes, changes in business policy and regulations are more easily reflected in process model updates. However, the relationship between business vocabulary, business rules and business process models is not yet fully understood. Languages such as the case handling paradigm (van der Aalst et al., 2005), OWL-S (The OWL Services Coalition, 2004), the constraint specification framework of Sadiq et al. (2005), the Web Service Modelling Ontology (WSMO) (Roman et al., 2005), the ConDec language (Pesic and van der Aalst, 2006) and the PENELOPE language (Goedertier and Vanthienen, 2006) can be categorised as languages for declarative process modelling. However, none of these languages is expressive enough to cover all real-life business concerns that exist

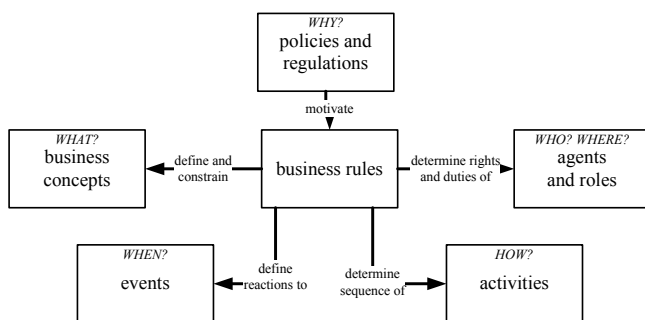
about business processes. For instance, the ConDec language and the PENELOPE language only allow expressing business rules about sequence and timing constraints, i.e., control-flow. Moreover, these languages make use of very different knowledge representation paradigms. For instance, the ConDec language is expressed in linear temporal logic (LTL) whereas the PENELOPE language is expressed in terms of the event calculus. Finally, these languages do not have an explicit execution model or have an execution model that explicitly assumes either human or machine-mediated enactment. The case handling paradigm, for instance, assumes humans to perform atomic tasks but has an orchestration engine to perform the orchestration (coordination) work. The above-mentioned declarative modelling approaches remain fragmentary and are difficult to compare and combine. What is needed are meaningful ways to modularly combine several methods for knowledge representation and reasoning in a general ontology that standardises modelling constructs. The Object Management Group (OMG) has two new standards for business modelling currently under development: the Semantics of Business Vocabulary and Business Rules (SBVR) (Object Management Group, 2006a) specification and the Business Process Definition Metamodel (BPDM) (Object Management Group, 2007). Although the SBVR will allow for documenting the semantics of SBVR of business processes, it is not tailored towards declarative process modelling because it lacks a built-in vocabulary for expressing process-related concepts such as ‘agent’, ‘activity’, ‘event’ or ‘deontic assignment’. The BPDM enables integration with business rules through so-called ‘fact change conditions’, but it can be categorised as a metamodel for procedural process notations and an alignment of the SBVR and the BPDM is deferred to a future request for proposal.

The contribution of this article consists of an extension to the SBVR for declarative business process modelling, called the enterprise modelling using business rules, agents, activities, concepts and events (EM-BrA²CE) vocabulary. The intent is to serve as a foundation in integrating existing and developing new forms of declarative business process modelling. In addition, the article demonstrates how declarative process models can be enacted in the context of a service-oriented architecture. The remainder of the article is structured as follows. A first section defines the fundamental building blocks of the EM-BrA²CE vocabulary. This vocabulary can be used to construct different types of business rules that are defined in a next section. Subsequently, we informally discuss the execution semantics behind EM-BrA²CE process models and discuss how declarative process models can be enacted in a service-oriented architecture. Finally, we evaluate the proposed vocabulary, execution model and architecture by relating it to the literature.

2 The EM-BrA²CE vocabulary

Enterprise models (EM) are abstractions of different aspects of an enterprise, typically with a purpose to understand and share the knowledge of how the enterprise is structured and how it operates (Bajec and Krisper, 2005) and can be a foundation for model-driven architecture. In the context of rule-based business process modelling, five building blocks are particularly relevant: business rules, agents, activities, concepts and events (EM-BrA²CE). Business policy and regulations can be internalised and made explicit in terms of the EM-BrA²CE building blocks, as displayed in Figure 1.

Figure 1 The EM-BrA²CE framework



Declarative process models expressed with the EM-BrA²CE vocabulary should be on the one hand comprehensible so that they can be understood by business people and on the other hand formal so that they can be enforced by information systems. The SBVR (Object Management Group, 2006a) is a language for business modelling that has such property. On the one hand, the SBVR provides a number of conceptual vocabularies for modelling a business domain as a vocabulary and a set of rules that, among others, can be expressed in SBVR structured English. On the other hand, the SBVR has a vocabulary to describe the semantic structure of expressions in terms of formal logic that can be used by automated reasoning mechanisms. This property makes the SBVR suitable as a base language for declarative process modelling. Unfortunately, the current SBVR specification (Object Management Group, 2006a) does not have a built-in vocabulary for expressing process-related concepts such as agent, activity, event or deontic assignment. In the next sections, we extend the SBVR vocabularies with the EM-BrA²CE vocabulary.

