Business Requirements
Normative Approach to Behavior Modeling

Askhat Omarov, Rustem Kamun and Timur Umarov
Department of Computer Engineering, Kazakh-British Technical University, 59 Tole bi str., Almaty, Kazakhstan
{*askhat.omarov91, r.kamun*}*@gmail.com, t.umarov@kbtu.kz*

Keywords: Business Modeling, Organizational Semiotics, Deontic Logic, Responsibility, Requirements Formalization, Normative Positions, MEASUR.

Abstract: This paper attempts to formalize abstract requirements by amalgamating the MEASUR methodology with deontic logic and the limited version of the theory of normative positions. This amalgamation is performed in a manner that would potentially decrease the complexity inevitably embedded in the formalized abstract requirements so as to be able to target a wide range of non-technical users involved in the initial stages of software development lifecycle. This amalgamation is embraced by an artifact which represents a business model that avoids specifying software systems as a highly technical document and is rather referred to as a computational independent model of the MDA framework of software development.

1 INTRODUCTION

Business process engineering represents a critical area of concern in enterprise. It is becoming increasingly important to specify structured initial requirements for the business model which will (i) best fit the environment and the market demands and (ii) adapt to different changes that may happen. Since organizations are modeled using different theories, approaches, and solutions, it is our belief that, whichever approach is taken, the questions of complexity of modeling tools and insufficient depth of information description still remain.

Different new trends in requirements elicitation and gathering for the overall process of information systems' engineering were revealed. Among these approaches we can underline the one, which relates organizations to a set of interacting agents and entities while recognizing these objects via signs. This paradigm of modeling is regarded as organizational semiotics and represents an emergent discipline that studies the nature, characteristics, and features of information and how this information can be interacted between agents in the most optimal way within the contexts of organizations and business domains. OS finds its roots in semiotics and treats organizations as information systems, within which information is created, processed, distributed, stored and used.

The concept of organizational business modeling of the enterprise is driven by principles of adapting business processes to ever-changing market conditions in order to provide better services and products to clients. This is practically achieved by modeling individual business tasks and connecting them into logically organized activities to form workflows that would rapidly and effectively respond to the environment changes. Participants (humans or machines) are intimately related to these workflows by controlling and executing them in one way or another, subject to the knowledge possessed and the roles that these participants play. Equally important is the concept of contractual relationships formed among participants over the course of their interaction with an emphasis on social aspects of an organizational setting. However, most of the business processes with different functional ontological descriptions and annotations still lack complete operational semantics being defined within the requirements specifications.

In our attempt to address this problem, we relate to the following theories. One school of thought, led by Stamper and Liu, espouses a natural framework for the notions of behavioral patterns, roles, and relationships, conceptualized via the philosophy of language and communication, the theory of speech acts, and computational semiotics. Another school of thought, led by Kanger, Pörn, and Lindahl, supports so called the theory of normative positions which is a combination of deontic logic and logic of action and agency to provide a formal account of complex social interactions within organizations. As such concepts are

Omarov A., Kamun R. and Umarov T.
Business RequirementsNormative Approach to Behavior Modeling.
DOI: 10.5220/0005425901860195
In *Proceedings of the Fourth International Symposium on Business Modeling and Software Design (BMSD 2014)*, pages 186-195
ISBN: 978-989-758-032-1

new and not usually discussed in the wider fields of enterprise software engineering. Similarly, use of formal methods such as B and Event-B is a rare practice in the software engineering community for their inherent complexity in the software specification process. On the contrary, there also exist informal business process specification languages such as BPMN, BPEL, and WWF which are easier and handy in use.

There are several attempts to formalize requirements and provide formal semantics for unstructured or semi-formal requirements specifications. The authors in (Ponsard and Dieul, 2008) address this by bridging the semantic gap between the semi-formal requirements and formal models using a goal-oriented approach and regulation modeling and further refine them into formal specifications. Besides formalization an important role in developing, for example, critical systems is played by validation. This is thoroughly addressed by (Cimatti et al., 2010), in which the authors describe their methodology and a series of techniques for formalization and validation of high-level abstract requirements. The language used is a combination of first-order, temporal, and hybrid logic. Reliability as one of the most important properties of any systems, including critical ones, is raised by (van Lamsweerde, 2001) while discussing requirements engineering in terms of developing high-quality requirements. The arguments is built in the light of the related techniques for goal-oriented description of requirements, multiparadigm specification and so on.

