A Mapping from Normative Requirements to Event-B to Facilitate Verified Data-Centric Business Process Management

Iman Poernomo, Timur Umarov

A Mapping from Normative Requirements to Event-B to Facilitate Verified Data-Centric Business Process Management

Iman Poernomo[1] and Timur Umarov[2]

[1] Department of Computer Science, King's College London
Strand, London, UK, WC2R2LS
(iman.poernomo@kcl.ac.uk)

[2] Department of Computer Engineering, Kazakh-British Technical University
59 Tole bi str., Almaty 050000 Kazakhstan
(t.umarov@kbtu.kz)

Abstract. This paper addresses the problem of describing and analyzing data manipulation within business process workflow specifications. We apply a model-driven approach. We begin with business requirement specifications, consisting of an ontology and an associated set of normative rules, that define the ways in which business processes can interact. We then transform this specification into an Event-B specification. The resulting specification, by virtue of the Event-B formalism, is very close to a typical loosely coupled component-based implementation of a business system workflow, but has the additional value of being amenable to theorem proving techniques to check and refine data representation with respect to process evolution.

1 Introduction

Business process management (BPM) is an increasingly challenging aspect of the enterprise. Middleware support for BPM, as provided by, for example, Oracle, Biztalk and the recent Windows Workflow Foundation (WWF), has met some challenges with respect to performance and maintenance of workflow.

The central challenge to BPM is complexity: business processes are becoming widely distributed, interoperating across a range of inter- and intra-organizational vocabularies and semantics. It is important that complex business workflows are checked and analyzed for optimality and trustworthiness prior to deployment. The problem becomes worse when we consider the enterprise's demand to regularly adapt and change processes. For example, the growth of a company, changes to the market, revaluation of tasks to minimize cost. All these factors often require reengineering or adaptation of business processes along with continuous improvement of individual activities for achieving dramatic improvements of performance critical parameters such as quality (of a product or service), cost, and speed [1]. Reengineering of a complex workflow implementation is dangerous, due to existing dependencies between tasks.

Formal methods can assist in meeting the challenge of complexity, as their mathematical basis facilitates analysis and refining of a system specification. However, complex systems often involve a number of different aspects that entail separate kinds of analysis and, consequently, the use of a number of different formal methods.

A business process implementation within a BPM middleware requires detailed treatment of both information flow and information content. The abstraction gap is identified by Hepp and Roman in [2]: an abstract workflow that ignores information content provides an abstract view of business processes that does not fully define the key aspects necessary for BPM implementation.

We argue that this abstraction gap can be addressed by developing event-driven data models in the Event-B language from an initial business process requirements specification. We employ a Model Driven Architecture approach.

The initial CIM might be written as models within a number of requirements specification frameworks. We use the ontologies and normative language of the MEASUR method [3]. The method has a 20 year history and is widely used within the organizational semiotics community, but less well-known in Computer Science. Its roots lie in the philosophical pragmatism of Peirce, the semiotics of Saussure and Austin's speech act theory. It is model-based, with ontologies and normative constraints forming the central deliverables of a requirements document. We employ MEASUR notation because our starting point is information systems analysis, where MEASUR has found the most application. Its normative constraints lend themselves to transformation into our PIM languages. However, our approach should be readily adaptable to a number of similar notations in use in the multi-agents and normative specification research communities.

The Event-B language is used in specifying, designing, and implementing software systems. The language is used to develop software by a process of gradual refinement, from an abstract, possibly nonexecutable system model, to intermediate system models that contain more detail on how to treat data and algorithms, to a final, optimized, executable system. In this process, (i) the first abstract model in this refinement chain is verified for consistency and (ii) each step in the refinement chain is formally checked for semantic preservation. Consistency will then be preserved throughout the chain. Therefore, the final executable refinement can be trusted to implement the abstract specification.

Our approach addresses the semantic gap by defining a transformation of MEASUR models to Event-B machines, permitting: (i) a full B-based formal semantics for vocabularies and data manipulation that is carried out within the modeled workflow, which can be validated for consistency; and (ii) an initial, abstract B model that can be refined using the B-method to a final optimal executable system in an object-oriented workflow middleware, such as WWF.

A notion of semantic compatibility holds over the transformed models, so that any property derived over the normative-ontological view of the system will hold over potential processes that arise from the Event-B machine.

The paper proceeds as follows:

- In Section 2, we sketch the nature of our CIM, the normative ontology language of MEASUR.
- Section 3 provides a brief introduction to Event-B specifications, focusing on the main points relevant to our formal semantics.

- Section 4 then outlines the transformation approach to generating Event-B specification. We discuss how the resulting specification provides a formal semantics of our data-centric business process, and how this enables consistency validation checks.
- Section 5 discusses related work and conclusions.

2 MEASUR Models

The MEASUR can be used to analyze and specify an organization's business processes via three stages [3]: (i) articulation of the problem, where a business requirements problem statement is developed in partnership with the client; (ii) semantic analysis, where the requirements problem statement is encoded as an ontology, identifying the main roles, relationships and actions; and (iii) norm analysis, where the dynamics of the statement are identified as social norms, deontic statements of rights, responsibilities and obligations.