2.1 Type-level and instance-level concepts

The EM-BrA²CE 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 an activity. Figure 3 is a MOF/UML class diagram representation of the instance-level concepts in the EM-BrA²CE vocabulary. Likewise, Figure 4 represents the type-level concepts. Whereas all instance-level concepts extend SBVR:

individual concept, all type-level concepts extend SBVR: concept type. Notice that to each instance-level individual concept a particular type-level concept type corresponds. In the following paragraphs, these type-instance pairs are defined.

2.2 Activity and activity type concepts

The pair activity – activity type (or service instance – service capability) represents two of the most central concepts in the EM-BrA²CE vocabulary. A business process consists both of work and coordination work. Composite activities represent coordination work, the act of coordinating a set of sub-activities indicated by the ‘can consist of’ and ‘is parent of’ fact types. Atomic activities represent actual work. Each activity must be uniquely identifiable by at least one business identifier that consists of a set of business concepts.

2.3 State space and state

Each process (called activity) can be modelled by describing its state space and the set of business rules that constrain movements in this state space. The EM-BrA²CE vocabulary defines a *state space* as the description of a set of discrete states of an activity type (or service capability) in terms of the concepts and facts types that occur in its state space. A *state* is a specific configuration of facts about concepts corresponding to a specific situation of an activity. An *abstract state* is a logical specification that covers a set of states within a state space. Notice that state is a relative concept: only those concepts and facts are considered that are related to the task at hand. A *business rule* is an assertion about an abstract state being a necessity, an obligation, a prohibition or a possibility.

Consider, for instance, an order-to-cash business process depicted in Figure 2. The state space of this process model can be described in terms of the following EM-BrA²CE concepts:

- *Roles*: buyer, seller
- *Composite activity types*: coordinate purchase order, coordinate sales order
- *Atomic activity types*: place order, accept order, reject order, pay, ship
- *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* deadline.

