

Normative Ontologies for Data-Centric Business Process Management

Iman Poernomo

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

Timur Umarov

Department of Computer Science
King's College London
Strand, London, UK, WC2R2LS
timur.umarov@kcl.ac.uk

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 a Petri Net workflow model and, separately, an Event B specification. The former models can be submitted to further behavioural analysis to ensure, for instance, satisfaction of liveness and safety properties. The latter specifications are important as we can use theorem proving techniques to check and refine data representation with respect to process evolution. An important property of the transformation is semantic equivalence between the Petri net model and Event-B model.

Keywords—Petri nets, MEASUR, Business Process Management, Formal Specification, Semantic Augmentation

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 Framework, 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 [16]. 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 can assist in analysing and refining 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.

Petri nets are a formal method that has successfully assisted in workflow design and analysis. While Petri nets are good for expressing the dynamics of a workflow, the representation of data as tokens do not provide the full depth of specification necessarily to by developers. Petri nets model the *possible flow* of information in a business process, but do not specify the nature of the information nor how information is to be manipulated during the business process. Petri net lack modeling power and mechanisms for data abstraction and refinement [6].

In contrast, 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 [9]: 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 *semantically compatible* PN models and data models from an initial business process requirements specification. We employ a Model Driven Architecture approach.

The overall framework is depicted in Fig. 1. For our purposes, we consider transformations between models of three languages, a Computation-Independent Model (CIM) and two Platform Independent Models (PIMs). The CIM is an initial model of business requirements. It describes business functionality without treating any architectural or computational aspects of the system implementation. The two PIMs

describe complementary aspects of the overall structure of the system to be implemented: workflow descriptions from PN models and data and data exchange mechanisms from Event B specifications.

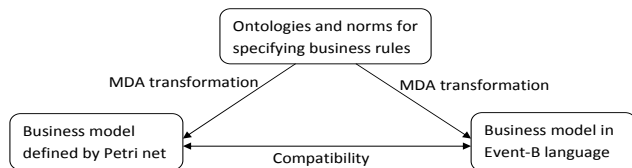


Figure 1. The data-centric workflow framework.

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 [12]. 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 Pierce, 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. We have also found 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 may be 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, optimised, executable system. In this process,

- the first abstract model in this refinement chain should be verified for consistency and
- each step in the refinement chain should be formally checked for semantic preservation.

Consistency will then be preserved throughout the chain. This means that the final executable refinement can be trusted to implement the initial abstract specification. This is the main reason why we are using Event-B method: once we have an initial abstract model of the system we can refine it to get more concrete and executable model. The efficiency of using formal methods in software development is proven by significantly low number of errors in a developed system and high degree of reliability. Development-by-proving ap-

proach allows develop more efficient and less error-prone software system.

Our approach defines a transformation MEASUR models to

- PN models, permitting the usual workflow analysis results
- Event B machines, permitting
 - a full B-based formal semantics for vocabularies and data manipulation that is carried out within the modelled workflow, which can be validated for consistency.
 - an initial, abstract B model that can be further refined using the B method to a final optimal executable system in an object-oriented workflow middleware, such as Windows Workflow Foundation.

A notion of semantic compatibility holds over the transformed models, so that any property derived over the PN 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 and the notion of semantic consistency between PNs and Event B specifications.
- Section 4 then outlines the transformation approach to generating B specification and Petri nets from our ontologies. 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 analyse and specify an organization’s business processes via three stages [12]:

1. Articulation of the problem, where a business requirements problem statement is developed in partnership with the client.
2. Semantic Analysis, where the requirements problem statement is encoded as an ontology, identifying the main roles, relationships and actions.

3. 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. We hereafter refer to the combination of an 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 Pierce, 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* possess the structure prescribed by the ontology and obey any further constraints imposed an associated set of norms.

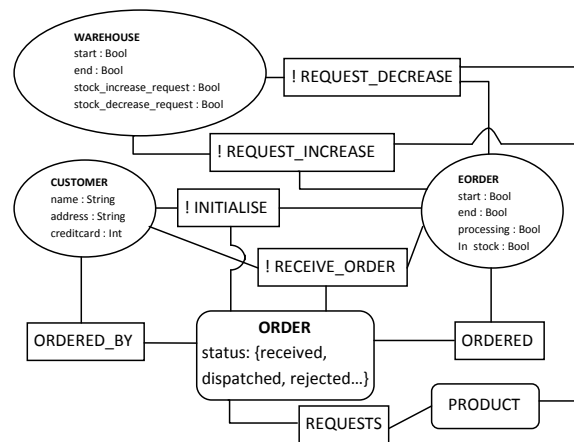


Figure 2. Example normative ontology

Example 2.1 The ontology for a purchasing system is given in Fig. 2. Agents are represented as ovals and business entities as rectangles with curved edges. Communication acts and relations 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 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 ProductWarehouse, 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). A customer can initialise the order processing system with a given order, denoted by a communication act !immortalize between the corresponding customer and eOrder agents. 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.