This paper describes our approach and explains bearing of the modeling methodologies on information systems. We try to describe how the relatively informal methodology can incorporate normative techniques to make this language structured and computer interpretable for future machine use to implement model transformations. Because these concepts form a common root for modern logic and therefore formal methods, we find that we can potentially exploit our framework via model transformation ideas to derive a method for developing better business processes models from business requirements models. Our approach amalgamates the MEASUR methodology with the limited version of the theory of normative positions. This provides an additional level of semantics by "attaching" the notion of responsibility to the agents. The paper proceeds as follows:

- Section 2 provides an overview of the methodology MEASUR and gives a flavor of the normative approach in business modeling;
- Section 3 describes the proposed approach of formalizing MEASUR and defines the so-called normative tableaux;
- Section 4 provides an example of applying the approach in the example of making an electronic purchase and gives normative tableaux of several communication acts; and
- Section 5 briefly elaborates on future work of the research group.

2 BACKGROUND

In his definition of speech acts (Searle, 1969; Searle, 1971), Searle emphasizes the importance of so called proper circumstances in which speech acts have to be used. Uttering speech acts in proper circumstances causes construction or alteration of the world, e.g. creating an obligation. Searle also calls these proper circumstances rules of use. In our following discussion, term norms will be used to designate rules of using speech acts effectively.

2.1 Methodology MEASUR

Method for Eliciting, Analyzing and Specifying User Requirements (MEASUR) represents a radically new set of norm-oriented methods for business systems modeling and requirements specification for software development (Liu, 2000). MEASUR is a result of more than 30 years of research. The idea of MEASUR is that it sees the organizations as information systems with the set of agents, their corresponding set of potential actions (affordances) and set of norms which govern agents behavior. MEASUR offers five phases for business modeling and software development of which three the most important phases are problem articulation methods, semantic analysis method and norm analysis method.

The general ontological theory is concerned with fundamental questions of classifying everything that exists in the world into different categories, describing that everything while it exists, and seeking to elicit possible hierarchies and dependencies among the things that make up that everything. Liu (Liu, 2000) defines the term everything as constituent notions such as thing, entity, individual, universal, particular, substance, event, process and state. These notions are embraced with the general ontological study.

Although there are different types of ontologies related to knowledge and its representation, the only type of ontology that semantic analysis is relevant to is that which recognizes only our own behavior in accordance with our own ambiance. This type of ontology can be defined as a collection of representational data that models a certain domain of knowledge or discourse. Everything that exists in the world is dependent on the agents behavior. In other words, the

meaning of a thing is related to the ability of the agent to recognize that thing as a particular entity.

Knowledge representation is an image of the world in our minds and is created by our perceptions of reality. The representation that we perceive, a surrogate, represents a substitute for the real entity itself. The correspondence between the entities and their surrogates forms the semantics for the representation. What we learn about the surrounding world forms our understanding about it. This understanding allows us to recognize the things that we know. Other things that we do not possess any knowledge about and hence cannot recognize them, do not exist in the sense of such entities in the world. By recognition we mean assigning meaning to things and knowing their properties. As we learn more about the world, the ontology changes accordingly. In other words, the possessed knowledge shapes a repertoire of our behavior.

With relation to the semantics, Liu (Liu, 2000) introduces the term affordance and defines it as a pattern of behavior of a human (or agent) which he learns through perceiving an ambient world. What an agent does is directed by his knowledge. The process of learning, or gaining knowledge about a certain set of facts, is in intimate relationship with the behavior and this relationship is bipartite. On the one hand, an agent learns the environment through his actions. On the other hand, the agents set of actions constrained by his behavior manifests knowledge that he possesses.

In any particular environment, whether an organization or a community, actions of its participants 1 must conform to the internal rules of behavior. Here the term behavior ensues from a social or a business context. Every agent has a set of certain actions to perform in accordance with the external conditions. The rules that members must adhere to is also referred to as norms. In the technical context, we are modeling the interaction between machines and humans, which can universally be referred to as agents. In terms of (Liu, 2000), the process of acting within the required norms represents invariant patterns of behavior or mechanisms of agents. Agents interact with each other which inevitably creates different relation patterns. The questions of modeling and formalizing these relations between agents is studied by the theory of normative positions.

The MEASUR methodology is a set of methods for business modeling and requirements specification for information systems. MEASUR can be used to analyze and specify an organization's business processes. As a traditional software system development lifecycle, MEASUR methodology of information systems modelling and requirements specification for software system development is divided into the following three stages (Liu, 2000):

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.

The processes of the first stage are comparable to other well known approaches to requirements specification. The last two stages require some elaboration and have their own unique associated notations.

The ultimate deliverable of these three stages is what we will refer to as a *normative ontology*. A normative ontology consists of what might be called a semi-formal requirements document, in the sense that it can be understood readily by clients, business analysts and developers. This model breaks down an information system into a set of business data, communicating agents (stakeholders, departments, computer programs) and business processes that agents can invoke in order to manipulate data between each other. The model consists of a role-and-relationships ontology together with a set of norms that formally define the structure and expected and permitted interactions within an organization and its processes.