Figure 2 The shipment-after-payment process

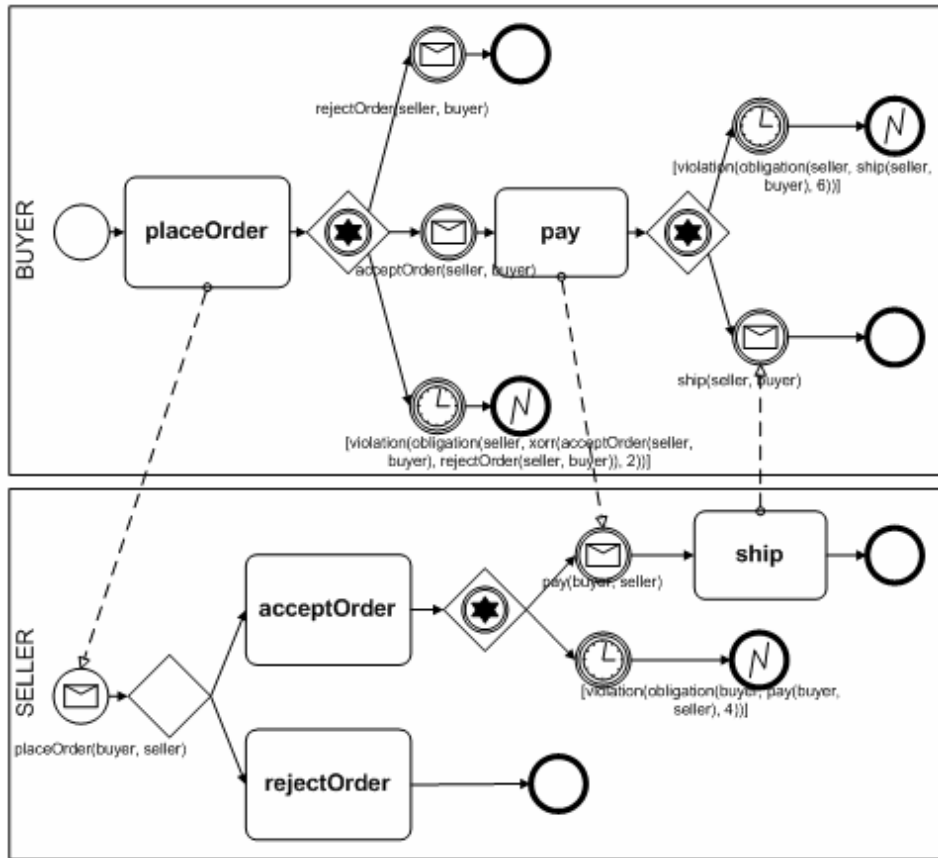


Figure 3 Instance-level concepts

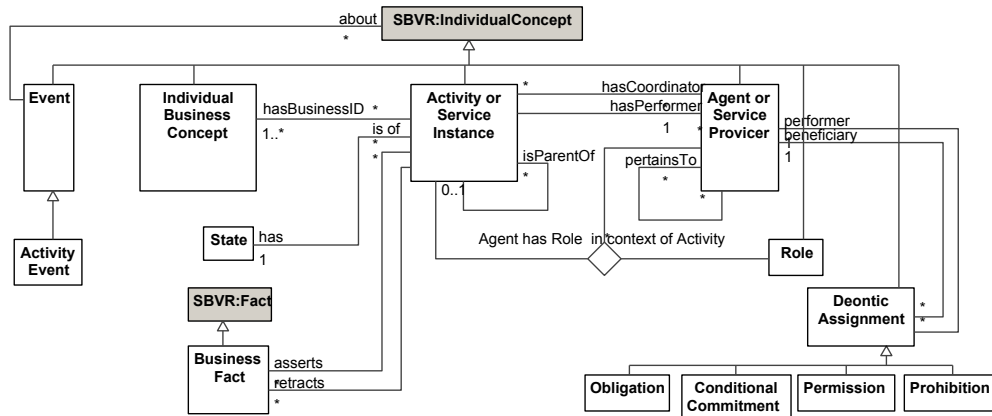
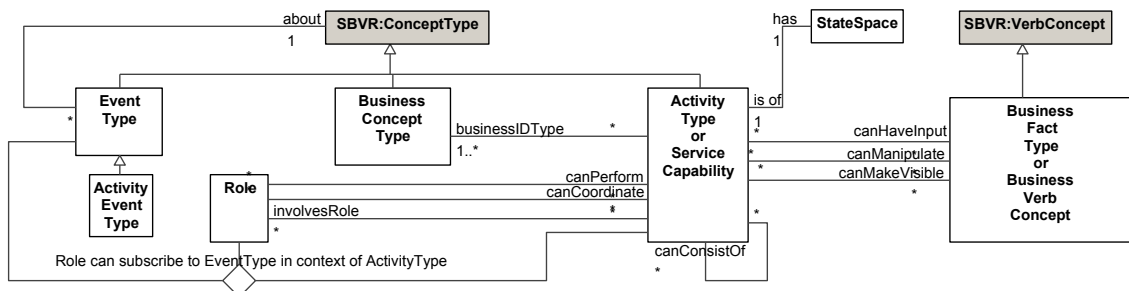


Figure 4 Type-level concepts



2.4 Agent and role concepts

One of the requirements of declarative process modelling is that it is agnostic of the nature of the actor who performs the (coordination) work. To this end it is useful to abstract from the differences between humans and machines through the use of the agent metaphor. This agent metaphor is present in many other ontologies. The agent (or service provider) concept in the EM-BrA²CE vocabulary does not only represent individual workers or machines, but also organisational structures and ad-hoc groups of agents, such as for instance an entire department or a group of workers. This is expressed with the ‘pertains to’ fact type. In the context of a business process an agent can fulfil a particular role that represents an authorisation to perform a number of activities. This is represented by the ‘agent has role in context of activity’ fact type. This conception of role is consistent with the Role Based Access Control (RBAC) standard (Ferraiolo et al., 2001).

2.5 Business fact and business fact type

The pair business fact – business fact type represents the case data that is linked to a business process. The EM-BrA²CE vocabulary adopts the fact-oriented modelling style of the SBVR. Fact-orientation perceives the world in terms of facts rather than in terms of objects, attributes and relationships. The finer granularity of facts compared to objects facilitates modelling and postpones decisions about grouping facts into objects and physical data structures.

2.6 Event and event type concepts

In the EM-BrA²CE vocabulary, the state of an activity (or service instance) includes the history of events related to the activity or its sub-activities. Unlike many ontologies for business modelling, such as for instance the Unified Foundational Ontology (UFO) (Guizzardi and Wagner, 2005), the EM-BrA²CE vocabulary makes a distinction between activities and events. Activities are performed by agents and have a particular duration whereas (primitive) events occur instantaneously and represent a discrete state change in the world. Changes to the life cycle of an activity are reflected by means of activity events. Activity events allow process modellers to distinguish between the activity state transitions that occur when creating, scheduling, assigning, starting, completing, skipping or aborting an activity.

Although events occur instantaneously, they are added to the state of an activity and are assumed not to be retracted. As such activity events make up the history of an activity. For the purpose of declarative process modelling such an event history allows for a greater expressiveness compared to procedural process languages such as the BPMN or Petri nets. 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 behaviour, the event history has to be reflected into the local state of the tokens. van Hee et al.

(2006) have shown that such history-dependent behaviour is in general difficult to model using Petri nets and propose to include event history into the state of a process instance.

Furthermore, the distinction between activity and event allows for reactive behaviour. 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 effectively 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.

2.7 Deontic assignment

A deontic assignment represents among others the obligation or permission of an agent to perform a particular activity by a particular due date. Obligations and permissions keep track of the legal commitments of agents towards each other. It can be useful to include such information within a process model. For instance, Yolum and Singh (2004) demonstrate how business interactions can be modelled as the coming into existence or ceasing to exist of commitments of agents towards each other. In this context, the state space, or rather the commitment space contains the performances of agents and the commitments that result from it. Goedertier and Vanthienen (2006) have used this idea to illustrate the use of deontic assignments in the modelling of compliant business processes.

3 Types of business rule

This section identifies 16 types of business rule. They refer to one of the three aspects of business process modelling that are generally considered (Curtis et al., 1992): the control-flow, the data and the organisational 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 *resource aspect* provides information about the organisational structure in the form of human and machine roles responsible for executing tasks.

Ross (2003) has advocated that business rules should not always express necessities but also guidelines or possibilities. Likewise, the SBVR standard classifies business rules according to the intended modality as being either structural or operative (Object Management Group, 2006a). *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 advise – what *ought to* be true – from strict obligation – what *should* be true. In the remainder of this section, 16 business rules, as defined as specialisations of structural or operative business rules.