Another communication act, !receive_order, corresponds to the initial reception of data.

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 and normative logic to express logical constraints over business processes, using ontologies as atomic classes, relations, objects and actions for the logic. Our constraints take the form

$$A, B := R(\bar{a}) \mid \neg A \mid A \vee B \mid A \wedge B \mid A \rightarrow B \mid \forall x : C.A(x) \mid \exists x : C.B(x) \mid ObA \mid PA \mid NPA \mid E_xA \quad (1)$$

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 PA is that A is permitted to happen; the meaning of NPA is that A is prohibited to happen; the meaning of E_xA 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 behavioural norm is the general form for a constraint over our ontologies, and has the following form (Liu, 2000):

$$\text{Trigger} \rightarrow \text{pre-condition} \rightarrow E_{\text{agent}} Ob/P/NP \text{post-condition} \quad (2)$$

The informal meaning of the norm might be written:

if *Trigger* occurs and
the *pre-condition* is satisfied,
then *agent* performs an action so that
the *post-condition* is
Obliged/Permitted/Prohibited from resulting

The idea of a behavioural 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 behavioural norms to expected dynamic interaction protocols between subsystems.

Example 2.2 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 following behavioural norm:

$$\forall oo : Order. \forall e : eOrder. \neg \text{ordered}(oo, e) \rightarrow E_e Ob \text{receive_order}(oo, e) \quad (3)$$

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. We define them as further definitions of norms which contain expressions semantically equivalent to the effects of communication acts. In this case, the reception of an order entails a change of state in the order (its status becomes set to "received") and order processing system (its processing attribute is set to true). This norm definition is depicted below:

$$\forall oo : Order. \forall e : eOrder. \text{receive_order}(oo, e) \rightarrow \text{ordered}(oo) \wedge oo.\text{status} = \text{received} \wedge e.\text{processing} = \text{true} \quad (4)$$

When the eOrder system is processing an order, it will request an increase in the stock from the warehouse. This is prescribed by the following norm:

$$\forall oo : Order. \forall e : eOrder. \forall p : ProductWarehouse. \neg e.\text{processing} = \text{true} \rightarrow E_e Ob \text{request_increase}(oo, p)$$

where the communication act entails a change of state in the warehouse subsystem:

$$\begin{aligned} \forall oo : Order. \forall e : eOrder. \forall p : ProductWarehouse. \\ request_increase(oo, e) \rightarrow \\ p.stock_increase_request = true \end{aligned}$$

A number of other norms are required to define the entire business process. For example, after processing this information the system sends response to the customer depending on data provided. If the data are valid, then system sends an appropriate availability request to the warehouse. If the product is available, it is sent to the customer. If not validated, the order is rejected. The structure of the norms

There have been a number of attempts to use semantic analysis normative ontologies as the language for a business process management engine. The most widely used is Liu's NORMBASE system [12]. 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 intend to implement these requirements using a standard business process management infrastructure. We believe that further refinement and analysis a necessary step to this goal. In particular, it is important to ensure that

- the possible communication act traces permitted by a set of norms do not deadlock unexpectedly (in our example, this might happen if the order processing system waits indefinitely for a response from the warehouse that stock is available);
- the ontology and its associated norms do not allow for an inconsistent state of the system (in our example, this happens if an action entails that an order is processed and rejected at the same time).

The first kind of error can be removed by providing the normative ontology with a Petri Net representation and applying standard behavioural analysis techniques. The second kind of error can be eliminated by checking and ensuring that our generated invariants and guard conditions of the machines are not overlapping.

3 Event B and Petri nets

This section provides an overview of the Event B notation. We define the notion of semantic consistency between Petri Nets and Event B specifications.

3.1. Event B

Event B 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 non-deterministic: 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.

Fig. 3 demonstrates the structure of an Event-B model. 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 of the names of each enumerated set to its values and declaration of rules according to which constants are defined as not being equal to each other. There can also be one or several theorems defined in the Context of a model definition.

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 initialisation construct and all events of the machine. Each event contains one or several guards and one or several actions.

