An organizational ontology for multiagent-based Enterprise Process modeling and automation

Yishuai Lin

▶ **To cite this version:**

Yishuai Lin. An organizational ontology for multiagent-based Enterprise Process modeling and automation. Computers and Society [cs.CY]. Université de Technologie de Belfort-Montbeliard, 2013. English. NNT : 2013BELF0206 . tel-00977726

HAL Id: tel-00977726

https://theses.hal.science/tel-00977726

Submitted on 11 Apr 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

SPIM

Thèse de Doctorat

école doctorale sciences pour l'ingénieur et microtechniques
UNIVERSITÉ DE TECHNOLOGIE BELFORT-MONTBÉLIARD

An organizational ontology for multiagent-based Enterprise process modeling and automation

Application to SCRUM

Yishuai LIN

SPIM

Thèse de Doctorat

école doctorale **sciences pour l'ingénieur et microtechniques**

UNIVERSITÉ DE TECHNOLOGIE BELFORT-MONTBÉLIARD

N° | X | X | X |

THÈSE présentée par

Yishuai LIN

pour obtenir le

Grade de Docteur de
l'Université de Technologie de Belfort-Montbéliard

Spécialité : **Informatique**

An organizational ontology for multiagent-based Enterprise process modeling and automation

Application to SCRUM

Soutenue le X septembre 2013 devant le Jury :

Nada MATTA	Rapporteur	Maître de conférences-HDR à l'Université de Technologies de Troyes
Vincent CHEVRIER	Rapporteur	Maître de conférences-HDR à l'Université de Lorraine, Nancy
Djamal BENSLIMANE	Examinateur	Professeur des Universités, Université Claude Bernard, Lyon
Nicolas GAUD	Examinateur	Maître de conférences à l'Université de Technologie de Belfort-Montbéliard
Vincent HILAIRE	Examinateur	Professeur des Universités, Université de Technologie de Belfort-Montbéliard
Pierre-Alain MÜLLER	Examinateur	Professeur des Universités, Université de Haute Alsace
Samuel GOMES	Examinateur	Professeur des Universités, Université de Technologie de Belfort-Montbéliard

ACKNOWLEDGMENTS

At the final stage of preparing the documentation of my work, this section gives me the opportunity to express my gratitude to all the people who supported and assisted me.

First of all, my research would not have been possible without the generous funding of CSC-China Scholarship Council programme[1].

[1] For more information, please refer to http://en.csc.edu.cn

CONTENTS

List of Figures

LISTINGS

List of Tables

I

CONTEXT

1

INTRODUCTION

1.1/ CONTEXT

In the 21st century, enterprises face an increasingly competitive market place. In the environment of which, the successes of enterprises depend critically on the quality and efficiency of their product business processes. By enterprise, we mean an entrepreneurial economic organization or a business organization.

Enterprises generally aim at delivering products or services to given markets. The engineering of such products/services are the result of business processes, which involve staff of the enterprise. These processes define the activities, organizational structures, organizational management and aim at collective achievements in the form of products/services.

Specifically, software companies are a special kind of IT enterprises delivering software products or services. The delivery processes deployed in these enterprises frequently follow one specific software-development process such as the traditional Waterfall Model, Spiral Model, Scrum, Extreme Programming (XP), OpenUP, etc.

Project teams, which are the collective units responsible for software products/services' design, have to face to increasing product complexity, process intricacy, markets globalization, distributed organizations and ever-changing customers' orientation. All these parameters ask for a well organized/framed approach and a strong support to business processes within enterprises.

With the development of Web 2.0, enterprises have entered the web generation, particularly through the use of collaborative technologies. It is thus significant to think about how to model the actual business processes and to manage occurring interactions in business processes. Moreover, this kind of technology could be the basis building block for an assistance system servicing for related processes' participants who join in these business processes.

1.2/ OBJECTIVES AND CONCERN OF THIS WORK

The major concern of the work could be summarized as:

Propose an approach for
(i) modeling and conceptualizing various business processes in enterprises and,
(ii) (based on the result of the previous modeling and conceptualization) designing and implementing an intelligent assistance system to support human activities within business processes.

In order to help human actors of business processes to understand how these processes proceed, the proposed approach should not only conceptualize items in enterprises, like their department structures, human relationships and so on, but also model actual workflows occurring within these processes.

To answer these concerns, we have built an organizational ontology named K-CRIO [Lin et al., 2011],[LIN et al., 2011b] for modeling enterprises. Moreover, for automating the business processes, our approach consists in designing and implementing a software assistance system that acts as a tool for human workers. This assistance system must take into account the availability and geographic distribution of actual users and must ensure means of communication. Obviously, there are too many kinds of business processes for producing a generic tool. However, to be efficient, such a tool should possess knowledge about the supported business process. This knowledge is provided by the overall process description provided with the instantiation of K-CRIO Ontology's concepts.

K-CRIO Ontology aims at proposing an organizational ontology for gathering and conceptualizing knowledge, which is defined by concepts and relationships of these concepts, which are respectively described with `owl:class` and `owl:ObjectProperty` in Ontology Web Language (OWL). Furthermore, we apply K-CRIO to conceptualize knowledge appearing in human activities. To explain the usage and guidelines of K-CRIO, we introduce two case examples, e.g. the Waterfall Model and the SCRUM Method, these two models are two well-known software-development processes, widely used in software engineering. Through these examples related to software engineering, we also aim at showing the capabilities of K-CRIO for modeling wider issues within the field of business processes.

In order to design and implement an intelligent assistance system to support business processes we have designed and implemented a web-based system relying on a Multi-Agent System. The web-based system allows the collaboration of human actors, and the MAS enables an autonomous and pro-active assistance owing to agents' capabilities. The MAS monitors the actual processes and interactions and has the knowledge of the business process described by instances of the K-CRIO ontology.

Lastly, for validating our proposed approach, we deployed our approach with the SCRUM process refining the K-CRIO ontology and adapting our MAS to support this process.

1.2.1/ Towards a method of Organizational Ontology K-CRIO for one aspect of Enterprise Modeling: business processes

An Enterprise Model is a computational representation of the structure, activities, processes, information, resources, people, behavior, goals and constraints of a business, government or other enterprises [Fox and Grüninger, 1997].

From a design perspective, an enterprise model should provide understandable language used to explicitly define an enterprise. From an operations perspective, the enterprise model should be able to represent what is planned, what has happened and what will happen?

As stated in [Bottazzi and Ferrario, 2009], an Ontology of organizations is the first, fundamental, and ineliminable pillar on which to build a precise and rigorous Enterprise Model.

Therefore, as a first step of our contribution we have defined an ontology for enterprises that rely on organizational concepts that are now considered a standard by the majority of the approaches in the field Agent-Oriented Software Engineering. This organizational ontology is named K-CRIO.

The context of this ontology is that many enterprises are on the path towards the automation of their business processes. More specifically, the targeted enterprises are the ones that are dedicated to the design of a product. The conceptualization of such enterprises results in an organizational ontology, named K-CRIO, which defines a rich and semantic model for business processes. This model allows the description of business processes in terms of peoples involved and their interactions. Indeed, expressing related people and their interactions is the basic work for modeling and conceptualizing business processes.

1.2.2/ Case Studies for conceptualization with K-CRIO

The definition and instantiation of the K-CRIO ontology are illustrated with the conceptualization of a simple sequential process, namely the Waterfall Model, often used in traditional software-development processes. In this process, the progress is seen as flowing steadily downwards (like a waterfall) through the phases of Requirement, Design, Production/Implementation, Testing/Verification and Maintenance. This example is not completely detailed, but it acts as a starting point for understanding K-CRIO fundamentals.

In order to improve and validate the K-CRIO Ontology and provide some guidelines for using this ontology to model business processes, we used K-CRIO Ontology to fully conceptualize a famous agile and iterative process, namely the SCRUM Method. This latter is widely used in current software-development processes. The provided ontology is a rather complete case study directly related to the implementation work described hereafter.

1.2.3/ Towards an assistance system for Scrum Project Teams

Ontologies in software engineering are generally used to represent (a partial) knowledge of a particular domain [Calero et al., 2006]. This knowledge constitutes a starting point for requirements analysis.

Considering our work, the use of K-CRIO will be twofold. On the one hand, it is a basis for requirements analysis. And adopting this stance, it should help defining what kind of help the assistance system could provide. On the other hand, K-CRIO is a rich and semantic model that will be used by the assistance system for reasoning on business process operations.

Multi-Agent Systems have proven to be a suitable paradigm for modeling autonomous, distributed, dynamic and complex systems.

They are thus a good candidate for the analysis, design and implementation of the aimed assistance tool. Hence, we have developed a web-based system, allowing a distributed, including a Multi-Agent System for automating a software-development process: SCRUM. It is a preliminary attempt to practice our built ontology K-CRIO as the basis ontology for assistance software. Specifically, the SCRUM Tool is providing a platform for distributed SCRUM team members, which supports actual human activities and work-flows, such as the SCRUM project phases and steps, and SCRUM MASTER[1] related tasks occurring in the SCRUM project process. Among the tasks of the SCRUM MASTER, we focused on project monitoring, cost estimations and team member efficiency evaluation. These different elements provide help for the SCRUM MASTER for decision making and project management.

1.3/ PLAN OF THE DOCUMENT

After this brief overview of the proposals presented in this thesis, this section introduces the overall organization of this manuscript. According to the objectives set out in section 1.2, this thesis is organized into three main parts. The various chapters in this document and their respective organization are summarized in Figure 1.1.

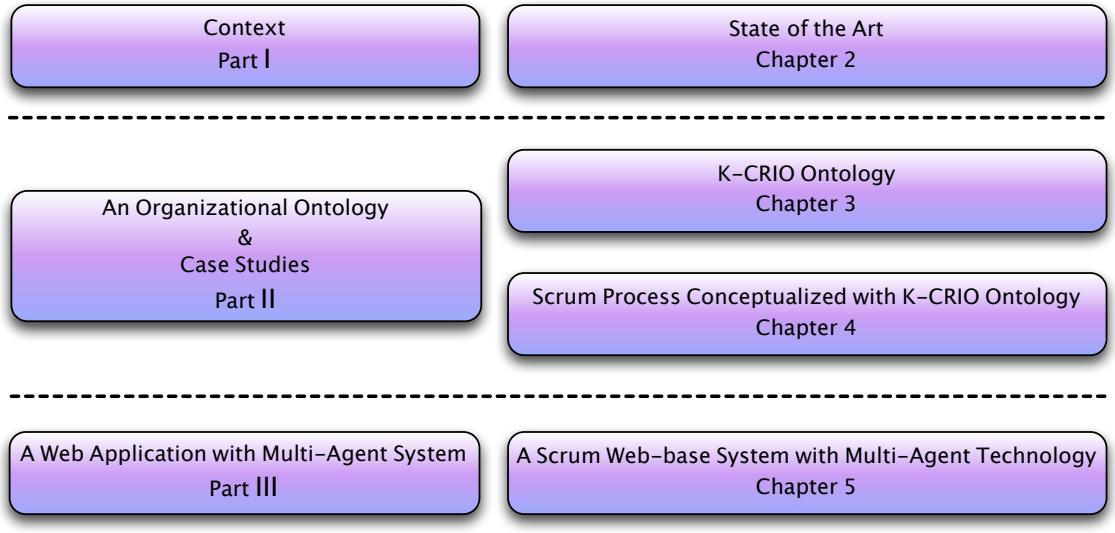

Figure 1.1: Reading Directions

Chapter 2, and more generally the first part of this thesis, presents all the necessary background for comprehending our work, more precisely the state of art about existing ap-

[1] SCRUM project team's managers

proaches based upon Ontology, MAS or Web Technologies used in Enterprise business process management and modeling. It also analyses and compares these presented models/methods.

The second part (Chapters 3 and 4) is the core of our contribution and presents the K-CRIO Ontology, and its illustrations on the Waterfall model and SCRUM software development process.

Chapter 3 presents the definition of all the concepts and their relationships in K-CRIO Ontology. K-CRIO Ontology is an organizational ontology described with Ontology Web Language, the designing inspiration of which is CRIO Meta-model that an existing meta model used in MAS. After the presentation about definition, there is a pellucid example of using K-CRIO to conceptualize the Waterfall Model software development process in order to help readers to understand K-CRIO Ontology.

Chapter 4 presents the popular agile software-development process: SCRUM, and its conceptualization with K-CRIO.

The third and last part, represented by **Chapter 5**, illustrates shows how to exploit the concepts provided by K-CRIO to develop a multiagent-based web application to assist SCRUM MASTER and SCRUM teams in managing their software-development processes and associated technical tasks (task assignment, product delivery, ...). Precisely, this chapter provides a general description of the developed tool in terms of functionalities and architecture, and a feature-based comparison with existing SCRUM tools.

2

State of the Art

2.1/ Introduction

This State-of-the-Art chapter presents the three different domains this work is the convergence of namely, the business or enterprise models, ontologies and Multi-Agent Systems. Indeed, many enterprises are on the path towards the automation of their business processes. As stated in the introduction, the main hypothesis of this work is that to be fully efficient this kind of automation needs to rely on (i) rich models of business processes, and (ii) software tools that support this automation and allow it to take place in a transparent fashion.

The literature in terms of business/enterprise models is nowadays huge as it can concern many different domains. We will restrict our study to the models that can be exploited by software agents. These models were generally defined in the computer science, or software engineering communities.

These models differ first by the chosen expression language. Each language has it owns semantic and expression capabilities and thus allow the conceptualization of the different kind of models. Ontologies are the most-used languages in order to produce such conceptualizations. Hence, we also present in this chapter ontology background and more specifically Ontology Web Language (OWL) which is widely used in knowledge engineering and domain conceptualization.

Concerning the software tool aspect, many technologies can be used. Our claim is that Multi-Agent Systems, because of the intrinsic capabilities of agents such as autonomy, re-activity and pro-activity allow the implementation of complex systems like the aimed software assistance tool.

This chapter is structured to cover the three main areas constituting the foundations of this thesis. Section 2.2 quickly presents the concept of ontology. Section 2.3 introduces the multi-agent systems. And finally, Section 2.4 looks at the heart of this thesis, the business process approaches and their associated tools.

2.2/ ONTOLOGY

2.2.1/ OVERVIEW OF THE CONCEPT OF ONTOLOGY

The word Ontology has been taken from Philosophy, where it means a systematic explanation of Existence. In terms of etymology, it comes from the Greek ”ontos” for being and ”logos” for word.