3.1 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 expresses the creation or destruction of deontic assignments based on the (non-)occurrence of events. The rules describe behaviour from a third-person perspective. Temporal rules can be represented using many different knowledge representation paradigms. Goedertier and Vanthienen (2006) show how temporal deontic rules can be expressed using an event calculus-based temporal logic.

- Permission: It is necessary that initially the buyer has the permission to place an order.
- Permission: It is necessary that when the buyer places an order, the seller has the obligation to either accept or reject the order within two time units.
- Conditional commitment: It is necessary that when the buyer places an order, the buyer has the obligation to pay the seller within two time units after the seller ships the goods.
- Conditional commitment: It is necessary that when the seller accepts the order, the seller must ship the goods within two time units.

The displayed temporal deontic rules categorise the external business regulation shipment-after-payment visualised in Figure 2. 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 processes.

3.2 Activity precondition

An *activity precondition* is a business rule that defines the conditions required prior to the start of an activity of a particular activity type. This constraint is similar to a pre-condition in the WSMO (Roman et al., 2005).

- acceptOrder: It is necessary that the order exists.
- rejectOrder: It is necessary that the order exists.
- ship: It is necessary that there exists an accepted order.
- pay: It is necessary that there exists a proof of delivery for the order.

3.3 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 in terms of some relevant fact types. This constraint is equivalent to a post-condition in the WSMO (Roman et al., 2005) and subsumes a mandatory constraint in the case handling paradigm (van der Aalst et al., 2005).

- placeOrder: A new order exists.
- acceptOrder: The order has the acceptance of the seller.
- rejectOrder: The order has the rejection of the seller.
- ship: There exists a proof of delivery for the order.
- pay: There exists a proof of payment.

3.4 Dynamic integrity constraint

A dynamic integrity constraint is a business rule that defines the admissible state changes of a business fact (Object Management Group, 2006a).

- Dynamic integrity constraint: During shipping or after an order has been shipped, the order lines of the order can no longer be changed.

3.5 Activity cardinality constraint

An 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. This business rule type can be expressed in the ConDec language with so-called existence constraints (Pesic and van der Aalst, 2006).

- placeOrder: There exists exactly one place order activity in the context of a handle purchase order activity.
- acceptOrder: There exists at most one accept order activity in the context of a handle sales order activity.

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

- Serial activity constraint: The activities place order, accept order, ship order reject order must not be performed in parallel.

3.7 Activity order constraint

An activity order constraint is a business rule that imposes that activities of particular activity types must be performed in a specified order. 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 (Pesic and van der Aalts, 2006). Notice that an activity order constraint on a set of activity types implies that these activity types are serial.

- Activity order constraint: An accept order activity can only be performed after a place order activity.

3.8 Activity exclusion and inclusion constraints

An activity exclusion constraint is a business rule that imposes that two activities of a particular activity type are mutually exclusive. This can, for instance, be expressed with an exclusion or inclusion constraint in the constraint specification framework of Sadiq et al. (2005) or with different types of negation constraints in the ConDec language (Pesic and van der Aalts, 2006).

- Activity exclusion constraint: It is necessary that the activities accept order and reject order are mutually exclusive.

3.9 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, there is little benefit in modelling business processes using only reaction rules. Process models that are only composed of reaction rules risk being over-specified and can be regarded as prescriptive. What is needed is a hybrid approach, in which the freedom of choice that is left by other business rules is filled in, if required, by a small set of reaction rules. 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 behaviour for a given situation. 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 shipment-after-payment 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 modelling 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:

- 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.

In natural language, the following policy might be formulated:

- Reaction rule: When an order times out, reorder with the same supplier.
- Reaction rule: When an order is rejected reorder with a different supplier.
- Reaction rule: When an order is rejected or times out and the order is critical only notify a purchase representative.

Notice that the reorder policy is formulated using a kind of defeasible rules (Antoniou et al., 2000) in which there are priority relations between the rules and in which defeasible rules can be overridden by strict rules. In particular, the third reaction rule overrides the two previous reaction rules in the case that the order is critical as implied by the ‘only’ keyword. These formulations are common in natural language, but not always practical in logic. For instance, the above definition of the ECA rules does not support defeasibility. Moreover, defeasible rules by themselves are difficult to visualise, verify and validate. To overcome these problems a transformation mechanism is needed that transforms defeasible rules into standard, non-defeasible rules.

Decision tables can be such a transformation, visualisation and verification mechanism. Decision tables are one of many visualisations (Antoniou et al., 2004) of rule sets. Table 1 displays an example of a decision table. Graphically, a decision table consists of four quadrants. The two upper quadrants make up the condition sphere and the two lower quadrants represent the conclusion sphere. Properly built decision tables contain columns that represent exhaustive and mutually exclusive conjunctions of conditions and that have one or more conclusions. A set of input rules, possibly expressed using a defeasible logic, determines which combination of conditions in the upper right quadrant leads to which conclusion values in the lower right quadrant.