2.2 The Normative Perspective

The theory of normative positions is based on the fundamental principles of deontic logic. Deontic logic is a formal system and represents a field of symbolic logic which is concerned with normative concepts such as obligation, permission, and prohibition used in contractual relationships for classifying actions and states of affairs. The word deontic comes from Greek and means "of that which is binding". Standard Deontic Logic (SDL, KD, or simply D) is the most studied and axiomatically defined system of deontic logic. SDL 1) represents a monadic deontic logic because its deontic operator is a one-place operator, 2) builds on propositional logic, and 3) is formally specified by a Kripke-style semantics. SDL uses the following notation to denote its main concepts: Ob*A* stands for it is obligatory that *A* and Pe*A* – it is permissible that *A*.

Deontic logic have not yet found wide application in technical areas such as Computer Science. However, deontic modalities could be efficiently used in specifying technical requirements for different business domains. For instance, modality obligatory may

well model the requirement that a system must check whether a buyer have enough money in his back account before making a purchase in an on-line or physical store; and modality permissible can be applied to specify a rule that a bank may check (it is permissible that bank checks) credit history of a client before issuing a loan. More on the definition and a Kripke-style possible world semantics for SDL can be found in (McNamara, 2006).

The theory of normative positions was originally inspired by analytical study of law and originated in the works of Stig Kanger (Kanger, 1972; Kanger, 1985), Ingmar Pörn (Pörn, 1970; Pörn, 1977), and Lars Lindahl (Lindahl, 1977; Lindahl, 1994). The Kanger- Lindahl theory is characterized by attempts to apply modal logic – mainly standard deontic logic, the field of logic concerned with obligations and permissions – and the logic of action and agency to the concepts of legal and normative relations which Wesley Newcomb Hohfeld (1879–1918), an American jurist, had regarded to as the fundamental legal conceptions of jurisprudence (Hohfeld, 1964). These normative relations are alternatively referred to as normative positions that take the forms of obligations, permissions, duties, and rights of agents of a community, society or some other form of organization. By agents here we mean humans, machines, or both. The Kanger-Lindahl theory also embraces the formal representation of more complex normative positions such as entitlement, authorization and responsibility.

Besides the areas of legal knowledge representation (e.g. representation of laws, regulations, legal contracts, etc.), where the theory of normative positions found its initial application, there are also other areas such as Computer Science, where the theory contributed much for formal representation of relations between agents. For example, Jones and Sergot (Jones and Sergot, 1992; Jones and Sergot, 1993) describe a modified version of the Kanger-Lindahl theory and attempt to apply it to the problem of access control and security policies specifications and analysis for databases. In (Jones and Sergot, 1993), authors illustrate by an example of library regulations for governing the procedures of loaning books that the use of formal methods in developing system specifications have to be taken seriously whenever it is necessary to analyze an ideal case and an actual one and see how the actual behavior deviates from that of ideal. Use of such formal methods as deontic logic can help in revealing the possibility of violations, i.e. those deviations that had actually occurred. According to Jones and Sergot (Jones and Sergot, 1993), the use of deontic logic will in general allow (i) to reason with the specifications developed, (ii) to be able to test the internal consistency of the system specifications as a whole, and (iii) to use theorem provers to implement and test different components of the system.

In the works of Kanger, Pörn, and Lindahl (Kanger, 1972; Kanger, 1985; Pörn, 1970; Pörn, 1977; Lindahl, 1977; Lindahl, 1994), deontic logic was merged with logic of action and agency to provide a formal account of complex social interactions within organizations, which can be related to the technical context by applying it to the multi-agent environment. They introduce a so-called relativised modal operator which is designated as E_a, where a represents a responsible agent. The approach is partially similar to that of dynamic logic in the sense that it also assumes that an action, if performed, should bring about a certain state of affairs. For example, expression $[a]A$ would mean that after performing action a it is necessarily the case that A holds. In other words, a must bring about A. Analogously, $\langle a\rangle A$ means that after performing action a it is possibly the case that A holds, or a might bring about A.

However, in the theory of normative positions all actions are associated with their respective responsible agents which makes the semantics comparatively more expressive. When modeling business systems using this theory we now have to deal with the element of agent's responsibility embedded in the process of bringing about new states of affairs. The overall concept is formalized by

$$E_a A, \tag{1}$$

which is read as "agent a sees to it that A is the case" or similarly "agent a brings it about that A". It is important to note that actions in this case represent a relationship between agents and the state of affairs that these agents bring about.