Depending on the considered area and the adopted point of view, numerous definitions may be considered. Some of the main ones are described below. Within the Artificial Intelligence field, [Neches et al., 1991] defines the notion of ontology as: ”*An ontology defines the basic terms and relations comprising the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary*”. Gruber in Gruber [1995] defines an ontology as ”*an explicit specification of a conceptualization*”. This definition is the most referenced definition in the literature. Moreover, Borst [1997] slightly improved Grubers' definition by considering that ”*Ontologies are defined as a formal specification of a shared conceptualization*”. Based on these descriptions, Studer et al. [1998] gave a more exhaustive explanation of ontology as that ”*Conceptualization refers to an abstract model of some phenomenon in the word by having identified the relevant concepts of that phenomenon. Explicit means that the type of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine-readable. Shared reflects the notion that an ontology captures consensual knowledge, that is not private to some individual but accepted by a group*”. From the perspective of taxonomy, ontology could be seen as ”*a set of types, properties, and relationship types*” [Garshol, 2004]. Basically, an ontology defines a set of concepts in a specific area and their relationships.

In Swartout et al. [1996], the authors bind ontology with a knowledge base, so that , ”*an ontology is a hierarchically structured set of terms for describing a domain that can be used as skeletal foundation for a knowledge base*”. This definition is based upon the face that builds domain-specific ontologies by identifying the relevant terms to a particular domain in the ontology SENSUS (which includes more than 50000 terms). Then, they prune the SENSUS ontology using a kind of heuristics. A different approach is taken by Bernaras et al. [1996]. They build a preliminary ontology from a knowledge base, which is refined and augmented with new definitions if new applications are built. This is the cause why the propose the following definition. ”*An Ontology provides the means for describing explicitly the conceptualization behind the knowledge represented in a knowledge base.*”

Since ontology raised, it has haven widespread use. As a set of well-defined constructs, it could be used to build structured knowledge. Ontologies include rich relationships among terms, rich taxonomies, and enable intelligent researches. For humans, ontologies enable better access to information and promote knowledge reuse and shared understanding. For computers, ontologies facilitate comprehension of information and more extensive processing (Ontology Engineering). Being a technology used for Knowledge Management, ontology defines the terms used to describe and represent an area of knowledge. Meanwhile, the ontology specifications can be passed as parameters in agent conversations. It is established as a powerful tool to enable knowledge sharing, and a growing number of applications has benefited from the use of ontology as a means to achieve semantic interoperability among heterogeneous, distributed systems.

2.2.2/ ONTOLOGY DESCRIPTION LANGUAGE

As a language for describing ontology, OWL stands for Ontology Web Language. In fact, it is a family of knowledge representation languages for authoring ontologies. OWL is supported by the World Wide Web Consortium. OWL is based upon RDF and RDFS, which are extensions of XML for describing web resources. Compared with RDF, OWL has more features that allow a greater machine interpret-ability, and comes with a larger vocabulary and richer syntax. OWL has three increasingly expressive sub-languages:

- OWL Lite, which supports those users primarily needing a classification hierarchy and simple constraints;
- OWL DL, which supports those users who want the maximum expressiveness while retaining computational completeness and decidability. OWL DL includes all OWL language constructs, but they can be used only under certain restrictions.
- OWL Full, which is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees.

In the following, we introduce the logical constructs of OWL. Classes provide an abstraction mechanism for grouping resources with similar characteristics. Every OWL class is associated with a set of individuals (called the class extension). The individuals in the class extension are called the instances of the class. OWL classes are described through ”class descriptions”, which can be combined into ”class axioms”. Specifically, OWL distinguishes six types of class descriptions:

- a class identifier (a URI reference),
- an exhaustive enumeration of individuals (that together form the instances of a class),
- a property restriction (including value constraints and cardinality constraints),
- the intersection of two or more class descriptions,
- the union of two or more class descriptions and the complement of a class description.

In terms of class axioms, OWL contains three language constructs for combining class descriptions, which are `rdfs:subClassOf`, `owl:equivalentClass` and `owl:disjointWith`. In addition, OWL distinguishes between two main categories of properties that an ontology designer may want to define, which are Object properties defining relationships between classes and Datatype (properties linking individuals to data values). Equally, OWL also supplies various types of property axioms to define additional characteristics of properties, which are as following: RDF Schema constructs (`rdfs:subPropertyOf`, `rdfs:domain` and `rdfs:range`), relations to other properties (`owl:equivalentProperty` and `owl:inverseOf`), global cardinality constraints (`owl:FunctionalProperty` and `owl:InverseFunctionalProperty`), logical property characteristics (`owl:SymmetricProperty` and `owl:TransitiveProperty`). Finally, individuals are defined with individual axioms (also called ”facts”), where are two types of facts: one about class membership and property values of individuals and the other about individual identity (that details `owl: sameAs`, `owl:differentFrom` and `owl:AllDifferent`).

2.2.3/ Ontology used in Knowledge Presentation

Various methods have been devised to support knowledge organization and interchange. Controlled vocabularies provide a standardized dictionary of terms for use during, for example, indexing or retrieval. Dictionaries can be organized according to specific relations to form taxonomies. Ontology further specifies the semantics of a domain in terms of conceptual relationships and logical theories [Jurisica et al., 2004].

Take an instance, if you are interested in the knowledge of health care, the patient, disease, symptom, diagnosis, and treatment might be the primitive concepts upon which you could describe the domain. These concepts related to their meanings, and relationships could define an ontology for health care. Such an ontology could be used as common knowledge for facilitating communication among medical workers. Additionally, it could also be used during the development of hospital information systems or decision support systems.

Earlier work in computational ontologies includes the Cyc project [Lenat and Guha, 1990] and the ARPA Knowledge Sharing effort [Neches et al., 1991]. The Knowledge Interchange Format effort provides a declarative language for describing knowledge [Genesereth, 1991]. The National Library of Medicine has assembled a large multidisciplinary, multi-site team to work on the Unified Medical Language System, aimed at reducing fundamental barriers to the application of computers to medicine [Humphreys et al., 1998]. Similarly, an ontology for manufacturing may consist of (industrial) process, resource, schedule, product and the like [Vernadat, 1996].

Ontologies may be constructed for different purposes. Some ontologies aim at describing very general concepts like space, time, matter, object, event, action, etc [Guarino and Guarino, 1997] . They are called top-level ontologies (also named upper ontologies, or foundation ontologies), which are independent of a particular problem or domain, such as DOLCE [1]. Specially, when we want to enable sharing and reuse, we define an ontology as a specification used for making ontological commitments [Gruber, 1993]. Ontological commitment is an agreement to consistently use a vocabulary with respect to a theory specified by an ontology. In order to support a specification, we define an ontology as a conceptualization, i.e., an ontology defines entities and relationships among them. Every information is based on either an implicit or an explicit conceptualization.

Research within artificial intelligence has formalized many interesting ontologies and has developed techniques for analyzing such knowledge. Along a very different path, [Wand and Weber, 1990] studied the adequacy of information systems to describe applications based on a general ontology, such as the one proposed by [Bunge, 1977].

In the area of ontology used for knowledge management, especially from the perspective of information system,[Jurisica et al., 2004] proposed four broad ontological categories, which respectively deal with static, dynamic, intentional and social aspects of the world. Their claim is that, for a large class of applications, the representation of relevant knowledge can be based on primitive concepts derived from these four ontological categories. For example, if someone wants to model a university environment, we may choose entities and relations to model static aspects of the domain and processes to model dynamic aspects. Our classification of ontological concepts into four categories has been derived from a broad survey of modeling techniques in computer science [Mylopoulos, 1998]. The four ontological categories are:

[1] http://www.loa.istc.cnr.it/DOLCE.html

- Static ontologies describe static aspects of the world, for example, what are the existing things, their attributes and relationships.
- Dynamic ontologies describe changing aspects of the world, of which, typical primitive concepts include state, state transition and processes.
- Intentional ontologies encompass the world of motivations, intents, goals, beliefs, alternatives, choices, etc., of which, typical primitive concepts include issue, goal, supports, denies, subgoalOf, agent, etc. The example of this classification is the ”Softgoals ontology” [Chung et al., 1999] for modeling software nonfunctional requirements, like software usability, security, reliability, user friendliness, performance, etc. (”Softgoals” are defined as goals whose criteria for satisfaction are not crisply defined a prior.)
- Social ontologies cover social settings, organizational structures or shifting networks of alliances and inter-dependencies [Scott and Davis, 2006]. Traditionally, social ontologies have been characterized in terms of concepts such as actor, position, role, authority, commitment, etc. Speech acts theory offers an ontology for modeling communication among actors [Medina-mora et al., 1992]. Social ontologies are also of interest in distributed artificial intelligence. Some of the concepts have been formalized using a specialized logic [Castelfranchi and Müller, 1993].

In certain situation, these above categories of ontologies need to be used together. On the application side, the most prevalent ontology-based activity is developing static ontologies, such as taxonomies or controlled vocabularies in [Godfray, 2002], [Ashburner et al., 2000], [Gennari et al., 1995]. The goal is to standardize terminology and taxonomically organize concepts in specific domains to enable information sharing and system cooperation.

2.3/ Agent and Multi-Agent System

2.3.1/ Agent

Multi-Agent Systems (MAS) stand out as a paradigm for the design of Complex Systems. Indeed, this paradigm proposes new strategies for the analysis, modeling and implementation of such systems. Its elementary constituents are called ”agents”, i.e. software entities, which exhibit autonomous and flexible behaviors.

It is to be noted that the word ”Agent” may appear in serious different areas, like in philosophy, in sociology, in law enforcement, in economics, in medicine and so on. Indeed in these different areas, agents may be used to represent acting entities. In our case, we will restrict at agents in the area of computer science.

Autonomy it operates without the direct intervention of humans or others, and has some kind of control over its actions and internal state.

Social ability it interacts with other agents (and possibly humans) via some kind of agent communication language.

Reactivity it perceives its environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and it may respond to changes that occur in it.

Pro-Activeness it does not simply act in response to its environment, it also can exhibit goal-directed behavior by taking the initiative.

This minimal set of features that an entity must exhibit to be an agent is completed by the following definition from [Ferber, 1999] :

A clearly identifiable physical or virtual autonomous entity which: (i) is situated in a particular environment of which it has only a partial representation; (ii) is capable of perceiving (with sensors) and acting (with effectors) in that environment; (iii) is designed to fulfill a specific role; (iv) communicates directly with other agents; (v) possesses its own state (and controls it) and skills; (vi) offers services (in the sense of particular problem solving capabilities); (vii) may be able to reproduce itself; (viii) has a behavior that tends to satisfy its objectives.

2.3.2/ MULTI-AGENT SYSTEM

A Multi-Agent System could be defined as a loosely coupled network of agents who work together as a society aiming at solving problems that would generally be beyond the reach of any individual agent.

According to [Sycara, 1998], the characteristics of MAS are that:

- each agent has incomplete information or capabilities for solving the overall problem tackled by the system and, thus, has a limited viewpoint.
- there is no system global control: the collective behavior is the result of social rules and interactions and not of a supervising central authority.
- resources are decentralized: resources needed for the completion of the tasks assigned to the system are divided and distributed.

Multi-Agent System has advantages for solving two main classes problems distinguished by [Zambonelli et al., 2000] as:

- distributed problem solving systems in which the agents are explicitly designed to cooperatively achieve a given goal in a benevolent fashion.
- open systems in which agents are not necessarily co-designed to share a common goal and can dynamically leave and enter the system.

Frequently, MAS researches are concerned with the problem of coordinating the behaviors among a collection of intelligent agents. For example, what are the mechanisms for agent coordination, for sharing knowledge, coordinate goals' achievement, coordinate joint plans to solve problems [Van, 1989]. Hence, numerous works, in the MAS field, have paid attention to additional architectural problems to traditional artificial intelligence, including organizational structure of the agent society and its patterns (hierarchy, anarchy,

etc.), organizational interactions sustaining the structural patterns and organizational and environmental rules constraining the structures of the interactions.

In sum, a MAS is a system composed of multiple interacting intelligent agents. MAS could be better used to solve problems, which are difficult or impossible for an individual agent or monolithic system to solve. Some examples of problems considered as being appropriate to MAS include distributed problem solving involving real distributed actors (e.g. image processing [Bourjot et al., 2003], tracking [Gechter et al., 2006], scheduling [Maes, 1994], distributed constraints satisfaction [Dury et al., 1999], collaborative work [Adam et al., 2003]), management of distributed resources and means (e.g. network management [Sierra and Sonenberg, 2005], sensors data [Yao et al., 2009], information gathering [Adam and Mandiau, January 2005, Revised Selected Paper]) and simulation of complex systems ([Gaud et al., 2008, Helleboogh et al., 2007, Rodriguez et al., 2007]).

To support MAS implementation, an adapted platform is required. There are a lot of platforms designed for developing MAS, such as JADE [Bellifemine et al., 2007], ADK (Tryllian Agent Development Kit)[2], CybelePro[3], MASS (Multi-Agent Simulation Suit[4]) and so on. For the implementation of our MAS, we have selected the Janus Platform[5].

2.3.3/ Organizational Centered Multi-Agent Systems

Already in the eighties, links between human organizations and computational systems were suggested Fox [1981]. Since then, organizational approaches have become the subject of an increasing interest in the research community.

In MAS several approaches have been proposed inspired from a Social Metaphor, where terms like ”role”, ”group”, ”community” represent the main concepts of the model. We can realize the usefulness of the concepts when we consider the number of methodologies (e.g. GAIA Zambonelli et al. [2003] or MESSAGE Caire et al. [2002]) and (meta-)models (e.g. AGR Ferber and Gutknecht [1998] or CRIO Cossentino et al. [2007].

As Ferber Ferber and Gutknecht [1998], Ferber et al. [2003] points out Organizational approaches can contribute to Agent Software Engineering in the following points:

Heterogeneity of Languages : If each group is considered as an interaction space, inside each group we can find specific communication means such as KQML or ACL without modifying system-wide architectures.

Modularity : Organizations can be seen as modules that provide a description to obtain a particular behavior of the members. We can use them to define clear visibility rules that help in the design of MAS.