Definition 3.1 (Consistent B Machine) *An event B ma-*

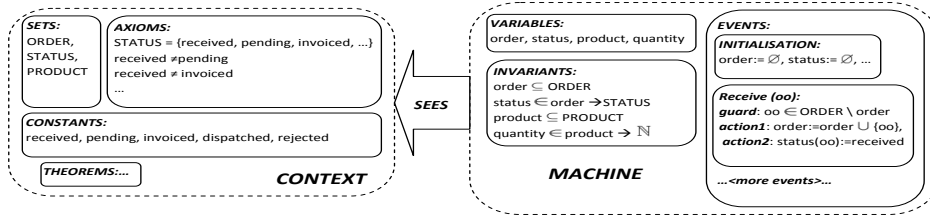


Figure 3. The structure of an abstract Event-B model.

chine 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 modelled system can be understood. Essentially, this is done by assuming 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, which space does not permit us to discuss here).

B machines do not lend themselves easily to simulation. This is because guards and actions are first order logical formulae and, consequently, the selection of a guard and the determination of how an action affects state is not decidable and requires human proof. Simulation of the possible ordering of tasks in a workflow are better handled via a Petri Net specification.

3.2. Petri nets and semantic compatibility

Space does not permit a full overview of the Petri Net notation. The reader is referred to, for example, [16] for an detailed introduction. A Petri net specifies a business process workflow in terms of *places*, representing a main business activity, tokens, representing some data or document that can be passed between activities, *transitions*, representing permitted flow of data between activities. Directed arcs are used to related transitions to places. An example Petri Net is given in Fig. 4: places are denoted by circles and transitions as rectangles. Petri nets permit the usual business process workflow notions of joins and forks from transitions to

places, allowing us to represent parallel and synchronizing processes and nondeterministic choice.

An Event-B machine is *semantically compatible* with a PN if there is a bijection from transitions of the PN to events of the Event-B machine such that all possible traces of the machine's events correspond to possible traces of transitions in the PN, and vice versa. That is, a machine is compatible with a PN if the PN simulates every possible sequence of events of the machine, and vice versa.

4 Semantic Embedding of Normative Ontologies in Event-B

This section describes in detail the actual mapping approach that was used to implement the transformation.

4.1 General Mapping Strategy

We now sketch our transformations from normative ontologies to PN and Event-B machines. The purpose of the transformations is threefold: 1) it provides a formal semantics for MEASUR's normative ontologies that can be analysed for consistency and correctness using B-based tools, 2) it serves to produce the first B model in a chain of refinements that leads to a final implementation and 3) Petri net model obtained from normative ontologies can be further formally analysed for liveness errors.

We have implemented our transformations using Kerma meta-modelling language.

The transformation from normative ontologies to Petri nets focuses on identifying the traces of communication acts that are possible from a collection of norms. The transformation involves *discarding* information about changes to states of entities and agents, and mapping communication acts to places. The transformation is *interactive*, not automatic, requiring a domain expert to identify the type of ordering expected to hold between communication acts. This is then used to define the possible transitions between places.

The domain expert need not be an expert in formal methods or workflow implementation – our transformation might

be incorporated at the requirements analysis stage immediately after the normative ontology has been developed. The resulting Petri Nets can be seen as a further refinement of the normative ontology requirements, further constraining workflow enabled by the norms.

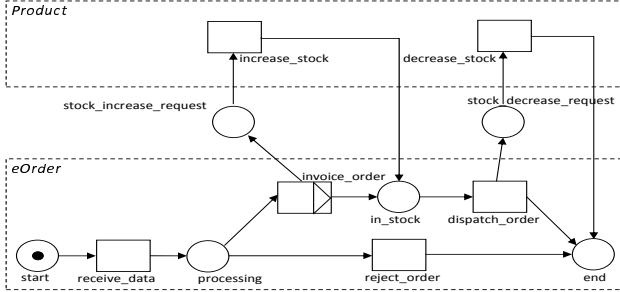


Figure 4. Petri net model for eOrder

Example 4.1 A Petri net model obtained from our example via the interactive transformation depicted in Fig. 4. The process *eOrder* is comprised of four main transitions, *receive_data*, *invoice_order*, *dispatch_order*, and *reject_order*, and four places, *start*, *processing*, *in_stock*, and *end*. The process begins with the place “*start*” by containing a token in it. This fires the transition “*receive_data*” after which the token transfers to the next place “*processing*”. This place contains an implicit condition which selects which transition to fire next. If the total price of the purchase is lower than allowed credit limit then “*invoice_order*” transition fires, otherwise the transition “*reject_order*” fires. If the ordered products are in stock the token moves to the place “*in_stock*”. The presence of the token in the place “*in_stock*” causes the transition “*dispatch_order*” to fire. Otherwise, if the product is currently not available in the stock then the token moves to the place “*stock_increase_request*” after which the transition “*increase_stock*” fires which starts an appropriate subprocess for increasing the stock and returns the token to the place “*in_stock*”. Successful execution of the transition “*dispatch_order*” decreases the stock and ends the process.

