Formalizing Agent-Oriented Enterprise Models

Ivan Jureta[1], Stéphane Faulkner[1], and Manuel Kolp[2]

[1] Information Management Research Unit, University of Namur,
8 Rempart de la vierge, B-5000 Namur, Belgium
{ivan.jureta, stephane.faulkner}@fundp.ac.be
[2] Information System Research Unit, University catholic of Louvain,
1 Place des doyens, B-1348 Louvain-la-Neuve, Belgium
kolp@isys.ucl.ac.be

Abstract. This paper proposes an agent-oriented metamodel that provides rigorous concepts for conducting enterprise modelling. The aim is to allow analysts to produce an enterprise model that precisely captures the knowledge of an organization and of its business processes so that an agent-oriented requirements specification of the system-to-be and its operational corporate environment can be derived from it. To this end, the model identifies constructs that permit capturing the intrinsic characteristics of an agent system such as autonomy, intentionality, sociality, identity and boundary, or rational self-interest; an agent being an organizational actor and/or a software component. Such an approach of the concept of agent allows the analyst to have a holistic perspective integrating human and organizational aspects to gain better understanding of business system inner and outer modelling issues. The metamodel has roots in both management theory and requirements engineering. It helps to bridge the gap between enterprise and requirements models proposing an integrated framework, comprehensive and expressive to both managers and software (requirements) engineers.

1 Introduction

Business analysts and IT managers have advocated these last fifteen years the use of enterprise models to specify the organizational and operational environment (outer aspects of the system) in which a corporate software will be deployed (inner aspects of the system) [1]. Such a model is a representation of the knowledge an organization has about itself or of what it would like this knowledge to be. This covers knowledge about functional aspects of operations that describe what and how business processes are to be carried out and in what order; informational aspects that describe what objects are to be processed; resource aspects that describe what or who performs these processes according to what policy; organizational aspects that describe the organizational architecture within which processes are to be carried out; and, finally, strategic aspects that describe why processes must be carried out. The specification of these key aspects of the core business of an enterprise is an effective tool to consider for gathering and eliciting software requirements. It may be used to [2, 3]:

- analyse the current organizational structure and business processes in order to reveal problems and opportunities;
- evaluate and compare alternative processes and structures;
- achieve a common understanding and agreement between stakeholders (e.g. managers, owners, workers) about different aspects of the organization;
- reuse knowledge available in the organization.

This paper proposes an integrated agent-oriented metamodel for enterprise modelling. The agent paradigm is a recent approach in software engineering that allows developers to handle the lifecycle of complex distributed and open systems required to offer open and dynamic capabilities in the latest generation enterprise software (see e.g. [4]).

The proposed metamodel takes inspiration from research works in requirements engineering frameworks (see e.g. [5-6]), management theory concepts found to be relevant for enterprise modelling (see e.g. [7-9]) and agent-oriented software engineering (see e.g. [4]). It leads to the reduction of the semantic gap between enterprise and requirements representations, providing a modelling tool that integrates the outer specification of the system together with its inner specification. Our proposal implicitly suggests a holistic approach to integrate human and organizational issues and gain better understanding of the representation of business processes and organizations representation. To this end, we introduce new concepts to enterprise modelling, related to authority, power and interest.

The rest of this paper is organized as follows. Section 2 describes the main concepts of our metamodel. Sections 3 and 4 detail some elements of the metamodel using the Z specification langage and discuss their relevance for enterprise modelling. Section 5 gives an overview of related works and Section 6 summarizes the results and points to further work.

2 An Agent-Oriented Enterprise Metamodel

The motivation of our proposal is to understand precisely the semantics of the organizational environment of the system and to produce an agent-oriented requirements specification for the software to build. The framework described in this section provides modelling constructs that permit the representation of the autonomy, intentionality, sociality, identity and boundary, as well as the rational self-interest of actors, i.e. agents in the real world and/or software agents. Actors are autonomous as their behaviour is not prescribed and varies according to their dependencies, personal goals and capabilities. They are intentional since they base their actions and plans on beliefs about the environment, as well as on goals they have to achieve. Being autonomous, actors can exhibit cooperative behaviour, resulting from similar goals and/or reciprocal dependencies concerning organizational roles they assume. The dependencies can either be direct or mediated by other organizational roles. Actors can have competing goals, which lead to conflicts that may result from competing use of resources. Actors have varying power and interest in the ways in which organizational goals contribute to their personal ones. Boundary and identity are closely related to power and interest of actors. We model variations in boundary and identity as resulting from changes in power and interest since these vary with respect to the modifications in the roles an actor assumes and the dependencies involving these roles. Actors can act according to their self-interest, as they have personal goals to achieve. They have varying degrees of motivation to assume organizational roles, according to the degree of contribution to personal goals these roles have in achieving organizational ones. Actors apply plans according to the rationale described in terms of personal, organizational goals and capabilities. The rationale of our actors is not perfect, but bounded [10-11], since they can act based on beliefs that are incomplete and/or inconsistent with reality. We provide constructs such as AndOr relationships, non-functional requirements [4] etc. to evaluate alternative deployments of the software in the organizational environment.