Multiple Architectures : An organizational approach makes no assumptions about the internal architecture of the agent, thus leaving the specification open for a number of models and implementations

Security of Applications : If all agents communicate without any external control it may lead to security problems. If we allow, when required, each group to control the

[2]http://www.tryllian.org.
[3]http://i-a-i.com/cybelepro
[4]http://mass.aitia.ai
[5]http://www.janus-project.org

access to the roles defined in the group, we can then reach a level of security without the need of a ”global” centralized control.

By considering organizations as blueprints that can be used to define a solution to a problem, we believe that an organizational approach encourages a reusable model.

Among the possible metamodels that define organizational concepts we have chosen CRIO Cossentino et al. [2007]. As stated in Isern et al. [2011], it seems to be, currently, the more complete available. The meaning of completeness here is understood as the expression capability of the underlying concepts. Indeed, the CRIO metamodel was defined specifically to design complex organizations (sometimes holonic) according to structural, functional and behavioral viewpoints. Moreover, CRIO is integrated in a methodological process from analysis to implementation, namely ASPECS Cossentino et al. [2010b, 2013], and is supported by a deployment platform, namely Janus Gaud et al. [2009].

The CRIO metamodel relies upon four main fundamental concepts: Capacity, Role, Interaction and Organization (see Figure 2.1). An organization is composed of Roles, which are abstract behaviors interacting following defined interactions within scenarios while executing their Role plans. An organization has a context that is described in terms of an ontology. Roles participate to the achievement of their organization goals by means of their Capacities.

An organization is defined by a collection of roles that take part in systematic institutionalized patterns of interactions with other roles in a common context. This context consists in a shared knowledge, social rules/norms, social feelings, and it is defined according to an ontology. The aim of an organization is to fulfill some requirements. An organization can be seen as a tool to decompose a system and it is structured as an aggregate of several disjoint partitions. Each organization aggregates several roles and it may itself be decomposed into sub-organizations.

A Role defines an expected behavior as a set of role tasks ordered by a plan, and a set of rights and obligations in the organization context. The goal of each Role is to contribute to the fulfillment of (a part of) the requirements of the organization within which it is defined. Roles use their capacities for participating to organizational goals fulfillment; a Capacity is a specification of a transformation of a part of the designed system or its environment. This transformation guarantees resulting properties if the system satisfies a set of constraints before the transformation. It may be considered as a specification of the pre- and post-conditions of a goal achievement. This concept is a high level abstraction that proved to be very useful for modeling a portion of the system capabilities without making any assumption about their implementations as it should be at the initial analysis stage.

A Capacity describes what a behavior is able to do or what a behavior may require to be defined. As a consequence, there are two main ways of using this concept:

- it can specify the result of some role interactions, and consequently the results that an organization as a whole may achieve with its behavior. In this sense, it is possible to say that an organization may exhibit a capacity.
- capacities may also be used to decompose complex role behaviors by abstracting and externalizing a part of their tasks into capacities (for instance by delegating these tasks to other roles). In this case the capacity may be considered as a behavioral building block that increases modularity and reusability.

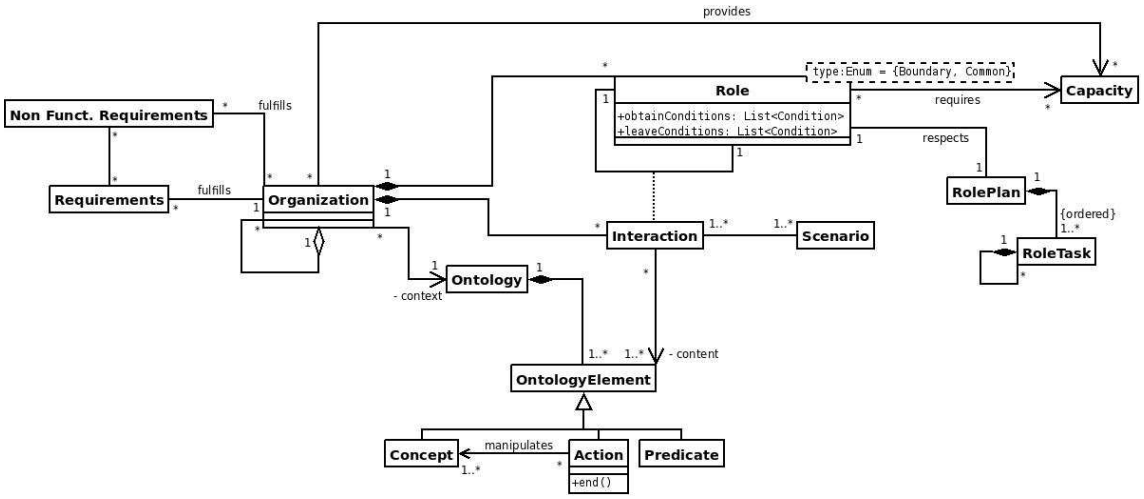

Figure 2.1: Problem Domain of the CRIO metamodel

2.4/ BUSINESS PROCESS IN ENTERPRISES

2.4.1/ AIMS AND OBJECTIVES

Considering the current economic market, in order to achieve the success in product competition, enterprises are often faced with the following questions concerning about developing and producing products /services:

- How would it be possible to reduce the time needed to turn an idea of one customer into a final product/service?
- How it be possible to make the processes to produce product/service more efficient and normative?

Therefore, business processes have been widely recognized as the key capital of enterprises that contributes to enterprise competitiveness and provides the basis for long-term growth, development and existence.

Generally speaking, a business process is any system or procedure that an organization uses to achieve a larger business goal. If we break it down, in enterprises, a business process is a collection of related, structured activities or tasks, executed in a specific order, that produce a specific service or product (serve a particular goal) for a particular customer or customers. In enterprises, business processes are found everywhere and all the time. For example, a business process could be simple as a vendor sells an item to a customer. However, business processes can also be complex. For example, a car production business process.

In the field of software enterprises, business processes are mainly related to software-development processes. They are also called software-development life-cycle (SDLC) or software process, which is a process structure imposed upon the development of a software product. There are several software processes, like Waterfall Model, Spiral model, Iterative and incremental development Model, Agile development Model, Rapid application development Model and so on. Each of these processes describes various kinds

of tasks or activities that take place during the software-development process. These predictable processes provide delivery schedule and planning. People manage projects following these software processes in software enterprises, in order to develop products and avoid delays in products' delivery or budgets that become greater than initial estimations.

Accompanying with rising market competition, business processes are also becoming more and more complex. Therefore, enterprises must require some assistant models, techniques, and software to support business processes, particularly, to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information.

2.4.2/ DEFINITION OF CRITERIA FOR COMPARISON

The first question one must answer is: Considering a model/method related to business processes, what are the desired characteristics ? We have identified the following list:

- It is advisable that a business process model can be understood by the various stakeholders involved in a manner as straightforward as possible. For example, this is achieved through the use of graphical representations.
- It is also advisable that business process models have a formal foundation. Well-known reasons in [Van der Aalst], [Wodtke and Weikum, 1997] include the fact that formal models do not leave any scope for ambiguity, and formal models increase the potential for analysis.
- It is advisable that behaviors in a business process can be explained in terms of a formal semantics. As remarked in [Kiepuszewski et al., 2002], the lack of a formal semantics has resulted in different interpretations by vendors of even basic control flow constructs, definitions in natural language such as the ones provided by the Workflow Management Coalition are not precise enough.

Moreover, each model/method used in business processes is designed and described according to a specific expression mean. Following these different description patterns, we could classify models/methods for business processes by:

- Descriptive Models/Methods, which are described by a descriptive language, and that always come with a graphical representation. For example, the UML Activity Diagram provides a graphical presentation to support design business process;
- Procedural Models/Methods, which are described by a procedural language. For example, XPDL applies a procedural language to automate business processes;
- Formal Models/Methods, which are described by formal language. For example, the MOISE+ Model use a formal language to describe relationships between individuals and organizations in business processes;
- Ontology-based Models/Methods, which are described by an ontology-based/semantic languages, for example, TOVE and BPAL both use a semantic approach to model business processes.

The previous criteria concern general aspects of model/method for business processes. We have also to detail what are the internal features of these model/methods. For example, a complex business process may involve serious element kinds. In order to model/describe a business process, these aspects have to be considered, including:

- Individual, that means a single human being, as distinguished from a group. For modeling individual, the attributes of individuals may be modeled, like abilities of individuals;
- Actor, that means role played by an entity that interacts with the subject. For business processes, it may be considered as a participant;
- Organization Structure, that means the structure of organization (organization means a group of interacting entities organized for some goal or common work common, like enterprise, special department, project team, etc.).
- Behavior, that means the individual behaviors in the human activities in business processes, such as a person sending a mail to other persons.
- Data flow, that means the related parameters about human activities, such as the input parameter and output parameter.
- Workflow Pattern, that means the pattern of all the relational activities in a process, from start to finish. Activities may be triggered by external events or by other activities. The pattern between activities may be sequential, choice, exclusive, iterative, in parallel and so on.

2.4.3/ MODELS/METHODS USED FOR BUSINESS PROCESS

In the following section, we briefly state and discuss some existing methods/models used for modeling business processes, which are organized according to the four groups identified by description pattern (languages) in the previous section. Some models/methods are designed with two description patterns, such as TOVE and PSL Ontology which are generally described by ontological languages, but use formal languages for assistant description. The chosen approach is to classify by their core description pattern.

2.4.3.1/ DESCRIPTIVE MODELS/METHODS

UML Activity Diagram Activity diagrams are graphical representations of work-flows of step-wise activities and actions with support choice, iteration and concurrency. UML (Unified Modeling Language) activity diagrams are typically used for describing business processes, for modeling operational step-by-step work-flows of components within a system or a single use-case, or modeling the detailed logic of a business rule.

For example, the business flow activity of order processing, based on the example from [OMG, 2007] is represented by the diagram of figure 2.2 [6]. The figure presents a RequestedOrder as input object of the sequence and then the possible orders of activities and data exchange.

[6] Please see complete example from http://www.uml-diagrams.org/activity-diagrams-examples.html#process-order

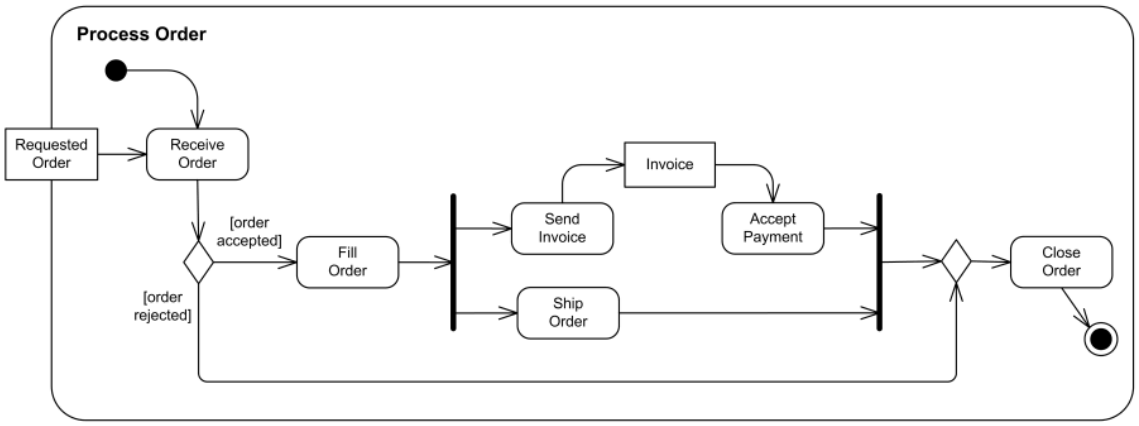

Figure 2.2: UML Activity Diagram: An example of business flow activity to process order

BPMN BPMN stands for Business Process Model and Notation. It is a graphical representation for specifying business processes in a business process model, and a standard for business process modeling notations. BPMN aims to provide a standard notation readily understandable by all business stakeholders. These stakeholders may be the business analysts who create and refine the processes, the technical developers responsible for implementing them, and the business managers that monitor and manage them.

The core constructs defined in BPMN as following, related graphical notations could see official document of BPMN[7].

- Event, including Start Event, Intermediate Event, End Event;
- Activity, including Task, Sub Process, Call Activity;
- Gateway (for decision), including Exclusive Gateway without Marker, Exclusive Gateway with Marker, Inclusive Gateway, Parallel Gateway, Complex Gateway, Event-Based Gateway, Event-Based Gateway to Start a Process, Parallel Event-Based Gateway to Start a Process;
- Flow (Connecting Object), including Sequence Flow, Message Flow, Association, Data Association;
- Data, including Data Object, Data Object Collection, Data Input, Data Input Collection, Data Output, Data Output Collection, Data Store Artifact;
- Artifact, including Group, Text Annotation;
- Swimlane (for actor), including Pool, Lane.

BPMN could be used for very complex business processes. Here, we just take a simple example of a shipment process of a hardware retailer, as presented in Figure 2.3. This example uses only one pool (Hardware Retailer) and different lanes (Warehouse Worker, Clerk, Logistics Manager) for the peoples (roles) involved in this process.