Normative ontologies are mapped to a B machine in the following way.

The mapping of affordances is straightforward. Actors are mapped to machines. Business entities and relations are mapped to Event-B sets and relations.

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

- A *trigger* and *pre-condition* correspond to a *guard*. The former define 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 $Ob/Ppost - condition$ identifies whether the action corresponding to *post - condition* should be necessarily performed, or whether execution of another (skip action) is possible instead. The *NP* 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.

Fig. 5 shows an excerpt of the metamodel for normative ontology written in Kermeta (the source model). The

```

package OBRMM;

class OBRModel
{
    attribute name : kermeta::standard::String
    attribute isDefinedBy : Affordance[1..*]
    attribute cntsV : Variable[0..*]
    attribute cntsN : Norm[0..*]
    attribute containsAttributes : DataType[0..*]
}

class Affordance
{
    attribute name : kermeta::standard::String
    reference defines : OBRModel[1..1]#isDefinedBy
    reference connBy : Association[0..*]#connects
    reference has : Property[0..*]#isFor
    reference contains : Type[0..*]#isOfType1
    reference rIn : Norm[1..1]#cntsA
    attribute rUses : Predicate[1..1]
    reference definesVar : Term[1..*]#isOfType
    reference domain : Affordance[1..1]
    reference range : Affordance[1..1]
}

class Agent inherits Affordance
{
    attribute cntsE : Entity[0..*]
    attribute cntsR : Relationship[0..*]
    reference pIm : Effect[0..*]#by
}

class Relationship inherits Affordance
{
    reference rIn : Agent[1..1]#cntsR
}

```

Figure 5. Metamodel for a normative ontology

main containing metaclass is *OBRModel* that is mainly defined by affordances, which can effectively be an agent, a relationship, an entity, a communication act, etc. Fig. 6 depicts an excerpt of the metamodel for an Event-B machine written in Kermeta (the target model). The main containing metaclass in this metamodel is *EBModel* which is defined

```

package EBMM;

class EBModel
{
    attribute name : kermeta::standard::String
    attribute isDefinedBy : Machine[1..*]
}

class Machine
{
    attribute name : kermeta::standard::String
    attribute evnts : Event[0..*]
    reference defines : EBModel[1..1]#isDefinedBy
    attribute sees : Context[1..1]
    attribute variables : EBVariable[1..*]
}

class Event
{
    attribute name : kermeta::standard::String
    attribute blks : Block[0..*]
    attribute grds : EBUFF[0..*]
    attribute invs : EBUFF[0..*]
    attribute prmt : EBVariable[1..*]
    reference eIn : Machine[1..1]#evnts
}

class Block
{
    attribute insts : Instruction[0..*]
    reference bIn : Event[1..1]#blks
}

```

Figure 6. Metamodel for an Event-B machine

by machine, set of events and variables. Events are defined by well-formed formulas for preconditions and blocks of instructions for actions.

4.2 Mapping Rules

Mapping from normative ontologies to Event-B machines consists of several rules to be implemented in the transformation φ . The following rules are used:

$$R : Entity \times Agent \mapsto_{\varphi} R'_{A'}, \quad (5)$$

where R is from $Entity \times Agent$ superset and represents a relationship type between $Entity$ and $Agent$, R maps to R' , which represents a set variable inside the machine A' ;

For any variable a from the entity E

$$a : E \mapsto_{\varphi} a' \in E', \quad (6)$$

a maps to a' which is a variable from the generated set E' ;

Predicate P of the form $P(a, b)$ maps to a local variable a from the set variable P :

$$P(a, b) \mapsto_{\varphi} a \in P, \quad (7)$$

where b is used for identifying the machine, which contains the set variable;

$$\neg F \mapsto_{\varphi} \neg \varphi(F) \quad (8)$$

According to the rule (5) any given relationship r holding between a given entity e and a certain agent a maps to a set

variable r' in the set e' of the machine a' . For example, let us consider the following expression of a norm:

$$\neg \text{ordered}(oo : ORDER, e : EORDER)$$

This statement declares that a particular order oo has not been ordered, or is not yet in the system (agent) $EORDER$. We can transform this expression to several constructs and expressions in Event-B. By applying rule (5) we first generate a set (state) variable $ordered$ and machine $EORDER$ (if it is not already created).