Table 1 Transforming defeasible reaction rules into ECA rules

| | <i>reorder policy</i> | | | | | | | |
|---------------------------------|-----------------------|---|---|---|---|---|---|---|
| when the order is accepted | Y | | N | | | | | |
| when the order is rejected | N | | Y | | N | | | |
| when the order times out | N | | N | | Y | | N | |
| the order is critical | Y | N | Y | N | Y | N | Y | N |
| reorder with the same seller | . | . | – | . | – | × | . | . |
| reorder with a different seller | . | . | – | × | – | . | . | . |
| notify a purchase rep. | . | . | × | . | × | . | . | . |

Several tools provide support in constructing decision tables (Vanthienen et al., 1998; Spreeuwenberg et al., 2000). In Prologa (Vanthienen et al., 1998) decision tables are constructed by providing conditions with mutually exclusive condition labels, conclusions and input rules. Although Prologa fits in a propositional logic framework, Prologa proves a useful tool to visualise and transform a number of input rules, expressed using Prologa's own defeasible logic, into table rules. In particular, Prologa provides the user with a number of features that facilitate knowledge modelling, such as the reordering of condition labels to expand or contract the table, the syntactical verification of rules and the helpful visualisations that allow to semantically validating a rule set. Notice that not every conjunction of conditions is necessarily meaningful. In other words, it might be the case that a specific condition is not meaningful in conjunction with other specific conditions. For instance, the column in which an order is both accepted and reject is not meaningful as it is not allowed by the above mentioned activity exclusion constraint. To express this, a decision table can make use of so called condition impossibility expressions.

The columns in Figure 6 represent the four ECA rules that are obtained by transforming the three defeasible rules of the reorder policy. These rules translate in the following natural language statements.

- Reaction rule: When an order is rejected and if the order is critical, notify a purchase representative.
- Reaction rule: When an order is rejected and if the order is not critical, reorder with a different seller.
- Reaction rule: When an order times out and if the order is critical, notify a purchase representative.
- Reaction rule: When an order times out and if the order is not critical, reorder with the same seller.

3.10 Static integrity constraint

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). Static integrity constraints are expressed with the existing SBVR vocabularies.

- Static integrity constraint: It is necessary that each order has at least one order line.

3.11 Derivation rule

A derivation rule is a business rule that defines new business fact types in terms of existing business fact types (Wagner, 2003). Notice that a derivation rule can both express a necessity, obligation or guideline. For instance, a business rule recommends a particular price, but leaves a salesperson with the freedom of choice of determining a custom-tailored price. Derivation rules are expressed with the existing SBVR vocabularies.

- Static integrity constraint: It is advisable that the unit price is twice the bottom price.

3.12 Activity authorisation constraint

An activity authorisation 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. In conformance with dynamic access control (Strembeck and Neumann, 2004), this business rule allows to constrain the role-based authorisations that can be granted to an agent.

- Activity authorisation constraint: It is necessary that a junior sales representative cannot accept or reject orders larger than EUR 2000.

3.13 Visibility constraint

A visibility constraint is a structural business rule that dynamically constrains the visibility of business facts within the context of 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.

- Visibility constraint: It is necessary that the reason of rejection is only visible to the customer who placed the order.

3.14 Event subscription constraint

An 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. 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 deontic assignment that results from the event can be enforced. Event subscription does not stipulate how an event is communicated to an agent. This can be realised either by a push or a pull mechanism.

- Event subscription: It is necessary that a reject order activity is visible for the customer who has placed the order.

3.15 Activity allocation rule

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.

- Activity allocation rule: A head of the department sales should not be assigned activities of activity type archive documents.

4 Service-oriented execution

As mentioned above, the state of an activity (or service instance) consists of the properties of the activity and the business concepts and history of events that are within the scope of the activity. The possible movements within a business process' state space can be described by 12 generic activity state transitions. Because they are generic, these 12 activity state transitions provide a means for defining execution semantics for processes modelled with the EM-BrA²CE vocabulary. In particular, the following state transitions are considered:

- *create(Aid,AT,BId,PIId,CoordinatorId)*: requests the creation of a new activity with identifier *Aid* of type *AT* with business identifiers *BId*, parent activity *PIId* by coordinator *CoordinatorId*. *Activity event type*: created.
- *schedule(Aid,DueDate,CoordinatorId)*: requests the due date of activity *Aid* to be set to *DueDate* by coordinator *CoordinatorId*. *Activity event type*: scheduled.
- *assign(Aid,AgentId,CoordinatorId)*, *revoke(Aid,AgentId,CoordinatorId)*: requests the assignment or revocation of the assignment of activity *Aid* to an agent *AgentId* by coordinator *CoordinatorId*. *Activity event type*: assigned, revoked.
- *start(Aid,WorkerId)*: requests an activity *Aid* to be started by worker *WorkerId*. *Activity event type*: started.
- *addFact(Aid,C,WorkerId)*, *removeFact(Aid,C,WorkerId)*, *updateFact(Aid,C1,C2,WorkerId)*: requests the addition or removal of business fact *C* or the update of a business fact *C1* by *C2* within the context of activity *Aid* by worker *WorkerId*. *Activity event type*: factAdded, factUpdated, factRemoved.
- *complete(Aid,WorkerId)*: requests the completion of activity *Aid* by worker *WorkerId*. *Activity event type*: completed. 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 coordinator *CoordinatorId*. *Activity event type*: skipped, aborted, redone.

Notice that these 12 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*, *updateFact* and *complete* 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 2 indicates which business rule types constrain which state transition types.

