

An Agent-Oriented Enterprise Model for Early Requirements Engineering

Ivan J. JURETA

*Information Management Research Unit
University of Namur, Rempart de la Vierge, 8
B-5000 Namur, Belgium
iju@info.fundp.ac.be*

Stéphane FAULKNER

*Information Management Research Unit
University of Namur, Rempart de la Vierge, 8
B-5000 Namur, Belgium
stephane.faulkner@fundp.ac.be*

Manuel KOLP

*ISYS Information Systems Research Unit,
University of Louvain, 1 Place des Doyens,
B-1348 Louvain-la-Neuve, Belgium
kolp@isys.ucl.ac.be*

Keywords: Enterprise Modeling, Agent Technology, Agent Software, IS Structure, IS Models, Requirements Definition

An Agent-Oriented Enterprise Model for Early Requirements Engineering

Abstract. This chapter introduces an agent-oriented enterprise model for conducting enterprise modeling during the early stages of information system requirements engineering. The enterprise model integrates a set of concepts and relationships that the analyst instantiates when building a model of the part of the organization in which the future information system will operate. The aim is to allow the analyst to produce an enterprise model which captures knowledge about an organization and its business processes, and which can be used to build an agent-oriented requirements specification of the future system and of its organizational environment. Compared to similar models, the present one integrates concepts and relationships allowing the analyst to capture the relevant intrinsic characteristics, such as autonomy and intentionality of human and software agents that are to participate in the future system.

INTRODUCTION

Business analysts and IT managers increasingly recognize that the ability to correctly and often extensively specify and analyze early requirements about an Information System (IS) is critical for gaining organizational acceptance of the future system and achieving a close match between the expected and observed quality thereof. Within the Requirements Engineering (RE) effort which initiates and subsequently guides the development and deployment of any IS within an organization, *early* RE is its first step, focusing on the representation and analysis of the organizational environment before the future system is introduced, dealing with the definition of desired behaviors and qualities of the future system that would fit this environment, and finally anticipating the effects that its introduction is likely to have on the performance of the organization. In order to analyze such organizational environments, it is necessary to understand the objectives, organizational processes, roles and interdependencies of different stakeholders. Although errors and misunderstandings at this level of requirements are frequent and costly, early RE is usually done informally.

In this chapter, we propose to address this issue by suggesting a precisely and formally defined enterprise model to facilitate the modeling and analysis of early requirements for IS. This enterprise model allows the representation of the structures, organizational processes, resources, actors, work roles, behaviors, goals and constraints of the organizational setting in which the future IS will function. It can be both descriptive and definitional, i.e., spanning what is and what should be. One of its key characteristics is its support for the agent software engineering paradigm which allows developers to handle the life cycle of complex, distributed and open systems required to offer open and dynamic capabilities in the latest generation of enterprise IS (see, e.g., Castro et al., 2002). By instantiating the concepts and relationships provided in the enterprise model, the analyst can:

- analyze the current organizational structure and business processes in order to reveal problems and opportunities;
- evaluate and compare alternative organizational processes and structures;
- achieve common understanding and agreement between stakeholders (e.g., managers, owners, workers, etc.) about different aspects of the organization;
- build a database along the structure of the enterprise model for use in collecting, managing, and reusing the knowledge available in the organization.

The proposed enterprise model draws on research in RE frameworks (e.g., Yu 1994; Dardenne et al., 1993), management theory found to be relevant for enterprise modeling (e.g., Simon 1976; Johnson & Scholes, 2002; Brickley et al., 2001; Uschold et al., 1997) and agent

oriented software engineering (e.g., Castro et al., 2002). It aims to reduce the semantic gap between enterprise and requirements representations, providing a conceptual foundation for modeling organizational IS. Through agent-orientation our proposal advances current research results towards an integrative approach to the representation of human and organizational issues found relevant to the RE of organizational IS, all in the aim of arriving at a better understanding of the setting in which the IS will be used.

The following section motivates the use of the agent paradigm to model and design IS. Section 3 gives an overview of related works. Section 4 introduces the enterprise model. In Sections 5 to 9, all elements of the enterprise model are defined and discussed, and, to increase precision, specified using the Z specification language. Section 10 summarizes the results and points to further work.

Agent-Orientation in Enterprise Models for Modern Organizational IS

The characteristics and expectations of new application areas for the enterprise such as electronic and mobile commerce, supply-chain management, peer-to-peer computing, or web services are deeply affecting IS engineering. Most of the IS designed for these application areas are now concurrent and distributed. They tend to be open and adaptable, in that they exist in a changing organizational and operational environment where new components can be added, modified, or removed at any time.

Given these new needs, many researchers (e.g., Yu, 1994; Castro et al. 2002; Zambonelli et al., 2003) have suggested and discussed novel paradigms that would enable more appropriate conceptualization, design and implementation systems that can operate efficiently and effectively in such circumstances. The paradigm of agent orientation is being increasingly applied when the aim is to design complex yet flexible organizational IS that adapt to the changing operating conditions of modern organizations. Such systems are composed of *agents*, i.e., open, modular, interoperable, and self-contained components, commonly characterized as intelligent, in that they may be autonomous, proactive, and exhibit some degree of learning and adaptation to operating environment conditions.

Agent orientation does not represent a radical departure from current software and IS engineering thinking; instead, legacy system can be incorporated in agent systems under limited cost. The cited benefits of agent orientation, along with the broad range of potential applications, lead to the conclusion that agent orientation does have the capacity to succeed as mainstream software and IS engineering paradigm.

It appears relevant to enrich concepts and relationships employed in enterprise modeling with those appropriate for representing information proper to an agent-oriented perspective. Following the established results in agent-oriented software engineering (e.g., Yu, 1994; 2001; Zambonelli et al., 2003; Jennings, 2000) modeling constructs are introduced to enable the representation of autonomy, intentionality, sociality, identity and boundary, of human and/or software agents. In the paragraph below, the terms in italics refer to concepts or relationships in the agent-oriented enterprise model proposed in the remainder of the chapter.

Actors are autonomous as their behavior 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 behavior, 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*

goals. 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 goals*. *Actors* apply *plans* according to the rationale described in terms of *personal goals*, *organizational goals*, and *capabilities*. The rationale of our *actors* is not perfect, but bounded (Simon, 1976; 1979), as they can act based on *beliefs* that are incomplete and/or inconsistent with reality.

RELATED WORK

The discussion is organized around five types of frameworks for enterprise modeling relevant to the results presented in this chapter. There is no clear distinction of enterprise modeling from requirements elicitation since their objective is similar: to improve the organization through the representation of knowledge about its main constituents, processes, purpose, etc. In addition, most frameworks that are clearly intended for RE involve the modeling of the organizational context in which the future IS will be implemented. The relevant modeling frameworks are distinguished on the basis of their main modeling concepts and their overall purpose. Methodological issues related to how the modeling elements are instantiated are not considered herein.

Activity-oriented Models

The various business process modeling techniques fall into activity-oriented modeling (for an overview, see, e.g., Kettinger et al., 1997).

Activity-based frameworks such Activity Diagrams, DFDs, and IDEF0 (see e.g., (Kamath et al. 2003, Elmagarmid et al., 1998, Mentzas et al. 2001, Sheth et al., 1999) describe 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. Workflows have received considerable attention in the literature (for an extensive overview, see, zur Muehlen, 2002).

In the generic Business Process Modeling (BPM) approach employed in (Kettinger et al., 1997) to evaluate a range of available BPM techniques, BPM proceeds through stages, starting, broadly, from the identification of a business process to change, process redesign, towards change management to move in reality to the new process structure, and the monitoring and evaluation of the performance of the new process. The variety of techniques that can be applied (e.g., brainstorming, visioning, force field analysis, Delphi technique, Business Systems Planning, Critical Success Factors, etc.– more than 50 techniques are cited in Kettinger et al., 1997) and their usually informal presentation makes it particularly difficult to elaborate on the comparison with the enterprise model introduced herein. It can, nevertheless, be argued that the enterprise model here has a double interest for the researchers and the practitioners of BPM. First, it indicates that establishing sound conceptual bases is necessary for ensuring that the methodology that can later be constructed is grounded in well understood foundations. Basing a methodology on informal grounds can only entail difficulties in its use for lack of clarity and precision is bound to increase the cost of applying it within realistic settings: different people will understand and use it differently, making cooperation difficult. Second, this paper in itself indicates how the definition of conceptual foundations can be performed.

In activity-oriented models, agents have been treated mostly as a computational paradigm, with focus on the design and implementation of agent systems. Compared to our enterprise model, they do not incorporate social metaphors for agent systems and there is a limited treatment of inconsistencies.

Ontology-driven Approaches to Enterprise Modeling

Ontological modeling is concerned with capturing the relevant entities of a domain in an ontology using an ontology specification language based on a small set of basic, domain-independent ontological categories (Guizzardi et al., 2002). The ontology is used to share common understanding of the structure of information among people or software agents, to enable reuse of domain knowledge, to analyze domain knowledge, and so on.

The aim of the TOVE project (Fox and Grüninger, 1997) is the creation of enterprise ontology for creating deductive enterprise models. Such enterprise model build onto generic enterprise models by adding axioms expressed in first-order logic and a deduction engine, so that the model integrates some degree of deductive capability. Deductive enterprise models can be used for extracting information about the organization using “common-sense” queries that require limited deduction, in order to support functions such as accounting, forecasting, etc.

The Enterprise Ontology (Uschold et al. 1997) defines the ontology used in the Enterprise Project (Uschold and Grüninger, 1996; Stader, 1996). The overall aim of the project is to improve and where necessary replace existing modeling methods with a framework for integrating methods and tools which are appropriate to enterprise modeling and the management of change. The project resulted in a toolset which is used to modeling processes, supporting agent development, matching agents with process tasks, and communicating among people and software.

The main difference of our framework from both Tove and the Enterprise Project is that they are not oriented towards RE. Tove intends to create a computable model of the organization by using axioms defined in first-order logic and a deduction engine. Our aim is closer to engineering software than creating computable enterprise models. Tove does not treat potential inconsistencies in processes, has limited support for process modeling (Koubarakis and Plexousakis, 2002), and does not treat the strategic dimension of agent interactions (i.e. it does not integrate dependency relationships). Inconsistencies are not treated in the Enterprise Ontology. In addition, our model provides more specific specialization of goals which is particularly useful in the context of open systems with self-interested agents. Even though the aim of our model is different to some extent, these frameworks have served as a source of inspiration for providing clear and unambiguous definitions of the terminology used in our enterprise model.

Goal-driven Modeling

Goal-based modeling focuses on goals that the IS should achieve within an organization. The concept of goal has been argued easy to understand and can be used at different activities of the RE process (Kavakli, 1999).

The KAOS framework for RE (Dardenne et al., 1993) provides a specification language, an elaboration method, and meta-level knowledge used for guidance while the method is applied (van Lamsweerde et al., 1998). 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.) 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) (Kavakli and Loucopoulos, 1999) is used primarily in modeling 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 realise business goals.

Other goal-based approaches have also been proposed. (Kavakli, 1999) provides an extensive discussion of a number of other approaches.