The start event is ”goods to ship”, which indicates that this preparation should be done now. After the instantiation of the process, there are two things done in parallel, as the parallel gateway indicates:

[7]Please refer to http://www.bpmn.org

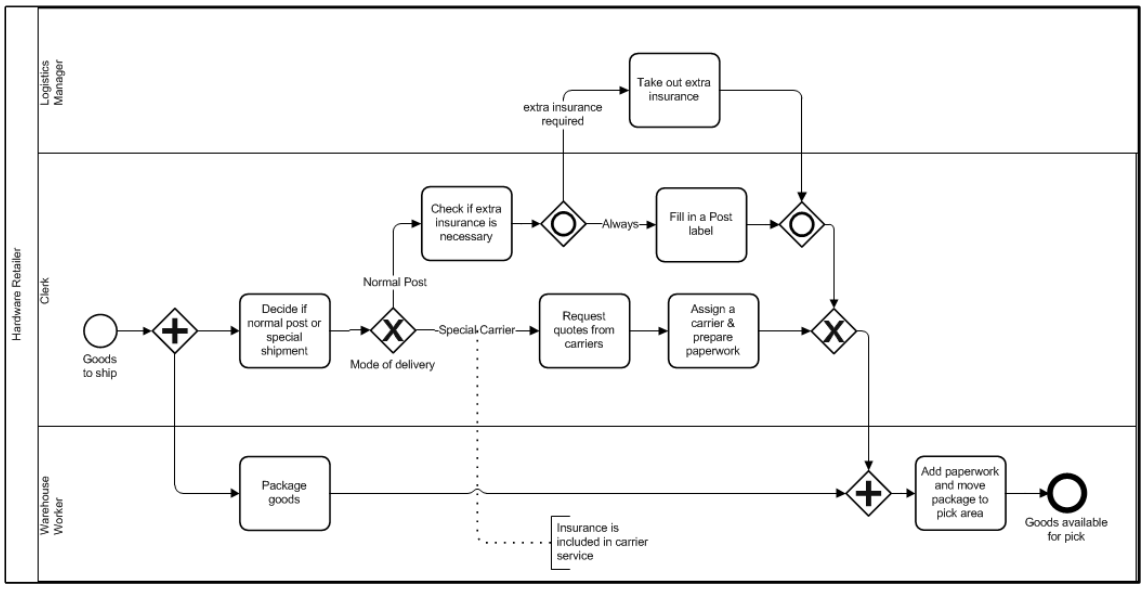

Figure 2.3: BPMN: an Example of a shipment process of a hardware retailer [BPM, 2010]

- the Warehouse Worker can already start packaging the goods.
- the Clerk has to decide whether this is a normal postal or a special shipment. This Clerk's task, which is followed by the exclusive gateway "mode of delivery" (exclusive gateway means only one of the following two branches can be traversed). The gateway only works as a router, which is based on the result of the previous task, and provides alternative paths. These two alternative paths are:
  - If we need a special shipment, the Clerk requests quotes from different carriers, then assigns a carrier and prepares the paperwork.
  - If we need a normal post shipment, the Clerk needs to check if an extra insurance is necessary. If that extra insurance is required, the Logistics Manager has to take out that insurance. In any case, the Clerk has to fill in a postal label for the shipment. Because of this parallelism, there is a synchronizing inclusive gateway right behind "Fill in a Post label" and "Take out extra insurance". In this scenario, the inclusive gateway will always wait for both "Fill in a Post label" and "Take out extra insurance" to be completed.

Furthermore, there is a synchronizing parallel gateway before the latest task " add paperwork and move package to pick the area", to make sure that everything has been fulfilled before the last task is executed.

2.4.3.2/ PROCEDURAL MODELS/METHODS

XPDL The XML Process Definition Language (XPDL) is a WfMC standard for interchanging process models among process definition tools and workflow management systems. It provides the modeling constructs of BPMN and allows a BPMN process to be specified as an XML document. Precisely, XPDL provides a file format that supports every aspect of the BPMN process definition notation, including graphical

descriptions of the diagram, as well as executable properties used at run time. With XPDL, a software can write out a process definition with full fidelity, and another software can read it in and reproduce the diagram that was sent.

BPEL Business Process Execution Language (BPEL), and its extension to Web Services, Web Services Business Process Execution Language (WS-BPEL) is a standard executable language for specifying actions within business processes with web services.

According to BPEL, processes can be described as executable processes, modeling the behavior of a participant in a business interaction, or as abstract processes, specifying the mutually visible message exchange between the parties involved in the protocol, without revealing their internal behavior. To obtain an executable BPEL process, modelers need to specify primitives and structured activities, execution ordering, messages exchanged, fault and exception handling.

There are existing tools for support BPEL, such as BPEL Process Manager by Oracle, BPWS4J by IBM and so on.

2.4.3.3/ FORMAL MODELS/METHODS

MOISE+ Model The MOISE+ Model [Hübner et al., 2002] structure is built up in three levels: one is the behaviors that an agent playing a role is responsible for (individual), the other is the structure and interconnection of the roles with each other (social), and the last is the aggregation of roles in large structures (collective). In MOISE+, as in MOISE, three main concepts, roles, role relations, and groups, are used to build, respectively, the individual, social, and collective structural levels of an organization. Furthermore, the MOISE original structural dimension is enriched with concepts such as inheritance, compatibility, cardinality, and sub-groups.

- **Individual level** is formed by the roles of the organization. A role means a set of constraints that an agent ought to follow when it accepts to enter a group playing that role. These constraints are defined in two ways: in relation to other roles (in the collective structural level) and in a deontic relation to global plans (in the functional dimension). In order to simplify the specification, like in Object-Oriented (OO) terms, there is an inheritance relation between roles. If a role p' inherits a role p (denoted by $p \subset p'$), with $p \neq p'$, p' receives some properties from p, and p' is a sub-role, or specialization, of p. In the definition of the role properties presented in the sequence, it will be precisely stated what one specialized role inherits from another role. For example, in the soccer domain, the attacker role has many properties of the player role ($p_{player} \subset p_{attacker}$). It is also possible to state that a role specializes more than one role, i.e., a role can receive properties from more than one role. The set of all roles are denoted by R_{ss}. Following this OO inspiration, we can define an abstract role as a role that cannot be played by any agent. It has just a specification purpose. The set of all abstract roles is detonated by $R_{abc}(R_{abc} \subset R_{ss})$. There is also a special abstract role $P_{soc}$where $\forall_{(p \in R_{ss})} P_{soc} \subset P$, trough the transitivity of $\subset$, all other roles are specializations of it.
- **Social level** defines other kinds of relations between roles that directly constrain the agents. Those relations are called links and are represented by the

predicate $link_{(p_s,p_d,t)}$ where p_s is the link source, p_d is the link destination, and $t \in acq, com, aut$ is the link type. In case the link type is acq (acquaintance), the agents playing the source role p_s are allowed to have a representation of the agents playing the destination role p_d (p_d agents, in short). In a communication $link(t = com)$, the p_s agents are allowed to communicate with p_d agents. In an authority $link(t = aut)$, the p_s agents are allowed to have authority on p_d agents, i.e., to control them. An authority link implies the existence of a communication link that implies the existence of an acquaintance link:

$$link_{p_s,p_d,aut} \Longrightarrow link_{p_s,p_d,com} link_{p_s,p_d,com} \Longrightarrow link_{p_s,p_d,acq} \tag{2.1}$$

Regarding the inheritance relation, the links follow the rules:

$$(link_{(p_s,p_d,t)} \wedge p_s \subset p_{s'}) \Longrightarrow link_{(p_{s'},p_d,t)} (link_{(p_s,p_d,t)} \wedge p_s \subset p_{d'}) \Longrightarrow link_{(p_s,p_{d'},t)} \tag{2.2}$$

For example, if the coach's role has authority on the player's role $link_{(p_{coah},p_{player},aut)}$ and player has a sub-role $(p_{player} \subset p_{attacker})$, by Eq. (4), a coach has also authority on attackers. Moreover, a coach is allowed to communicate with players (by Eq. (1)) and it is allowed to represent the players (by Eq. (2)).

- **Collective level**: the links constrain the agents after they have accepted to play a role. However, we should constrain the roles that an agent is allowed to play depending on the roles this agent is currently playing. This compatibility constraint $p_a \bowtie p_b$ states that the agents playing the role are also allowed to play the role p_b (it is a reflexive and transitive relation). As an example, the team leader role is compatible with the back player role $p_{leader} \bowtie p_{back}$. If it is not specified that two roles are compatible, by default they are not. Regarding the inheritance, this relation follows the rule

$$(p_a \bowtie p_b \wedge p_a \neq p_b \wedge p_a \sqsubseteq p') \Longrightarrow (p' \bowtie p_b) \tag{2.3}$$

Hence, there should be a series of rules and relationships defined within the collective level.

AIC Model The AIC Model represents such a system as a tripartite graph with hyperedges [Mika, 2007]. The set of vertices is partitioned into the three (possibly empty) disjoint sets $A = \{a_1, a_2, ..., a_k\}$, $C = \{c_1, c_2, ..., c_i\}$, $I = \{i_1, i_2, ..., i_m\}$ corresponding to the set of actors (users), the set of concepts (tags, keywords) and the set of annotated objects (bookmarks, photos, etc.). It extends the traditional bipartite model of ontology (concepts and instances) by incorporating actors in the model. In a social tagging system, users' tag objects with concepts, creating ternary associations between the user, the concept and the object. Thus, the folksonomy is defined by a set of annotations $T \in A \times C \times I$. Such a network is most naturally represented as a hypergraph with ternary edges, where each edge represents the fact that a given actor is associated with a certain instance and a certain concept. In particular, the author defines the representing hypergraph of a folksonomy T as a (simple) tripartite hypergraph $H(T) = (V, E)$ where $V = A \cup C \cup I$, $E = \{\{a, c, i\} \mid (a, c, i) \in T\}$.

2.4.3.4/ ONTOLOGY-BASED MODELS/METHODS

TOVE One of the earliest initiatives was the TOVE project that aimed at developing a set of integrated ontologies for modeling all kinds of enterprise, such as commercial

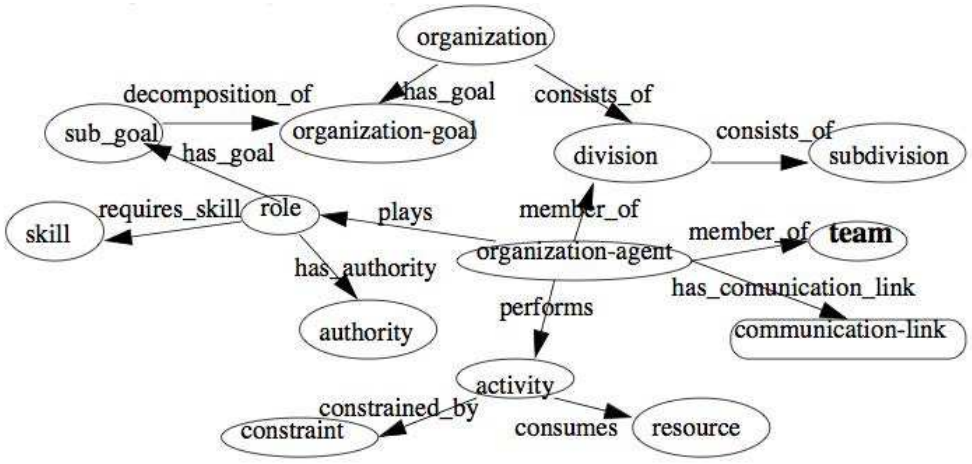

Figure 2.4: Organizational object taxonomy in TOVE [Fox et al., 1997]

and public ones. The TOVE Organization Ontology [Fox et al., 1996] for Enterprise Modeling is one of these, which puts forward a number of conceptualizations such as agents, roles, positions, goals, communication, authority, commitment. Precisely, one organization consists of a set of Organization-Agents (OA) having two sub-classes: Individual-Agents and Group-Agents, a set of Organization-Units as recursive sub-components and an Organization-Goal tree (that could divide goals into sub-goals). An Organization-Role defines a prototypical function of an agent in an organization. Each Organization-Role played by OA has Organization-Goals or Role-Goals, Role-Skills, Role-Process or Organization-Activity, Role-Policy, Role-Communication-Link. Moreover, an Organization-Position defines a formal position that can be filled by OA in the Organization. The figure 2.4 represents a taxonomy of organizational object in TOVE.

Ontologies based upon SPEM The approach proposed in Rodríguez et al. [2010] defines Ontologies based upon SPEM Meta-Model with Ontology Web Language. The software & Systems Process Engineering meta-model (SPEM) allows the Modeling of software processes using OMG (Object Management Group) standard such as the MOF (meta-object facility) and UML: making possible to represent software processes using tools compliant with UML. Rodríguez et al. [2010] represents generic processes modeled with SPEM using an underlying ontology based upon the OWL representation together with data derived from actual projects.

SPEM is generally used to design generic software processes such as Scrum. Therefore, Rodríguez et al. [2010] discussed a first approach to create an ontology from the Scrum process as example using SPEM. In [Rodríguez et al., 2010], Scrum Ontology extends from the *Role* class in the method-content ontology in SPEM. *Method-Content: Role* has *scrum:Pig* and *scrum:Chicken* as sub-classes. The *productOwner*, *scrumMaster* and *team* were instances of the *Scrum Chicken role*. Moreover *WorkProduct* as one class in the method content ontology has three sub-classes: *Artifact*, *Deliverable* and *Outcome*. The term like SprintBacklog is as an instance or individual of *Artifact* as the part of the scrum Ontology. Furthermore, the process ontology of SPEM has *Activity* class with *Iteration* and *Phase* classes defined as sub-classes. Relatively, the *PreGame*, *Game* and *PostGame* are defined in the scrum ontology.

BPAL BPAL (Business Process Abstract Language) is an ontological framework for business process, in the context of BPMN business process modeling method. It is primarily conceived to provide a formal semantics to BPMN, an informal business process modeling method that is emerging in the business world [Nicola et al., 2007].The modeling categories of BPAL are based on well accepted business notions, such as *activity*, *decision*, *role*. And within BPAL, relationships among these notions are also defined, such as

- *pre(activity/decision, activity/decision)*: a precedence relation between activities, decisions, or an activity and a decision,
- *perf(role, activity)*: a relation that indicates which role/roles are dedicated to which activities,
- *msg(obj, sourceNode, destNode)*: a message, characterized, for instance, by a content (obj), a sending activity (sourceNode), and a receiving activity (destNode).

PSL Ontology The Process Specification Language (PSL) is a set of logic terms used to describe processes. The primary component of PSL is an ontology designed to represent the primitive concepts that, according to PSL, are adequate for describing basic manufacturing, engineering, and business processes [Schlenoff et al., 1999b]. Precisely, the PSL ontology provides a rigorous mathematical characterization of process information as well as a precise expression of the fundamental logical properties of that information in the PSL language [Schlenoff et al., 1999a], and the PSL Ontology consists of a set of first-order logic theories within which there is a distinction between core theories and definitional extensions [8]. All core theories within the ontology are consistent extensions of PSL Core. Within PSL Ontology, four kinds key elements in PSL Core are as:

- Activity: a class or type of action.
- Activity-Occurrence: an event or action that takes place at a specific place and time. An instance or occurrence of an activity.
- Timepoint: a point in time.
- Object: anything that is not a timepoint or an activity.