The following Fig. 7 – 12 show the excerpts of algorithms of transformation implementation. Fig. 7 depicts the transformation algorithm for this mapping.

```

var ebvar1 : EBVariable init EBVariable.new
var check : Boolean
var guard : EBUFF init EBUFF.new
ebvar1.name := pred.name
//checking for possible duplicates of variables
check:=ebvar.cntsT.exists(e | e.name==ebvar1.name)
if (check==false) then
    ebvar.cntsT.add(ebvar1)
else
    stdio.writeln("Variable already exists")
end
machine.variables.add(ebvar)

```

Figure 7. Algorithm for set variables

By applying rule (6) we generate a "general" guard $oo \in ORDER$ which declares local variable oo of type $ORDER$. Applying rule (7) will associate local variable oo with the set (state) variable $ordered$: $oo \in ordered$. Fig. 8 represents

```

if (pred.notApplies.getMetaClass==Not) then
    guard.expression:=
        localvar.name + " /: " + ebvar1.name
else
    guard.expression:=
        pred.left.name + " : " + ebvar1.name
end
event.grds.add(guard)

```

Figure 8. Algorithm for set variables

the algorithm for this mapping rule. It considers both cases: with and without negation. Since our example expression contains “ \neg ” sign, which means that the order oo has not been ordered, then (applying rule (8)) the resultant expression will effectively take the form of $oo \notin ordered$. This *guard* will reside in the event which is generated from the behavioural norm (3) following the responsibility modality. The content of the event is generated from the behavioural norm definition (4), which is semantically equivalent to the meaning of the effect of the norm. The norm’s expression $e : EORDER$ is used only to identify the responsible agent that maps to a corresponding machine where the event resides. Hence, we are not applying the rules to this statement. Fig. 9 shows how we assign names to the newly created events from predicates’ type information. For example, a predicate *receive_order* is of type COMMUNICATION

```

if(pp.getMetaClass==Implies) then
  if(pp.E.action.getMetaClass==DeonticWFF) then
    var deontic : DeonticWFF
    deontic:=pp.E.action
    if (deontic.modality=="Ob") then
      deontic.cntsWFF.cntsP.each(pp |
        //assigning event names
        event.name := pp.isOfType.isOfType1.name
      )
    end
  end
else if(pp.getMetaClass==ForAll) then
  ...

```

Figure 9. Algorithm for norms' effects

ACT RECEIVE_ORDER, taken from normative ontology (see Fig. 2), generates a new event with the same name, given that the deontic modality is "Ob". These newly created events from normative ontologies are added to the ma-

```

else if(pp.getMetaClass==ForAll) then
  var check : Boolean
  pp.cntsT.each(tt |
    if(tt.isOfType.getMetaClass==Entity) then
      localvar.name := tt.name
    else if(tt.isOfType.getMetaClass==Agent) then
      if(nn.cntsA!=CommunicationAct) then
        var checkMachine : Machine init Machine.new
        checkMachine.name := tt.isOfType.name + "_M"
        check:=result.isDefinedBy.exists(e |
          //checking for possible duplicates of machines
          e.name==checkMachine.name
        )
        if(check==false) then
          checkMachine.name := tt.isOfType.name + "_M"
          context.name := tt.isOfType.name + "_C"
          checkMachine.sees := context
          checkMachine.evnts.add(event) //adding event
          result.isDefinedBy.add(checkMachine) //adding machine
          machine:=checkMachine
        else
          if(machine.name!=checkMachine.name) then
            machine:=checkMachine
            machine.evnts.add(event) //adding event
          else
            machine.evnts.add(event) //adding event
          end end
        end end end
      )
    )
  )

```

Figure 10. Adding machines and events

chines as shown in Fig. 10. This part of the transformation is responsible for creating new machines, adding new events to these machines and assigning new contexts to these machines. The events are further elaborated by adding new

```

nd.meaning.cntsT.each(aa |
  var prmts : EBVariable init EBVariable.new
  if(aa.isOfType.getMetaClass==Entity) then
    prmts.name:=aa.name
    prmt.cntsT.add(prmts)
  end
)

```

Figure 11. Adding parameters for events

parameters to them as shown depicted in Fig. 11. These parameters are taken from norm definitions, which contain necessary variables as input arguments.