Actors appear in EKD without explicit treatment of their autonomy and sociality (Yu, 2001). In KAOS, actors (i.e. agents) interact with each other non-intentionally, which reduces the benefits of using them as modeling constructs. They are considered as specialization of objects. In our approach, actors are considered clearly as autonomous and social entities which exhibit intentional behavior. As noted in (Koubarakis and Plexousakis, 2002), an important limitation of EKD is the lack of formal support for its conceptual models meaning that formal specification of the IS cannot be derived without considerable additional effort. While KAOS relies on an elaborated formalism, it does not integrate social concepts for multi-agent systems, such as organizational role, group, organization, etc. Consequently, it does not benefit from the advantages of such concepts. Our framework integrates such social metaphors since they aid in dealing with system complexity. They are also easy to understand by developers and users and stakeholders, and help to reduce the concept distance between the systems in the “real world” and models that are developed on the basis of these concepts (Mao and Yu, 2004).

Strategy maps proposed in management literature (Kaplan and Norton, 2000; 2004) can also be categorized as goal-driven modeling: a strategy map is a diagram built to indicate how an organization creates value. Strategic objectives are related through cause and effect relationships with each other, and this within the four balanced scorecard perspectives (i.e., financial, customer, internal, and learning and growth perspective). Compared to the enterprise model proposed herein, a strategy map can be argued to feature only goals and cause and effect relationships as modeling primitives. As such, it is of very limited use in the RE of organizational IS. The enterprise model herein does not integrate causal relationships per se, but the contribution relationships. This kind of relationships is weaker, but more realistic knowing the complexity of organizational environments and the difficulty in precisely understanding causal links therein.

Role-driven Modeling

Roles seem to be a suitable concept for the development of agent-oriented IS, in particular for the engineering of interactions between agents (Cabri et al., 2004). Roles are used to abstractly model the participants of the organization, without considering their specific characteristics. Roles promote the organizational view of the IS, which helps in understanding the main functions of the system and can be translated in an agent-based design model.

Gaia (Zambonelli et al., 2003) is a methodology for agent-oriented analysis and design. The Gaia process consists in orderly constructing a series of models aimed at describing both the macro (societal) aspects and the micro (intra-agent) aspects of a multi-agent system, generally conceived as an organized society of individuals (i.e., a computational organization of autonomous entities) (Cernuzzi et al., 2004). Roles are specified in terms of permissions (which express the resources available to it, i.e. what a role can or cannot use), responsibilities (which specify the expected behavior of the role) and protocols and activities (which specify interactions involving the role). Gaia integrates the concepts of organizational rule, which is

used to specify the responsibilities of the entire agent organization. Organizational structure of the IS is defined through a role model that defines the topology of interaction patterns and the control regime of organization activities (Zambonelli et al., 2003).

A number of other role-based approaches have been proposed, such as Aalaadin, Kendall, RoleEP, TRUCE and ROPE. A comprehensive overview and critique of these is provided in (Cabri et al., 2004; Ferber and Gutknecht, 1998; Ferber et al., 2003).

Compared to Gaia, our model integrates the possibility of competitive behavior that may lead to conflicts, resulting from the pursuit of actor's personal goals. In a recent paper, (Cabri et al., 2004) affirm the need for treating roles as concepts in order to fully exploit the advantages of the concept at analysis, design and implementation phases. In general, when compared to Gaia, Aalaadin, Kendall, RoleEP, TRUCE and ROPE, our model is conceptually closer to users, clients and domain experts, while remaining usable to software architects, software developers, and development tools.

Agent-driven Modeling

Most of the discussed frameworks in this section incorporate the concept of agent. However, a truly agent-based framework must consider an agent as its basic concept, both in the requirements elicitation and subsequent steps of the requirements and software engineering process. Currently there seems to be only *i** (Yu, 1994; 2001) and Tropos which are truly agent-based. The *i** modeling framework (Yu, 1994) has been proposed for business process modeling and reengineering. Processes, in which IS are used, are viewed as social systems populated by intentional actors which 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. In this context, agent-based approaches provide significant advantages: agents are autonomous, intentional, social, etc. (Yu, 1994) which is of particular importance for the development of open distributed IS in which change is ongoing.

Tropos (Castro et al., 2002) uses *i** in its early requirements phase of the software development process. One of the significant differences between Tropos and the other methodologies (such as e.g., Gaia) is its strong focus on early requirements analysis where the domain stakeholders and their intentions are identified and analysed.

Our enterprise model makes it possible to combine the strengths of *i** notably in terms of strategic dependency analysis among the process' organizational roles, with the analysis of the realization of the process as a series actions. *i** includes neither concepts such as group or organization, nor conflicts, which limits its expressivity. As Tropos uses *i** during early requirements, these same limitations are present.

OVERVIEW OF THE ENTERPRISE MODEL

The proposed enterprise model is built with a set of primitives common to business management (that are comprehensible to clients, users, domain experts) and system development (that are comprehensible to system analysts and developers). The set of primitives and their relationships are represented in Fig.1 using the UML (Bennett et al., 2002) notation. The rest of this section overviews the enterprise model from the point of view of business managers (the management perspective) and of systems developers (the information-system perspective). It also presents and discusses the different kinds of primitive used to describe the enterprise model.

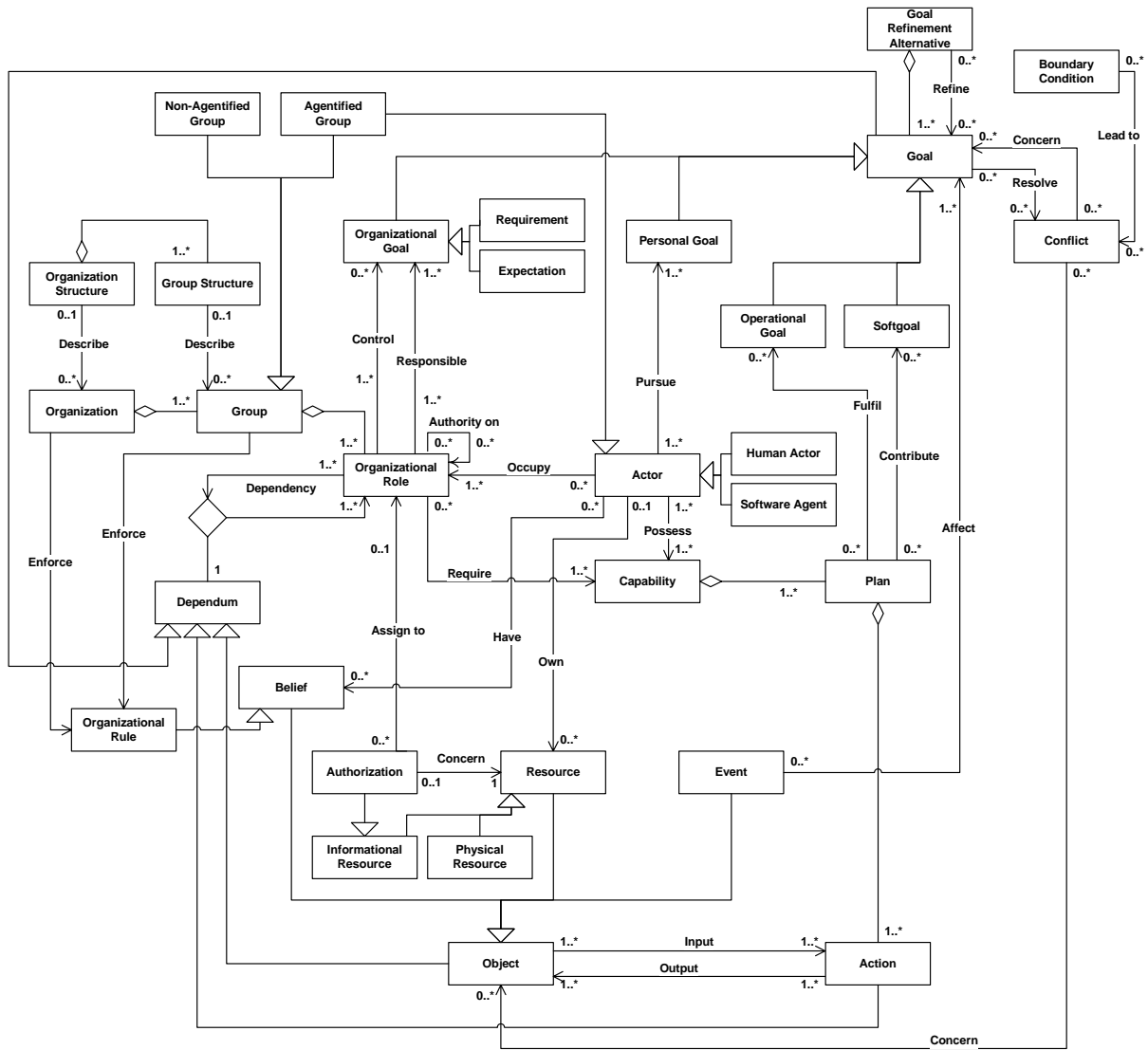


Fig.1: The Agent-Oriented Enterprise Model.

The Management Perspective

Our enterprise model provides common concepts used to describe strategic, behavioral, and informational aspects of an organization. *Goals* represent the purposes of the organization and of its processes. They are the responsibility of *Organizational Roles* which are defined as *Groups* formed in the *Organization* for structural and/or functional reasons (i.e. to establish a clear hierarchy and/or to accomplish specific, non-recurrent business projects). *Organizational* and *Group Structure* describe the internal structure of the *Organization* and of its *Groups* in terms of strategic *Dependencies* that influence the interactions among their members. *Organizational Rules* are enforced at the organizational and group levels, defining policies and constraining the behavior of *Actors* so that the expression of their self-interest (expressed with *Personal Goals*) does not limit the performance of the organization. *Actors* occupy *Organizational Roles* if they possess the required *Capabilities*. Actors discharge the responsibilities assigned through organizational roles by executing *Actions*. Actions accomplished sequentially or in parallel, specify how processes are structured. *Events* affect the

goals of the organization making it responsive to changes in its internal or external environment.

The Information-System Perspective

Our enterprise model also provides widely-used concepts for specifying the architecture of agent-oriented IS. *Software Agents* are the components of the IS. They act according to their *Beliefs*, *Goals* and *Capabilities*. *Beliefs* represent their information about the environment in which they exist. *Goals* determine the desired states of the environment. *Capabilities*, *Plans* and *Actions* represent the intentional state of an agent, i.e. means for satisfying its external and internal stimuli. *Objects* are non-intentional entities that are manipulated by agents and/or that influence their behavior, and that are significant for the organization. *Organizational Roles* provide the building blocks for agent social systems and the requirements by which agents interact. Each agent is related to other agents by the roles that it occupies and according to the responsibilities that these roles assume. When related organizational roles are assembled, agent *Groups* can be created, according, e.g., to a specific and reusable (patterned) group structure. Groups can be characterized as single agents (i.e., “agentified”) to benefit from synergies in capabilities of its member agents. An *Organization* of agents can be seen as a framework which is open and dynamic, integrating cooperative and self-interested agents (i.e. those pursuing also *Personal Goals*). The *Group* and *Organization* concepts aid in increasing the modularity of the system and in managing its complexity. *Organizational Rules* are applicable to agent organizations and govern the running of the whole multi-agent system, expressing how the organization is expected to work.