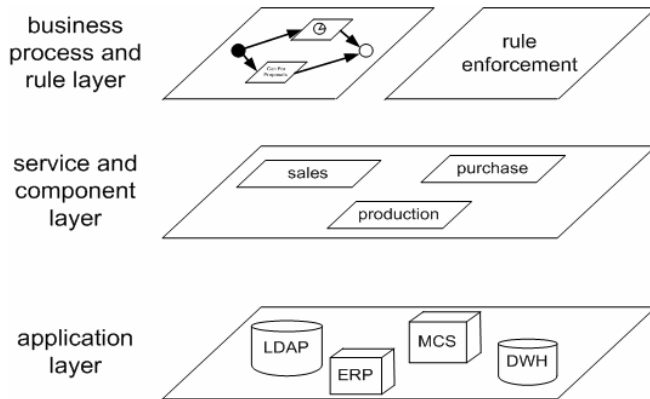
Table 2 Relating transition types to business rule types

| | <i>Create</i> | <i>Schedule</i> | <i>Assign</i> | <i>Revoke</i> | <i>Start</i> | <i>AddFact</i> | <i>RemoveFact</i> | <i>UpdateFact</i> | <i>Complete</i> | <i>Skip</i> | <i>Abort</i> | <i>Redo</i> |
|----------------------------|---------------|-----------------|---------------|---------------|--------------|----------------|-------------------|-------------------|-----------------|-------------|--------------|-------------|
| Temporal deontic rule | x | x | | | x | | | | | x | | x |
| Activity precondition | x | | | | x | | | | | | | x |
| Activity postcondition | | | | | | | | | x | | | |
| Dynamic integrity | x | | | | 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 | | | | x | | | | | | | x |
| Reaction rule | x | x | | | x | | | | | | | x |
| Static integrity | | | | | | | x | x | x | | | |
| Derivation rule | | | | | | | x | | x | | | |
| Activity authorisation | | | x | x | | | | | | | | |
| Visibility constraint | | | | | | | | | | | | |
| Event subscription | | | | | | | | | | | | |
| Activity allocation rule | | | x | x | | | | | | | | |

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 rule-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 5, 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.

Figure 5 A service-oriented architecture stacks



‘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 6:

- *Service providers* are software services that can perform activities or support the performance of activities. To each service provider an EM-BrA²CE agent can be mapped. A service provider is a recursive structure: it can represent an entire organisation 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 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-BrA²CE activity type can be mapped. Each activity service has 12 generic operations that implement the 12 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 2. 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-BrA²CE 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 7 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 2. There are two service providers, *aBuyer* and *aSeller*, that represent two organisations. The *place order* and *accept order* activity coordination services manage the state transitions of the activities for the buyer and the seller organisations 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 organisation 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.

Figure 6 Activity coordination service and concept service

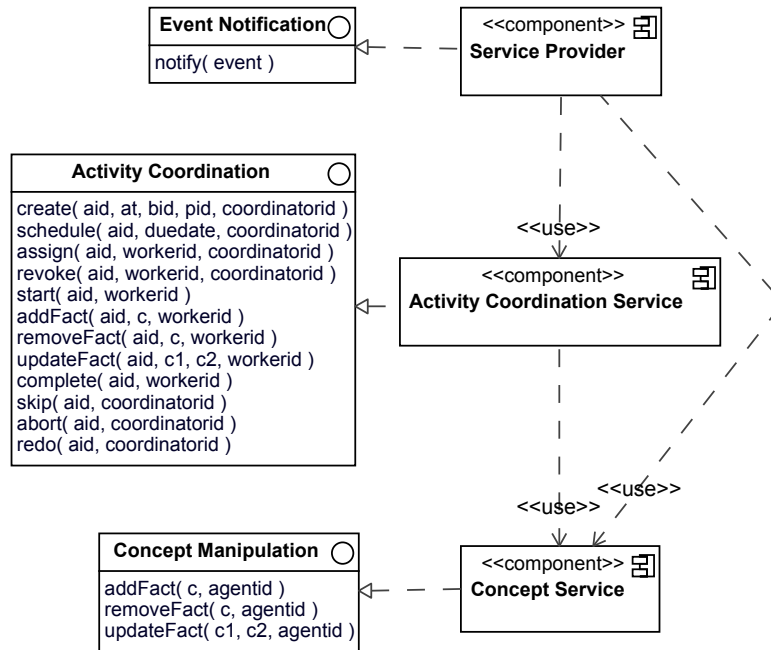
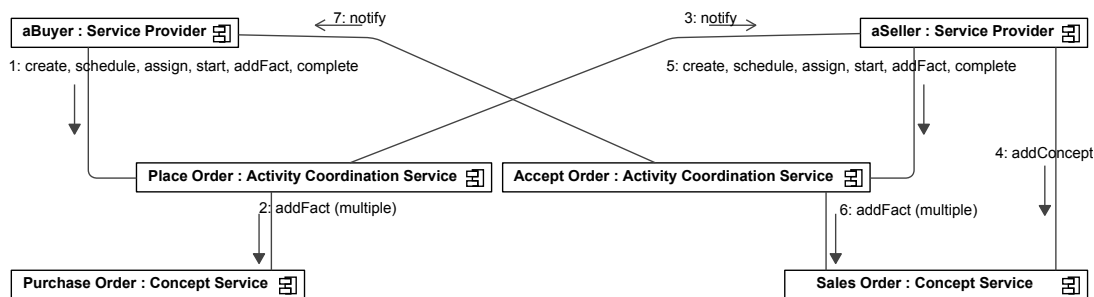


Figure 7 Example: a communication diagram



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

- The execution maintains the advantages of declarative modelling as it allows to enforce business rules without requiring a specification of when and how to check for rule violations.
- 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 synchronisation between process instances obsolete. 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.
- Process models in the EM-BrA²CE 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, organisational politics, legacy applications, quality-of-service requirements and the like must be taken into account. These concerns are outside the scope of this paper.