Space does not permit us to detail the first stage. Its processes are comparable to other well known approaches to requirements specification. The last two stages require some elaboration. For our purposes, they provide a Computation Independent Model, consisting of an ontology and collection of norms, that formally define the structure and potential behaviour of an organization and its processes from a non-technical perspective. We hereafter refer to the combination of a MEASUR ontology and associated norms as a *normative ontology*.

2.1 Ontologies

The ontologies of semantic analysis are similar to those of, for example, OWL, decomposing a problem domain into roles and relationships. As such, our ontologies enable us to identify the kinds of data that are of importance to business processes. A key difference with OWL is the ability to directly represent *agents* and *actions* as entities within an ontology. This is useful from the perspective of business process analysis, as it enables us to identify tasks of a workflow and relate them to data and identify what agent within the organization has responsibility for the task.

Semantic Analysis has its roots in semiotics, the philosophical investigation of signs. MEASUR applies to information system analysis a number of ideas and approaches from philosophy of language, drawing on the pragmatism of Peirce, semiotics of Saussure and the epistemology of Wittgenstein and Austin. The method's core assumption is knowledge and information exists only in relation to a knowing *agent* (a single human or a social organization). There is no Platonic reality which defines Truth. Instead, Truth is a derived concept that might be defined as *agreement* between a group of agents. An agent is *responsible* for its knowledge. When a group of agents agree on what is true and what is false, they accept responsibility for that judgement. Following Wittgenstein, MEASUR considers an information system as a "language game": a form of activity involving a party of agents that generates meaning. In an information-system-as-language-game, the meaning of data derives from usage by agents, rather than from a universal semantics.

Semantic Analysis represents the information system as language game in the form of an ontology diagram, identifying agents, the kinds of actions agents can perform and the relationships and forms of knowledge that can result from actions. These concepts are identified as types of *affordance*. An affordance is a collection of patterns of behaviour that define an object or a potential action available to an agent. Every concept in a MEASUR ontology is an affordance.

MEASUR subclasses the notion of affordance as follows. A *business entity* – such as a user account or a bank loan – is an affordance in the sense that it is associated with a set of permissible behaviours and possibilities of use. For the purpose of business process analysis, business entities are used to identify the main kinds of data that are of importance in an organization's processes. A *relationship* – such as a contract – between business entities or agents is an affordance in the sense that it is defined by the behaviour it generates for the parties involved in the contract. *Agents* are affordances in terms of the actions they can perform and the things that may be done to them. Agents then occupy a special status in that they take responsibility for their own actions and the actions of others and can authorize patterns of behaviour. The structure of a business entity, relationship or agent is given via a list of associated properties, called *determiners*. Determiners are properties and attributes of affordances, such an address or telephone number associated with a user account. *Units of measurement* are typical data types that type determiners and other values associated with affordances. The latter two concepts are considered as affordances as their values constrain the possible behaviour of their owners.

In our treatment, affordances can be treated as types of things within a business system, with an ontology defining a type structure for the system. An actual executing system consists of a collection of affordance *instances* that possess the structure prescribed by the ontology and obey any further constraints imposed an associated set of norms.

Example 1. An ontology for the purchasing system is given in Fig. 1. Agents are represented as ovals and business entities as rectangles with curved edges. Communication acts and relations are shown as rectangles, with the former differentiated by the use of an exclamation mark *!* before the act's name.

All affordances (including agents and business entities) have a number of typed attributes, defining the kinds of states it may be in. We permit navigation through an affordance's attributes and related affordances in the object-oriented style of the OCL. The system also involves processes that cross the boundaries of two subsystems: an order processing system, and a product warehouse system. These two subsystems are represented as agents in the ontology, *eOrder* and *Warehouse*, respectively. By default all agents contain start and end attributes.

Orders are *requests* for *products*, both represented as entities in the ontology with a *requests* relationship holding between them (multiplicities could be associated with the relationship to define the possibility of a number of products contained in an order). An order is associated with its customer, defined by the *ordered_by* relationship holding between the *customer* agent and *order* entity. An order can stand in an *ordered* relationship with the *eOrder* agent, after it has been successfully processed. Communication act *!receive_order* corresponds to the initial reception of data. The *Processing* communica-