Kinds of Primitives

The enterprise model uses different kinds of primitives: concepts (*Goal*, *Actor*, *Object*, etc.), relationships (*possess*, *require*, *pursue*, etc.), attributes (*Power*, *Interest*, *Motivation*, etc.), and constraints (e.g., “*an actor occupies a position if and only if that actor possesses all the capabilities required to occupy it*”).

Concepts represent abstractions of real-world entities significant to model the “inner“ and “outer” aspects of an IS. Relationships define interaction among concepts. Attributes describe significant properties of concepts. Finally, constraints impose restrictions upon concepts, relationships and attributes.

For consistency, any primitive of the enterprise model has two mandatory attributes:

- *Name*, which allows unambiguous reference to the instance of the concept;
- *Description*, which is a precise and unambiguous description of the corresponding instance of the concept. The description should contain sufficient information and be precise so that, if required, a formal specification can be derived for use in requirements specifications for a future IS.

Fig.1 shows only concepts and relationships. Attributes and constraints are specified using the Z state-based specification language (Spivey, 1992; Bowen, 1996). We use Z as 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 (Faulkner, 2004; Bowen, 1996). In addition, it is widely accepted in the software development industry and has been used in large-scale projects.

In the following sections, we provide definitions of primitives, and discuss their relevance for enterprise modeling and early RE. For clarity, we have subdivided the enterprise model into five sub-models:

- *Organizational sub-model*, describing the organization in terms of the actors, their organizational roles, groups that they constitute, and their 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*, representing 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.

ORGANIZATIONAL SUB-MODEL

The Organizational Sub-Model is presented in Fig. 2. It is used to specify the *Groups* in which *Actors* coexist and interact, by *occupying* diverse *Organizational Roles*. Each *Actor* occupies one or more *Organizational Roles* according to the *Capabilities* that are *required* by these roles, and that the *Actor possesses*. The following sub-sections provide details on each of the elements of the sub-model.

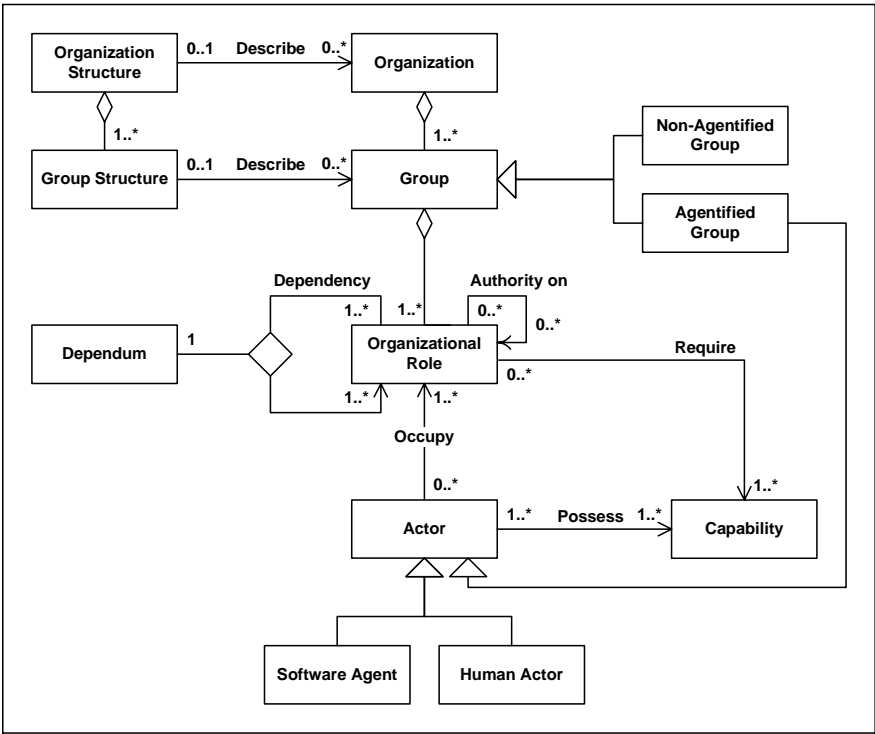


Fig. 2. Organizational Sub-Model.

Actor

Fig. 3 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 *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 schemas. A schema groups a collection of related declarations and predicates into a separate namespace or scope. The schema in Fig. 3 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 of the schema 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.

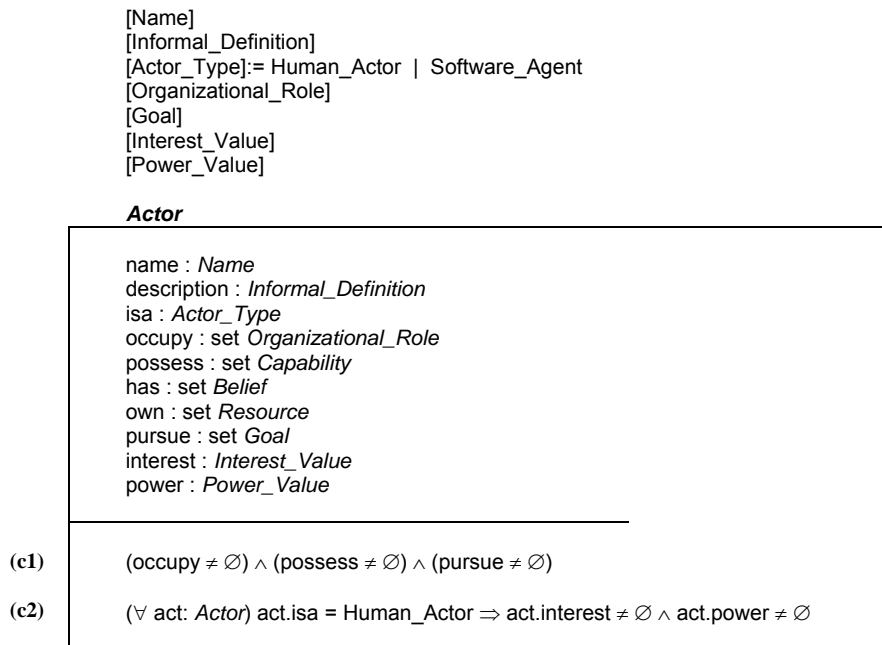


Fig. 3. Formal specification of the *Actor* concept.

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, i.e., that have an influence on its resources, its goals, etc. A *Software Agent* represents a software component of the multi-agent IS for which the requirements are being specified.

Whenever an *Actor* enters the system, its *Capabilities* must be known (c1) so that it can *occupy* at least one *Organizational Role*. There can be no *Actor* in the system that does not *occupy* at least one role, since the *Organizational Role* defines the purpose of the *Actor* in the *Organization*, through the responsibilities it carries.

A *Human Actor* is characterized with two specific attributes (c2): *Interest* and *Power* (Johnson & Scholes, 2002). *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 *Capa-*

bilities. For instance, when automating a business process, the values of *Interest* and *Power* 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 benefited 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 the priority of already specified *Goals*. *Interest* and *Power* help to find *Human Actors* that will play a crucial role in the software-to-be.

Organizational Role

An *Organizational Role* is an abstract characterization of expected behavior 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*.

The concept of *Organizational Role* is important to abstractly model the agents in an agent system and helpful to manage system complexity without considering the concrete details of agents. Roles enable the separation between different functionalities of software agents (e.g., mobility from collaboration), or between different phases of the development process (e.g., functions in the design from methods in the implementation). Fig. 4 shows the Z formal specification of the *Organizational Role* concept.

[Goal_Control_Status]:= Fulfilled | 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) $(\forall \text{act: Actor; } r: \text{Organizational_Role}) r \in \text{act.occupy} \Rightarrow r.\text{require} \subset \text{act.possess}$
//An Actor *act* that occupies the Organizational Role *r* possesses the Capabilities required by the Organizational Role *r*./
- (c5) $(\forall \text{act: Actor; } r: \text{Organizational_Role})$
 $\text{act.has} \subset r.\text{leave_condition} \Rightarrow r \notin \text{act.occupy}$
//If the Leave Condition is true, then the Actor *act* no longer occupies the Organizational Role *r*./
- (c6) $(\forall r: \text{Organizational_Role; } g: \text{Goal})$
 $g \in r.\text{responsible} \Rightarrow g.\text{sec_isa} = \text{Organizational_Goal}$
//If Organizational Role *r* is responsible of Goal *g*, then *g* is an Organizational Goal./
- (c7) $(\forall r: \text{Organizational_Role; } g: \text{Goal})$
 $(g.\text{prim_isa} = \text{Operational_Goal} \wedge g.\text{sec_isa} = \text{Organizational_Goal} \wedge$
 $g \in r.\text{control} \wedge g.\text{status} = \text{Fulfilled})$
 $\Rightarrow (\exists b!: \text{Belief}) (g.\text{status} = \text{Fulfilled}) \in b.\text{term} \wedge (g, \text{Fulfilled}) \in r.\text{control}$
//If an Organizational Operational Goal *g* is fulfilled, then the Organizational Role *r* which controls the fulfillment of *g* outputs a new Belief *b* which indicates that the Goal *g* has been fulfilled./
- (c8) $(\forall r_1, r_2: \text{Organizational_Role; } g: \text{Goal; } a_1, a_2: \text{Actor}) (g.\text{sec_isa} =$
 $\text{Organizational_Goal} \wedge g \in r_1.\text{responsible} \wedge g \in r_2.\text{control} \wedge r_1 \neq r_2 \wedge r_1 \in$
 $\text{act.occupy} \wedge r_2 \in \text{act.occupy}) \Rightarrow a_1 \neq a_2$
//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

Fig. 4. Formal specification of the *Organizational Role* concept

Each *Organizational Role* requires a set of *Capabilities* to fulfill 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)ⁱ. 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 *fulfillment*. In case an *Organizational Goal* has been fulfilled, the *Actor*, occupying the *Organizational Role* which *controls* that *Goal*, executes a *Plan* in which an *Action* outputs a new *Belief* to mark the goal fulfillment (c7). This control procedure requires that a single *Actor* can never occupy distinct *Organizational Roles* that are *responsible* of and *control* the fulfillment of the *Organizational Goal* (c8).

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.

Capability

A *Capability* specifies the behaviors 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. 5 shows that a *Capability* can be structured as a set of *Plans* and/or other *Capabilities*. This increases system modularity as 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* *possess*. 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 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 ≠ ∅

(c10) (∀ cap: Capability)
∃ act: Actor ; cap ∈ act.possess ⇒ cap.availability = available
//If there is some Actor *act* that possesses Capability *cap*, then *cap* is available.//

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

Group and Group Structure

A *Group* is an aggregation of *Organizational Roles*. Each *Organizational Role* is part of one or more groups. In its most basic form, the *Group* is only a way to tag a set of *Roles*. In a more developed form, in conjunction with the *Actor* definition, groups can be used for partitioning the organization and organizing actors with some common goals together.