The following expressions are properties for the action operator. The first axiom schema implies that the action operator is a *success* operator:

$$\vdash E_a A \rightarrow A. \tag{2}$$

It is read as: if agent a brings it about that A then A is indeed the case. The second property represents a rule of inference:

$$If \vdash A \leftrightarrow B \, then \vdash (E_a A \leftrightarrow E_a B). \tag{3}$$

Although the approach of combination of deontic logic with action logic is reminiscent of that of dynamic logic by its rules and operators, the theory of normative positions provides higher expressivity in a way that, unlike in dynamic logic: (i) by using operator $E_a A$ one can express different atomic positions agent a can be in with respect to a particular state of affairs A; and (ii) using E_a operator gives another important advantage, namely, one can also formalise a

normative interpersonal relationship by means of iterating the action operators:

$$E_a E_b A. \tag{4}$$

The examples of using this form of iteration can be demonstrated by the following:

$$E_a \neg E_a A \quad and \quad \neg Pe E_b \neg E_a A \tag{5}$$

where the former effectively implies that agent *a refrains* from seeing to it that *A* and the latter means that the agent *b* is forbidden to prevent agent *a* from seeing to it that *A*. The detailed definition is described by Jones and Sergot in (Jones and Sergot, 1993).

The main building block of our proposed methodology for formalizing requirements is a tableau. It effectively uses the MEASUR constructs and the normative positions for describing the roles of the agents. A deeper analysis of the approach is required for clearer understanding of how MEASUR and the normative perspective are combined. We describe it in the next section.

3 FORMALIZING MEASUR

MEASUR and, in particular, NORMA, is a curiously mongrel beast. Its originator, Roland Stamper, while having a firmly applied background in system development, was influenced by ideas from philosophy of language and the expressive possibilities of deontic logic. However, in spite of drawing upon these ideas to develop the language and approach, it was always, first and foremost, a semiformal approach to requirements analysis, lacking a full formal semantics for analysis and reasoning.

The way in which MEASUR currently treats norms is analogous to the use of OCL within UML, that is syntaxes of OCL and UML are based on formal languages, but within their respective methods they do not have a precise semantics, which we give for MEASUR. In this paper, we provide what might be called a faithful formalization of MEASUR by restricting its norms to precisely the kind of logical language that inspired its notion of behavioral norm: that is, a limited version of deontic logic, combined with notions of agency inspired by the theory of normative positions. This will be defined as a first-order logic together with a straightforward semantics.

3.1 Normative Tableaux

The language proposed in this paper is somewhat more complex, due to its relational nature and conformance to an ontology. However, we need not study the full set of formulae given: we *restrict* our attention to a subset of normative formulae that correspond to the informal schema for behavioural norms given in Definition 3.1.

A MEASUR normative definition is of the form:

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

Importantly, MEASUR norms never contain deontic quantifiers within the trigger, pre-condition or post-condition statements. Furthermore, actual communication acts are only mentioned in the post-condition: the trigger and pre-condition statement only refer to relations from the ontology. All three elements of the definition can refer to agents and entities, however. Finally, the prescription of an agent's responsibility and a given deontic obligation are only given in the implication of the constraint.

We can therefore restrict our attention to a subset of behavioral norms for a given ontology that naturally preserves these syntactic constraints within our formalization, now defined.

Definition 3.1 (Formal behavioral norms for an ontology)**.** *The set of* behavioral norms *is defined to be any formula of the form*

$$G \rightarrow E_a \mathbf{D} Post \tag{6}$$

where

- **D** *is a deontic operator Ob (obligatorily) or Pe (permissibly).*
- *The only free variables occurring in G and DEF are agent or entity variables from* Var_{AGENT} *and* Var_{ENTITY}.

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. 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 specifications of data and behavioral norms to expected dynamic interaction protocols between subsystems.

MEASUR and, in particular, our logical restriction of MEASUR, allows much flexibility when detailing the intended meaning of communication acts. This can be done by clarifying assertions. When it comes to the question of implementation of a communication act, an analyst will always ask the client: what is entailed by this act? The client will then explain what changes the act is expected to make on the

elements of the ontology. We encode this description as a *definition* DEF of the act A, of the form

$$A \rightarrow DEF \tag{7}$$

and

$$DEF \rightarrow A \tag{8}$$

henceforth abbreviated as

$$A \leftrightarrow DEF \tag{9}$$

where DEF is any MEASUR formula not involving communication acts, both A and DEF sharing the same free variables.

We are now ready to define our formal notion of a MEASUR requirement analysis document. Essentially, it consists of an ontology and pairs of behavioral norms and definitions, called normative tableaux.