The axioms among these four elements are as everything is either an activity, an activity-occurrence, an object, or a timepoint, and activities, activity-occurrences, objects, and timepoints are all distinct kinds of things. Within PSL Ontology, primitive relations for the PSL Core are as:

- Before (timepoint, timepoint): this relation is used to impose a total ordering on timepoints. For example, *Before(t1,t2)*: time point *t1* precedes time point *t2* on the time line.
- Occurrence-of (activity-occurrence, activity): the activity-occurrence is a particular occurrence of the given activity.
- Participates-in (object, activity-occurrence, timepoint): the given object plays some role in the given occurrence of an activity at the given timepoint.

[8] The complete set of axioms for the PSL Ontology can be found at http://www.mel.nist.gov/psl/psl-ontology/.

All the concepts, relationships and functions in PSL core are explained in [Schlenoff et al., 1999a].

DOLCE DOLCE [C. Masolo and Schneider, 2002] is an important existing work about ontological model. It stands for Descriptive Ontology for Linguistic and Cognitive Engineering, which is implemented by First Order Logic, KIF, OWL languages. The taxonomy of the most basic categories of particulars assumed in DOLCE includes, for example, endurant, perdurant/occurrence, quality, quality region, abstract entities.

Different with all the above ontological models, DOLCE is not an ontology, which is specially for modeling business processes. Oppositely, it is a top-level ontology, which may describe very general concepts that are the same across all domains. Accroding to the level of dependence on a particular task or point of view, [Guarino and Guarino, 1997] distinguished between top-level ontologies, domain ontologies, task ontologies and application ontologies:

- Top-level ontologies describe very general concepts like space, time, matter, object, event, action, etc., which are independent of a particular problem or domain: it seems therefore reasonable, at least in theory, to have unified top-level ontologies for large communities of users.
- Domain ontologies and task ontologies describe, respectively, the vocabulary related to a generic domain (like medicine, or automobiles) or a generic task or activity (like diagnosing or selling), by specializing the terms introduced in the top-level ontology.
- Application ontologies describe concepts depending both on a particular domain and task, which are often specializations of both the related ontologies. These concepts often correspond to roles played by domain entities while performing a certain activity, like replaceable unit or spare component.

Hence, there are some existing special ontologies extending foundational ontology DOLCE. Precisely, [Bottazzi et al., 2005] have presented a preliminary proposal of an ontology of organizations based on DOLCE. Compared to [Bottazzi et al., 2005], K-CRIO is an organizational ontology specially for modeling/conceptualizing business processes. Therefore, K-CRIO is considering not only structural aspects, but also many more entities about processes (workflows), like interaction patterns, data-flow, etc.

	Model	Description	Scope of modeling						Related Tool
			Individual	Actor	Organization Structure	Behaviors	Data Flow	Workflow Pattern	
Descriptive Model/Method	UML Activity Diagram	Graphical representations of work-flows of step-wise activities and actions with support for choice, iteration and concurrency.		Yes		Yes	Yes	Yes	Rational Rose, Vision, PowerDesign
	BPMN	Graphical representation for specifying business processes. Provide a standard notation readily understandable by all business stakeholders.		Yes		Yes	Yes	Yes	XPDL
Procedural Model/Method	XPDL	Provide a file format that supports every aspect of the BPMN process definition notation including graphical descriptions of the diagram, as well as executable properties used at run time.		Yes		Yes	Yes	Yes	
	BPEL	A standard executable language for specifying actions within business processes. With BPEL, processes can be described as executable processes, modeling the behavior of a participant in a business interaction, or as abstract processes.		Yes		Yes	Yes	Yes	Oracle BPEL Process Manager, IBM BPWS4J
Formal Model/Method	AIC Model	A tripartite relationship with actors (users), concepts (tags, keywords) and annotated objects (bookmarks, photos etc).	Yes						
	MOISE+ Model	The behaviors that an agent playing a role is responsible for (individual); the structure and interconnection of the roles with each other (social), and the aggregation of roles in large structures (collective)	Yes		Yes				
Ontology-based Model/Method	TOVE	An Organization Ontology for Enterprise Modeling, which puts forward a number of conceptualizations as agents, roles, positions, goals, communication, authority, commitment.	Yes	Yes	Yes	Yes	Yes		
	Ontologies based on SPEM	Design/Build Ontologies based upon SPEM Meta-Model with Ontology Web Language.		Yes		Yes	Yes	Yes	
	BPAL	An ontological framework for business process, in the context of BPMN business process modeling method.		Yes		Yes	Yes	Yes	
	PSL Ontology	Provide a rigorous mathematical characterization of process information as well as precise expression of the basic logical properties of that information in the PSL language, for describing basic manufacturing, engineering, and business processes.		Yes		Yes	Yes	Yes	

Table 2.1: Summary and Comparison of Business Process Models and Methods

Lastly, we summarize and compare previously presented methods/models in Table 2.1. None of the existing methods do cover all the criteria considered advisable for a complete Business Process modeling approach. From the table, we could see that, although there are serious existing models/methods described with different features, used for modeling/conceptualizing business processes, these models/methods lack abilities for modeling/conceptualizing one or more aspects in business processes. This simply means that their scope of modeling couldn't cover all the involved elements in business processes. For example, XPDL lacks the ability for modeling individual and organization structure in business processes. TOVE could model organization structure and individual but can not express workflow pattern. Furthermore, as the top-level ontology, DOLCE is general. However, as the ontological model specially for conceptualizing business processes, it is too general for providing the actual support to human activities in business processes. Therefore, these above observations led us to develop the K-CRIO ontology and its associated approach in an attempt to model/conceptualize all the aspects of business processes, including individual, actor, organization structure, behaviors, data flow and work pattern.

2.5/ CONCLUSION

This chapter has presented a state of the art in terms of enterprise models, ontologies and MAS. The contributions of this thesis are at the convergence of these three domains. Indeed, among the hypothesis of this work, we have emphasized in this chapter the need of rich and semantic models of enterprises in order to build assistance tools.

Among the possible representations of such rich and semantic models, ontologies exhibit satisfying features and expressive capabilities. Specifically, the OWL language has been chosen.

In order to design assistance tools that can support business processes in a transparent way, MAS is a fitted paradigm. We have thus presented the necessary background to understand what are the building blocks of the tool that is one of the contributions of this thesis.

The following chapters will define respectively: the K-CRIO ontology, that is our contribution in terms of an enterprise model, the conceptualization of SCRUM methodology with K-CRIO and a SCRUM assistance tool based on MAS.

II

An Organizational Ontology and Case Studies

3

K-CRIO ONTOLOGY

3.1/ INTRODUCTION

The K-CRIO Ontology is an Ontology dedicated to the study of organizations. In other words, it is used to understand, analyze and reason about organizations. The targeted organizations are those composed of individuals involved in the design of a product and, to do so, following a business process. This ontology is to be used to support process assistance within the described organizations. More specifically, the ontology could provide means for reasoning, annotating resources, monitoring business processes, enabling searches and pro-actively proposing tips and proper content. This kind of approach was already deployed, for example, in [Kiesel et al., 2008, Limpens et al., 2008], in the knowledge management field.

The ontology defined in this chapter is based upon the elements of an existing organizational meta-model, namely CRIO, already used for the description of Multi-Agent Systems (MAS) organizations [Cossentino et al., 2007]. In our case, the concepts of this meta-model are used to model human activities.

The definition of K-CRIO was a two steps process. First, the concepts of the CRIO meta-model that were identified as pertinent for the description for our objectives are conceptualized through an ontology formalism, namely OWL [McGuinness and Van Harmelen]. Second, the result of this conceptualization is enriched by a set of classes and relationships that cover the gaps for human processes' description.

This chapter is organized as follows, in Section 3.2, the basic meta-model CRIO will be shortly presented. After that, Section 3.3 defines the K-CRIO Ontology. Using K-CRIO a simple example related to Software-Development Process, specifically the Waterfall process, is presented in Section 3.4. Finally, Section 3.5 concludes.

3.2/ BACKGROUND: THE CRIO META-MODEL

The CRIO meta-model relies upon four fundamental concepts: Capacity, Role, Interaction and Organization (see Figure 2.1). An organization is composed of Roles, which are abstract behaviors interacting following defined interactions within scenarios while executing their Role plans. An organization has a context that is described in terms of an ontology. Roles participate in the achievement of their organization goals by their Capacities.

Definition 1 *An organization is defined by a collection of roles that take part in systematic*

institutionalized patterns of interactions with other roles in a common context. This context consists in shared knowledge and social rules/norms, social feelings, and is defined according to an ontology. The aim of an organization is to fulfill some requirements.

An organization can be seen as a tool to decompose a system, and it is structured as an aggregate of several disjoint partitions. Each organization aggregates several roles and it may itself be decomposed into sub-organizations.

Definition 2 *A role is an expected behaviour (a set of role tasks ordered by a plan) and a set of rights and obligations in the organisation context. The goal of each Role is to contribute to the fulfilment of (a part of) the requirements of the organisation within which it is defined.*

Inside organizations, roles are supposed to interact. These interactions denote whatever happen between roles of a same organization.

Definition 3 *An interaction is a dynamic, not a priori known sequence of events (a specification of some occurrence that may potentially trigger effects on the system) exchanged among roles, or between roles and entities outside the agent system to be designed. Roles may react to the events according to theirs behaviours.*

The concept of capacity abstracts a specific know-how. It is a high level abstraction that proved to be very useful for modeling a portion of the system capabilities without making any assumption about their implementations as it should be at the initial analysis stage.

Definition 4 *A capacity is the specification of a transformation of a part of the designed system or its environment. This transformation guarantees resulting properties if the system before the transformation satisfies a set of constraints. It may be considered as a specification of the pre- and post-conditions of a goal achievement.*

Roles use their capacities for participating to organizational goals' fulfillment.

A Capacity describes what a behavior can do or what a behavior may require to be defined. As a consequence, there are two main ways of using this concept:

- it can specify the result of some role interactions, and consequently, the results that an organization as a whole may achieve with its behavior. In this sense, it is possible to say that an organization may exhibit a capacity.
- capacities may also be used to decompose complex role behaviors by abstracting and externalizing a part of their tasks into capacities (for instance, by delegating these tasks to other roles). In this case, the capacity may be considered as a behavioral building block that increases modularity and re-usability.

Let's take an example to illustrate these concepts. If we have a *Motion control* organization composed of two roles: *Route requester* role that asks for a route between two locations and *Route provider* role that provides routes. The capacity to find the shortest path in a weighted directed a cyclic graph $\mathcal{G}(N, E)$, from a source node to a destination

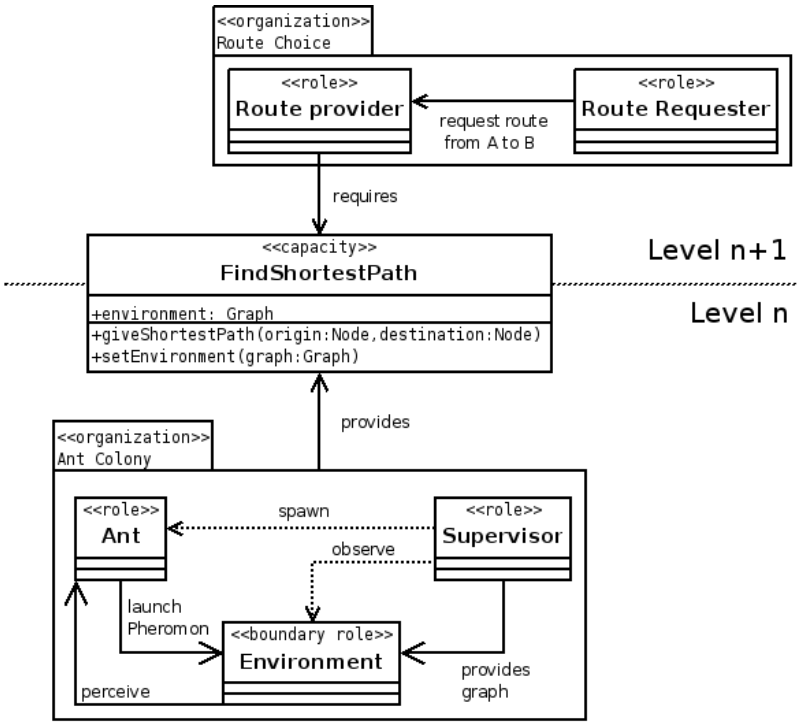

Figure 3.1: Example of Organizations, Roles and Capacity diagram

node d may be required by the role *Route provider*. This capacity may be realized in various ways, by using classical graph algorithms, if the know-how of a single entity is considered, or it can also be modeled by an organization. The *Ant Colony* is a well-known organization able to find a solution to the problem of finding the shortest path in a graph [Attiratanasunthron and Fakcharoenphol, 2008]. The solution (the shortest path) emerges from interactions between Ants in their environment. Let us suppose that the environment represents the graph $\mathcal{G}$, the source node s is mapped to the Ant Hill and the destination d to a food source. At the level $n+1$, the *Route Choice* organization is responsible for providing the best route between two given points to another organization (for instance the *Motion Control* organization). This capacity provides the solution of a problem that is effectively solved at a lower level of abstraction (level n). The ant colony organization that is located at a lower level in the organizational hierarchy can realize this capacity. The capacity concept thus allows to define how an organization at level n may contribute to the behavior of a role at level $n+1$.

3.3/ DEFINITION OF THE K-CRIO ONTOLOGY

The K-CRIO Ontology is based upon the elements of an existing organizational meta-model, namely CRIO [Cossentino et al., 2007], already used for the description of MAS organizations. In the case of K-CRIO Ontology, the concepts and relationships of this meta-model are used to model human activities. K-CRIO stands for Knowledge-Capacity Role Interaction Organization.