Groups are formed or dissolved according to the states of the environment described in their *Formation Condition* and *Dissolution Condition* attributes. For example, a group can be formed if some specific goal has not yet been achieved, and may be dissolved as soon as the goal has been achieved. Whenever a *Group* is formed, a series of *Organizational Roles* will be added to it. The *Access Condition* specifies diverse criteria for granting access to an *Organizational Role* to a *Group*. For example, an *Organizational Role* can be granted access to the *Group* if it possesses the *Capabilities* required and not available to the *Group*. Another example is the constitution of a *Group* containing only similar *Organizational Roles*.

```
[Group_Structure]
[Group_Type]= Agentified_Group | Non-Agentified_Group
[Organizational_Rule]
```

Group

```
name : Name
description : Informal_Definition
isa : Group_Type
composed_of : set Organizational_Roles
formation_condition : set Belief
access_condition : set {Capability, Belief}
dissolution_condition : set Belief
described_by : Group_Structure
enforce : set Organizational_Rule
```

- (c11) $(\text{composed_of} \neq \emptyset) \wedge (\text{access_condition} \neq \emptyset)$
- (c12) $(\forall \text{act} : \text{Actor} ; r : \text{Organizational_Role} ; gr : \text{Group} ; sb : \text{Organizational_Rule})$
 $r \in \text{act.occupy} \wedge r \in \text{gr.composed_of} \wedge sb \in \text{gr.enforce} \Rightarrow sb \in \text{act.has}$
//If an Actor act occupies an Organizational Role r which composes the Group gr and if Organizational Rule sb is enforced by gr, then act has sb.//
- (c13) $\exists r : \text{Organizational_Role} \wedge (\neg \exists \text{act} : \text{Actor} | r \in \text{act.occupy}) \wedge$
 $(\exists \text{act}_1, \dots, \text{act}_n : \text{Actor} | r.\text{require} \subset \text{act}_1.\text{possess} \cup \dots \cup \text{act}_n.\text{possess})$
 $\Rightarrow \exists gr! : \text{Group} (r_1!, \dots, r_n! : \text{Organizational_Role} \wedge r_1 \cup \text{act}_1.\text{occupy} \wedge \dots \wedge r_n$
 $\cup \text{act}_n.\text{occupy}) \wedge \text{gr.composed_of} = \{r_1, \dots, r_n\} \wedge \text{agentify_group}(gr)$
//If there is no single Actor act which can occupy the Organizational Role r, and if there is a set of Actors act₁, ..., act_n which together do possess the capabilities required to occupy r, then a Group gr is formed, as composed of this set of Organizational Roles r₁, ..., r_n, attributed to Actors act₁, ..., act_n. Group gr is agentified.//

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

Whenever an *Actor* becomes a member of a *Group*, it must conform to the constraints for the *Group*. The *Group* thus enforces a set of *Organizational Rules* upon every *Actor* that is its member (c12).

The constraint (c13) is used for the agentification of a group. Its use will be clarified in the next sub-section.

Agentified and Non-Agentified Groups

An *Agentified Group* possesses all the features that any *Actor* might have (Odell et al., 2004). Such a group can be considered by other *Actors* in the organization as a single *Actor*. This pushes further the modularity and encapsulation principles in the organization. It facilitates interaction between groups and between a group and individual actors, by establishing standard interaction points for each group.

An *Agentified Group* can occupy an *Organizational Role*, and have all features that are associated with it. This makes it possible to combine the individual capabilities of an actor in order to constitute more complex actors which can occupy highly important roles in the organization. Fig. 7 is an axiomatic description of the *agentify_group* function, used to agentify a group. It indicates that the *Actor* formed by agentifying a *Group* possesses all the *Capabilities* that the individual *Actors* composing the *Group* possess (c14). This function has been used in the constraint (c13) to agentify a group formed in order to fulfil responsibilities associated with an organizational role still unoccupied by a single actor can be fulfilled.

(c14)	<pre> agentify_group : Group → Actor ----- (∀ i = 1,...,n act_i: Actor ; j = 1,...,m r_j: Organizational_Role ; gr : Group) ∃ r_j ∈ act_i.occupy ∧ gr.composed_of = {r₁,...,r_m} agentify_group(gr) = (gr.isa = Agentified_Group ∧ actr! actr.possess = act₁.possess ∪ ... ∪ act_n.possess) //A Group gr is agentified into an Actor actr that possesses all the Capabilities that individual Actors composing the Group gr possess.// </pre>
-------	---

Fig. 7. Axiomatic description of the *agentify_group* function

A *Non-Agentified Group* is any group that is formed for purposes such as intra-group synergies or to partition the organization. When an actor wishes to interact with such group, it must interact directly with one of its members, i.e. actors which occupy organizational roles that compose the group.

Group Structure

A *Group* can be established according to a specific *Group Structure*. The *Group Structure* is the abstract description of a *Group* in terms of *Organizational Roles* that compose it and *Dependencies* that exist among these *Organizational Roles*. Its aim is to define generic structures that can be reused in the formation of groups.

The *Composed Of* attribute in Fig. 8 explicitly identifies the set of roles that compose the *Group Structure*. On the basis of identified *Organizational Roles*, the *Dependencies* among them can be derived. They define the *Internal Structure* of the group (c16). A *Dependency* among *Organizational Roles* exists when some of these roles depend on the other ones to provide a *Dependum*, which can be an *Object*, a *Goal* or an *Action*. The *Dependum* concept is further discussed in Section 4.6.

In the simplest case, the *Group Structure* will specify only the *Organizational Roles* which compose the group (c15). That means that a *Group Structure* might be instantiated in a partial form in the actual moment: group dynamics might imply that not all roles defined in the group structure will be present at a given moment.

[Dependum]
 [Dependency]:=
 (set Organizational_Role) × Dependum × (set Organizational_Role)

Group Structure

name : *Name*
 description : *Informal_Definition*
 composed_of : set *Organizational_Role*
 internal_structure : set *Dependency*

(c15) composed_of ≠ ∅

(c16) (∀ gs: *Group_Structure* ; d: *Dependency* ; r₁, r₂: *Organizational_Role* ∧ r₁ ≠ r₂ ∧
 dpd: *Dependum*) {r₁, r₂} ∈ d ∧ (d ≡ r₁ × dpd × r₂) ∧ {r₁, r₂} ∈ gs.composed_of
 ⇒ d ∈ gs.internal_structure
 //If Organizational Role r₁ depends on r₂ and if they both compose the Goal Structure
 gs, then the Dependency d among these Organizational Roles defines the Internal
 Structure of the Group Structure.//

Fig. 8. Formal specification of the *Group Structure* concept

Organization and Organization Structure

An *Organization* is a collection of *Groups* that have certain relationships to one another and take part in systematic institutionalized patterns of interactions with other *Groups*. This concept specifies the macro organization information needed to describe the *Organization structure* and the *Organizational Rules* that must be satisfied by all *Groups* in the organization. The main differences between *Group* and *Organization* concepts are:

- An *Organization* cannot be agentified;
- An *Organization* cannot be part of another *Organization*. An *Agentified Group* can be part of another *Group*, by playing some *Organizational Role* in that other *Group*.

Relationships among *Groups* that compose an *Organization* are determined by *Dependency* relationships among their component *Groups*. In order to enable open and dynamic systems, it is not necessary to specify a priori the *Dependencies* in a *Group*. However, this is necessary when defining *Organizational Structures* as discussed below.

[Organizational_Structure]

Organization

name : *Name*
 description : *Informal_Definition*
 composed_of : set *Groups*
 access_condition : set {*Capability*, *Belief*}
 described_by : *Organizational_Structure*
 enforce : set *Organizational_Rule*

(c17) (composed_of ≠ ∅) ∧ (access_condition ≠ ∅)

(c18) (∀ gr : *Group* ; org: *Organization* ; sb: *Organizational_Rule*)
 gr ∈ org.composed_of ∧ sb ∈ org.enforce ⇒ sb ∈ gr.enforce
 //A Organizational Rule sb that is enforced in the Organization org is enforced in every
 Group gr that compose org.//

Fig. 9. Formal specification of the *Organization* concept

Actors that enter the *Organization* must have adequate *Capabilities* and *Beliefs*, as specified in the *Access Condition* attribute in Fig. 9. For example, it can be used to forbid entry to actors that have beliefs which are contradictory to some organizational goals in order to avoid conflicts in the organization.

As for *Groups*, an *Organization* can be an instance of a specific *Organizational Structure*. The *Organizational Structure* provides an abstract description of the architecture of an *Organization*. As discussed above, the structure of an *Organization* can be entirely specified as a set of *Groups* and inter-group *Dependencies*, as shown in Fig. 10.

Further analogy with *Group Structure* can be made as the intention with *Organizational Structure* is to use it to define generic reusable organizational structures. In this respect, work on organizational patterns (Kolp et al., 2003) in the context of TROPOS can be reused here.

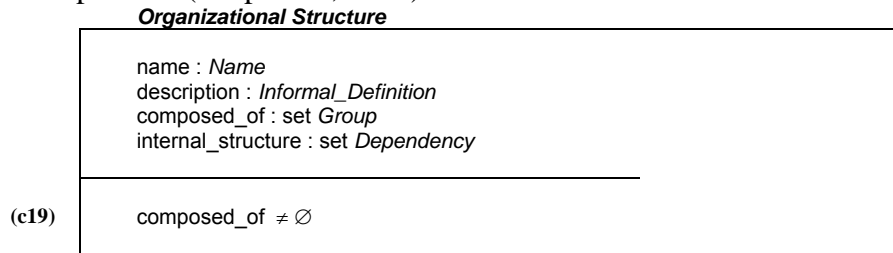


Fig. 10. Formal specification of the *Organizational Structure* concept

Dependum and Dependency

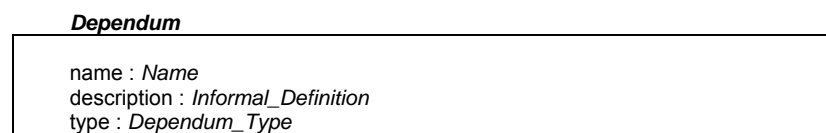
It has been widely accepted (see the literature on i^* , e.g., (Yu, 1994; Liu & Yu, 2004; Yu, 2001) that the representation of dependencies among members of an organization makes it possible to provide a better understanding of its strategic and social dimensions. Indeed, members of the organization are involved in numerous interactions: as they have limited capabilities and limited access to resources, they will depend on one another in order to realize their responsibilities.

The *Dependency* relationship was defined in Fig. 8 as involving *Organizational Roles* that depend on other *Organizational Roles* for a *Dependum*. A *Dependum* can be an *Organizational Goal*, an *Object*, or an *Action*. In a *Dependency*, the *Organizational Role* that expects the *Dependum* is called the depender, While the *Organizational Role* that is expected to supply the *Dependum* is called the dependee (c21). We define the following dependency types:

- *Organizational-Goal* dependency: the depender *depends* on the dependee to *fulfill* and/or *contribute* to an *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*.

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 (c22).