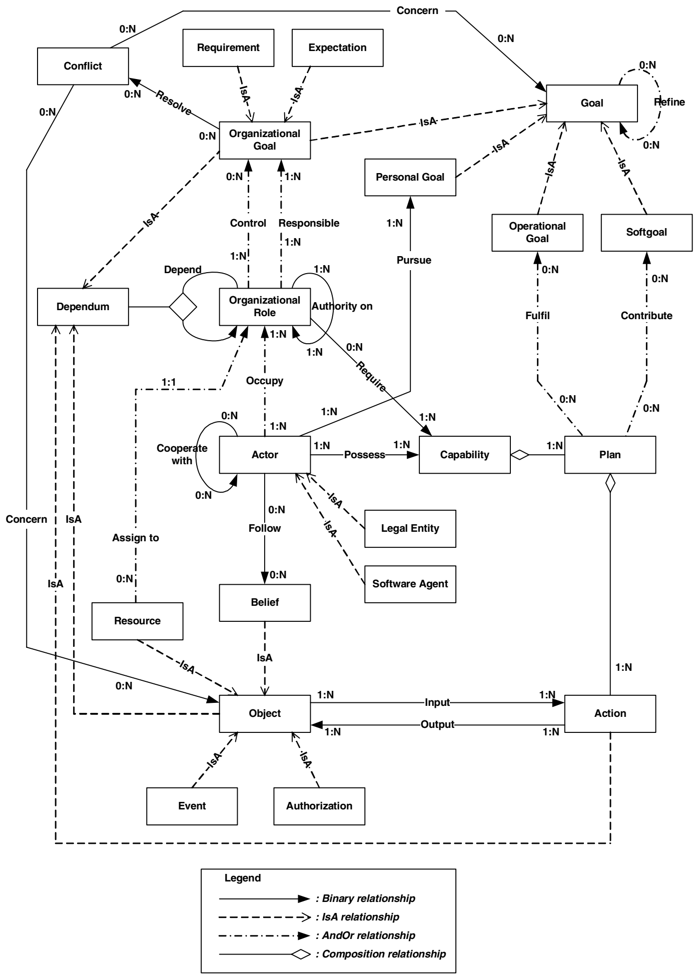

Fig. 1. The agent-oriented metamodel

Fig. 1 introduces the main entities and relationships of our metamodel. For clarity, we have subdivided it into five sub-models:

- *Organizational sub-model*, describing the actors of the organization, their organizational roles, responsibilities and capabilities.
- *Goals sub-model*, describing enterprise and business process purposes, i.e. what the actors are trying to achieve and why.
- *Conflict sub-model*, indicating inconsistencies in the business process.
- *Process sub-model*, describing how actors achieve or intend to achieve goals.
- *Objects sub-model*, describing non-intentional entities and assumptions about the environment of the organization and the business processes.

Due to a lack of space, the paper only details the organizational and goal sub-models, their integration and discusses their relevance for enterprise modelling. We first sketch the metamodel from the point of view of system developers and of organization managers.

2.1 Information System Development Perspective

The metamodel provides widely-used constructs for specifying the architecture of an agent-oriented information system: *Actors* are agents of the system. They *possess Capabilities* composed of *Plans*, each *Plan* representing a sequence of atomic *Actions*. When applying *Plans*, *Actors fulfil* or *contribute* to system *Goals*. *Actors follow Beliefs* which represent assertions about aspects of the organization and/or its environment. *Actions* can take *Objects* as *input* from the system or its environment. New *Objects* can be produced or existing ones modified by carrying out *Actions*, i.e. they can be *output* from *Actions*. *Objects* represent any thing of interest for the system: *Resources*, *Beliefs*, *Authorizations* or *Events.*

2.2 Management Perspective

The metamodel provides common terms used to describe an organization. *Organizational Roles* are *responsible* of *Organizational Goals*, which may be either *Operational* (i.e. can be actually fulfilled) or *Softgoals* (such as e.g. broadly specified business objectives). *Organizational Roles* can *depend* on one another for the provision of *Dependums* - *Actions*, *Objects*, or *Organizational Goals*. An *Actor*, being a *Human Actor* or a *Software Agent*, can *occupy Organizational Roles*, as long as it *possesses* the *required Capabilities* to do so. *Actors* exhibit intentional behaviour since they act according to *Goals* and *Beliefs* about their environment. Since *Beliefs* may be incoherent, and as they pursue *Personal Goals*, *Actors* can exhibit competitive behaviour. They will exhibit cooperative behaviour when they are responsible of identical *Organizational Goals*. *Actors* execute *Plans*, composed of *Actions*, in order to fulfil and contribute to *Goals*. By doing so, they comply with the responsibilities of *Organizational Roles* they *occupy*. As a matter of organizational policy, *Resources* in the organization are assigned to *Organizational Roles*. The allocation of Resources is determined by both *authority* among *Organizational Roles* and *Authorizations* that may be *input* or *output* of specific *Actions*.

Common ground between both points of view resides in the sense that the information system can be developed to automate some (part of) business processes (e.g. administrative tasks) or to radically modify ways in which *Goals* are fulfilled (e.g. reorganizing customer relationship management by deploying e-commerce facilities). The model provides an unambiguous representation serving both software staff and organization strategic management.