One norm definition can contain several instructions to execute. Fig. 12 demonstrates how parallel executions are generated from such norms.

```

result.isDefinedBy.each(mm |
  mm.evnts.each(ee |
    nd.meaning.compound.each(cc |
      if(cc.getMetaClass==And) then
        var andOperator : And
        andOperator:=cc
        var parallel : ParallelExecution
        init ParallelExecution.new
        var ebterm : EBTerm init EBTerm.new
        if(ee.name==andOperator.cntsRPredicate.
          isOfType.isOfType1.name
          and ee.name==andOperator.cntsLPredicate.
          isOfType.isOfType1.name) then
          ee.prmt.add(prmt) //adding parameter to event
          parallel.left:=PredicatesToInstructions
            (andOperator.cntsLPredicate)
          parallel.right:=PredicatesToInstructions
            (andOperator.cntsRPredicate)
          ebterm.containsInstruction:=parallel
          instruction.isBuiltOf:=ebterm
          block.insts.add(instruction)
          ee.biks.add(block)
        end
      else if(cc.getMetaClass==Predicate) then

```

Figure 12. Handling parallel executions

Given a PN associated with a normative ontology, it is possible to further extend our transformation so that the PN and the Event B machine are semantically consistent.

It is possible for norms to specify invariant properties over a system: these take the form of norms whose trigger is always true: the invariant is then specified as a post-condition in first-order logic whose Deontic modality is *Ob*. These invariants are then mapped to invariants of the B machine associated with the responsible agent. Initial conditions are handled in a similar fashion.

Example 4.2 *The example order processing normative ontology can be transformed to an Event-B model, part of which is shown in Fig. 13. Each normative act within the ontology is mapped to Event-B events.*

Fig. 14 shows the context for the model defined in Fig. 13.

Variable order (a subset of ORDER) is a set of order data currently in the system. Variable product (a subset of PRODUCT) is a set of products currently involved in the ordering process. Variable status is a total function over the variable order and represents the status of the order. Variable quantity describes the quantity of ordered products within a given order and is a function over the product variable.

One of the generated events which is shown in Fig. 13 is receive. This event creates a new instance of the order and assigns the status of that order to received. The receive event contains "any-where-then-end" substitution construct. See the whole definition of "eOrder" and "Product" and their operations in [15].

One of the requirements of technical specification development is to prove that this specification does not violate its initial requirements and is internally consistent, and that eventually the final software system which is generated in the course of successive refinement steps is indeed correct.

```

MACHINE EOS_0
SEES EOS_C0
VARIABLES order status product quantity
INVARIANT
  inv1: order  $\subseteq$  ORDER inv2: product  $\subseteq$  PRODUCT
  inv3: status  $\in$  order  $\rightarrow$  STATUS
  inv4: quantity  $\in$  product  $\rightarrow$   $\mathbb{N}_1$ 
EVENTS
Initialisation
  begin
    act1: order :=  $\emptyset$  act2: status :=  $\emptyset$ 
    act3: product :=  $\emptyset$  act4: quantity :=  $\emptyset$ 
  end
Event receive  $\hat{=}$ 
  any oo
  where
    grd1: oo  $\in$  ORDER  $\setminus$  order
  then
    act1: order := order  $\cup$  {oo}
    act2: status(oo) := received
  end
END

```

Figure 13. Machine for the model defined in Event-B

On the one hand, we are checking the correctness of the generated model. Since, we are performing transformation from normative ontologies to the Event-B model and the Petri net model, it is of high importance to check whether the both target models do not contain errors and are internally consistent. If this is the case, then we can state that the generated Event-B model is compatible with the generated Petri net model and vice versa. On the other hand by demonstrating the compatibility between two generated models we can also assert that our transformation is indeed correct. As [10] states, to be useful at all, an transformation must have specific characteristics. The most important characteristic is that a transformation should preserve meaning between the source and the target model. In our case, it is

```

CONTEXT EOS_C0
SETS
  ORDER STATUS PRODUCT
CONSTANTS
  received pending invoiced dispatched rejected
AXIOMS
  axm1: STATUS = {received, pending, ...}
  axm2: received  $\neq$  pending
  axm3: received  $\neq$  invoiced ...
  axm11: dispatched  $\neq$  rejected
END

```

Figure 14. Context for the model defined in Event-B

Table 1. Proof obligations

Machine	Proof Obligations	Automatic	Interactive
eOrder	22	22	0
Total	22	22	0

done by checking whether the target model functions in a way, as it was “prescribed” by the source model.

We are particularly interested in checking whether events and initialization preserve the guards and invariants of the generated Event-B model. The Event-B language (similarly to B-method) defines proof obligations for substitutions (events and initializations). Discharging these proof obligations presents a form of specification validation.

There are several theorem provers used for specification validation. We used the Rodin platform, an Eclipse-based IDE [1], to generate the proof obligations. As it was mentioned earlier, the proof obligations generated were only for initializations and events, since only these elements of the abstract machine modify state variables.