[Dependum_Type]:= Organizational_Goal | Object | Action



	depender : set <i>Organizational_Role</i> Dependee : set <i>Organizational_Role</i>
(c20)	$(type \neq \emptyset) \wedge (depender \neq \emptyset) \wedge (dependee \neq \emptyset)$
(c21)	$(\forall d: Dependency; dpd: Dependum; r_1, r_2: Organizational_Role)$ $r_1 \neq r_2 \wedge (d \equiv r_1 \times dpd \times r_2) \Rightarrow (depender = r_2 \wedge dependee = r_1)$
(c22)	$(\forall d: Dependency; dpd: Dependum; r_1, r_2: Organizational_Role)$ $r_1 \neq r_2 \wedge$ $(d \equiv r_1 \times dpd \times r_2) \wedge (dpd.type = Authorization) \Rightarrow r_1 \in r_2.authority_on$ <i>//If the Dependum is an Authorization, then Dependee r_2 has authority on Depender r_1.//</i>
(c23)	$(\forall obj: Object; a_1, a_2: Actor; cap_1, cap_2: Capability; pl_1, pl_2: Plan; actn_1, actn_2:$ <i>Action; $r_1, r_2: Organizational_Role$)</i> $(a_1 \neq a_2 \wedge cap_1 \neq cap_2 \wedge pl_1 \neq pl_2 \wedge actn_1 \neq actn_2 \wedge$ $(actn_1 \in pl_1.composed_of \wedge pl_1 \in cap_1.composed_of \wedge cap_1 \in a_1.possess) \wedge$ $(actn_2 \in pl_2.composed_of \wedge pl_2 \in cap_2.composed_of \wedge cap_2 \in a_2.possess) \wedge$ $obj \in actn_1.postcondition \wedge obj \in actn_2.input \wedge r_1 \in a_1.occupy \wedge r_2 \in a_2.occupy \wedge$ $\{r_1, r_2\} \neq \{a_1.occupy \cap a_2.occupy\}) \Leftrightarrow (\exists dm: Dependum \wedge dm.type = Object \wedge$ $dm.name = obj.name \wedge dm.depender = r_2 \wedge dm.dependee = r_1)$ <i>//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.//</i>

Fig. 11. Formal specification of the *Dependum* concept

The constraint (c23) in Fig. 11 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 (c23) 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.
- 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 analyzed for vulnerabilities and alternative process structures can be evaluated.

Constraint (c23) makes it possible to combine the strengths of the *i** dependency representation 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.

GOALS SUB-MODEL

The aim of the Goals sub-model illustrated in Fig. 12 is to explicitly show the purpose of the process that is being modeled. It allows to describe the actual or desired state of the process in terms of the goals that it should satisfy. The process is then analyzed according to its performance in the fulfillment and contribution of goals, and modified so that its performance is improved.

- Goals have been recognized as an essential component of the RE process (see van Lamswerde, 2001 for an overview). They are used in a number of RE frameworks: e.g., *i** (Yu, 1994), KAOS (Dardenne et al., 1993), NFR (Chung et al., 2000). In man-

agement, their importance has been recognized in frameworks such as Management By Objectives, the Balanced Scorecard (Kaplan & Norton, 2000) and Enterprise Knowledge Development (Kavakli & Loucopoulos, 1999).

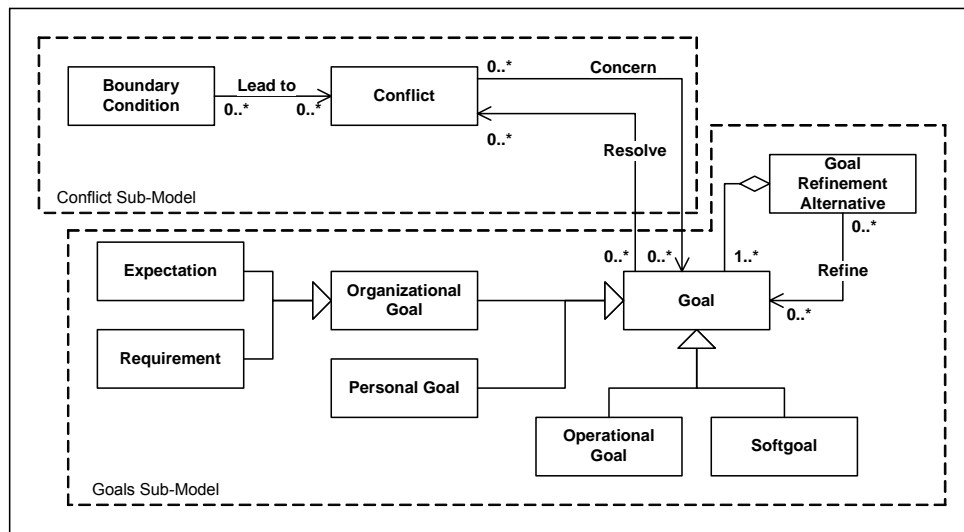


Fig. 12. Goals and Conflicts Sub-Models

Goal

A *Goal* describes a desired or undesired state of the environment. The environment is the context where 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 (van Lamsweerde, 2000), 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 behavior 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. 13). *State* describes the state of the environment in which the *Operational Goal* is fulfilled (c25). *Status* indicates whether the *State* of the *Operational Goal* has been achieved, i.e., whether the *Goal* has been fulfilled or not (c26).

A *Softgoal* also describes a desired or undesired state of environment, but its fulfil criteria (i.e., how achieve the desired state) may not be formally specified. A consequence of this is that *Plans* that are otherwise applied to *fulfil Operational Goals* can only *contribute* (positively or negatively) to *Softgoals*. For example, “increase customer satisfaction”, “implement a flexible IS”, “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 behavior of *Actors*. An *Organizational Goal* is either a *Requirement* or an *Expectation* (c27). A *Requirement* is an *Organizational Goal* under the responsibility of an *Organizational Role* occupied by a *Software Agent* (c28). An *Expectation* is an *Organizational Goal* under the responsibility of an *Organizational Role* occupied by a *Human Actor* (c29). This distinction between a requirement of the IS 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* (c30) 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 behavior). It can require competitive behavior 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*. We use *fulfil* and *contribute* relationships to show how *Plans* fulfil and contribute to both *Personal Goals* that the *Actor* pursues and *Organizational Goals* for which its *Organizational Roles* are responsible.

Temporal Behavior of Goals

A behavioral pattern is associated to each *Goal*. The possible patterns are: *achieve*, *cease*, *maintain*, and *avoid* (Dardenne et al. 1993). 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 behavior of the *Actors* concerning the *Goal*: *achieve* and *cease* generate behavior, *maintain* and *avoid* restrict behavior.

```
[Primary_Goal_Type]:= Operational_Goal | Softgoal
[Secondary_Goal_Type]:= Organizational_Goal | Personal_Goal
[Org_Goal_Type]:= Requirement | Expectation
[Goal_Pattern]:= Achieve | Cease | Maintain | Avoid
[Object]:= Resource | Authorization | Belief | Event
[Goal_Status]:= Fulfilled | Unfulfilled
[Refinement_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
```

(c24)	$(\text{prim_isa} \neq \emptyset) \wedge (\text{sec_isa} \neq \emptyset) \wedge (\text{pattern} \neq \emptyset)$
(c25)	$(\forall g: \text{Goal}) g.\text{prim_isa} = \text{Operational_Goal} \Rightarrow g.\text{state} \neq \emptyset$ //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.//
(c26)	$(\forall g: \text{Goal}) g.\text{prim_isa} = \text{Operational_Goal} \wedge \exists \text{oset} = \{\text{ob}_1, \dots, \text{ob}_n : \text{Object}\} \wedge g.\text{state} \subseteq \text{oset} \Rightarrow g.\text{status} = \text{Fulfilled}$ //If there is a set of Objects oset , such that the state of Goal g is a subset of oset , then g is fulfilled.//
(c27)	$(\forall g: \text{Goal}) g.\text{sec_isa} = \text{Organizational_Goal} \Leftrightarrow g.\text{org_isa} \neq \emptyset$ //If the Goal g that is an Organizational Goal, then g must be either a Requirement or an Expectation.//
(c28)	$(\forall g: \text{Goal}; r: \text{Organizational_Role}; \text{act}: \text{Actor})$ $(g.\text{sec_isa} = \text{Organizational_Goal} \wedge r \in \text{act.occupy} \wedge g \in r.\text{responsible} \wedge \text{act.isa} = \text{Software_Agent}) \Rightarrow g.\text{org_isa} = \text{Requirement}$ //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 .//
(c29)	$(\forall g: \text{Goal}; r: \text{Organizational_Role}; \text{act}: \text{Actor})$ $(g.\text{sec_isa} = \text{Organizational_Goal} \wedge r \in \text{act.occupy} \wedge g \in r.\text{responsible} \wedge \text{act.isa} = \text{Human_Actor}) \Rightarrow g.\text{org_isa} = \text{Expectation}$ //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 .//
(c30)	$(\forall g: \text{Goal}) g.\text{sec_isa} \neq \text{Organizational_Goal} \Rightarrow g.\text{resolve} = \emptyset$ //If Goal g is not an Organizational Goal, then g cannot resolve Conflicts.//

Fig. 13. Formal specification of the *Goal* concept

Goal Refinement Alternative

Refining a goal consists in identifying a set of sub-goals that, when achieved, imply that the refined goal has been achieved. As a goal can often be refined into alternative sets of goals that may lead to its achievement, we introduce *Goal Refinement Alternative* in order to better organize goal refinement information, which is significant when evaluating alternative process structures. Each set of alternatives is identified through goal refinement and represented as a *Goal Refinement Alternative*.

[Status_Value]:= Complete | Incomplete

Goal Refinement Alternative

name : *Name*
description : *Informal_Definition*
contains : set *Goal*
status : *Status_Value*

(c31)	$\text{contains} \neq \emptyset$
(c32)	$(\forall \text{gra}: \text{Goal_Refinement_Alternative}; g_1, \dots, g_n: \text{Goal}; g_k: \text{Goal})$ $(\bigwedge_{1 \leq i \leq n} g_i.\text{prim_isa} = \text{Operational_Goal}) \wedge g_k.\text{prim_isa} = \text{Operational_Goal} \wedge$ $\text{gra.contains} = \{g_1, \dots, g_n\} \wedge g_k \notin \{g_1, \dots, g_n\} \wedge \text{gra} \in g_k.\text{refined_by} \wedge (\bigwedge_{1 \leq i \leq n} g_i.\text{status} = \text{Fulfilled}) \Rightarrow \text{gra.status} = \text{Complete}$ //If some Goal Refinement Alternative gra composed of Operational Goals g_1, \dots, g_n refines Operational Goal g_k and if all Goals g_1, \dots, g_n are fulfilled, then Goal Refinement Alternative gra is complete.//

Fig. 14. Formal specification of the *Goal Refinement Alternative* concept

A *Goal Refinement Alternative* contains a set of *Goals* (c31). Its *Status* indicates whether the alternative is sufficient or not (i.e. complete or incomplete) to fulfil the *refined Operational Goal* (c32).

CONFLICT SUB-MODEL

Identification and resolution of inconsistencies is a necessary condition for successful development of a system. From the management perspective, the representation of *Conflicts* makes them clearly visible to stakeholders, serving as a basis for the negotiation among stakeholders that would lead to the introduction of *Organizational Goals* that *resolve Conflicts*. The conflicts sub-model is shown in Fig. 12.