- 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.

5 Evaluation

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. Languages such as the case handling paradigm (van der Aalst et al., 2005), OWL-S (The OWL Services Coalition, 2004), the constraint specification framework of Sadiq et al. (2005), the WSMO (Roman et al., 2005), the ConDec language (Pesic and van der Aalst, 2006) and the

PENELOPE language (Goedertier and Vanthienen, 2006) can be considered declarative process languages. The EM-BrA²CE framework unifies the aspects of declarative process modelling that are present in these languages and by itself provides an overview of the relevant literature in the field. It has the following comparative advantages:

- The proposed EM-BrA²CE framework addresses control flow, data and resource aspects. This is realised by 16 different types of business rule that need to be enforced during the lifecycle of an activity. In contrast, popular process languages like BPMN (Object Management Group, 2006b) and UML Activity Diagrams (Object Management Group, 2005) predominantly focus on the control-flow perspective of business processes. In such process languages it might be possible to simulate data aspects using a control-flow-based modelling construct. For instance, the enforcement of a derivation or integrity constraint can be directly modelled in BPEL as a calculation or input validation step. However, by modelling how and when business rules must be enforced, process models are unnecessarily overburdened with procedural details.
- The EM-BrA²CE vocabulary models the event subscriptions and business concept visibilities of agents, but does not model how and when business events are communicated or how business facts are consulted. In contrast, procedural process models are overburdened with communication activities intended to notify an external business partner about the occurrence of a relevant business event.
- Few languages for declarative process modelling 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.
- Several authors have indicated the use of *reaction rules* to model behaviour (Brooks, 1991; Herbst, 1996). The rule-based process model, nonetheless, has some distinct features that enhance maintainability and expressivity over existing reaction rule-based approaches. It recognises that even a small number of reaction rules can be difficult to verify and validate. To improve maintainability, the process model modularises reaction rules into abstract states or decision points. Expressivity is also enhanced, because reaction rules are able to express composite events and activity events reflect the entire lifecycle of an activity instead of only the completion of an activity.

Declarative process modelling allows to include a lot of functional and non-functional 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, 2006). 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.

Keeping business processes and business rules separate at modelling time, raises the problem of how to link the enforcement of business rules, the manipulation of data and the enactment of processes at execution time. Several approaches to this problem are described in the literature. For instance, D'Hondt and Jonckers (2004) consider business rule enforcement a crosscutting concern in aspect-oriented programming. Business rule enforcement in this paper is situated at the level of business processes rather than at the level of individual applications.

Service-oriented architectures can be used to integrate the functionality of different applications, process support and business rule support while maintaining a strong decoupling. Rosenberg and Dustdar (2005) show how business rules can be integrated in the business process execution language (BPEL4WS) using a rule interceptor service that intercepts each incoming and outgoing BPEL web service call to automatically apply business rules. Goedertier and Vanthienen show how a process support service can enact a process model by decomposing the processing of each business event as a number of consecutive queries on an inference engine. There are several advantages to the EM-BrA²CE execution model:

- In the EM-BrA²CE execution model activities can undergo multiple state transitions. Unlike execution semantics of BPEL4WS, for instance, the lifecycle of activities does not merely involve the successful invocation of web services. Instead, 12 different state transitions are considered at which different types of business rule need to be enforced. Consequently, the EM-BrA²CE execution models offers more interception points – so-called point cuts in the context of aspect-oriented programming – than Rosenberg and Dustdar's solution does.
- Like OWL-S (The Owl Services Coalition, 2004) and WSMO (Roman et al., 2005) the EM-BrA²CE execution model allows to construct a suitable execution plan at runtime via the create, schedule and assign transitions. Because the execution model includes the event history into the state space of a (composite) activity, more expressive business rules can be formulated to constrain dynamic orchestration.
- The state of an activity does not live within an orchestration engine. Instead, facts about concepts (agents, activities and business concepts and events) are managed by concept services and can be used in the context of different activities. This eliminates the need

for maintaining consistency between the state of an orchestration engine and the state of the other information system components.

6 Conclusions

A rule-based 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. Rule-based process modelling allows including a lot of useful information that otherwise would remain implicit in procedural process models. The advantages of rule-based business process modelling manifest themselves during both the design and enactment phase of the BPM lifecycle. By documenting the governing business concerns, organisations 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. Rule-based process models can also be declaratively enacted in the context of service-oriented architectures. The proposed execution model offers much flexibility because the state of a business process has real business meaning and is not maintained within the context of an explicit process engine. In this paper, we have demonstrated how rule-based business process modelling and enactment contributes to flexible and compliant business processes, but we certainly do not claim to have identified and solved all the important problems. In particular, much research is still required regarding the parsing, visualisation, verification of declarative process models and the model-driven design of the service-based execution models.