Example 4.3 *Continuing our example, we can validate the generated specification. Validation of the specification requires that initialization T is guaranteed to establish the invariant I . It is also necessary to prove that all events preserve the invariant. In other words, if invariant I and precondition P are both true when the event is executed, then the event should be guaranteed to re-establish I : $I \wedge P \Rightarrow [S]I$.*

Table 1 illustrates a summary of statistics for proof obligations. According to these numbers all 22 generated proof obligations for the Event-B model were proven automatically.

5 Conclusion and Related Work

In this paper, we have shown how normative ontologies can be used for generating data-aware and semantically-rich business process models in the form of B specifications. We have shown the results MDA transformation from normative ontologies to both Petri nets model and Event-B machines. Petri nets specify possible flow of the information but do not specify the nature of the information. In order to avoid this, we are generating additional Event-B machines from normative ontologies to provide missing semantics for the business processes.

There are a number of related approaches for enriching workflow models [3, 4]. 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 [2]. They are using an 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.

Another work is using the example of BPEL processes which should be converted to semantically enriched specifications. All data (stored in process models) must be augmented by references to ontologies [9]. They refer this augmentation to as *ontological lifting* because input business processes must be expressed using richer constructs provided by ontologies. However, from the perspective of web services, systems support only part of the process space representation which is reduced to the patterns of message exchange (choreography) and the control and data flow in the combination of multiple Web services (orchestration) [8].

[4] advocates the idea of ensuring the correctness of a workflow by making protocol specifications data-aware through expressing actual data content rather than message names. In other words, workflow validation cannot be complete unless this abstraction is eliminated. They present CTL-FO+ tool, an extension over Computation Tree Logic that includes first-order quantification on state variables in addition to temporal operators, and which is adequate for expressing data-aware constraints.

There is also a variety of research directed towards semantic enriching of Petri net business processes. The authors were proposing to enrich the semantics of Petri nets by combining it with OWL language. Representing Petri nets in combination with OWL is a way to make data computer-interpretable for flexibility, ease of integration and significant level of automation of loosely coupled business processes [11]. In this work, authors are trying to define Petri net models using OWL framework which entails horizontal way of integration. In other words, they are defining semantic metadata for business processes described by Petri nets. [3] was proposing an approach for (semi-)automatic detection of synonyms and homonyms of the process element names in order to support semantic process model interconnectivity and interoperability with use of OWL.

There were also several results on integrating Petri nets and Z [6, 7]. In [6] describes a thorough integration definition of Petri nets and Z which results in so called PZ nets for specifying concurrent and distributed systems. In this work, Petri nets are used to define the overall structure, control flow and dynamic properties and Z is applied for specifying tokens, labels and constraints of the system. This result is based on the previous more preliminary work on integrating Petri nets and Z outlined in [5]. Another work in [17] have used Z to specify certain aspects of restricted hierarchical coloured Petri nets. Namely, the authors have used Z schemas to define the metamodel of a hierarchical coloured

Petri net and operation for specifying the transitions in a specific coloured Petri net.

[13] describes the model-driven transformation between the Semantic Web Rule Language with Web Ontology Language (OWL/SWRL) and Object Constraint Language with UML (UML/OCL). The implementation of the transformation is performed by using ATLAS Transformation Language involving several MOF based metamodels, XML schemas, and EBNF grammars. Whereas, [19] presents a new interchange format for rules for integrating the Rule Markup Language, the Semantic Web Rule Language and the Object Constraint Language. Since these languages are capable of providing a rich syntax for expressing rules, it is possible to make conceptual distinctions of different types of terms and different types of atoms. They also adopt the Model-Driven Approach, particularly specifying the computation-independent level as a domain containing set of rules and business policies.

The advantage of our work over the before-mentioned approaches is that we are incorporating norms into our source models and for this purpose using the MEASUR language to provide semantical information and additional constraints to the business processes of the source model (the CIM). Another advantage of our approach is that from the CIM model we are generating two PIM models defined in Petri nets and in Event-B. The Petri net model can be checked for liveness errors. With respect to the Event-B model we can use theorem proving techniques to check and refine data representation to obtain an executable system.

In multiagent systems, there are several quite successful works on developing and using norms in order to specify the expected behaviour of agents in a certain organization. For instance, one of the most interesting works in this area is [21, 20] which describes the agent architecture SMART which is based in an agent specification framework developed in the Z modeling language. Interesting aspect of this work is that it provides an analysis of different kinds of norms and agent societies based on these norms. Moreover, they are modelling norms as objects rather than as static constraints. As a result, these norms can have several states which in its turn completes the Norm lifecycle.