Primitives of our framework are of different types: meta-concepts (*Goal*, *Actor*, *Object* etc.), meta-relationships (*possess*, *require*, *pursue* etc.), meta-attributes (*Power*, *Interest*, *Motivation* etc.) and meta-constraints (e.g. "*an actor occupies a position if that actor possesses all the capabilities required to occupy it*").

All meta-concepts, meta-relationships and meta-constraints have the following mandatory meta-attributes:

- *Name*, which allows unambiguous reference to the instance of the meta-concept (e.g. "European Commission" for the Actor meta-concept).
- *Description*, which is a precise and unambiguous description of the corresponding instance of the meta-concept. The description should contain sufficient information so that a formal specification can be derived for use in requirements specifications for a future information system.

Fig.1 shows only meta-concepts and meta-relationships. Meta-attributes and meta-constraints are specified in the next sections using the Z state-based specification language [12, 13]. We use Z since it provides sufficient modularity, abstraction and expressiveness to describe in a consistent, unified and structured way an agent-oriented IS and the wider context in which it is used. It has a pragmatic approach to specifications by allowing a clear transition between specification and implementation of software [13]. In addition, it is widely accepted in the software development industry and has been used in large-scale projects.

3 Organizational Sub-model

The Organizational sub-model is used to identify the relevant *Actors* of the organization, the *Organizational Roles* they *occupy*, the *Capabilities* they *possess* and the *Dependum* for which *Actors depend on* one another.

3.1 Actor

Fig. 2 shows the Z formal specification of the *Actor* concept. The first part of the specification represents the definition of types. A given type defines a finite set of items. The *Actor* specification first defines the type Name (which represents the *Name* attribute) by writing [Name]. Such a declaration introduces the set of all names, without making assumptions about the type (i.e. whether the name is a string of characters and numbers, or only characters, etc.). Note that the type Actor_Type is defined as being either a Human_Actor or a Software_Agent. Defining types in such way indicates either that further detail about the type would not add significant descriptive power to the specification or that a more elaborate internal representation is not required.

More complex and structured types are defined with schemata. A schema groups a collection of related declarations and predicates into a separate namespace or scope. The schema in Fig. 2 is entitled ***Actor*** and is partitioned by a horizontal line into two sections: the declaration section above and the predicate section below the line. The declaration section introduces a set of named, typed variable declarations. The predicate section provides predicates that constrain values of the variables, i.e. predicates are used to represent constraints. In order to clarify the Z formal specifications of the concepts, we will refer in the text to specific Z schema predicates by using identifiers placed left of the schema in the form e.g. "(c1)" to refer to predicate, i.e. constraint (c1) of the schema.

[Name]
[Informal_Definition]
[Actor_Type]:= Human_Actor | Software_Agent
[Organizational_Role]
[Goal]
[Interest_Value]
[Power_Value]

Actor

name : *Name*
description : *Informal_Definition*
isa : *Actor_Type*
occupy : set *Organizational_Role*
possess : set *Capability*
has : set *Belief*
own : set *Resource*
pursue : set *Goal*
interest : *Interest_Value*
power : *Power_Value*

(c1) $(\text{occupy} \neq \emptyset) \wedge (\text{possess} \neq \emptyset) \wedge (\text{pursue} \neq \emptyset)$

(c2) $(\forall\ \text{act}: \textit{Actor})\ \text{act.isa} = \text{Human\_Actor} \Rightarrow \text{act.interest} \neq \emptyset \wedge \text{act.power} \neq \emptyset$

Fig. 2. Formal specification of the *Actor* concept

An Actor applies *Plans* (which are part of his *Capabilities*) to *fulfil* and/or *contribute* to *Organizational Goals* for which the *Organizational Role* he/she *occupies* is *responsible* and *Personal Goals* he/she *pursues* (i.e. wishes to achieve). As the *Actor* exists in a changing environment, it *follows Beliefs* about the environment in order to adapt its behaviour to environmental circumstances.

An *Actor* is either a *Human Actor* or a *Software Agent*. A *Human Actor* is used to represent any person, group of people, organizational units or other organizations that are significant to the organization we are modelling, i.e. that have an influence on its resources, its goals etc. A *Software Agent* is used to represent a software component of an information system(-to-be). An *Actor* can *cooperate with* another *Actor* to fulfil and/or contribute to *Organizational Goals* common to the *Organizational Roles* that each of these *Actors occupies*.

Besides standard meta-attributes, a *Human Actor* is characterized with two specific meta-attributes: *Interest* and *Power*. *Interest* is the degree of satisfaction of an actor to see *Organizational Goals* positively contributing to its *Personal Goals*. *Power* is the degree to which the actor is able to modify the objectives of the organization or its business processes through its *Capabilities*. For instance, when automating a business process, the values of *Interest* and *Power* meta-attributes of *Human Actors* change: in the new configuration of the process, some actors will gain decision power while maintaining the same level of interest; others that previously benefitted from high power in the initial process structure might become less powerful. It is crucial to take these changes into account when eliciting software requirements. It may lead otherwise to introducing *Goals* not identified during the initial requirements analysis, and/or changing *Priority* of already specified *Goals*. *Interest* and *Power* help to find *Human Actors* that will play a crucial role in the software-to-be. For example, focus in some business process might shift to *Human Actors* that were not considered very significant during the inception phase and whose needs were not specified in depth. This would result in that these now crucial processes would not be fully exploited and would lead to the overall failure of the requirements specification efforts.