Conflict

The *Conflict* concept represents inconsistencies that may exist in a business process and/or in the system to build. *Conflicts* may result from *Goals* that cannot be concurrently fulfilled and/or contributed to (e.g., when a *Goal* is fulfilled and/or contributed to, other *Goals* cannot be fulfilled and/or contributed to), and/or from inconsistencies among *Objects* (e.g., *Beliefs* that cannot be true in the same state of the environment, *Resources* that are concurrently input into several *Actions*).

The *concern* relationship identifies the *Objects*, *Goals* and *Boundary Conditions* that are involved in the *Conflict*.

In addition to standard attributes, we further characterize *Conflict* with a degree of *Criticality* (which describes the severity of consequences of the *Conflict*), and *Likelihood* (which describes how likely the *Conflict* occurrences are). Their values are domain-specific.

Taking inspiration from the work on inconsistencies in the context of the KAOS framework (Dardenne et al., 1993; van Lamsweerde, 1998; Spivey, 1992) we consider three types of *Conflict*: *Object Inconsistency* (which exists whenever several *Objects* cannot be considered true together (c34)), *Divergence* (existing whenever a set of *Objects* and a *Boundary Condition* cannot hold true together (c38)), and *Obstruction* (which is a particular case of divergence where a single *Object* and a single *Boundary Condition* cannot be true together (c39)).

After *Conflicts* have been identified, diverse methods for their resolution can be applied (see e.g., (van Lamsweerde A., Darimont R. & Letier E., 1998)). In the *Conflict* schema, we provide two resolution methods. When a *Boundary Condition* is source of *Conflict*, we can introduce some *Operational Organizational Goal* which eliminates the possibility of the *Boundary Condition* being true after such *Goal* has been fulfilled (c36). In case a *Goal Refinement Alternative* is source of *Conflict*, we can substitute that alternative with another one which does not lead to *Conflict* (c37).

The constraint (c35) is used to derive goals that are concerned by the conflict from objects that have already been identified as involved in the conflict, since goals are specified as sets of objects that describe the desired state of the environment.

```
[Boundary_Condition]
[Conflict_Type]:= Object_Inconsistency | Divergence | Obstacle
[Criticality_Value]
[Likelihood_Value]
```

Conflict

```
name : Name
description : Informal_Definition
```

	<p>concern : set <i>Object</i> \cup set <i>Goal</i> \cup set <i>Boundary_Condition</i> isa : <i>Conflict_Type</i> criticality : <i>Criticality_Value</i> likelihood : <i>Likelihood_Value</i></p>
(c33)	concern $\neq \emptyset$
(c34)	<p>$(\forall \text{obj}_1, \dots, \text{obj}_n: \text{Object}) \neg \exists (\wedge_{1 \leq i \leq n} \text{obj}_i) \wedge (\neg \exists \text{obj}_i \mid \neg \exists (\wedge_{j \neq i} \text{obj}_j))$ $\Rightarrow \exists \text{cfl}: \text{Conflict} \wedge \text{cfl.concern} = \{\text{obj}_1, \dots, \text{obj}_n\} \wedge \text{cfl.isa} = \text{Object_Inconsistency}$ //If there is a set of Objects $\text{obj}_1, \dots, \text{obj}_n$ which is minimal – i.e. there is no Object obj_i such that its elimination from the set results in maintaining the inconsistency – and if it is impossible that all <i>Objects</i> in the set be true together, then there is an Object Inconsistency Conflict <i>cfl</i> which concerns the set of Objects $\text{obj}_1, \dots, \text{obj}_n$./</p>
(c35)	<p>$(\forall \text{g}_1, \dots, \text{g}_m: \text{Goal}; \text{cfl}: \text{Conflict}) \forall i=1, \dots, m \text{g}_i.\text{state} \subseteq \text{cfl.concern} \wedge \forall i \neq j \text{g}_i \neq \text{g}_j$ $\Rightarrow \text{cfl.concern} == \text{cfl.concern} \cup \{\text{g}_1, \dots, \text{g}_m\}$ //If there is a set of different Goals $\text{g}_1, \dots, \text{g}_m$ such that the set of Objects that defines the state of each Goal g_i of the set is a subset of Objects that are concerned by Conflict <i>cfl</i>, then the Conflict concerns also the set of Goals $\text{g}_1, \dots, \text{g}_m$./</p>
(c36)	<p>$(\forall \text{bc}: \text{Boundary_Condition}; \text{cfl}: \text{Conflict}) \text{bc} \in \text{cfl.concern} \Rightarrow (\exists \text{g}: \text{Goal} \wedge \text{g.prim_isa} = \text{Operational_Goal} \wedge \text{g.sec_isa} = \text{Organizational_Goal} \wedge (\text{g.status} = \text{Fulfilled} \Rightarrow \neg \text{bc}) \wedge \text{cfl} \in \text{g.resolve})$ //For any Conflict <i>cfl</i> which contains a Boundary Condition <i>bc</i> exists an Operational Organizational Goal <i>g</i> which, when fulfilled, resolves Conflict <i>cfl</i>, by eliminating the Boundary Condition <i>bc</i>./</p>
(c37)	<p>$(\forall \text{g}: \text{Goal}; \text{gra}_1: \text{Goal_Refinement_Alternative}; \text{cfl}: \text{Conflict})$ $(\text{g.refined_by} = \text{gra}_1 \wedge \text{gra}_1.\text{status} = \text{Complete} \wedge \text{cfl.concern} = \text{gra}_1.\text{contains})$ $\Rightarrow (\exists \text{gra}_2: \text{Goal_Refinement_Alternative} \wedge \text{g.refined_by} = \text{gra}_2 \wedge \text{cfl.concern} \neq \text{gra}_2.\text{contains})$ //Whenever a Goal Refinement Alternative gra_1 of Goal <i>g</i> generates Conflict <i>cfl</i>, there exists another Goal Refinement Alternative gra_2 of Goal <i>g</i> which does not generate Conflict <i>cfl</i>./</p>

Fig. 15. Formal specification of the *Conflict* concept

Boundary Condition

A *Boundary Condition* leads to *Conflict*. We use this concept to represent specific circumstances (in terms of conditions specified on *Objects*) which make *Goals* and/or *Objects* conflicting. Consequently, in situations other than those described in a *Boundary Condition*, the *Conflict* does not exist. The *state* of the *Boundary Condition* specifies the circumstances under which the *Conflict* exists.

	<p>Boundary Condition</p> <p>name : <i>Name</i> description : <i>Informal_Definition</i> state : set <i>Object</i></p>
(c38)	<p>$(\forall \text{obj}_1, \dots, \text{obj}_n: \text{Object}; \text{bc}: \text{Boundary_Condition})$ $\neg \exists ((\wedge_{1 \leq i \leq n} \text{obj}_i) \wedge \text{bc}) \wedge \exists ((\wedge_{1 \leq i \leq n} \text{obj}_i) \wedge \neg \text{bc}) \wedge (\neg \exists \text{obj}_i \mid \neg \exists ((\wedge_{j \neq i} \text{obj}_j) \wedge \text{bc}))$ $\Rightarrow \exists \text{cfl}: \text{Conflict} \wedge \text{cfl.concern} = \{\text{obj}_1, \dots, \text{obj}_n, \text{bc}\} \wedge \text{cfl.isa} = \text{Divergence}$ //If there is a set of Objects $\text{obj}_1, \dots, \text{obj}_n$ which is minimal – i.e. there is no Object obj_i such that its elimination from the set results in maintaining the inconsistency – and for which there is a Boundary Condition <i>bc</i> such that both the set of Objects and the Boundary Condition can never be true together, then there is a Divergence Conflict <i>cfl</i> which concerns the set of Objects $\text{obj}_1, \dots, \text{obj}_n$ and the Boundary Condition <i>bc</i>./</p>
(c39)	<p>$(\forall \text{obj}: \text{Object}; \text{bc}: \text{Boundary_Condition}) \neg \exists (\text{obj} \wedge \text{bc}) \wedge \exists (\text{obj} \wedge \neg \text{bc})$ $\Rightarrow \exists \text{cfl}: \text{Conflict} \wedge \text{cfl.concern} = \{\text{obj}, \text{bc}\} \wedge \text{cfl.isa} = \text{Obstruction}$ //If there is an Object <i>obj</i> and a Boundary Condition <i>bc</i> such that <i>obj</i> and <i>bc</i> cannot be true together, then there is an Obstruction Conflict <i>cfl</i> which concerns the Object <i>obj</i> and the Boundary Condition <i>cfl</i>./</p>

Fig. 16. Formal specification of the *Boundary Condition* concept

PROCESS SUB-MODEL

The process sub-model describes how goals of the organization are achieved through transformation of objects. To fulfil and contribute to goals, actors apply *Plans*. *Plans* are composed of *Actions*, which realize the transformation of inputs into outputs, according to their specific internal procedures.

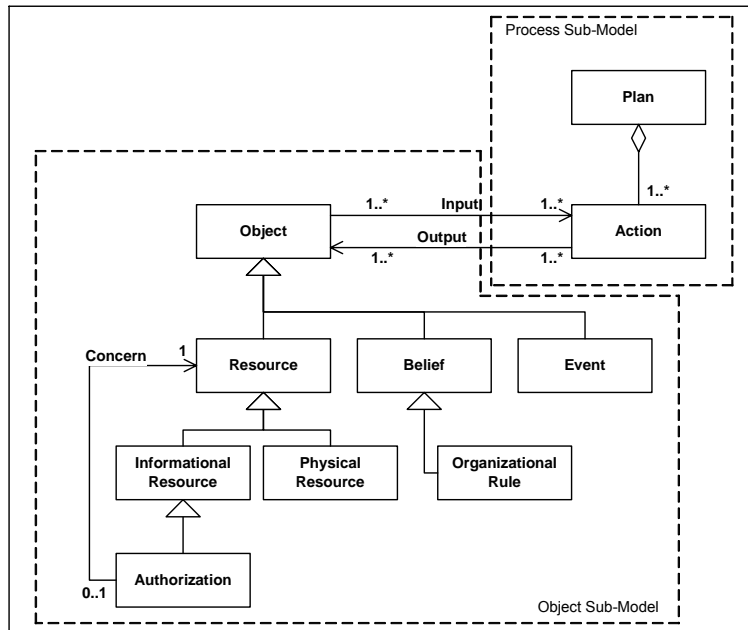


Fig. 17. Process and Object Sub-Models

Action

An *Action* is a transformation of an *input Objects* to an *output Objects*. It has the following mandatory attributes:

- *PreCondition*, describes the necessary *Objects* (*Beliefs*, *Resources*, etc.) required by the *input* in order to apply the *Action*.
- *Trigger* describes the sufficient *Objects* required by the *input* in order to apply the *Action*.
- *PostCondition* describes the *Objects* being *output* after the *Action* has succeeded.
- *Contingency* describes the *Objects* being *output* after the *Action* has failed.