Definition 3.2 (Normative tableaux)**.** *A requirements specification prescribes the action an agent is obliged (or permitted) to perform given the preconditions hold and consists of pairs of behavioural norms, each paired with definitions of the form*

$$REQ = \{(G_i \rightarrow E_{b\,i}\, D_i\, A_i, A_i \leftrightarrow DEF_i \mid i = 1,\ldots,n\} \tag{10}$$

where

- *each D_i is a deontic operator Ob or Pe.*
- *A_i is a single communication act.*
- *The only free variables occurring in G_i, DEF_i and A_i are agent or entity variables from Var_{AGENT} and Var_{ENTITY}.*

Each pair is called a normative tableaux.

A model $\mathcal{M}$ is a collection of these requirement pieces prescribed for some system and said to satisfy REQ if it validates the quantified conjunction of all REQ's normative tableaux. That is, $\mathcal{M}$ satisfies REQ when

$$\mathcal{M} \models \left(\begin{array}{l} \forall x_1 : T_1 \bullet \ldots, x_n : T_k \bullet \\ G_1 \rightarrow E_{b_1}\, D_1\, A_1 \wedge A_1 \leftrightarrow DEF_1 \\ \wedge \ldots \wedge \\ G_n \rightarrow E_{b_n}\, D_1\, A_n \wedge A_n \leftrightarrow DEF_n \end{array} \right) \tag{11}$$

where $x_1 : T_1, \ldots, x_n : T_k$ is the list of all free variables contained in the formulae of the tableaux.

A model for an ontology together with a set of behavioral norms $B_1, \ldots, B_n$ is one in which each norm is true for M. Depending on the model and the interpretation, this might be an abstract representation of a system execution, or might actually be an implementation of the specification: for example, one in which each possible world corresponds to an actual system state.

4 THE METHODOLOGY IN USE

This section will describe the methodology of specifying norms within the framework of new improved MEASUR. These norms specify, given certain conditions are true, an agent is obliged / permitted / impermitted to see to it that certain state of affairs is true. These types of normative specifications are not descriptive enough for requirements engineers to model changes in states of affairs with the necessary depth of data definition. Therefore, we need additional annotations for our norms, which we refer to as *definitions*. We hereafter present this norm/definition pair as so called *normative tableaux*.

In formalizing MEASUR, we illustrate norms as communication acts that are enabled by an agent, using the elements of agency and action of the theory of normative positions. In the framework of new improved MEASUR, in order to specify norms we use so called normative tableaux. Tableau is a table for defining norms and their data definitions. In other words, each normative tableau illustrates a norm as a pre-condition and a responsibility expression that relates an agent to a particular communication act taken from MEASUR ontology. A tableau also provides further notation for these norms in the form of definitions, so that each norm is paired with its definition. All norms have their definitions to obtain a certain depth of data definition and to be able to manipulate with these data. We need these definitions within our new MEASUR in order to provide ways for requirement engineers to define and describe data in conjunction with ontologies.

4.1 Case Study

The subject of our example is to process an order made by a customer using her credit card. The meaning of processing an order embraces herein such activities as receiving a new order, processing its data (price, customer's credit card information), invoicing and dispatching an order, interactions with warehouse and rejecting an order. In performing these activities in the order specified, we examine interaction between three involved players in an organized manner.

The first player, which starts the whole process of ordering is Customer. This player orders products, thereby creating a new instance of order and sends it to the electronic ordering system, which further processes it in a sequence specified below. The electronic ordering system eOrder initiates one activity after another and at some point starts interaction with yet another player called Warehouse. All these activities and communications between different players shift

the order from one state to another. All orders have a strict correspondence with its particular customer and all orders are stored in the system during their processing until they are rejected. The activity of rejection removes the order from the system and cancels the correspondence with its particular customer.

The activities of making a purchase using our system is of the following order:

1. When a customer makes an on-line order, he initiates an interaction with the electronic ordering system by creating a new instance of the order and changing the status of this order to "received".
2. Electronic ordering system initiates the next activity which processes this new order by checking order's data and customer's solvency. If this check is successful, then the status of this order changes from "received" to "pending". Otherwise, if the order's total cost is higher than customer's credit card limit, then the status of his order changes to "rejected", which starts the rejecting process by removing the order from the system.
3. When the order's state changes to "pending" and if the product ordered is in stock, the system initiates the process of invoicing it by changing its status to "invoiced".
4. If the order is not available in the stock, then the electronic ordering system initiates a communication to the warehouse by starting the "request_increase" activity.
5. After successful invoicing, the order's status changes to "dispatched" to indicate that the order is dispatched.

The diagram for this example may be similar to UML's sequence or activity diagrams. But, it is different in the way of providing more expressive definitions for activities, interactions, and the notion of responsibility. We will employ this example to illustrate the use of the MEASUR methodology its formalized version.