3.2 Organizational Role

An *Organizational Role* is an abstract characterization of expected behaviour of an *Actor* within some specified context of the organization. An *Actor* can *occupy* multiple roles and a role can be *occupied* by multiple *Actors*.

From an agent orientation perspective, *Organizational Roles* provide the building blocks for agent social systems and the requirements by which agents interact. The concept of *Organizational Role* is important to abstractly model the agents in multi-agent systems and helpful to manage its complexity without considering the concrete details of agents (e.g. implementation architectures and technologies).

Fig. 3 shows the Z formal specification of the *Organizational Role* concept. Each *Organizational Role requires* a set of *Capabilities* to fulfil or contribute to *Organizational Goals* for which it is *responsible*. An *Actor* can *occupy* the *Organizational Role* only if it *possesses* the required *Capabilities* (c4)[1]. In addition to entering *Organizational Roles*, *Actors* should be able to leave roles at runtime. The attribute *Leave Condition* is used to specify the *Belief* that has to be true in order for the *Actor* to leave the *Organizational Role* (c5).

Organizational Roles are *responsible* for *Organizational Goals* (c6) and can *control* their *fulfilment*. In case an *Organizational Goal* has been fulfilled, the *Actor*, occupying the *Organizational Role* that *controls* that *Goal*, executes a *Plan* in which an *Action output*s a new *Belief* to mark the goal fulfilment (c7). This control procedure requires that a single *Actor* can never occupy distinct *Organizational Roles* that are *responsible* of and *control* the fulfilment of the *Organizational Goal* (c8).

[1] To clarify the formal specifications, we embed the comments on predicates between two "//" signs.

[Goal_Control_Status]:= Fulfilled I Unfulfilled
[Belief]

Organizational Role

name : *Name*
description : *Informal_Definition*
require : set *Capability*
leave_condition: set *Belief*
responsible : set *Goal*
control : set (*Organizational_Goal*, *Goal_Control_Status*)
authority_on : set *Organizational_Role*

(c3) $(\text{require} \neq \emptyset) \wedge (\text{leave\_condition} \neq \emptyset) \wedge (\text{responsible} \neq \emptyset)$

(c4) $(\forall$ act: *Actor*, r: *Organizational_Role*$)$
$r \in \text{act.occupy} \Rightarrow \text{r.require} \subset \text{act.possess}$
//An Actor *act* that occupies the Organizational Role *r* possesses the Capabilities required by the Organizational Role *r*.//

(c5) $(\forall$ act: *Actor* ; r: *Organizational_Role*$)$
$\text{act.has} \subset \text{r.leave\_condition} \Rightarrow r \notin \text{act.occupy}$
//If the Leave Condition is true, than the Actor *act* no longer occupies the Organizational Role *r*.//

(c6) $(\forall$ r: *Organizational_Role* ; g: *Goal*$)$
$g \in \text{r.responsible} \Rightarrow \text{g.sec\_isa} = \text{Organizational\_Goal}$
//If Organizational Role *r* is responsible of Goal *g*, then *g* is an Organizational Goal.//

(c7) $(\forall$ r: *Organizational_Role*; g: *Goal*$)$
$(\text{g.prim\_isa} = \text{Operational\_Goal} \wedge \text{g.sec\_isa} = \text{Organizational\_Goal}$
$\wedge\, g \in \text{r.control} \wedge \text{g.status} = \text{Fulfilled})$
$\Rightarrow (\exists\, b!: \textit{Belief})\ (\text{g.status} = \text{Fulfilled}) \in \text{b.term} \wedge (g, \text{Fulfilled}) \in \text{r.control})$
//If an Organizational Operational Goal *g* is fulfilled, then the Organizational Role *r* which controls the fulfilment of *g* outputs a new Belief *b* which indicates that the Goal *g* has been fulfilled.//

(c8) $(\forall\, r_1, r_2$: *Organizational_Role* ; g: *Goal* ; a_1, a_2: *Actor*$)$
$(\text{g.sec\_isa} = \text{Organizational\_Goal} \wedge g \in r_1.\text{responsible} \wedge g \in r_2.\text{control} \wedge r_1 \neq r_2 \wedge r_1 \in \text{act.occupy} \wedge r_2 \in \text{act.occupy}) \Rightarrow a_1, \neq a_2$
//There can be no Actor *a* which occupies both the Organizational Role r_1 which is responsible for Organizational Goal *g*, and the Organizational Role r_2 which controls the fulfilment of Organizational Goal *g*.//

Fig. 3. Formal specification of the *Organizational Role* concept

Organizational Roles can have different levels of authority. Consequently, an *Organizational Role* can have *authority on* other *Organizational Roles*. The *authority on* relationship specifies the hierarchical structure of the organization. For instance, in the context of multi-agent systems, it can be used to define security policies that differ according to authority attributed to software agents.

3.3 Capability

A Capability specifies the behaviours that Organizational Roles should have in order to be responsible for or to control their Organizational Goals. An Actor possesses Capabilities. The formal specification in Fig. 4 shows that a Capability can be structured as a set of Plans and/or other Capabilities. This increases system modularity since libraries of capabilities can be built up and then combined to provide complex functionalities.