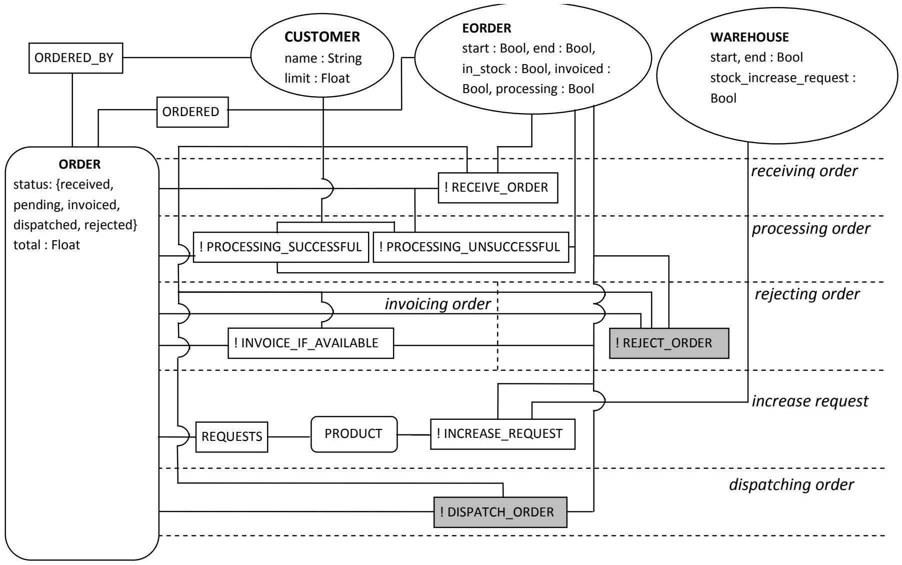

Fig. 1. Example normative ontology .

tion act further deals with the newly arrived order and checks whether the clients credit limit allows for the purchase. Namely, it checks whether the total cost of the purchase is less than a customer's credit limit. This condition results in the following outcomes: if the credit limit is lower than the total cost then the system rejects the order, otherwise it initiates the invoicing process (denoted by the *invoice_if_available* communication act). It does so if the stock contains enough amount of the product for the order. If not, then the system requests to increase the stock by initiating *request_increase*. This is followed by the actual process of increasing the stock *increase_stock*. Finally, the system dispatches the order by *dispatch_order*.

2.2 Norms

Norms are constraints and rules that determine how agents interact and control affordances. They also control the initialization and termination of particulars (affordance instances). We have adopted a typed language of deontic logic, logic of action, and the theory of normative positions to express logical constraints over business processes, using ontologies as atomic classes, relations, objects and actions for the logic. Our constraints take the form

$$\begin{aligned} A, B := R(\bar{a}) \mid \neg A \mid A \vee B \mid A \wedge B \mid A \rightarrow B \mid \\ \forall x : C \bullet A \mid \exists x : C \bullet B \mid Ob\, A \mid Pe\, A \mid Im\, A \mid E_x\, A\,, \end{aligned} \tag{1}$$

where C is an affordance (that acts as a type of a particular instance); $R(\bar{a})$ is an affordance with one or two antecedents $\bar{A}$ and $\bar{a}$ is one or two particular instances of $\bar{A}$; the meaning of ObA is that A is obliged to happen; the meaning of PeA is that A is permitted to happen; the meaning of ImA is that A is prohibited (impermissible) to happen; the meaning of $E_x A$ is that A results from, and is the responsibility of, agent particular x; the meaning of the other connectives follows standard first order logic. A conditional description of behaviour or conditional act description, which we otherwise regard as a *behavioral* norm, represents the general form for a constraint over our ontologies [3]:

$$(\text{Trigger} \wedge \text{pre-condition}) \rightarrow E_{\text{agent}} Ob/Pe/Im\, post\text{-}condition. \tag{2}$$

The informal meaning of the norm is the following: if *Trigger* occurs and the *pre-condition* is satisfied, then *agent* performs an action so that *post-condition* is Obliged/Permitted/Impermissible from resulting.

The idea of a behavioral norm is to associate knowledge and information with agents, who produce and are responsible for it. From a philosophical perspective, truth is then defined as something that an agent brings about and is responsible for. As shall be seen, from the perspective of determining how to *implement* a normative ontology as a workflow-based system, we view agents as corresponding to subsystems, business entities to specify data and behavioral norms to expected dynamic interaction protocols between subsystems.

Example 2. Consider the communication act *!receive_order* from our example, corresponding to the initial reception of data by the order processing system. The idea that this reception can only occur over orders that are not yet processed is captured by the behavioral norm shown in Table 1. Both relationships and communication acts are represented as logical relations in our language, but communication acts are not used in pre-conditions, and may only be placed *after* a Deontic operator.

Communication acts often define resulting changes of state on related agents and entities. As shown in our ontology in Fig. 1, *receive_order* relates three affordances: agents *Customer* and *eOrder* and business entity *Order*, instances of which are used as arguments for this communication act. As such, this communication act should affect the following relationships *ORDERED* and *ORDERED_BY* that are involved in relating the pertinent affordances. Therefore, the reception of an order entails a change of state of affairs to include a newly arrived order, the status becomes set to "received", and the system initiates the processing stage by setting its attribute to *true*. This is formalized in Table 1.

Table 1. Communication act *!receive_order*: norm and definition.