We first represent knowledge as a domain ontology encompassing such entities as agents, actions, and relationships between them. The ontologies of MEASUR's semantic analysis are similar to those of, for example, OWL: they decompose a problem domain into roles and relationships. 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.

In our treatment, affordances are viewed as *classifications* 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* which possess the structure prescribed by the ontology and obey any further constraints imposed by an associated set of norms.

In our case study, we model an electronic ordering system. The purchasing system player *eOrder* is related to the *Customer* agent and the *Warehouse* agent. *Customer* orders products from *eOrder* by initializing the interaction. *eOrder* receives data describing the order and performs further processing which normally includes checking availability of the product in the system, customer's solvency, invoicing, dispatching, and rejecting. If processing is successful, the system files an invoice for the purchase and subsequently dispatches it. If the product is unavailable, *eOrder* sends a request to the warehouse to increase the stock to further proceed with the order. If processing is unsuccessful, *eOrder* rejects the order.

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 involves processes that cross the boundaries of two subsystems: a customer agent, an order processing system, and a product warehouse system. These three subsystems are represented as agents in the ontology, *Customer*, *eOrder* and *Warehouse*, respectively. By default all agents except for *Customer* contain start and end attributes.

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* communication act further deals with the newly arrived order and checks whether the client's credit limit allows for the purchase. Namely, it checks whether the total cost of the purchase is less than the credit limit of the customer. This condition results in the following outcomes: if the credit limit is lower than the total cost of the purchase then the system rejects the order, otherwise it initiates the invoicing process (denoted by

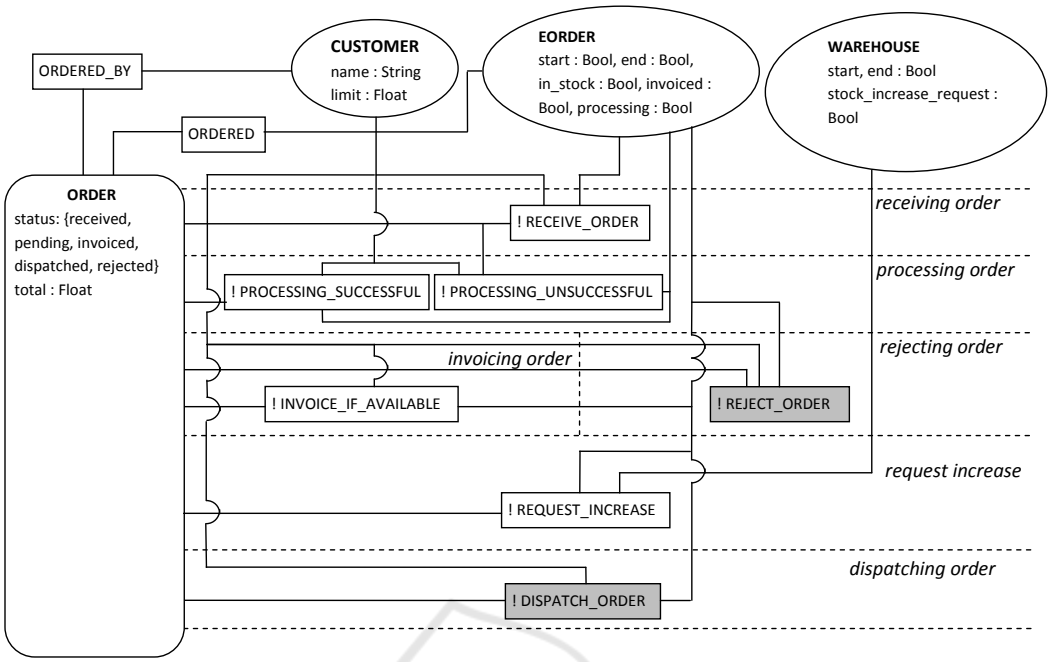

Figure 1: Example normative ontology.

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*, which sets flag *stock_increase_request* to true. Finally, the system dispatches the order by *dispatch_order*.

4.2 Behavioral Norms and Definitions

All norms described here strictly follow the norm-definition relationship (9) and take form according to (10). 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 normative tableaux shown in (12) and (13). 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. The meaning of the behavioral norms are extracted from additional definitions, which represent their equivalent meaning. An equivalent meaning of 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 by the norm

$$\forall cc : Customer \bullet \forall oo : Order \bullet \forall e : eOrder \bullet \\ \neg ordered\_by(oo, cc) \wedge \neg ordered(oo, e) \rightarrow \\ E_e \; Ob \; \mathbf{receive\_order}(\mathbf{oo}, \mathbf{cc}, \mathbf{e}) \quad (12)$$

and its definition

$$\forall cc : Customer \bullet \forall oo : Order \bullet \\ \forall e : eOrder \bullet \mathbf{receive\_order}(\mathbf{oo}, \mathbf{cc}, \mathbf{e}) \leftrightarrow \\ ordered\_by(oo, cc) \wedge ordered(oo, e) \wedge \\ oo.status = received \wedge e.processing = \top \quad (13)$$

The norm (12) and its definition (13) form a single normative tableau for the communication act *!receive_oder*. Note that we have employed the conventions and assumptions just given. So determinacy is implicit, and we are quantifying over individual formulae, to indicate types of variables. However, importantly, all variables – for example, *oo* and *cc* – should be understood as denoting the same possible interpreting object in the semantics.