PreCondition differs from *Trigger* in the sense that an *Action* may be applied when the *PreCondition* holds, whereas it must be applied when the *Trigger* holds (e.g., a stock-trading agent might sell some company's stock if it starts to fall, but must sell it if it falls under some threshold value).

```
[Action_Outcome]:= Succeeded | Failed
[Cost_Value]
[Duration_Value]
```

Action

```
name : Name
description : Informal_Definition
```

	<pre> precondition : set Object trigger : set Object postcondition : set Object contingency : set Object status : Action_Outcome input : set Object output : set Object precedent : set Action concurrent : set Action alternative : set Action cost : Cost_Value duration : Duration_Value </pre>
(c40)	$(\text{trigger} \neq \emptyset) \wedge (\text{postcondition} \neq \emptyset)$
(c41)	<pre> (∀ a: Action) ∃ obj₁,...,obj_n: Object ∧ {obj₁,...,obj_n} = a.postcondition ⇒ a.status = Succeeded //If the set of Objects {obj₁,...,obj_n} is the postcondition of an Action a, and if this set exists, then the Action a has succeeded. Otherwise, the Action a has failed.// </pre>
(c42)	<pre> (∀ a: Action ∧ obj: Object) obj ∈ a.input ⇔ obj ∈ a.precondition ∪ a.trigger //Any Object obj that is input into Action a is in the precondition or trigger of that Action.// </pre>
(c43)	<pre> (∀ a: Action ∧ obj: Object) obj ∈ a.output ⇔ obj ∈ a.postcondition ∪ a.contingency //Any Object obj that is output from Action a is in the postcondition of that Action.// </pre>
(c44)	<pre> (∀ a₁, a₂: Action ; obj: Object) obj ∈ a₁.input ∧ obj ∈ a₂.output ⇒ (a₂ ∈ a₁.precedent) //If Action a₂ outputs Object obj and obj is input into Action a₁ then Action a₂ precedes Action a₁.// </pre>
(c45)	<pre> (∀ a₁, a₂: Action) a₁.trigger = a₂.trigger ⇒ a₂ ∈ a₁.concurrent ∧ a₁ ∈ a₂.concurrent //Any two Actions a₁ and a₂ that have identical triggers are concurrent.// </pre>
(c46)	<pre> (∀ a₁, a₂: Action ; obj: Object) obj ∈ a₁.output ∧ obj ∈ a₂.output ⇒ ((a₂ ∈ a₁.alternative) ∧ (a₁ ∈ a₂.alternative)) //Any two Actions a₁ and a₂ that can both output the same Object obj are alternative to one another.// </pre>

Fig. 18. Formal specification of the *Action* concept

Whenever an *Action* has been applied, its *Status* attribute indicates if it has succeeded (c41), resulting in either *Postcondition* or *Contingency* being achieved.

Relations between *Actions* can be explicitly modelled by using the optional attributes *Precedent*, *Concurrent*, and *Alternative*. Respectively, *Actions* can be accomplished sequentially (when an *Action* precedes some other *Action* (c44)), concurrently (when they are accomplished in parallel (c45)), and can be alternative to one another (when they output identical *Objects* (c46)). This is of particular interest since activity diagrams, (such as those in e.g., UML (Bennett et al., 2002)) can be derived directly from the model.

An *Action* may also be described using the following optional attributes: *Cost* (which specifies the material cost of applying the *Action*), and *Duration* (which specifies time necessary to apply the *Action*). These attributes can be used to evaluate each alternative in terms of cost and time necessary to fulfil *Goal* or achieve a process, which helps in alternative selection when specific budget and performance constraints apply.

Plan

A *Plan* represents a way of doing something (e.g., a sequence of *Actions*). It can *fulfil* and/or *contribute* to a *Goal*. An *Actor* selects *Plans* according to the *Goals* that it pursues and *Beliefs* that it *follows*. An *Actor* can select only among *Plans* which compose the *Capabilities* that it *possesses*.

When a *Plan* is applied, each of the *Actions* that compose it can either succeed or fail. Consequently, we say that a *Plan* has succeeded only if all *Actions* that compose it have succeeded (c50). When a *Plan* has failed, the *Action* of that *Plan* which has failed *outputs* the set of *Objects* specified in its *Contingency* attribute (c51).

[Plan_Outcome]:= Succeeded | Failed

Plan	
	name : <i>Name</i> description : <i>Informal_Definition</i> composed_of : seq <i>Action</i> fulfil : set <i>Goal</i> contribute : set <i>Goal</i> status : <i>Plan_Outcome</i>
(c47)	composed_of ≠ ∅
(c48)	$(\forall pl: Plan \wedge g: Goal) g \in pl.fulfill \Rightarrow g.prim_isa \neq Softgoal$ //If a Goal <i>g</i> can be fulfilled, then <i>g</i> is not a Softgoal.//
(c49)	$(\forall pl: Plan \wedge g: Goal) g \in pl.contribute_to \Rightarrow g.prim_isa \neq Operational_Goal$ //If Plan <i>pl</i> contributes to Goal <i>g</i> , then <i>g</i> is not an Operational Goal.//
(c50)	$(\forall pl: Plan) \forall a_1, \dots, a_n: Action \wedge \langle a_1, \dots, a_n \rangle = pl.composed_of \wedge$ $\wedge_{1 \leq i \leq n} a_i.status = Succeeded \Rightarrow pl.status = Succeeded$ //If some Plan <i>pl</i> is composed of a sequence of Actions $\langle a_1, \dots, a_n \rangle$ and if each of the Actions in the sequence has been succeeded, then the Plan <i>pl</i> has succeeded.//
(c51)	$(\forall pl: Plan) \forall a_1, \dots, a_n: Action \wedge \langle a_1, \dots, a_n \rangle = pl.composed_of \wedge \exists a_i \in \langle a_1, \dots, a_n \rangle \wedge$ $a_i.status = Failed \Rightarrow pl.status = Failed \wedge \exists objcont! = \{obj_1, \dots, obj_m: Object\} \wedge$ $objcont = a_i.contingency$ //If some Action <i>a_i</i> which is in the sequence of Actions that compose Plan <i>pl</i> has failed, then the Plan <i>pl</i> has failed and the Action <i>a_i</i> outputs the set of Objects <i>objcont</i> , which belong to its contingency meta-attribute.//

Fig. 19. Formal specification of the *Plan* concept

OBJECT SUB-MODEL

The Object sub-model describes the *Objects* in the environment relevant for the process and the organization. An *Object* is a non-intentional entity of interest for the organization or involved in its business processes. *Objects* can be *input* and *output* of *Actions*. An *Object* can be a *Resource*, a *Belief*, or an *Event*. Contrary to *Actors*, *Objects* exhibit neither intentional nor social behavior.

Resource

A *Resource* is an *Object* that can be used or consumed during the performance of an *Action*. Resources can be owned by actors and can be assigned to organizational roles. As the characteristics of a resource can change with its utilization and/or with time, the *state* of a resource

is used to specify how and when changes occur. The *state* of the resource is specified as a *Belief* concerning the resource.

Usually the resources are needed when organizational roles intend to fulfil and/or contribute to organizational goals. They are often rare and used by a number of organizational roles. The exchange of rights to use resources is done through *Authorizations*: whenever an organizational role requires the resource, it must receive the authorization to use it from the organizational role occupied by the owner of the resource.

We distinguish *Informational Resources* from *Physical Resources*. An *Informational Resource* is a piece of data or information used as *input* in or produced as *output* by one or more *Actions*. We can characterize an *Informational Resource* with two optional attributes:

- *Relevance* describes the degree to which the informational resource aids the *Action*.
- *Information Quality* describes the quality of the informational resource in terms of domain-specific quality criteria.

A *Physical Resource* is a tangible *Resource* used as *input* in *Actions* or produced by them as *output*. We do not propose specific attributes for *Physical Resources* as they are domain-specific (e.g., different sets of attributes would be used to describe resources used in steel industry than those used in pharmaceuticals).

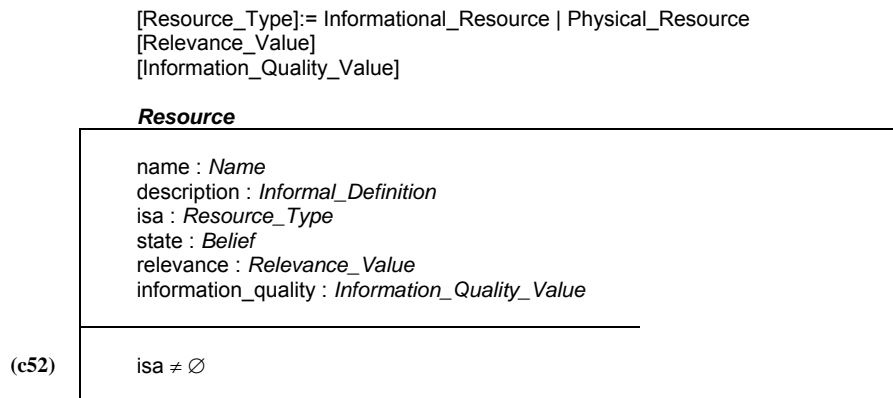


Fig. 20. Formal specification of the *Resource* concept

Authorization

An *Authorization* is a specific type of *Informational Resource*. It is assigned to *Organizational Roles* and enables them to use the *Resource* that it *concerns*. It is similar to the concept of permission in Gaia (Zambonelli et al., 2003) since:

- It identifies the resources that can legitimately be used to carry out a role as it is *assigned to* organizational roles.
- It states the resource limits within which a role must operate, since an authorization has limited validity: it “expires” as soon as the belief specified in its *Valid Until* attribute becomes true. An authorization may last until e.g., the resource has been consumed to a certain level or up to some point in time.

As indicated in Fig. 3, an *Actor* may own a resource, and that resource may be required by other actors in order to fulfil their responsibilities. Consequently, the owner of the resource must be capable of providing authorizations to organizational roles that require the execution of actions that use or consume that resource (c54).

Authorization	
	name : <i>Name</i> description : <i>Informal_Definition</i> assigned_to : <i>Organizational_Role</i> concern : <i>Resource</i> valid_until : <i>Belief</i>
(c53)	$(\text{assigned_to} \neq \emptyset) \wedge (\text{concern} \neq \emptyset)$
(c54)	$(\forall \text{act: Actor; res: Resource}) \text{res} \in \text{act.own}$ $\Rightarrow \exists \text{actn: Action, pl: Plan, cap: Capability, azn: Authorization}$ $(\text{azn} \in \text{actn.postcondition} \wedge \text{actn} \in \text{pl.composed_of} \wedge \text{pl} \in \text{cap.composed_of} \wedge$ $\text{cap} \in \text{act.possess})$ <i>//If an Actor act owns the Resource r, then act is capable of executing an Action actn</i> <i>that has an Authorization azn as postcondition.//</i>

Fig. 21. Formal specification of the *Authorization* concept

Belief

A *Belief* corresponds to true or false information that an actor carries about the environment in which it exists. The environment is described through *Terms* which are variables or functions defined over other *Terms*. A *Term* can be an *Object*, *Actor*, *Organizational Role*, *Group* or *Organization*.

```
[Term]:= Function(Term,...)
        | Object
        | Actor
        | Organizational_Role
        | Group
        | Organization
```

Atomic Belief

name : <i>Name</i> description : <i>Informal_Definition</i> terms : seq <i>Term</i>

```
[Belief]:= Atomic_Belief
          | ¬ Atomic_Belief
          | Atomic_Belief Connective Atomic_Belief
          | Temporal_Pattern Atomic_Belief
[Connective]:= ^ | v | =>
[Temporal_Pattern]:= o | • | ◇ | ◆ | □ | ■ | W | U
```

Fig. 22. Formal specification of the *Belief* concept

A *Belief* is specified either as an *Atomic Belief*, a negated *Atomic Belief*, a series of *Atomic Beliefs* connected using logic connectives, or an *Atomic Belief* characterized with a temporal pattern. We use the following temporal patterns (van Lamsweerde et al., 1998): \circ (in the next state), \bullet (in the previous state), \diamond (some time in the future), \blacklozenge (some time in the past), \square (always in the future), \blacksquare (always in the past), \mathcal{W} (always in the future unless), and \mathcal{U} (always in the future until).