When exploring possible alternative business processes or organizational structures, newly identified Organizational Roles can require Capabilities that no Actor possesses. These Capabilities have to be confronted to those available in the organization (Capabilities that the Actors possess, see (c10)), in order to evaluate the proposed alternatives with respect to the current Roles and the way they use existing Capabilities. This is significant to determine which and how the proposed Capabilities and Roles will be finally introduced through the system-to-be. The availability of a Capability is formally expressed through the availability attribute, as indicated in the Capability schema.

[Cap_Atom]:= Plan | Capability
[Cap_Availability]:= Available | Unavailable

Capability

name : *Name*
description : *Informal_Definition*
composed_of : set *Cap_Atom*
availability : *Cap_Availability*

(c9) composed_of $\neq \emptyset$

(c10) ($\forall$ cap: *Capability*)
$\exists$ act: *Actor* ; cap $\in$ act.possess $\Rightarrow$ cap.availability = available
//If there is some Actor *act* that possesses Capability *cap*, then *cap* is available.//

Fig. 4. Formal specification of the *Capability* concept

3.4 Dependum

An *Organizational Role depends* on another *Organizational Role* for a *Dependum*, so that the latter may provide the *Dependum* to the former. A *Dependum* can be an *Organizational Goal*, an *Object* or an *Action*. In the *depend* meta-relationship, the *Organizational Role* that depends on is called the depender and the *Organizational Role* being depended upon is called the dependee.

We define the following dependency types:

- *Organizatonal Goal*-dependency: the depender *depends* on the dependee to *fulfil* and/or *contribute* to an *Organizational Goal*. The dependee is given the possibility to choose *Plans* through which it will *fulfil* and/or *contribute* to the *Organizational Goal*.

- *Action*-dependency: The depender *depends* on the dependee to accomplish some specific *Action*.
- *Object*-dependency: The depender *depends* on the dependee for the availability of an *Object*.

[Dependum_Type]:= Organizational_Goal I Object I Action

Dependum

name : *Name*
description : *Informal_Definition*
type : *Dependum_Type*
depender : set *Organizational_Role*
dependee : set *Organizational_Role*

(c11) $(\text{type} \neq \varnothing) \wedge (\text{depender} \neq \varnothing) \wedge (\text{dependee} \neq \varnothing)$

(c12) $(\forall$ d: *Dependency* ; dpd: *Dependum* ; r_1, r_2: *Organizational_Role*$)$
$r_1 \neq r_2 \wedge (d \equiv r_1 \times dpd \times r_2) \Rightarrow (\text{depender} = r_2 \wedge \text{dependee} = r_1)$

(c13) $(\forall$ d: *Dependency* ; dpd: *Dependum* ; r_1, r_2: *Organizational_Role* $)$ $r_1 \neq r_2 \wedge (d \equiv r_1 \times dpd \times r_2) \wedge (\text{dpd.type} = \text{Authorization}) \Rightarrow r_1 \in r_2.\text{authority\_on}$
//If the Dependum is an Authorization, then Dependee r_2 has authority on Depender r_1.//

(c14) $(\forall$ obj: *Object* ; a_1, a_2: *Actor* ; cap_1, cap_2: Capability ; pl_1, pl_2: *Plan* ; $actn_1, actn_2$: *Action* ; r_1, r_2: *Organizational_Role*$)$
$(a_1 \neq a_2 \wedge cap_1 \neq cap_2 \wedge pl_1 \neq pl_2 \wedge actn_1 \neq actn_2 \wedge (actn_1 \in pl_1.\text{composed\_of} \wedge pl_1 \in cap_1.\text{composed\_of} \wedge cap_1 \in a_1.\text{possess}) \wedge (actn_2 \in pl_2.\text{composed\_of} \wedge pl_2 \in cap_2.\text{composed\_of} \wedge cap_2 \in a_2.\text{possess}) \wedge obj \in actn_1.\text{postcondition} \wedge obj \in actn_2.\text{input} \wedge r_1 \in a_1.\text{occupy} \wedge r_2 \in a_2.\text{occupy} \wedge \{r_1,r_2\} \notin \{a_1.\text{occupy} \cap a_2.\text{occupy}\}) \Leftrightarrow (\exists$ dm: *Dependum* $\wedge$ dm.type = Object $\wedge$ dm.name = obj.name $\wedge$ dm.depender = r_2 $\wedge$ dm.dependee = $r_1)$
//Suppose that there are two different Actors a_1 and a_2 that respectively occupy two different Organizational Roles r_1 and r_2. These Actors possess respectively two different Capabilities cap_1 and cap_2, which respectively contain distinct Plans pl_1 and pl_2. These plans enable them to execute respectively the distinct Actions $actn_1$ and $actn_2$. If Action *$actn_1$* has Object *obj* in its postcondition, and Action *$actn_2$* outputs *obj*, then Organizational Role *r_2* depends on the Organizational Role *r_1* to provide the Object *obj*.//

Fig. 5. Formal specification of the *Dependum* concept

Object dependency allows us to represent any specialization of the *Object* concept as a *Dependum*. For example, an *Organizational Role r_1* might *depend* on another *Organizational Role r_2* for an *Authorization*. This has implications on the *authority on* relationship, as this dependency means that r_2 must have *authority on r_1* (c13).