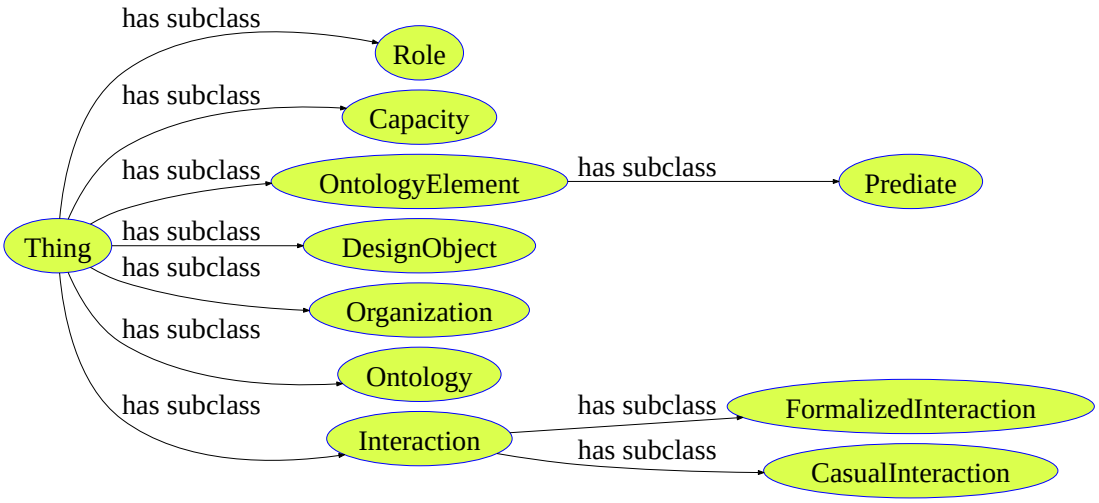

Figure 3.2: K-CRIO taxonomy

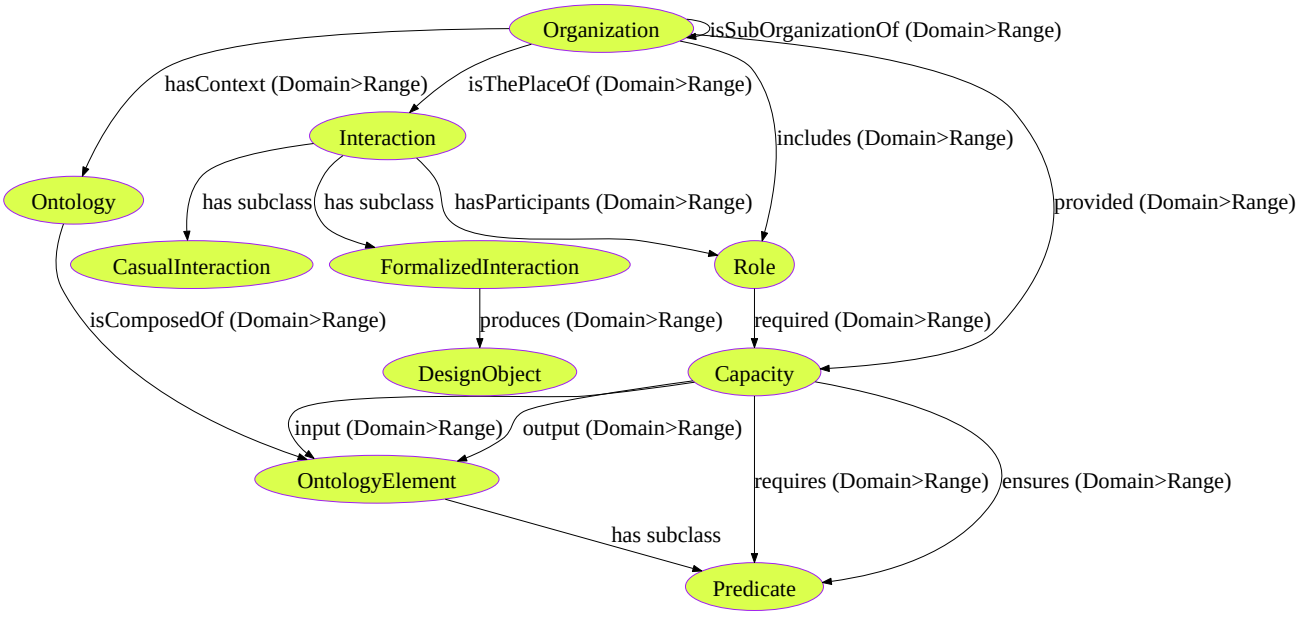

Figure 3.3: K-CRIO Ontology

The core content of K-CRIO is presented in Figure 3.2 from the point of view of a taxonomy and in Figure 3.3 from the relationships point of view.

In the following section, we will detail the conceptualization of the different K-CRIO concepts and their relationships respectively.

3.3.1/ ORGANIZATION

The targeted organizations are those dedicated to product design. We have then defined a concept named DesignObject (`owl:class`) which is the inheritance root class of all possible products that an organization can produce.

An organization can be seen as a set of interacting entities: sub-organizations or roles, which are regulated by social rules and norms. Taking a simple instance, an university could be seen as one organization. In this organization, the different departments could be also seen as organizations and more specifically university sub-organizations. As example, one can cite: Faculty of Computer Science, Faculty of Art, Faculty of Humanities, and so on. These organizations, for example, Faculty of Computer Science, are composed by some roles as student, professor, associate professor, lecturer, president of faculty, etc. All of these entities define both the university organizational structure and the global objectives that the university should fulfill. Therefore, we analyze the concept Organization from these two aspects.

- With respect to the ontology, an organization may be seen as a concept connected to other concepts by various kinds' relationships, such as *hierarchical* relations between organizations and sub-organizations, or *composed of* relation between an organization and its roles.
- With respect to the human processes in enterprises, an organization may be considered as a collective global system able to achieve particular goals through its collaborative members.

```
<owl:Class rdf:ID="Organization"/>
<owl:Class rdf:ID="Ontology"/>
<owl:Class rdf:ID="Capacity"/>
<owl:Class rdf:ID="Role"/>
<owl:Class rdf:ID="Interaction"/>
<owl:ObjectProperty rdf:ID="hasContext">
   <rdfs:range rdf:resource="#Ontology"/>
   <rdfs:domain rdf:resource="#Organization"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="includes">
   <rdfs:range rdf:resource="#Organization"/>
   <rdfs:domain rdf:resource="#Role"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="provided">
   <rdfs:range rdf:resource="#Capacity"/>
   <rdfs:domain rdf:resource="#Organization"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isThePlace">
   <rdfs:range rdf:resource="#Oraganization"/>
   <rdfs:domain rdf:resource="#Interaction"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isSubOrganizationOf">
   <rdfs:range rdf:resource="#Organization"/>
   <rdfs:domain rdf:resource="#Organization"/>
</owl:ObjectProperty>
```

Listing 3.1: Organization in K-CRIO

In the rest of this paper, we denote these definition with the following logical language:

Organization(X) means that *X* is an Organization.
DesignObject(X) means that *X* is a DesignObject.
Role(X) means that *X* is a Role.
Capacity(X) means that *X* is a Capacity.
Ontology(X) means that *X* is an Ontology.
Interaction (X) means that *X* is an Interaction.
isSubOrganizationOf (X,Y) means that *X* is a sub-Organization of *Y* and {*X: Organization(X), Y:Organization(Y)*}.
includes (X,Y) means that *X* includes Y and {*X: Organization(X), Y: Role(Y)*}.
provided (X,Y) means that *X* includes Y and {*X: Organization(X), Y: Capacity(Y)*}.
hasContext (X,Y) means that *X* hasContext Y and {*X: Organization(X), Y: Ontology(Y)*}.
isThePlaceOf (X,Y) means that *X* is the place of *Y* happening and {*X: Organization(X), Y: Interaction(Y)*}.

Presented in Figure 3.2, we also define some necessary restrictions between Organization and associated classes, which are represented by qualified cardinality restrictions in OWL (cardinality constraints, including `owl:cardinality`, `owl:minCardinality`, `owl:maxCardinality`[1]).
Specifically, one Organization may have an unspecified number of sub-organizations, (zero or more). Moreover, the *ObjectProperty: isSubOrganizationOf* is a transitive ObjectProperty. For example, on the one hand, if we consider an university as one Organization, different departments may be seen as its sub-organizations. On the other hand, one department is usually composed of diverse majors, which may be seen as sub-organizations of this department. Because of transitivity, these majors are also the sub-Organizations of the university. Expressed by an owl:Restriction, the sub-Organization "isSubOrganizationOf" its Organization with minCardinality 0 (`owl:restriction`). The logical expression of the transitivity of `ObjectProperty:isSubOrganizationOf` is:

$$isSubOrganizationOf(O_1, O_2),\ isSubOrganizationOf(O_2, O_3) \Rightarrow isSubOrganizationOf(O_1, O_3)$$

Moreover, one Organization must include one Role at least (inversely, one Role must be included by one Organization at least). Moreover, one (or more) interactions must occur in an organization. That means, describing with an `owl:Restriction`, that Organization "includes" Role with minCardinality 1 (`owl:restriction`) and "isThePlaceOf" interaction happening with minCardinality 1 (`owl:restriction`). The logic axiomatization of these facts are:

$$\forall O, Organization(O) \Rightarrow \exists R, \{Role(R) \wedge includes(O, R)\}$$

$$\forall R, Role(R) \Rightarrow \exists O, \{Organization(O) \wedge includes(O, R)\}$$

$$\forall O, Organization(O) \Rightarrow \exists I, \{Interaction(I) \wedge isThePlaceOf(O, I)\}$$

```
<owl:Class rdf:ID="Organization">
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</owl:minCardinality>
         <owl:onProperty>
            <owl:ObjectProperty rdf:ID="isSubOrganizationOf"/>
         </owl:onProperty>
      </owl:Restriction>
   </rdfs:subClassOf>
   <rdf:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing">
```

[1] http://www.cs.vu.nl/~guus/public/owl-restrictions/

```
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:minCardinality>
            <owl:onProperty>
                <owl:ObjectProperty rdf:ID="includes"/>
            </owl:onProperty>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty>
                <owl:ObjectProperty rdf:ID="isSubOrganizationOf"/>
            </owl:onProperty>
            <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:minCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Listing 3.2: Restrictions of Organization in K-CRIO

3.3.2/ ROLE

A role may identify a person, status or generic behavior. A role is a necessary part to achieve social objectives (goals of its organization). In order to fulfill this common target, each role of an organization may have specific individual capacities. An Organization (an `owl:class`) ”includes” (`owl:ObjectProperty`) Roles (an `owl:class`) which may ”required” (`owl:ObjectProperty`) Capacity (an `owl:class`). Following there are two expressions respectively as expressed by OWL in Figure 3.3 and logic language expression:

```
<owl:Class rdf:ID="Role"/>
<owl:ObjectProperty rdf:about="#includes">
        <rdfs:domain rdf:resource="#Organization"/>
    <rdfs:range rdf:resource="#Role"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="required">
    <rdfs:range rdf:resource="#Capacity"/>
    <rdfs:domain rdf:resource="#Role"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasParticipants">
    <rdfs:range rdf:resource="#Role"/>
    <rdfs:domain rdf:resource="#Interaction"/>
</owl:ObjectProperty>
```

Listing 3.3: Role in K-CRIO Ontology

required(X,Y) means *X* required *Y* and {*X: Role(X), Y: Capacity(Y)*}.

Moreover, an Interaction (`owl:class`) occurring in an Organization ”hasParticipants” (`owl:ObjectProperty`) that are Roles.

hasParticipants(X,Y) means the *X* hasParticipants *Y* happening and {*X: Interaction(X), Y: Role(Y)*}.

3.3.3/ CAPACITY

A capacity is a know-how and ability, which may be considered as an interface between the role, as generic behaviors, and its role-players. Capacities may be required by a role or provided by an organization to define their respective behaviors.

A Capacity is represented by an `owl:class`. This class is related to some *ContextOntologyElement* (which are parts of the Organization Ontology defining it context) divided into two sets ”input” and output” (both are `owl:ObjectProperty`). The Capacity class is also related to properties, which are conceptualized by the Predicate concept. A predicate is an assertion on a property of some *ContextOntologyElement*. There are two types of relationships between Capacity and Predicate. The first type is named ”`requires`” and represents the properties that are required by the capacity. The second type is named ”`ensures`” and represents the effects of the capacity. The whole concept Capacity and its related relationships could be expressed by OWL in Figure 3.4.

```
<owl:ObjectProperty rdf:ID="provided">
    <rdfs:range rdf:resource="#Capacity"/>
    <rdfs:domain rdf:resource="#Organization"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="ensures">
    <rdfs:domain rdf:resource="#Capacity"/>
    <rdfs:range rdf:resource="#Prediate"/>
</owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="required">
    <rdfs:domain rdf:resource="#Role"/>
    <rdfs:range rdf:resource="#Capacity"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="output">
    <rdfs:range rdf:resource="#Capacity"/>
    <rdfs:domain rdf:resource="#OntologyElement"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="input">
     <rdfs:range rdf:resource="#OntologyElement"/>
    <rdfs:domain rdf:resource="#Capacity"/>
</owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="requires">
    <rdfs:domain rdf:resource="#Capacity"/>
    <rdfs:range rdf:resource="#Prediate"/>
</owl:ObjectProperty>
```

Listing 3.4: Capacity in K-CRIO Ontology

3.3.4/ INTERACTION

As already identified in numerous works, such as [Matta et al., 2010], interactions are an essential component for modeling human collaborative processes. The aim of an interaction in this context is to achieve a goal or to contribute to a goal of one organization.

In this thesis a partition in two different types of interactions is proposed. These types are respectively *Casual Interaction* and *Formalized Interaction*. The former ones depict interactions between roles, which do not follow any specified pattern or sequence of activities in the model and that can take place in a non determined fashion. This kind of interactions usually do not produce any *DesignObject*, at least no identified ones. Among the possible examples of these interactions one can cite: chatting or mails exchange between different roles.

Definition 5 *A Casual Interaction is any interaction that is not specified by a sequence of events and that do not produce any DesignObject.*

On the contrary, formalized interactions identify, in a pre-determined way (frequently described in business processes), how different roles belonging to the same organization can interact with each other to achieve the common goal of the organization and in the meantime produce *DesignObject* of any sort.

Definition 6 *A Formalized Interaction is any interaction that satisfies the two conditions:*

1. *is specified by a sequence of events,*
2. *produce at least one DesignObject,*

The distinction between these two types of interactions is introduced to distinguish interactions that will be reasoned upon, the Formalized Interactions, and those that are either unknown or of no interest for the task at hand.

From the perspective of the K-CRIO ontology definition, *Formalized Interaction* and *Casual Interaction* are both defined as `owl:class`. These two classes are sub-classes of the Interaction (`owl:class`):

CasualInteraction(X) means *X* is CasualInteraction.
FormalizedInteraction(X) means *X* is FormalizedInteraction.

$$Interaction(X) \equiv CasualInteraction(X) \cup FormalizedInteraction(X)$$

The *FormalizedInteraction* class is related by the ”`produces`” (`owl:ObjectProperty`) relationships to the ”`DesignObject`” (an `owl:class`) concept:

produces(X,Y) means *X* produces *Y* and $\{$*X: FormalizedInteraction(X), Y: DesignObject(Y)*$\}$.

It may be expressed by OWL as detailed in Figure 3.5.