Organizational Rule

Organizational Rules are a specific type of *Belief* which defines global constraints in the group or organization. They govern interactions among actors and specify how the group or the organization is supposed to work.

Organizational rules are of particular importance in the context of open systems. They constrain the behavior of self-interested actors as they make it possible to distinguish legitimate from illegitimate expression of self-interest (Zambonelli et al., 2003).

Organizational rules are enforced both in groups and in the entire organizations. As indicated in (c18), an organizational rule specified for the organization is enforced in every group of the organization.

Event

An *Event* is a change of *Goals*. It is either *output* of an *Action*, or exogenous to the organization, resulting from an action not accomplished by actors in the organization (e.g., for a company, a change in the currency exchange rate is such an *Event*). An *Event* *affects* *Goals*, by dynamically adding, removing, or modifying them.

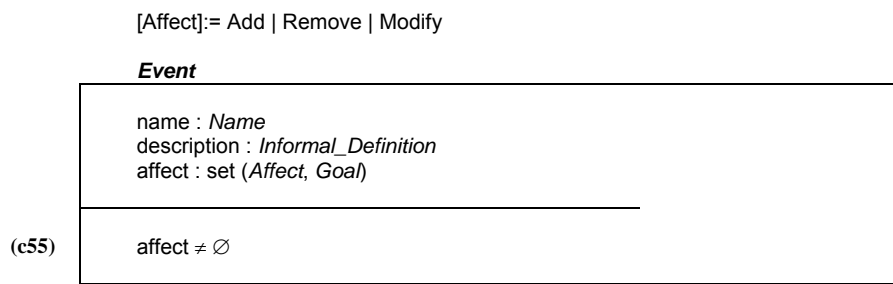


Fig. 23. Formal specification of the *Event* concept

CONCLUSION

Modeling 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., Kolp et al., 2003). Such models are usually founded on primitive concepts such as those of *actor* and *goal* (e.g., Yu, 1994). Unfortunately, no specific enterprise modeling framework really exists for engineering modern corporate IS. This chapter proposes an integrated agent-oriented enterprise model for enterprise modeling. Moreover, our approach differs primarily in the fact that it is founded on ideas from in RE frameworks, management theory concepts found to be relevant for enterprise modeling 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 enterprise model. For example, goals could be classified into further goal categories such as Accuracy, Security, Performance, etc. We also intend to define a strategy to guide enterprise modeling using our model as well as to define a modeling tool à la Rational Rose to visually represent the concepts.

REFERENCES

- Bennett S., McRobb S. & Farmer R. (2002) *Object-Oriented Systems Analysis and Design Using UML*. McGraw-Hill International.
- Bowen J. (1996) *Formal Specification and Documentation using Z: A Case Study Approach*, Thomson Publishing.
- Briand L., Melo W., Seaman C. & Basili V. (1995) Characterizing and Assessing a Large-Scale Software Maintenance Organization. *Proceedings of the 17th International Conference on Software Engineering*, Seattle, WA.
- Brickley J.A., Smith C.W. & Zimmerman J.L. (2001) *Managerial Economics and Organization Architecture*. McGraw-Hill Irwin.
- Cabri G., Ferrari L., Leonardi L., & Zambonelli F. (2004) Role-based Approaches for Agent Development. *Proceedings of the International Workshop on Agent-Oriented Software Engineering (AOSE)*.
- Castro J., Kolp M. & Mylopoulos J. (2002) Towards Requirements-Driven Information Systems Engineering: The Tropos Project. *Information Systems*, 27.
- Cernuzzi L., Juan T., Sterling L. & Zambonelli F. (2004) The Gaia Methodology: Basic Concepts and Extensions. In *Methodologies and Software Engineering for Agent Systems*, Kluwer.
- Chung L.K., Nixon B.A., Yu E. & Mylopoulos J. (2000) *Non-Functional Requirements in Software Engineering*, Kluwer Publishing.
- Dardenne A., van Lamsweerde A. & Ficklas S. (1993) Goal-directed requirements acquisition. In *Sciences of Computer Programming*, 20, 3-50.
- Elmagarmid A. & Du W. (1998) Workflow Management: State of the Art Versus State of the Products. In *Workflow Management Systems and Interoperability*, Springer Heidelberg.
- Faulkner S. (2004) *An Architectural Framework for Describing BDI-Multi-Agent Information Systems*, PhD Thesis, University of Louvain.
- Faulkner S., Kolp M., Coyette A. & Tung Do T. (2004) Agent-Oriented Design of E-Commerce System Architecture. *Proceedings of the 6th International Conference in Enterprise Information Systems Engineering*, Porto.
- Ferber J. & Gutknecht O. (1998) A meta-model for the analysis and design of organizations in multi-agent systems. *Proceedings of the Third International Conference on Multi-Agent Systems*.
- Ferber J., Gutknecht O. & Michel F. (2003) From Agents to Organizations: an Organizational View of Multi-Agent Systems. *Proceedings of the International Workshop on Agent-Oriented Software Engineering (AOSE)*.
- Fox M.S. & Grüninger M. (1997) On Ontologies And Enterprise Modeling. *Proceedings of the International Conference on Enterprise Integration Modeling Technology*, Springer-Verlag,.
- Guizzardi G., Herre H. & Wagner G. (2002) On the General Ontological Foundations of Conceptual Modeling. *Proceedings of the 21st International Conference on Conceptual Modeling (ER 2002)*, Springer-Verlag, Berlin.
- Jennings N. R. (2000) On Agent-Based Software Engineering. *Artificial Intelligence*, 117, 277-296.
- Johnson G. & Scholes K. (2002) *Exploring Corporate Strategy, Text and Cases*. Prentice Hall.
- Kamath M., Dalal N.P., Chaugule A., Sivaraman E. & Kolarik W.J. (2003) A Review of Enterprise Process Modeling Techniques. In Prabhu V., Kumara S., Kamath M.: *Scalable Enterprise Systems: An Introduction to Recent Advances*. Kluwer Academic Publishers, Boston.
- Kaplan S. & Norton P. (2000) *The Strategy-Focused Organization*, Harvard Business School Press.
- Kaplan, R. S., and Norton, D. P. (2004). *Strategy Maps: Converting Intangible Assets into Tangible Outcomes*. Boston, MA, Harvard Business School Press.
- Kavakli V. & Loucopoulos P. (1999) Goal-Driven Business Process Analysis Application in Electricity Deregulation. *Information Systems*, 24, 187-207.
- Kavakli E. (1999) *Goal-Driven Requirements Engineering: Modeling and Guidance*. PhD Thesis, University of Manchester.
- Kettinger W. J., Teng J. T. C. & Guha S. (1997) Business Process Change: A Study of Methodologies, Techniques, and Tools. *MIS Quarterly*.

- Kolp M., Giorgini P. & Mylopoulos J. (2003) Organizational Patterns for Early Requirements Analysis. *Proc. of the 15th Int. Conf. on Advanced Information Systems Engineering (CAiSE'03)*, Velden, Austria.
- Koubarakis M. & Plexousakis D. (2002) A formal framework for business process modeling and design. *Information Systems*, 27, 299-319.
- Liu L. & Yu E. (2004) Designing information systems in social context: a goal and scenario modeling approach. *Information Systems*, 29, 187-203.
- Mao X. & Yu E. (2004) Organizational and Social Concepts in Agent Oriented Software Engineering. *Proceedings of the International Workshop on Agent-Oriented Software Engineering (AOSE)*.
- Mentzas G., Halaris C. & Kavadias S. (2001) Modeling business processes with workflow systems: an evaluation of alternative approaches. *International Journal of Information Management*, 21, 123-135.
- Odell J., Nodine M. & Levy R. (2004) A Metamodel for Agents, Roles, and Groups, In *Agent-Oriented Software Engineering V* (pp. 78-92). Springer.
- Sheth A.P., van der Aalst W. & Arpinar I.B. (1999) Processes Driving the Networked Economy. *IEEE Concurrency*, 7, 18-31.
- Simon H. A. (1976) *Administrative Behavior : A Study of Decision-Making Processes in Administrative Organization*. New York: The Free Press.
- Simon H. A. (1979) Rational Decision Making in Business Organizations. *The American Economic Review*, 69(4), 493-513.
- Spivey J. M. (1992) *The Z Notation: A Reference Manual*. 2nd Edition, Prentice Hall International.
- Stader J.: Results of the Enterprise Project. *Proceedings of Expert Systems '96, the 16th Annual Conference of the British Computer Society Specialist Group on Expert Systems*, Cambridge UK, 1996.
- Uschold M., Grüninger M. (1996) Ontologies: Principles, Methods and Applications. *Knowledge Engineering Review*, 11(2).
- Uschold M., King M., Moralee S. & Zorgios Y. (1997) *The Enterprise Ontology*. AIAI, The University of Edinburgh.
- van Lamsweerde A. (2000) Requirements engineering in the year 00: a research perspective. In *Proceedings of the 22nd International Conference on Software Engineering*, Limerick Ireland. ACM, pp. 5-19.
- van Lamsweerde A. (2001) Goal-Oriented Requirements Engineering: A Guided Tour. *Proceedings RE'01, 5th IEEE International Symposium on Requirements Engineering*, Toronto, pp. 249-263.
- van Lamsweerde A., Darimont R. & Letier E. (1998) Managing Conflicts in Goal-Oriented Requirements Engineering. *IEEE Transactions on Software Engineering, Special Issue on Managing Inconsistency in Software Development*.
- Yu E. (1994) *Modeling Strategic Relationships for Process Reengineering*. Ph.D. thesis, Dept. of Computer Science, University of Toronto.
- Yu E. (2001) Agent-Oriented Modeling: Software Versus the World. *Proceedings of the International Workshop on Agent-Oriented Software Engineering (AOSE)*, Springer Verlag.
- Zambonelli F., Jennings N. R. & Wooldridge M. (2003) Developing multiagent systems: The Gaia methodology. *ACM Trans. Softw. Eng. Methodol.* 12(3): 317-370.
- zur Muehlen M. (2002) *Workflow-based Process Controlling*. Logos Verlag Berlin.

ⁱ To clarify the formal specifications, we embed the comments on predicates between two “//” signs.