The constraint (c14) in Fig. 5 shows that the existence of an *Object Dependum* among *Organizational Roles* has implications on the *Input* and *Postcondition* of *Actions* accomplished by *Actors* that *occupy* these *Organizational Roles*. This constraint provides a mapping rule between *depend* and *input/output* relationships. Its interest (c14) is twofold:

- If we know *Object* dependencies existing among several organizational roles, we can derive the activity diagram and the collaboration diagram (such as the ones in UML) without difficulties: actions that are related by dependencies (through their respective inputs/outputs) can be either sequential or parallel, which is sufficient to define the activity diagram. In addition, we know the actors that need to execute actions, as we know the organizational roles involved in dependencies.
- If we know the sequence of activities in a process, we can derive the dependencies among roles that participate in the realization of the process. Dependencies can then be analysed for vulnerabilities and alternative process structures can be evaluated.

This is an important difference of our approach compared to *i** [5]: we can use the link established between dependencies and actions in e.g. analyzing simultaneously the dependencies among organizational roles and the behavioural aspects of the process being analysed in terms of sequence of actions that compose it. This constraint makes it possible to combine the strengths of the *i** dependency representation, notably in terms of strategic dependency analysis among the process' organizational roles, with the analysis of the realization of the process as a series of sequential and/or parallel actions, that can be realized using e.g. UML activity and collaboration diagrams or scenario-based approaches.

4 Goals Sub-model

A *Goal* describes a desired or undesired state of the environment. The environment is the context in which actors live and interact with other actors. A state of the environment is described through the states of *Objects* (*Beliefs*, *Resources* etc.).

In addition to standard attributes, a *Goal* is characterized by the optional *Priority* attribute [14], which specifies the extent to which the goal is optional or mandatory. The values and the measurement of priority are domain specific.

To support qualitative and formal reasoning about goals, we classify them along two axes: *Operational Goals vs. Softgoals* and *Organizational Goals vs. Personal Goals*. In addition, we use patterns to specify the temporal behaviour of *Goals*. These classifications are treated in more detail below.

Operational Goal vs. Softgoal. An *Operational Goal* describes a desired or undesired state of the environment that can be achieved by applying *Plans*. An *Operational Goal* has been *fulfilled* if the state of the environment described by the *Operational Goal* has been achieved by a Plan. An *Operational Goal* has *State* and *Status* optional attributes (see Fig. 6). *State* describes the state of the environment in which the *Operational Goal* is fulfilled (c15). *Status* indicates whether the *State* of the *Operational Goal* has been achieved, i.e. whether the *Goal* has been fulfilled or not (c16).

A *Softgoal* also describes a desired or undesired state of environment, but its fulfilment criteria (i.e. how to achieve the desired state) may not be formally specified. A consequence of this is that *Plans* that are otherwise applied to *fulfil*

[Primary_Goal_Type]:= Operational_Goal I Softgoal
[Secondary_Goal_Type]:= Organizational_Goal I Personal_Goal
[Org_Goal_Type]:= Requirement I Expectation
[Goal_Pattern]:= Achieve I Cease I Maintain I Avoid
[Object]:= Resource I Authorization I Belief I Event
[Goal_Status]:= Fulfilled I Unfulfilled
[Refinment_Alternative]
[Priority_Value]
[Conflict]

Goal

name : *Name*
description : *Informal_Definition*
prim_isa : *Primary_Goal_Type*
sec_isa : *Secondary_Goal_Type*
org_isa : *Org_Goal_Type*
pattern : *Goal_Pattern*
state : set *Object*
status : *Goal_Status*
refined_by : set *Refinement_Alternative*
priority : *Priority_Value*
resolve : set *Conflict*

(c15) $(\forall$ g: *Goal*) g.prim_isa = Operational_Goal $\Rightarrow$ g.state $\neq \emptyset$
//If Goal *g* is an Operational Goal, then *g* must have a specified state, i.e. the environment in which *g* is fulfilled must be specified as a set of Objects.//

(c16) $(\forall$ g: *Goal*) g.prim_isa = Operational_Goal $\wedge \exists$ oset = {$ob_1,\ldots,ob_n$: *Object*} $\wedge$ g.state $\subseteq$ oset $\Rightarrow$ g.status = Fulfilled
//If there is a set of Objects *oset*, such that the state of Goal *g* is a subset of *oset*, then *g* is fulfilled.//

(c17) $(\forall$ g: *Goal*) g.sec_isa = Organizational_Goal $\Leftrightarrow$ g.org_isa $\neq \emptyset$
//If the Goal *g* that is an Organizational Goal, then *g* must be either a Requirement or an Expectation.//

(c18) $(\forall$ g: *Goal*; r: *Organizational_Role* ; act: *Actor*)
(g.sec_isa = Organizational_Goal $\wedge$ r $\in$ act.occupy $\wedge$ g $\in$ r.responsible $\wedge$ act.isa = Software_Agent) $\Rightarrow$ g.org_isa = Requirement
//An Organizational Goal *g* is a Requirement if there is some Software Agent Actor *act* which occupies the Organizational Role *r* which in turn is responsible for *g*.//