!RECEIVE_ORDER	
NORM	$\forall\, cc : Customer \bullet \forall\, oo : Order \bullet \forall\, e : eOrder \bullet$ $\neg ordered\_by(oo, cc) \wedge \neg ordered(oo, e) \rightarrow E_e\ Ob\ \mathbf{receive\_order}(\mathbf{oo}, \mathbf{cc}, \mathbf{e})$
DEFINITION	$\forall\, cc : Customer \bullet \forall\, oo : Order \bullet \forall\, e : eOrder \bullet \mathbf{receive\_order}(\mathbf{oo}, \mathbf{cc}, \mathbf{e}) \rightarrow$ $ordered\_by(oo, cc) \wedge ordered(oo, e) \wedge oo.status = received \wedge e.processing = \top$

There have been a number of attempts to use semantic analysis normative ontologies as the language for a business process management engine. The mostly widely used is

Liu's NORMBASE system [3]. In such systems, the ontology serves as a type system for data, while norms define the conditions under which tasks may be invoked to create and manipulate data.

Our approach is different: we treat normative ontologies as a useful and semantically rich requirements analysis document. However, we implement these requirements using a standard business process management infrastructure. We believe that further refinement and analysis is a necessary step to this goal. In particular, it is important to ensure that (i) the possible communication act traces permitted by a set of norms do not deadlock unexpectedly (in our example, this happens if the order processing system waits indefinitely for a response from the warehouse that stock is available); and (ii) the ontology and its associated norms do not allow for an inconsistent state of the system (this happens if an action entails that an order is processed and rejected at the same time).

3 Event-B

This section provides an overview of the Event-B notation which is inspired by the action systems approach [4] and represents an evolution of the B-method. The Event-B language specifies a software system in terms of encapsulated modules, called machines, that consist of a mutable state and a number of related operations, called events, whose execution changes the values of the state. Each event consists of a logical *guard* and an *action*. The guard is a first order logical statement about the state of the machine and defines the conditions under which an action may occur. The action defines the way the machine's state may be modified as a first-order logical statement relating the initial values of the state prior to the action occurring and the final values of state.

Machines therefore have a formal operational semantics, that models system execution as a sequence of events. If an event's guard holds over the machine's state, its action may be executed. This will change machine's state, which may cause another event's guard to hold, and an action to be executed. The sequence continues until the system has halted (it is deadlocked). Note that execution is potentially nondeterministic: when a number of event guards are true, then *one* of the corresponding event actions is chosen at random.

A common requirement over business process descriptions is the preservation of certain properties throughout the whole course of execution of events. These properties are called *invariants*: they represent predicates built on the state variables that must hold permanently. This is achieved by proving that under this invariant and guards of events, the invariant still holds after modifications made to the state variables associated with event executions.

Every model written in Event-B is represented as a machine/context pair. The relationship between these two constructs is that the machine "sees" the context (read-only access, with no modification possible). The context contains the main sets, constants, axioms, and theorems. Carrier sets and enumerated sets are declared in the Sets section. Since, an enumerated set is a collection of elements, additionally, its members are defined as constants in the Constants section. An axioms section contains assignments according to which constants are defined as unique values.

A machine consists of state variables, invariants, and events. State variables represent states which the machine can be in. The Invariants box is comprised of the conditions that should hold throughout the whole execution of the machine. The events box contains the initialization construct and all events of the machine. Each event contains one or several guards and one or several actions.

Definition 1 (Consistent Event-B Machine). *An Event-B machine is consistent if the following conditions hold:*

- *Invariant preservation : for any event, assuming the invariant and guard are true, then the invariant and action are consistent (do not result in a contradiction).*
- *Feasibility: given any event, if the guard holds, then it is possible for the action to be performed.*

It is possible to define an operational semantics for Event-B machines, over which the runtime execution of the modeled system can be understood. Essentially, this is done by assuming the the initialization constraints to hold over the state of the machine (actual values assigned to its set of variables), and then successively selecting events based on guard checks over the variables. Each event selection will result in the action condition changing the state of the system. The resulting sequence of events is a *trace* of the machine. A machine will usually have a potentially infinite number of traces, due to the nondeterminism of guard selection and the nondeterminism within actual actions.

4 Semantic Embedding of Normative Ontologies in Event-B

In MDA terms, ontologies and norms represent a computation independent model (CIM) and Event-B machines represent platform indepedent model (PIM). CIM does not contain information of computational nature and usually describes the state of affairs verbally or using high-level logical expressions. Whereas in PIM, one can find model descriptions defined in a more mathematical manner.

In this section, we are providing a stepwise detailed mapping of our normative rule to the Event-B machine. It is important to note that we are applying our formal definition of the mapping described in Appendix.

4.1 General Mapping Strategy

The mapping of affordances is straightforward. The general framework is shown in Fig. 2.

All the elements of the source and target models are marked with different patterns to be able to segregate separate mappings of the subclasses of affordances to Event-B constructs: agents are mapped to machines, business entities and relations are mapped to Event-B sets, relations and state variables, and communication acts are mapped to events. Generating events is a rather detailed transformation and we describe it more extensively below.