Acknowledgements

We are grateful to the anonymous reviewers for their many constructive remarks.

References

- Andrews, T. et al. (2003) *Business Process Execution Language for Web Services (BPEL4WS)*, version 1.1.
- Antoniou, G., Billington, D., Governatori, G. and Maher, M.J. (2000) *A Flexible Framework for Defeasible Logics*, in AAAI/IAAI, pp.405–410, AAAI Press/The MIT Press.
- Antoniou, G., Taveter, K., Berndtsson, M., Wagner, G. and Spreeuwenberg, S. (2004) *A First-Version Visual Rule Language*, Report IST-2004-506779, REWERSE.
- Bajec, M. and Krisper, M. (2005) 'A methodology and tool support for managing business rules in organisations', *Information Systems*, Vol. 30, No. 6, pp.423–443.
- Brooks, R.A. (1991) 'Intelligence without representation', *Artificial Intelligence*, Vol. 47, No. 1–3, pp.139–159.
- Curtis, B., Kellner, M.I. and Over, J. (1992) 'Process modelling', *Commun. ACM*, Vol. 35, No. 9, pp.75–90.
- D'Hondt, M. and Jonckers, V. (2004) 'Hybrid aspects for weaving object-oriented functionality and rule-based knowledge', in *AOSD '04: Proceedings of the 3rd International Conference on Aspect-Oriented Software Development*, ACM Press, New York, NY, USA, pp.132–140.
- 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.*, Vol. 4, No. 3, pp.224–274.
- Goedertier, S. and Vanthienen, J. (2006) 'Designing compliant business processes with obligations and permissions', *Business Process Management Workshops*, LNCS 4103, pp.5–14.
- Guizzardi, G. and Wagner, G. (2005) *Some Applications of a Unified Foundational Ontology in Business Modelling*, chapter in: *Ontologies and Business Systems Analysis*, M. Rosemann and P. Green (Eds.), pp.345–367, IDEA Publisher.
- 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*, Vol. 24, No. 2, pp.79–88.
- Herbst, H. (1996) 'Business rules in systems analysis: a meta-model and repository system', *Inf. Syst.*, Vol. 21, No. 2, pp.147–166.
- Kardasis, P. and Loucopoulos, P. (2005) 'A roadmap for the elicitation of business rules in information systems projects', *Business Process Management Journal*, Vol. 11, No. 4, pp.316–348.
- O'Connor, M. (2005) 'The implications of Sarbanes-Oxley for non-US IT departments', *Network Security*, Vol. 7, pp.17–20.
- Object Management Group (2005) *UML 2.0 Superstructure Specification*, OMG Document – formal/05-07-04.
- Object Management Group (2006a) *Semantics of Business Vocabulary and Business Rules (SBVR) – Interim Specification*, OMG Document – dtc/06-03-02.
- Object Management Group (2006b) *Business Process Modelling Notation (BPMN) – Final Adopted Specification*, OMG Document – dtc/06-02-01.
- Object Management Group (2007) *Business Process Definition Metamodel – Final Submission*, OMG Document – bmi/2007-03-01.
- Pesic, M. and van der Aalst, W.M.P. (2006) 'A declarative approach for flexible business processes management', *Business Process Management Workshops*, LNCS 4103, pp.169–180.
- 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 Modelling Ontology*, Vol. 1, No. 1, pp.77–106.
- Rosenberg, F. and Dustdar, S. (2005) 'Business rules integration in BPEL – a service-oriented approach', in *CEC*, IEEE Computer Society, pp.476–479.
- Ross, R.G. (2003) *Principles of the Business Rule Approach*, Addison-Wesley Professional.
- Sadiq, S.W., Orłowska, M.E. and Sadiq, W. (2005) 'Specification and validation of process constraints for flexible workflows', *Inf. Syst.*, Vol. 30, No. 5, pp.349–378.
- Spreeuwenberg, S., Gerrits, R. and Boekenoogen, M. (2000) *Valens: A Knowledge Based Tool to Validate and Verify an Aion Knowledge Base*.

- Strembeck, M. and Neumann, G. (2004) 'An integrated approach to engineer and enforce context constraints in rbac environments', *ACM Trans. Inf. Syst. Secur.*, Vol. 7, No. 3 pp.392–427.
- The OWL Services Coalition (2004) *OWL-S 1.1 release*, July, available at <http://www.daml.org/services/owl-s/1.1/>.
- van der Aalst, W.M.P., Weske, M. and Grünbauer, D. (2005) 'Case handling: a new paradigm for business process support', *Data Knowl. Eng.*, Vol. 53, No. 2, pp.129–162.
- van Hee, K., Oanea, O., Serebrenik, A., Sidorova, N. and Voorhoeve, M. (2006) 'Modelling history-dependent business processes', in *MSVVEIS*, pp.76–85.
- Vanthienen, J., Mues, C. and Aerts, A. (1998) 'An illustration of verification and validation in the modelling phase of KBS development', *Data Knowl. Eng.*, Vol. 27, No. 3, pp.337–352.
- Wagner, G. (2003) 'The agent-object-relationship metamodel: towards a unified view of state and behavior', *Information Systems*, Vol. 28, No. 5, pp.475–504.
- 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*, Vol. 42, No. 1–3, pp.227–253.