```
<owl:Class rdf:ID="Interaction"/>
<owl:Class rdf:ID="FormalizedInteraction">
   <rdfs:subClassOf rdf:resource="#Interaction"/>
</owl:class>
<owl:Class rdf:ID="CasualInteraction">
   <rdfs:subClassOf rdf:resource="#Interaction"/>
</owl:class>
<owl:ObjectProperty rdf:ID="produces">
   <rdfs:range rdf:resource="#DesignObject"/>
      <rdfs:domain rdf:resource="#FormalizedInteraction"/>
</owl:ObjectProperty>
```

Listing 3.5: Interaction in K-CRIO

In the following section, we define Interaction, in K-CRIO, from two different perspectives. Interaction may be seen as a description of possible patterns of actions and exchanges between the participant roles. This vision is similar to a workflow, the definition of which is in Section 3.3.4. On the other hand, we should consider about the state and the time-table of Interactions, the definition of which is in Section 3.3.4.

DEFINING INTERACTION FROM THE ASPECT OF WORKFLOW

FormalizedInteraction, seen as a description of possible patterns of actions and exchanges between the participant roles can be conceptualized by workflows. The concept of workflow is quite wide. In this thesis we have adopted the following definition of Beco et al. [2005]:

Definition 7 *Workflow can be defined as the orchestration of a set of activities to accomplish a larger and sophisticated goal. Examples of workflow include application processes, business processes, and infrastructure (e.g. "behind the scenes") processes.*

One can see that this definition fits well with the already defined *FormalizedInteraction*. In order to conceptualize *FormalizedInteraction*, we have thus chosen to reuse and improve an existing ontology, OWL-WS, dedicated to the description of workflow [Beco et al., 2005] and inspired by OWL-S (Semantic Markup for Web Services, [Martin et al., 2004]). OWL-WS is based on the assumption that a workflow is a kind of complex service and therefore it can be represented in OWL-WS as a full OWL-S Service. This service can be a simple one or composed of simpler services using the OWL-S control constructs such as: `Sequence`, `RepeatUntil`, `Split`, etc.

The reason behind the choice of such an (sub-)ontology for describing *FormalizedInteraction* is twofold. First, it allows to describe and specify the possible sequences of events that take place during a *FormalizedInteraction*. Considering this aspect only the contribution is not much as many formalisms, such as BPEL or BPMN, allow this kind of representations. Second, it allows to manipulate and reason upon the process in it workflow, as defined in this thesis, perspective. This last aspects is of interest as it is one of the goal of this thesis, defining an ontology for reasoning about productive organizations.

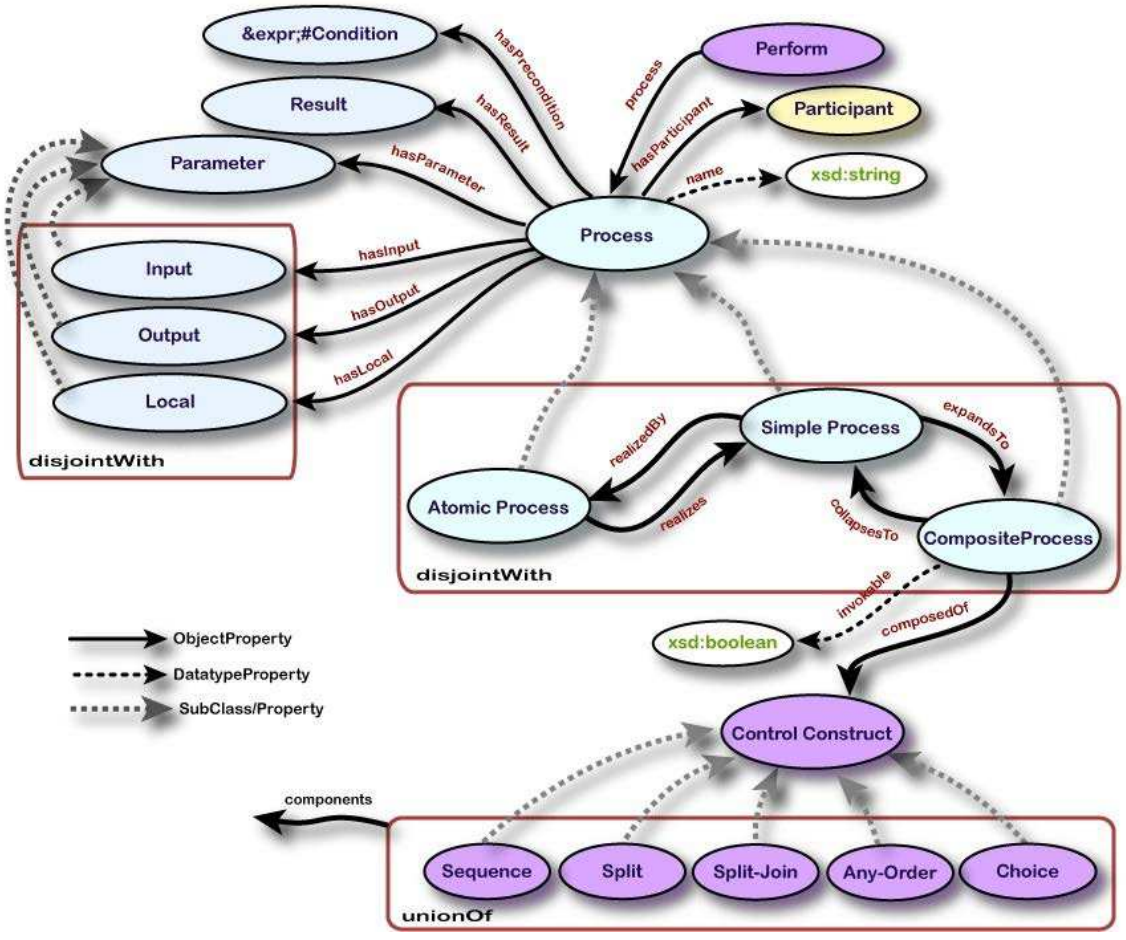

Figure 3.4: Inspiration from OWL-S [Martin et al., 2004]

Figure 3.4 illustrates the upper level of the process ontology in OWL-S [Martin et al., 2004]. As Figure 3.4 representing, the concept Process has three sub-classes: Atomic Process, Simple Process and Composite Process. The Composite Process collapses to Simple Process, which is realized by Atomic Process. The definitions of these three kinds of processes in [Martin et al., 2004] are: "Atomic processes correspond to the actions a service can perform by engaging it in a single interaction; composite processes correspond to actions that require multi-step protocols and/or multiple server actions; finally, simple processes provide an abstraction mechanism to provide multiple views of the same process". It is the difference between Atomic Processes and Simple Processes in service models. It means that for each atomic process, there must be provided a grounding that enables a service requester to construct messages to the process from its inputs and interpret or deconstruct a reply as an output of the service. On the contrary, simple processes are not invocable and are not associated with a grounding, but, like atomic processes, they are conceived as having single-step executions.

In our designed K-CRIO Ontology, because our purpose is to model business process, we conceptualize the concept of Process (`owl:class`) as being a class equivalent to FormalizedInteraction (`owl:class`), which has two sub-classes: Atomic Process (`owl:class`)) and Composite Process (`owl:class`). Each process has its participants and its parameters. Specifically, Participant as an `owl:class` is equivalent to the Role class. Parameter (`owl:class`) has two sub-classes: Input (`owl:class`) and Output (`owl:class`). The relationship between Process and Parameter are respectively hasInput (`owl:0bjectProperty`) and hasOutput (`owl:0bjectProperty`). These elements are formalized as follows.

$$Process(X) \equiv FormalizedInteraction(X)$$

$$Process(X) \equiv AtomicProcess(X) \cup CompositeProcess(X)$$

$$Participant(X) \equiv Role(X)$$

hasParticipant(X,Y) means *X* has participant *Y* and {*X: Process(X), Y: Participant(Y)* or Y:Role (Y)}.
hasParameter(X,Y) means *X* has parameter *Y* and {*X: Process(X), Y: Parameter(Y)*}.

Figure 3.5 helps us to understand how to define the FormalizedInteraction from the respective of workflow, which is referred by OWL-S.

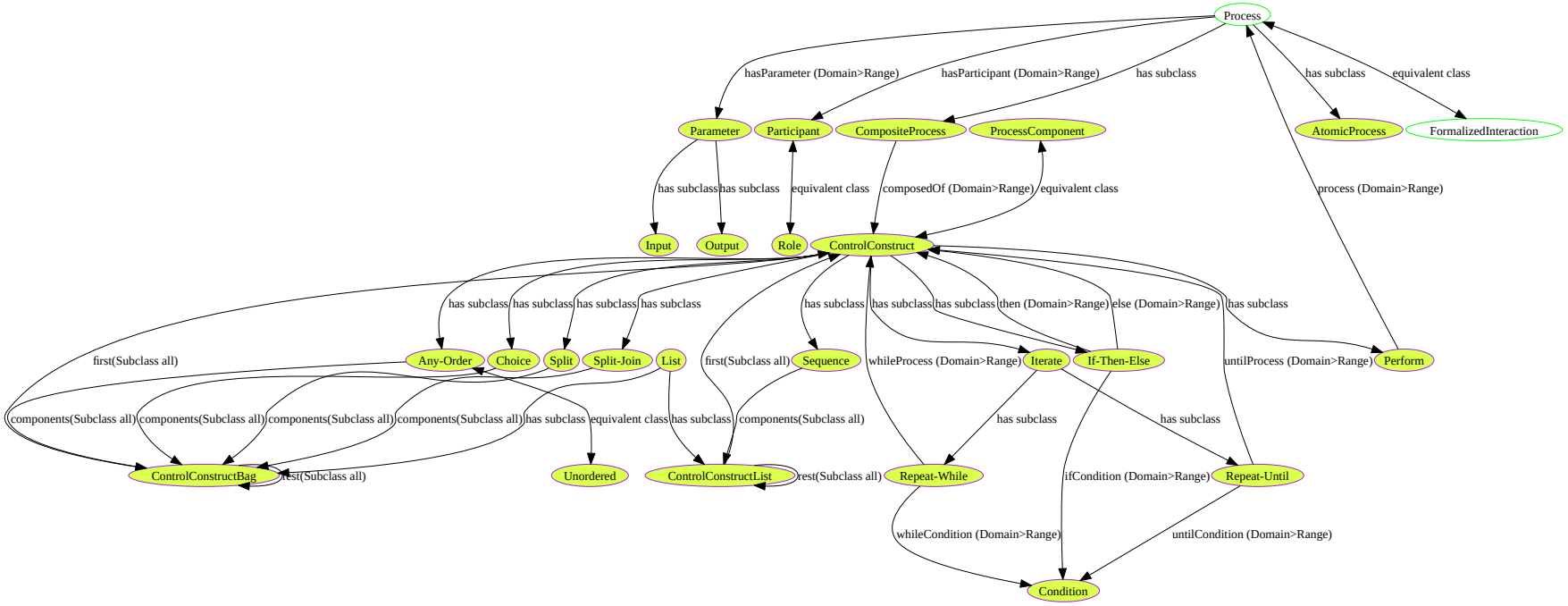

Figure 3.5: Definition of Formalized Interaction from the workflow aspect

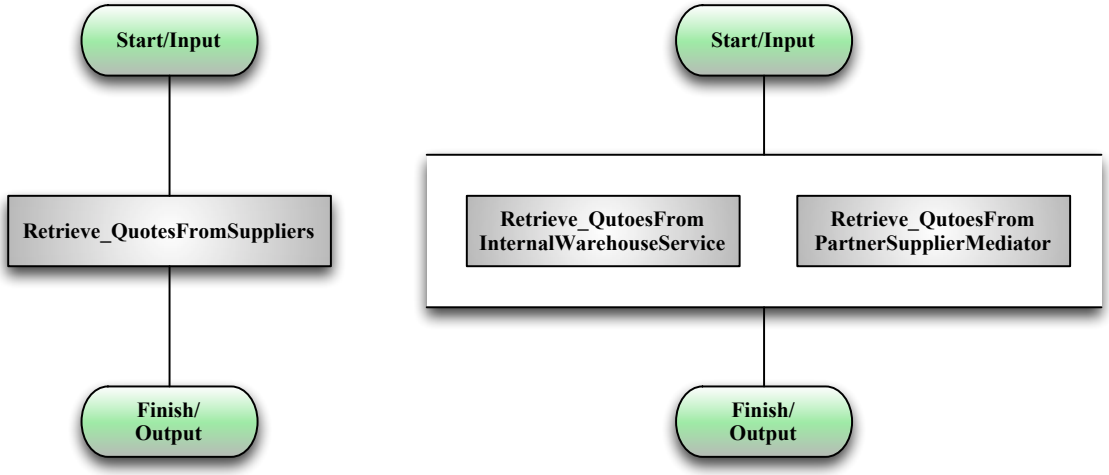

Figure 3.6: The format of a Split Process

Precisely, a process may be an atomic process which is a description of one (possibly complex) sent message and one returning (possibly complex) message in response. A process may also be a composite, which means that it can be divided into atomic processes describing a behavior (or a set of behaviors) sending and receiving a series of messages. The Composite Process is composed of (`owl:ObjectProperty`) Control Construct (`owl:class`) , which has several sub-classes, including Sequence (`owl:class`), Choice (`owl:class`), Split (`owl:class`), Split-Join (`owl:class`), Any-Order (`owl:class`), If-Then-Else (`owl:class`), Iterate (`owl:class`), Perform (`owl:class`). These different constructs are detailed in the following list.

- Sequence means a list of control constructs to be executed in the defined order. This construct is further defined by several components (`owl:ObjectProperty`) Control Construct List (`owl:class`). (Control Construct List and Control Construct Bag are two sub-classes of List (`owl:class`).) Moreover, the first (`owl:ObjectProperty`) element of Control Construct List could be another Control Construct and the rest (`owl:ObjectProperty`) of Control Construct List can be a Control Construct List defining a typical linked list.

  Examples of Sequence process could be comprehended easily. The example of Waterfall Model process is one of them in following section 3.4.