The transformation of normative constraints is more difficult. Conceptually, norms of the form (2) appear similar in form to a machine event:

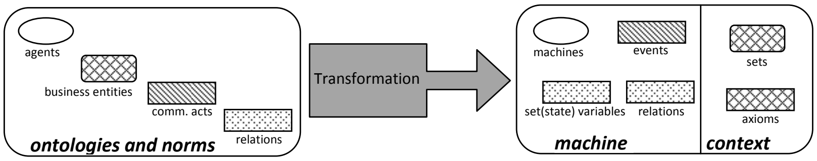

Fig. 2. The general framework of the transformation.

- A *trigger* and *pre-condition* correspond to a *guard*. The former defines the situation that must hold before an agent can act. The latter defines the state that must hold before a machine can perform an action.
- The responsibility modality E_a corresponds to the location of the event within the machine corresponding to agent *a*.
- The deontic modality *Ob/Pe post-condition* identifies whether the action corresponding to *post-condition* should be necessarily performed, or whether execution of another (skip action) is possible instead. The *Im* deontic modality means the negation of the post-condition holds.

Because the normative constraints are essentially abstract business rules, while the conditions of the B machine define further implementation-specific detail, the mapping will depend on how we interpret relations and functions of the ontology. For this purpose our transformation must be based on a given semantic mapping of individual relations and functions to B relations and functions. We assume this is defined by a domain expert with the purpose of wide reusability for the ontology's domain.

4.2 Example: A Stepwise Transformation of *receive_order*

Space does not permit the full definition of the MDA transformation. Instead, we illustrate the idea by showing how the transformation applies to a single norm – that of *receive_order* in the example.

Let us consider a norm for receiving an order described in Table 1. This norm generates event *receive_order* in machine *e*, because *e* is the responsible agent for the *effect* construct. Besides agent *e*, the norm contains another agent *cc*. Consequently, its corresponding mapping will also affect in certain ways the appropriate machine of *cc* as we illustrate below.

Norm *receive_order(oo,cc,e)* contains a composite guard, well-formedness constraint of which is defined as follows

$$\begin{aligned}&WFC'(GUARD(\neg ordered\_by(oo,cc) \wedge \neg ordered(oo,e))) \equiv\\ &WFC'(GUARD(\neg ordered\_by(oo,cc))) \cup WFC'(GUARD(\neg ordered(oo,e)))\,,\end{aligned} \tag{3}$$

and is recursively formalized as

$$\begin{aligned}&GUARD(\neg ordered\_by(oo,cc) \wedge \neg ordered(oo,e)) =\\ &GUARD(\neg ordered\_by(oo,cc)) \wedge GUARD(\neg ordered(oo,e))\,.\end{aligned} \tag{4}$$

Each pre-condition, *¬ordered_by(oo,cc)* and *¬ordered(oo,e)*, contain instances of *Entity* and *Agent*. Furthermore, one of them contains an instance of an agent different than *e*. In our approach, since these guards are of the form *R(a,c)* we formally define them as follows:

$$\neg ordered\_by(oo, cc) \overset{\varphi}{\mapsto} \ulcorner \neg oo \in ordered\_by \urcorner , \tag{5}$$

where each variable is rigorously defined as

$$\left\{ \begin{array}{r} \ulcorner oo \urcorner \in ExtVAR(cc), \\ \ulcorner ordered\_by \urcorner \in ExtVAR(cc), \\ \ulcorner oo \in Order \urcorner \in INV(cc), \\ \ulcorner ordered\_by \subseteq Order \urcorner \in INV(cc), \\ \ulcorner oo \urcorner \in ExtVAR(e), \\ \ulcorner ordered\_by \urcorner \in ExtVAR(e), \\ \ulcorner oo \in Order \urcorner \in INV(e), \\ \ulcorner ordered\_by \subseteq Order \urcorner \in INV(e) \end{array} \right\} \in WFC'(GUARD(\neg ordered\_by(oo, cc)))\,; \tag{6}$$

and

$$\neg ordered(oo, e) \overset{\varphi}{\mapsto} \ulcorner \neg oo \in ordered \urcorner , \tag{7}$$

where for each variable we have

$$\left\{ \begin{array}{r} \ulcorner oo \urcorner \in IntVAR(e), \\ \ulcorner ordered \urcorner \in IntVAR(e), \\ \ulcorner oo \in Order \urcorner \in INV(e), \\ \ulcorner ordered \subseteq Order \urcorner \in INV(e) \end{array} \right\} \in WFC'(GUARD(\neg ordered(oo, e)))\,. \tag{8}$$

Note that our transformation produces a well-formed condition that *oo* is both an external and an internal variable of machine *e*. This is not a problem, as we can assume that property of being external (*ExtVAR*) overrides being internal (*IntVAR*). This is because in Event-B a variable is rendered external simply by having its name shared between two machines.