[18] makes norms operational rather than purely declarative by focusing on how norms should be operationally implemented in MAS from an institutional perspective. [14] view an electronic institution based on agents as dialogical system where all the necessary interactions between agents are made through dialogic activities (message exchanges). These interactions, also called illocutions, follow a certain well-defined protocol and are structured through agent group meetings, scenes. Such a division of all possible interactions among agents in scenes is in line with modular approach of systems design with the classical modular design principles and methodologies (e.g. Modular Pro-

gramming and Object-Oriented Programming) taken as a foundation.

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 rich specifications involving data and operations will map naturally onto the modular technologies employed in, for example, Windows Workflow Foundation.

References

- [1] J.-R. Abrial, M. Butler, S. Hallerstede, and L. Voisin. An open extensible tool environment for event-b. *Proceedings of the Eighth International Conference on Formal Engineering Methods, ICFEM*, pages 588–605, 2006.
- [2] M. A. Aslam, S. Auer, and M. Böttcher. From bpm4ws process model to full owl-s ontology. *ESWC2006 Proceedings, Lecture Notes in Computer Science*, 2006.
- [3] M. Ehrig, A. Koschmider, and A. Oberweis. Measuring similarity between semantic business process models. *Proceedings of the fourth Asia-Pacific Conference on Conceptual Modelling, Ballarat, Australia*, 67:71–80, 2007.
- [4] S. Hallé, R. Villermaire, O. Cherkaoui, and B. Ghandour. Model-checking data-aware temporal workflow properties with ctl-fo+. *forthcoming*, 2007.
- [5] X. He. Pz nets - a formal method integrating petri nets with z. *Proceedings of the 7th International Conference of Software Engineering and Knowledge Engineering SEKE'95*, pages 73–180, 1995.
- [6] X. He. Pz nets - a formal method integrating petri nets with z. *Information and Software Technology*, 43(1):1–18, 2001.
- [7] X. He and C. Yang. Structured analysis using hierarchical predicate transition nets. *Proceedings of the 16th International Computer Software and Applications Conference, Chicago*, pages 212–217, 1992.
- [8] M. Hepp, F. Leymann, J. Domingue, A. Wahler, and D. Fensel. Semantic business process management: A vision towards using semantic web services for business process management. *Proceedings of the IEEE ICEBE, Beijing, China*, pages 535–540, October 2005.
- [9] M. Hepp and D. Roman. An ontology framework for semantic business process management. *8th international conference Wirtschaftsinformatik, Karlsruhe*, 2007.
- [10] A. Kleppe, J. Warmer, and W. Bast. *MDA Explained. The Model Driven Architecture: Practice and Promise*. Pearson Education, Boston, USA, 2003.
- [11] A. Koschmider and A. Oberweis. Ontology based business process description. *Proceedings of the CAiSE-05 Workshops, Lecture Notes in Computer Science, Springer, Porto, Portugal*, (13):321–333, 2005.
- [12] K. Liu. *Semiotics in Information Systems Engineering*. Cambridge University Press, 2000.
- [13] M. Milanović and et al. On interchanging between owl/swrl and uml/ocl. *Proceedings of 6th Workshop on OCL for (Meta-)Models in Multiple Application Domains (OCLApps) at the 9th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, pages 81–95, 2006.
- [14] P. Noriega. Agent-mediated auctions: The fishmarket metaphor. *Number 8 in IIIA Monograph Series, Institut d'Investigació en Intelligència Artificial (IIIA), PhD Thesis*, 1997.
- [15] PALab. The predictable assemble laboratory. <http://palab.dcs.kcl.ac.uk/>, November 2007.
- [16] W. van der Aalst and K. van Hee. *Workflow Management: Models, Methods, and Systems*. The MIT Press, Cambridge (USA), London (England), 2002.
- [17] K. van Hee, L. Somers, and M. Voorhoeve. Z and high-level petri nets. *Lecture Notes in Computer Science*, 551:204–219, 1991.
- [18] J. Vázquez-Salceda, H. Aldewereld, and F. Dignum. Implementing norms in multiagent systems. *Multiagent System Technologies: Second German Conference, MATES 2004, Erfurt, Germany*, pages 313–327, September 2004.
- [19] G. Wagner, A. Giurca, and S. Lukichev. A usable interchange format for rich syntax rules integrating ocl, ruleml and swrl. *Proceedings of Reasoning on the Web*, 2006.
- [20] L. y López and M. Luck. Towards a model of the dynamics of normative multiagent systems. *Proceedings of the International Workshop on Regulated Agent-based Social Systems: Theories and Applications (RASTA '02)*, pages 175–194, July 2002.
- [21] L. y López, M. Luck, and d'Inverno. A framework for norm-based interagent dependence. *Proceedings of the Third Mexican International Conference on Computer Science*, pages 31–40, 2001.