To further proceed with the new order we define a norm for processing the data received. This norm

effectively defines the requirement to check the payment (in order to be able to proceed with the order in a successful manner, it is important that the *total* for the purchase does not exceed the *limit* value in the credit card) and order related data. It should also be noted that another requirement to enable this norm is that the order should already be *received* from a *particular* customer. In order to model a system which behaves in a discrete fashion, norm *processing* is split into two cases: *process_successful*

$$\forall cc : Customer \bullet \forall oo : Order \bullet \forall e : eOrder \bullet \\ ordered\_by(oo, cc) \wedge ordered(oo, e) \wedge \\ oo.status = received \wedge e.processing = \top \wedge \\ oo.total < cc.limit \rightarrow \\ E_e\ Ob\ \mathbf{process\_successful}(\mathbf{oo}, \mathbf{cc}, \mathbf{e}) \quad (14)$$

with its definition

$$\forall cc : Customer \bullet \forall oo : Order \bullet \forall e : eOrder \bullet \\ \mathbf{process\_successful}(\mathbf{oo}, \mathbf{cc}, \mathbf{e}) \leftrightarrow \\ oo.status = pending \quad (15)$$

and *process_unsuccessful*

$$\forall cc : Customer \bullet \forall oo : Order \bullet \forall e : eOrder \bullet \\ ordered\_by(oo, cc) \wedge ordered(oo, e) \wedge \\ oo.status = received \wedge e.processing = \top \wedge \\ oo.total \geq cc.limit \rightarrow \\ E_e\ Ob\ \mathbf{process\_unsuccessful}(\mathbf{oo}, \mathbf{cc}, \mathbf{e}) \quad (16)$$

with its definition

$$\forall cc : Customer \bullet \forall oo : Order \bullet \forall e : eOrder \bullet \\ \mathbf{process\_unsuccessful}(\mathbf{oo}, \mathbf{cc}, \mathbf{e}) \leftrightarrow \\ oo.status = rejected \quad (17)$$

If the pre-conditions of *process_successful* hold then the norm changes the status to *pending*, as it is depicted in (14), which enables the *!invoice_if_available* communication act. Otherwise, the system initiates rejecting of the order by setting *status* to *rejected* as it is illustrated in (16). Below we show the norm and its definition for invoicing.

$$\forall cc : Customer \bullet \forall oo : Order \bullet \forall e : eOrder \bullet \\ ordered\_by(oo, cc) \wedge ordered(oo, e) \wedge \\ oo.status = pending \wedge e.in\_stock = \top \rightarrow \\ E_e\ Ob\ \mathbf{invoice\_if\_available}(\mathbf{oo}, \mathbf{cc}, \mathbf{e}) \quad (18)$$

$$\forall cc : Customer \bullet \forall oo : Order \bullet \forall e : eOrder \bullet \\ \mathbf{invoice\_if\_available}(\mathbf{oo}, \mathbf{cc}, \mathbf{e}) \leftrightarrow \\ oo.status = invoiced \wedge e.order\_invoiced = \top \quad (19)$$

Invoicing the order and increasing the stock are followed by the dispatching process. We depict its definition.

$$\forall cc : Customer \bullet \forall oo : Order \bullet \forall e : eOrder \bullet \\ ordered\_by(oo, cc) \wedge ordered(oo, e) \wedge \\ oo.status = invoiced \rightarrow \\ E_e\ Ob\ \mathbf{dispatch\_order}(\mathbf{oo}, \mathbf{cc}, \mathbf{e}) \quad (20)$$

$$\forall cc : Customer \bullet \forall oo : Order \bullet \forall e : eOrder \bullet \\ \mathbf{dispatch\_order}(\mathbf{oo}, \mathbf{cc}, \mathbf{e}) \leftrightarrow \\ oo.status = dispatched \quad (21)$$

Dispatching relates *Customer* to *eOrder* and *Order* and uses their instances for pre-conditions and to change the state of affairs towards updating the status to *dispatched*. This communication act finishes the ordering process. If processing was not successful, in other words if pre-conditions for norm *process_unsuccessful* were true, then the *eOrder* system rejects the order in accordance with the norm in (22).