Variable *oo* is defined as shared variable (*ExtVAR*), being an instance of *Entity Order*. Additionally, it can be seen that in (6) and (8) a definition of *oo* as an instance of *Order* is represented as an invariant construct (denoted as *INV*). Guard definitions (5) and (7) with an invariant specification are used in generating Event-B invariants and sets, where only instances of entities are involved (note: instances of *Agent* are irrelevant in generating sets). Applying this to our norm yields an invariant of the form $oo \in Order$ for both machines, because variable *oo* is shared. Shared variable *ordered_by* and internal (to machine *e*) variable *ordered* are defined as subsets of *Order* and these constraints are specified as invariants $ordered\_by \subseteq Order$ and $ordered \subseteq Order$, respectively.

Furthermore, each machine has an event with its guard defined:

$$\ulcorner \neg oo \in ordered\_by \urcorner \in cc \tag{9}$$

and

$$\ulcorner \neg oo \in ordered \urcorner \in e\,. \tag{10}$$

Communication act *receive_order(oo,cc,e)* relates the instance of *Entity oo* to instances of *Agent cc* and *e*. For this reason we are first generating an action for invoking this event. The machine that is calling event *receive_order* is *cc* and the machine which contains this event is *e*, because the responsible agent in our norm is *e*. Therefore, for machine *cc* we have

$$\mathit{ACTION}(\mathit{receive\_order}(oo,cc,e)) \equiv \ulcorner \mathit{receive\_order}_{\mathit{call}} := \top \urcorner , \tag{11}$$

where we type a shared variable as:

$$\ulcorner \mathit{receive\_order}_{\mathit{call}} \in \mathit{BOOL} \urcorner \in \mathit{INV}(cc) \land \mathit{INV}(e) , \tag{12}$$

which represents an invariant for machines *cc* and *e*. Shared variable $\mathit{receive\_order}_{\mathit{call}}$ is constrained strictly to machines *cc* and *e*:

$$\mathit{receive\_order}_{\mathit{call}} \in \mathit{ExtVAR}(cc) \land \mathit{ExtVAR}(e) . \tag{13}$$

The definition of the calling event in machine *cc* will take the following form:

$$\begin{array}{l} \ulcorner \mathit{Event}\ \mathbf{receive\_order\_trigger}\ \ \widehat{=} \urcorner \\ \ \text{WHEN}\ \neg oo \in \mathit{ordered\_by} \qquad\qquad \in cc\ \ . \\ \ \text{THEN}\ \mathit{receive\_order}_{\mathit{call}} := \top \end{array} \tag{14}$$

The guard for $\ulcorner \varphi(\mathit{receive\_order}) \urcorner$ should include additionally to (10) the following constraint to enable the communication between machines *cc* and *e*:

$$\ulcorner \mathit{receive\_order}_{\mathit{call}} = \top \urcorner \in \mathit{GUARD}_{\mathit{FOR}}(\varphi(\mathit{receive\_order}), e) \tag{15}$$

under the side-condition that

$$\ulcorner \mathit{receive\_order}_{\mathit{call}} = \bot \urcorner \in \mathit{GUARD}_{\mathit{FOR}}(x, e) \tag{16}$$

for every event $x \in e$ such that $x \neq \varphi(\mathit{receive\_order})$ in order to prevent unwanted invocations of $\varphi(\mathit{receive\_order})$ by other events in *e*. $\ulcorner \varphi(\mathit{receive\_order}) \urcorner$'s final action should include

$$\ulcorner \mathit{receive\_order}_{\mathit{call}} := \bot \urcorner . \tag{17}$$

The mapping of meaning of the norm

$$\begin{array}{l} \forall\, cc : \mathit{Customer} \bullet \forall\, oo : \mathit{Order} \bullet \forall\, e : \mathit{eOrder} \bullet \mathbf{receive\_order}(\mathbf{oo}, \mathbf{cc}, \mathbf{e}) \rightarrow \\ \quad \mathit{ordered\_by}(oo, cc) \land \mathit{ordered}(oo, e) \land \mathit{oo.status} = \mathit{received} \land \mathit{e.processing} = \top \end{array} \tag{18}$$

yields the following rules

$$\begin{array}{l} \mathit{ordered\_by}(oo, cc) \overset{\varphi}{\mapsto} \ulcorner \mathit{ordered\_by} := \mathit{ordered\_by} \cup \{oo\} \urcorner \\ \ \in \mathit{ACTION}_{\mathit{FOR}}(\varphi(\mathit{receive\_order}), e) \end{array} \tag{19}$$

and

$$\begin{array}{l} \mathit{ordered}(oo, e) \overset{\varphi}{\mapsto} \ulcorner \mathit{ordered} := \mathit{ordered} \cup \{oo\} \urcorner \\ \ \in \mathit{ACTION}_{\mathit{FOR}}(\varphi(\mathit{receive\_order}), e) , \end{array} \tag{20}$$

and for function *status* with argument *oo* we effectively have

$$\begin{aligned} &\mathit{oo.status} = \mathit{received} \overset{\varphi}{\mapsto} \ulcorner \mathit{status}(\mathit{oo}) := \mathit{received} \urcorner \\ &\in \mathit{ACTION}_{\mathit{FOR}}(\varphi(\mathit{receive\_order}), e) \end{aligned} \tag{21}$$

and for function *processing* with argument *e*, we define

$$\mathit{e.processing} = \top \overset{\varphi}{\mapsto} \ulcorner \mathit{processing} := \top \urcorner \in \mathit{ACTION}_{\mathit{FOR}}(\varphi(\mathit{receive\_order}), e)\,. \tag{22}$$