(c19) $(\forall$ g: *Goal*; r: *Organizational_Role* ; act: *Actor*)
(g.sec_isa = Organizational_Goal $\wedge$ r $\in$ act.occupy $\wedge$ g $\in$ r.responsible $\wedge$ act.isa = Human_Actor) $\Rightarrow$ g.org_isa = Expectation
//An Organizational Goal *g* is an Expectation, if there is a Human Actor *act* which occupies an Organizational Role *r* which in turn is responsible for *g*.//

(c20) $(\forall$ g: *Goal*) g.sec_isa $\neq$ Organizational_Goal $\Rightarrow$ g.resolve = $\emptyset$
//If Goal *g* is not an Organizational Goal, then *g* cannot resolve Conflicts.//

Fig. 6. Formal specification of the *Goal* concept

Operational Goals can only *contribute* (positively or negatively) to *Softgoals*. For example, "increase customer satisfaction" and "improve productivity of the workforce" are *Softgoals*.

Organizational Goal vs. Personal Goal. An *Organizational Goal* describes the state of the environment that should be achieved by cooperative and coordinated behaviour of *Actors*. An *Organizational Goal* is either a *Requirement* or an *Expectation* (c17). A *Requirement* is an *Organizational Goal* under the responsibility of an *Organizational Role occupied* by a *Software Agent* (c18). An *Expectation* is an *Organizational Goal* under the responsibility of an *Organizational Role occupied* by a *Human Actor* (c18). This distinction between a requirement of the information system and the expectation of its human users contributes to the successful accomplishment of a process that generally involves interaction among them. *Organizational Goals* can solve *Conflicts* (c20) by specifying the state of the environment in which the *Conflicts* cannot be true.

A *Personal Goal* describes the state of the environment that an *Actor pursues* individually (i.e. without cooperative and coordinated behaviour). It can require competitive behaviour with other *Actors*.

We distinguish what is expected from the participation of the *Actor* in the process (through the *Organizational Role* it *occupies*) from what the *Actor* expects from its participation in the process (*fulfilment* of or *contribution* to its *Personal Goals*). In reality, consistency between the *Organizational Goals* and *Personal Goals* is not necessarily ensured. Consequently, it is important to reason about *Conflicts* that may arise between *Personal* and *Organizational Goals*, as well as about the degree to which an *Organizational Goal* assists in the pursuit of *Personal Goals*.

Temporal Behaviour of Goals. A behavioural pattern is associated with each *Goal*. The possible patterns are: *achieve*, *cease*, *maintain* and *avoid* [6]. For example, organizations tend to *avoid* "conflict of interest" (*Softgoal*) and *achieve* "replenish stock" (*Operational Goal*). When we associate a pattern to a *Goal*, we restrict the possible behaviour of the *Actors* concerning the *Goal*: *achieve* and *cease* generate behaviour, whereas *maintain* and *avoid* restrict behaviour.

5 Related Works

Process-oriented approaches such as Activity Diagrams, DFDs, IDEF0, workflows (see e.g. [11, 15-17]) describe an enterprise's business processes as sets of activities. Strong emphasis is put on the activities that take place, the order of activity invocation, invocation conditions, activity synchronization and information flows. Among these approaches, workflows have received considerable attention in the literature. In such a process-oriented approaches, agents have been treated as a computational paradigm, with a focus on the design and implementation of agent systems, not on the analysis of enterprise models.

Actor-oriented approaches emphasize the analysis and specification of the role of the actors that participate in the process [18]. The *i** modelling framework [5] has been proposed for business process modelling and reengineering. Processes, in which information systems are used, are viewed as social systems populated by intentional actors that cooperate to achieve goals. The framework provides two types of dependency models: a strategic dependency model used for describing processes as networks of strategic dependencies among actors and the strategic rationale model used to describe each actor's reasoning in the process, as well as to explore alternative process structures. The diagrammatic notation of *i** is semi-formal and has proved useful in requirements elicitation (see e.g. [8, 19-20]). In this context, actor-oriented approaches provide significant advantages over other approaches: agents are autonomous, intentional, social etc. [4], which is of particular importance for the development of open distributed information systems in which change is ongoing. However, actors have served mostly as requirements engineering modelling constructs for real-world agents, without assuming the use of agent software as the implementation technology nor the use of organizational actors for enterprise modelling.

Goal-oriented approaches focus on goals that the information system or a business process should achieve. Frameworks like KAOS [6, 21] provides a formal specification language for requirements engineering, an elaboration method and meta-level knowledge used for guidance while the method is applied [22]. The KAOS specification language provides constructs for capturing the various types of concepts that appear during requirements elaboration. The elaboration method describes steps (i.e. goal elaboration, object capture, operation capture etc. [22]) that may be followed to systematically elaborate KAOS specifications. Finally, the meta-level knowledge provides domain-independent concepts that can be used for guidance and validation in the elaboration process.