$$\forall cc : Customer \bullet \forall oo : Order \bullet \forall e : eOrder \bullet \\ ordered\_by(oo, cc) \wedge ordered(oo, e) \wedge \\ oo.status = rejected \rightarrow \\ E_e\ Ob\ \mathbf{reject\_order}(\mathbf{oo}, \mathbf{cc}, \mathbf{e}) \quad (22)$$

and its definition

$$\forall oo : Order \bullet \forall e : eOrder \bullet \\ \mathbf{reject\_order}(\mathbf{oo}, \mathbf{cc}, \mathbf{e}) \leftrightarrow \\ \neg ordered(oo, e) \wedge \neg ordered\_by(oo, cc) \quad (23)$$

According to Fig. 1, there exist other communication acts such as *invoice_if_available* and *dispatch_order* and their norm-definition pairs can also be defined in a similar way we have defined tableaux (12)–(23). The norm-definition relationship is important as expression in the definition part of the tableaux further elaborates each norm that precedes this definition. For example, the norm with its arguments *receive_order(oo,cc,e)* defined in (12) has a meaning of mapping certain objects together while defining a new instance of the order, in that:

- $ordered\_by(oo, cc)$ means that instance of order *oo* is mapped to the instance of an abstract machine *cc* (that models a customer),
- $ordered(oo, e)$ means that instance of order *oo* is mapped to the instance of machine *e* (that models a system),
- $oo.status = received$ means that the status should now change its value to *received*, and

- $e.processing = \top$ means that the processing status of the system should now change to *processing* to invoke the next norm.

All other subsequent norms are similar in the way they are read.

5 CONCLUSION

In this paper, we have shown an approach of combining the MEASUR methodology and the limited version of the theory of normative positions to model norms and normative ontologies. The level of abstraction pertinent to this approach effectively corresponds to early stages of business modeling. The model that we obtain as a result is a piece of knowledge which can (semi-formally) describe a functional structure of a company (e.g., stakeholders, departments, people, business processes, communications between these processes, etc.).

The approach described in this paper represents a combination of MEASUR with the theory of normative positions limited to the kinds of norms used in this methodology and describe business rules which specify interaction between two agents for achieving certain goals. Since the MEASUR methodology is a set of methods for business modeling and requirements specification for information systems, it can be used to analyze and specify an organization's business processes. The term affordance is introduced in this paper as a set of actions formed by the knowledge that an agent gains from existing in the environment and learning by interacting. The artifact developed can effectively be used as a high-level structured requirements specifications that can be turned into more concrete models using a range of different transformation and refinement techniques (e.g. Model-driven Architecture or refinement of B-based models), which will be the subject of our future work. We intend to see how the developed methodology can be useful in both developing formalized and structured requirements and semantically better fit the modular design specifications using formal languages such as B and Event-B.

REFERENCES

Cimatti, A., Roveri, M., Susi, A., and Tonetta, S. (2010). Formalization and Validation of Safety-Critical Requirements. *EPTCS*.

Hohfeld, W. (1964). *Fundamental Legal Conceptions As Applied in Judicial Reasoning*. Dartmouth Publishing.

Jones, A. and Sergot, M. (1992). Formal Specification of Security Requirements Using the Theory of Normative Positions. *Proceedings of the Second European Symposium on Research in Computer*.

Jones, A. and Sergot, M. (1993). On the Characterization of Law and Computer Systems: The Normative Systems Perspective. *Deontic Logic in Computer Science: Normative System Specification*.

Kanger, S. (1972). *Law and Logic*. Theoria.

Kanger, S. (1985). On Realization of Human Rights. *Action, Logic and Social Theory*.

Lindahl, L. (1977). *Position and Change – A Study in Law and Logic*. D. Reidel Publishing Company.

Lindahl, L. (1994). Stig Kanger's Theory of Rights. *Logic, Methodology and Philosophy of Science*.

Liu, K. (2000). *Semiotics in Information Systems Engineering*. Cambridge University Press.

McNamara, P. (2006). Deontic Logic. *Stanford Encyclopedia of Philosophy*.

Ponsard, C. and Dieul, E. (2008). From Requirements Models to Formal Specifications in B. *ReMo2V*.

Pörn, I. (1970). *The Power of Logic*. Blackwell Oxford.

Pörn, I. (1977). *Action Theory and Social Science: Some Formal Models*. D. Reidel, Dordrecht.

Searle, J. (1969). *Speech acts: An Essay in the Philosophy of Language*. Cambridge University Press.

Searle, J. (1971). *The Philosophy of Language*. Oxford University Press.

van Lamsweerde, A. (2001). Building formal requirements models for reliable software. *Reliable Software Technologies*.