We now have to include the definition of function *status* in our machine *e*

$$\mathit{status} \in \mathit{IntVAR}(e), \tag{23}$$

and since *status* is a function, the transformation creates an appropriate invariant for it:

$$\ulcorner \mathit{status} \in \mathit{Order} \to \mathit{STATUS} \urcorner \in \mathit{INV}(e)\,, \tag{24}$$

where *STATUS* is an enumeration set with the predefined values which are taken from the ontology and $\mathit{received} \in \mathit{STATUS}$.

The full definition of event *receive_order* of machine *e* will take the following form:

$$\begin{array}{ll} \ulcorner \mathit{Event}\ \textbf{receive\_order} \ \widehat{=} & \urcorner \\ \quad \text{WHEN}\ \mathit{receive\_order}_{\mathit{call}} = \top & \\ \qquad\qquad \neg \mathit{oo} \in \mathit{ordered} & \\ \quad \text{THEN}\ \ \mathit{ordered} := \mathit{ordered} \cup \{\mathit{oo}\} & \in e\ . \\ \qquad\qquad \mathit{ordered\_by} := \mathit{ordered\_by} \cup \{\mathit{oo}\} & \\ \qquad\qquad \mathit{status}(\mathit{oo}) := \mathit{received} & \\ \qquad\qquad \mathit{processing} := \top & \\ \qquad\qquad \mathit{receive\_order}_{\mathit{call}} := \bot & \end{array} \tag{25}$$

The overall mapping of norm *receive_order(oo, cc, e)* is shown in Table 2.

5 Conclusion and Related Work

In this paper, we have provided a detailed description of our MDA mapping approach that we have applied for implementing the transformation from normative ontologies and norms to Event-B machines in order to address the semantic gap. For our source model we have combined MEASUR with the action logic and deontic logic in the light of the theory of normative positions. We have shown how semantic embedding of normative ontologies can be performed in Event-B for norm *receive_order*. We have also given a description of the implementation by presenting different components of the transformation and their formal definitions, and provided a formal definition of the transformation in the appendix.

There are a number of related approaches for enriching workflow models [5,6]. One of them is showing transformation from BPEL4WS to full OWL-S ontology to provide missing semantics in BPEL4WS. BPEL4WS does not present meaning of a business process so that business process can be automated in a computer understandable way

Table 2. Norm mapping for receive_order

Norm	Generated Constructs	
	MACHINE cc	**MACHINE** e
	VARIABLES	**VARIABLES**
	$receive\_order_{call}$	$receive\_order_{call}, ordered$
	$oo, ordered\_by$	$oo, ordered\_by, status, processing$
	INVARIANTS	**INVARIANTS**
	$receive\_order_{call} \in BOOL$	$receive\_order_{call} \in BOOL$
	$oo \in Order$	$oo \in Order$
	$ordered\_by \subseteq Order$	$ordered\_by \subseteq Order$
		$ordered \subseteq Order$
		$status \in Order \rightarrow STATUS$
		$processing \in BOOL$
	Event receive_order_trigger $\widehat{=}$	**Event receive_order** $\widehat{=}$
$\forall\, cc : Customer \bullet \forall\, oo : Order \bullet \forall\, e : eOrder \bullet$	*WHEN*	*WHEN*
$\neg ordered\_by(oo, cc) \wedge \neg ordered(oo, e) \rightarrow$	$\neg oo \in ordered\_by$	$receive\_order_{call} = \top$
$E_e\ Ob$ $\mathbf{receive\_order(oo, cc, e)}$		$\neg oo \in ordered$
$\overset{\varphi}{\mapsto}$	*THEN*	*THEN*
$\forall\, cc : Customer \bullet \forall\, oo : Order \bullet \forall\, e : eOrder \bullet$	$receive\_order_{call} := \top$	$ordered := ordered \cup \{oo\}$
$receive\_order(oo, cc, e) \rightarrow ordered\_by(oo, cc) \wedge$		$ordered\_by := ordered\_by \cup \{oo\}$
$ordered(oo, e) \wedge oo.status = received \wedge$		$status(oo) := received$
$e.processing = \top$		$processing := \top$
		$receive\_order_{call} := \bot$
	END	*END*
	END	**END**

[7]. They are using overlap which exists in the conceptual models of BPEL4WS and OWL-S and perform mapping from BPEL4WS to OWL-S to avoid this lack of semantics.