Enterprise Knowledge Development (EKD) [18] is used primarily in modelling of business processes of an enterprise. Through goal-orientation, it advocates a closer alignment between intentional and operational aspects of the organization and links re-engineering efforts to strategic business objectives. EKD describes a business enterprise as a network of related business processes, which collaboratively realize business goals. This is achieved through several sub-models: an enterprise goal sub-model (expressing the causal structure of the enterprise), an enterprise process sub-model (representing the organizational and behavioural aspects of the enterprise) and an information system component sub-model (showing information system components that support the enterprise processes) [18]. Agents appear in the EKD methodology but without explicit treatment of their autonomy and sociality [4]. In KAOS, agents interact with each other non-intentionally, which reduces the benefits of using agents as modelling constructs.

6 Conclusion

Modelling the organizational and operational context within which a software system will eventually operate has been recognized as an important element of the engineering process (e.g. [1]). Such models are usually founded on primitive concepts such as those of *actor* and *goal*. Unfortunately, no specific enterprise modelling framework really exists for engineering modern corporate IS. This paper proposes an integrated agent-oriented metamodel for enterprise modelling. Moreover, our approach differs primarily in the fact that it is founded on ideas from in requirements engineering frameworks, management theory concepts found to be relevant for enterprise modelling and agent oriented software engineering.

We have only discussed here the concepts that we consider the most relevant at this stage of our research. Further classification of, for instance, goals is possible and can be introduced optionally into the metamodel. For example, goals could be classified into further goal categories such as Accuracy, Security and Performance. We also intend to define a strategy to guide enterprise modelling using our metamodel as well as to define a modelling tool à la Rational Rose to visually represent the concepts.

References

1. Castro J., Kolp M. and Mylopoulos J.: Towards Requirements-Driven Information Systems Engineering: The Tropos Project. In Information Systems (27), Elsevier, Amsterdam (2002)
2. Koubarakis M., Plexousakis D.: A formal framework for business process modelling and design. Information Systems 27 (2002). 299-319
3. Bernus P.: Enterprise models for enterprise architecture and ISO9000:2000. Annual Reviews in Control 27 (2003) 211–220
4. Yu E.: Agent-Oriented Modelling: Software Versus the World. Proceedings of the Agent-Oriented Software Engineering AOSE-2001 Workshop, Springer Verlag (2001)
5. Yu E.: Modelling Strategic Relationships for Process Reengineering. Ph.D. thesis, Dept. of Computer Science, University of Toronto (1994)
6. Dardenne A., van Lamsweerde A., Ficklas S.: Goal-directed requirements acquisition. Sci. Comput. Programming 20 (1993) 3-50
7. Brickley J.A., Smith C.W., Zimmerman J.L.: Managerial Economics and Organization Architecture. McGraw-Hill Irwin 2nd ed. (2001)
8. Faulkner S., Kolp M., Coyette A., Tung Do T.: Agent-Oriented Design of E-Commerce System Architecture. Proceedings of the 6th International Conference in Enterprise Information Systems Engineering, Porto (2004)
9. Simon H. A.: Rational Decision Making in Business Organizations. The American Economic Review, 69(4) (1979), 493-513
10. Johnson G., Scholes K.: Exploring Corporate Strategy, Text and Cases. Prentice Hall (2002)
11. Mentzas G., Halaris C., Kavadias S.: Modelling business processes with workflow systems: an evaluation of alternative approaches. International Journal of Information Management, 21 (2001) 123-135
12. Spivey J. M.: The Z Notation: A Reference Manual. 2nd Edition, Prentice Hall International (1992)
13. Bowen J.: Formal Specification and Documentation using Z: A Case Study Approach (1994)
14. Simon H.A.: Administrative Behavior : A Study of Decision-Making Processes in Administrative Organization. New York: The Free Press 3rd ed. (1976)
15. Kamath M., Dalal N.P., Chaugule A., Sivaraman E., Kolarik W.J.: A Review of Enterprise Process Modelling Techniques. In Prabhu V., Kumara S., Kamath M.: Scalable Enterprise Systems: An Introduction to Recent Advances. Kluwer Academic Publishers, Boston (2003)

16. Elmagarmid A., Du W.: Workflow Management: State of the Art Versus State of the Products. In Dogac A., Kalinichenko L., Tamer Ozsu M., Sheth A.: Workflow Management Systems and Interoperability. NATO ASI Series, Series F: Computer and Systems Sciences, 164, Springer Heidelberg (1998)
17. Sheth A.P., van der Aalst W., Arpinar I.B.: Processes Driving the Networked Economy. IEEE Concurrency, 7, (1999) 18-31
18. Kavakli V., Loucopoulos P.: Goal-Driven Business Process Analysis Application in Electricity Deregulation, Information Systems, 24 (1999) 187-207
19. Liu L., Yu E.: Designing information systems in social context: a goal and scenario modelling approach. Information Systems, 29 (2004) 187–203
20. Briand L., Melo W., Seaman C., Basili V.: Characterizing and Assessing a Large-Scale Software Maintenance Organization. In Procedings of the 17th International Conference on Software Engineering, Seattle, WA (1995)
21. van Lamsweerde A., Darimont R., Letier E.: Managing Conflicts in Goal-Oriented Requirements Engineering. IEEE Transactions on Software Engineering, Special Issue on Managing Inconsistency in Software Development (1998)
22. van Lamsweerde A.: The KAOS Metamodel –Ten Years After. Technical report, (2003).