Norms are used in different areas for regulating and constraining behavioral patterns in certain organized environments. For example, in the area of artificial intelligence and multi-agent systems [8], agents need to organize their action patterns in a way to avoid conflicts, address complexity, reach agreements, and achieve a social order. These patterns are specified by norms which constrain what may, must and must not be done by an agent or a set of agents. The fulfillment of certain tasks by agents can be seen as a public good if the benefits that they bring can be enjoyed by the society, organization or group [8,9].

There are several works in the scope of MDA devoted to CIM-to-PIM transformation. For example, Rodriguez, et al. [10] define CIM-to-PIM transformation using QVT mapping rules. The model of the information system that they obtain as a PIM are represented as certain UML analysis-level classes and the whole idea is reflected in a case study related to payment for electrical energy consumption. This work was continued [11] by extending CIM definitions of BPMN to define security requirements and transforming them into UML use cases using QVT mapping rules. Another approach described by Wil van der Aalst, et al. [12] meets the difficulty of BPEL. While being a powerful language, BPEL is difficult for end-users to use. Its XML representation is very verbose and only readable for the trained eye [12]. It describes implementation of the transformation from Workflow-Nets to BPEL which is built on the rich theory of Petri nets and can also be applied for other languages.

The Event-B language was successfully applied in several serious projects where there was a need for rigorous and precise specification of the system. For example, Rezazadeh, et al. [13] discuss redevelopment of the central control function display and information system (CDIS). CDIS is a computer-based system for controlling important airport and flight data for London terminal control center. The system was originally developed by Praxis and was operational but yet had several problems related to the questions of formalization. Namely, the problems included difficulty of comprehending the specifications, lack of mechanical proof of the consistency and difficulties in distribution and refinement. These problems were addressed in redeveloping the system using the advantages of the Event-B language and Rodin platform.

Future work will investigate how our B-based PIMs can be further transformed into an actual platform specific solution utilizing industrial BPM solutions. We hope that our specifications involving data and operations making them semantically richer will map naturally onto the modular technologies employed in, for example, WWF.

References

1. van der Aalst, W., van Hee, K.: Workflow Management: Models, Methods, and Systems. The MIT Press (2002)
2. Hepp, M., Roman, D.: An Ontology Framework for Semantic Business Process Management. In: Proceedings of the 8th International Conference Wirtschaftsinformatik 2007, Universitaetsverlag Karlsruhe (2007)
3. Liu, K.: Semiotics in Information Systems Engineering. Cambridge University Press (2000)
4. Back, R.J.: Refinement Calculus, Part II: Parallel and Reactive Programs. In: de Bakker, J.W., de Roever, W.P., Rozenberg, G. (eds.) Stepwise Refinement of Distributed Systems. LNCS, vol. 430, pp. 67–93. Springer, Heidelberg (1990)
5. Ehrig, M., Koschmider, A., Oberweis, A.: Measuring Similarity Between Semantic Business Process Models. In: Roddick, J.F., Hinze, A. (eds.) Fourth Asia-Pacific Conference on Conceptual Modelling (APCCM2007), Conferences in Research and Practice in Information Technology, vol. 67, pp. 71–80. Australian Computer Society, Inc. (2007)
6. Halle, S., Villemaire, R., Cherkaoui, O., Ghandour, B.: Model Checking Data-aware Workflow Properties with CTL-FO$^+$. In: Proc. of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), pp. 267–278. IEEE Computer Society (2007)
7. Aslam, M.A., Auer, S., Böttcher, M.: From BPEL4WS Process Model to Full OWL-S Ontology. In: Proc. of the 3rd European Semantic Web Conference, Budva, Montenegro (2006)
8. d'Inverno, M., Luck, M.: Understanding Agent Systems. Springer Series on Agent Technology. Springer, Heidelberg (2004)
9. Castelfranchi, C., Conte, R., Paolucci: Normative Reputation and the Costs of Compliance. Journal of Artificial Societies and Social Simulation 1(3) (1998)
10. Rodríguez, A., Fernández-Medina, E., Piattini, M.: CIM to PIM transformation: A reality. In: Xu, L.D., Tjoa, M., Chaudhry, S. (eds.) International Conference on Research and Practical Issues of Enterprise Information Systems (2). International Federation For Information Processing, vol. 255, pp. 1239–1249. Springer, Heidelberg (2007)
11. Rodríguez, A., Fernández-Medina, E., Piattini, M.: Towards CIM to PIM Transformation: From Secure Business Processes Defined in BPMN to Use-cases. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) Business Process Management, LNCS, vol. 4714, pp. 408–415. Springer, Heidelberg (2007)
12. van der Aalst, W., Lassen, K.: Translating Workflow Nets to BPEL. In: BETA Working Paper Series. 145. Eindhoven University of Technology, Eindhoven (2005)
13. Rezazadeh, A., Evans, N., Butler, M.: Redevelopment of an Industrial Case Study Using Event-B and Rodin. In: The British Computer Society - Formal Aspects of Computing Science Christmas 2007 Meeting Formal Methods In Industry, pp. 1–8 (2007)