- The components of a Split process are a bag of process components to be executed concurrently. The Split completes as soon as all of its component processes have been scheduled for execution.

  For example, the process of retrieving quotes from all suppliers could be seen as a *CompositeProcess: Retrieve_QuotesFromSuppliers*. It is a Split process with two components: *AtomicProcess: Retrieve_QuotesFromInternalWarehouseService* and *AtomicProcess: Retrieve_QuotesFromPartnerSupplierMediator*. These two processes are executed concurrently, and the *CompositeProcess: Retrieve_QuotesFromSuppliers* is not finished until these two component processes are finished. The format of Split process is shown in Figure 3.6.

- When a composite process is composed of a control construct of Split-Join, it means that the process consists of concurrent executions of a bunch of process compo-

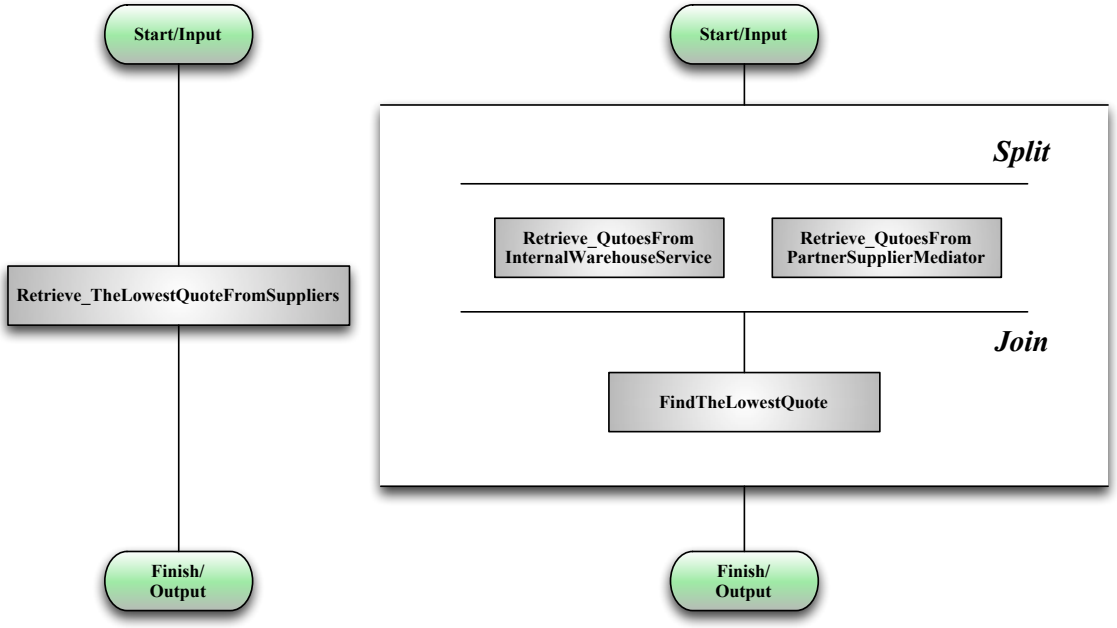

Figure 3.7: The format of Split+Join Process

nents with barrier synchronization. That is, Split+Join completes when all of its components processes have completed. With Split and Split+Join, we could define processes that have partial synchronization.

Extending the above example, the process of retrieving the lowest quote from all suppliers could be seen as a *CompositeProcess: Retrieve_TheLowestQuoteFromSuppliers*. It is a Split-Join process that two atomic processes, *AtomicProcess: Retrieve_QuotesFromInternalWarehouseService* and *AtomicProcess: Retrieve_QuotesFromPartnerSupplierMediator* are executed in parallel, and joins the responses into a single *AtomicProcess:FindTheLowestQuote*. The format of Split+Join process is shown in Figure 3.7.

- Any-order is the equivalent class of Un-order, which allows the process components (specified as a bag) to be executed in some unspecified order but not concurrently. Execution and completion of all components is required.

  For example, the process of mixing pigments could be seen as a *CompositeProcess: MixPigments*. In order to finish this process, we need to finish the processes *AtomicProcess: AddRedPigment*, *AtomicProcess: AddWhitePigment* and *AtomicProcess: AddYellowPigment*. The order of these three processes is not important.

- Choice calls for the execution of a single control construct from a given bag of control constructs (given by the components property). Any of the given control constructs may be chosen for execution.

  For example, the process of finding the fittest airplane ticket could be seen as a *CompositeProcess: FindTheFittestAirplaneTicket*. There are two choices for finishing this process: the *CompositeProcess: FindTheCheapestAirplaneTicket* and the *CompositeProcess: FindTheFastestAirplaneTicket*.

- The If-Then-Else class is a control construct that has properties ifCondition (`owl:ObjectProperty`), then and else holding different aspects of the If-Then-Else.

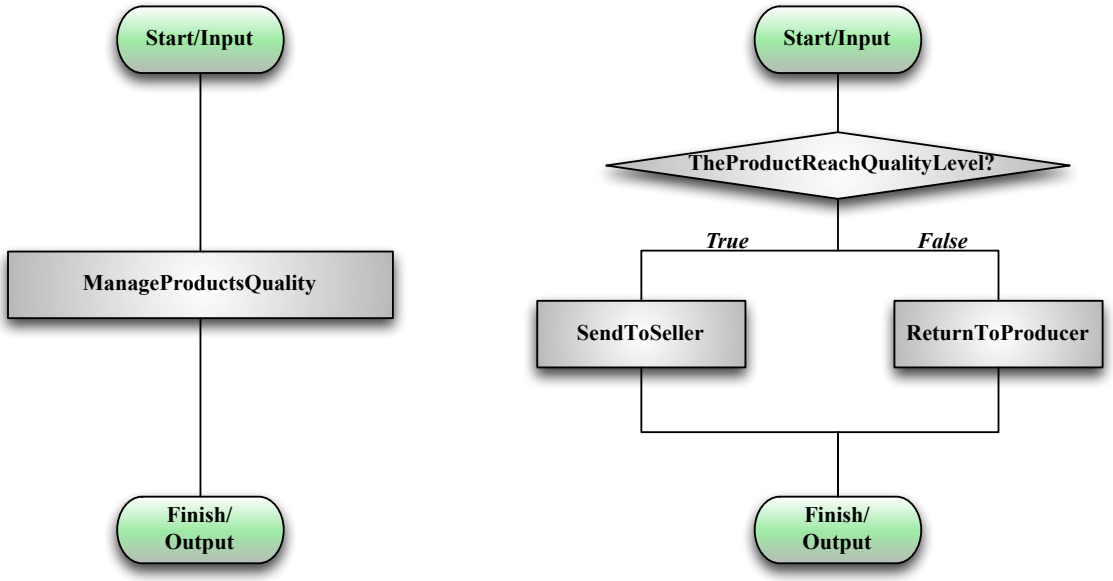

Figure 3.8: The format of a If-Then-Else Process

Its semantic means as that test the Condition (`owl:class`) of the related property If-condition; if True do Then, if False do Else.

For example, a process of managing products quality could be seen as an If-Then-Else *CompositeProcess: ManageProductQuality*, which has a *Condition: TheProductReachQualityLevel*. If the condition is true, execute the *AtomicProcess:SendToSeller*, otherwise execute the *AtomicProcess: ReturnToProducer*. The format of If-Then-Else process could is shown in Figure 3.8.

- Iterate construct makes no assumption about how many iterations are made or when to initiate, terminate, or resume. The Condition (`owl:class`) of initiation, termination or maintenance could be specified with a whileCondition (`owl:ObjectProperty`) or an untilCondition (`owl:ObjectProperty`). Both iterate until a condition becomes false or true, following the familiar programming language conventions. Repeat-While tests for the condition, exits if it is false and does the operation if the condition is true, then loops. Repeat-Until does the operation, tests for the condition, exits if it is true, and otherwise loops. Thus, Repeat-While may never act, whereas Repeat-Until always acts at least once.

  For example, the process of correcting coding mistakes could be seen as a Repeat-While *CompositeProcess: Debug* with the *Condition: HasBugs?*. The condition is true executes the *AtomicProcess: Correct* and then re-evaluates the condition. The format of Repeat-While process is shown in Figure 3.9. Additionally, if we define the CompositeProcess: Debug as a Repeat-Until process. The format of this Repeat-Until process is shown in Figure 3.10

- Perform could be considered as the command to execute a process. From the ontology point of view, one Perform (`owl:class`) could process (`owl:ObjectProperty`) one Process (`owl:class`). It could concerns either one Atomic Process or one Composite Process.

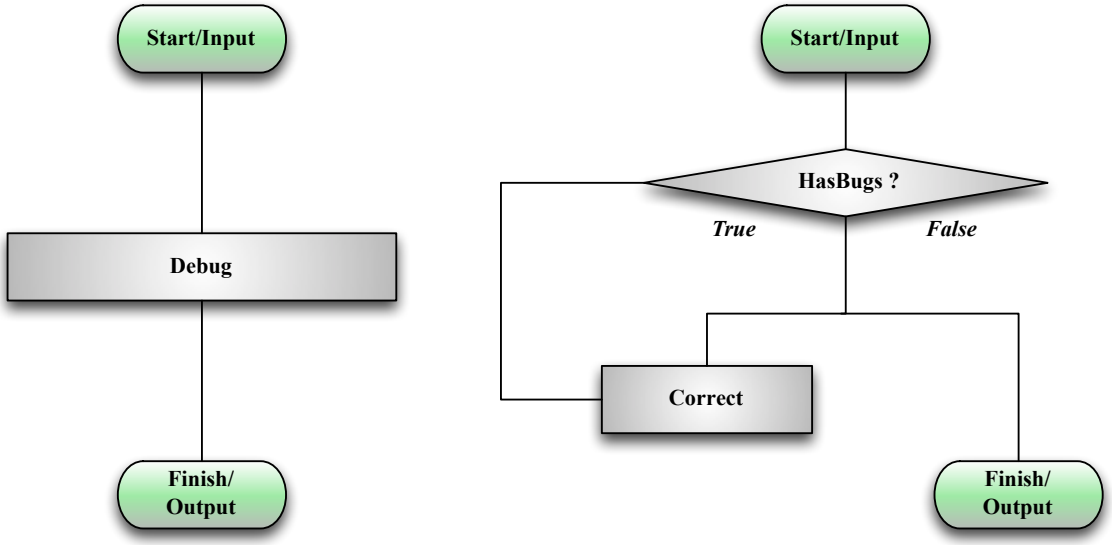

Figure 3.9: The format of a Repeat-While Process

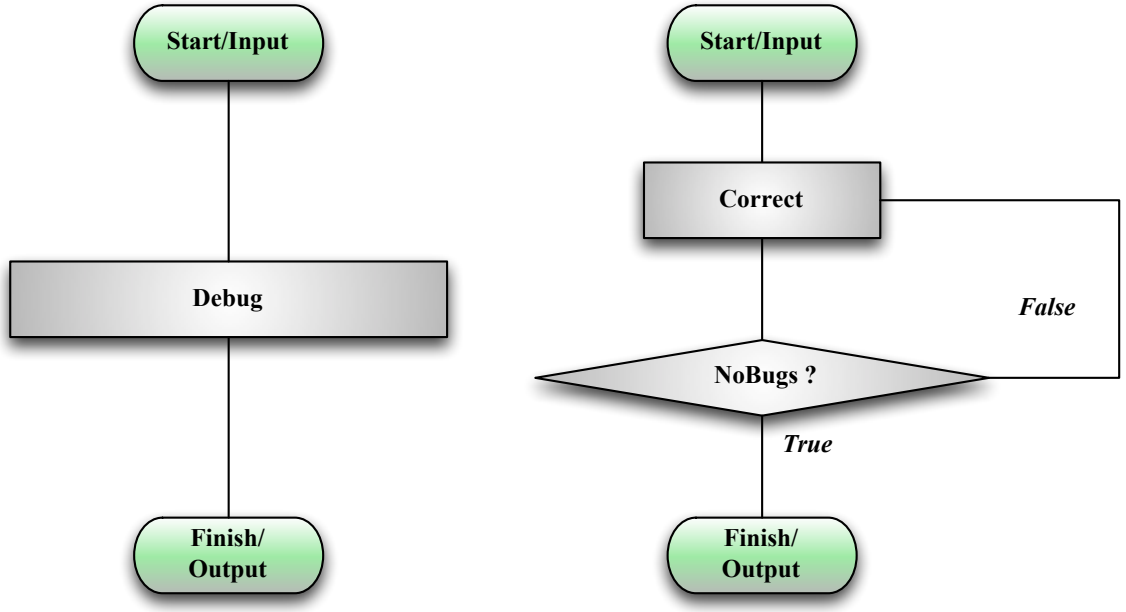

Figure 3.10: The format of a Repeat-Until Process

DEFINING INTERACTION FROM THE ASPECT OF SCHEDULE

Additionally, if we need to pay more attention to the state and time-table of each formalized interaction, we could use an extension of the K-CRIO ontology, which use a state machine representation for FormalizedInteraction. We define for FormalizedInteraction a specific relationship ”is in” (an ObjectProperty) some State (an owl:class), which has three sub-classes: NotStarted, Doing, Done. In certain situations, a formalized interaction has its pre-Interaction (hasPre-Interaction is an ObjectProperty, the domain and range of which are both FormalizedInteraction):

is in(X,Y) means *X* is in *Y* state and {*X: FormalizedInteraction(X), Y: State(Y)*}.
hasPre-Interaction(X,Y) means *X* has pre-interaction *Y* and {*X: FormalizedInteraction(X), Y: FormalizedInteraction(Y)*}.

In order to test whether the interaction is executed following the schedule or not, we consider the the FormalizedInteraction may respect time constraints. Therefore, we defined an `owl:class` Time which is related with FormalizedInteraction by the `ObjectProperty:follows`. More specifically, Time has four sub-classes: BeginningTime, EndTime, RealBeginningTime, RealEndTime, in which, BeginningTime and EndTime express the planning schedule and RealBeginningTime and RealEndTime expresses the real schedule. Comparing the subtraction of two types of beginning time and end time, we could express whether the process is earlier or later than the schedule actually:

For $Follows(FormalizedInteraction, BeginningTime) \wedge (FormalizedInteraction, EndTime) \wedge (FormalizedInteraction, RealBeginningTime) \wedge (FormalizedInteraction, RealEndTime)$:

$(EndTime - BeginningTime) > (RealEndTime - RealBeginningTime)$ means the FormalizedInteraction is earlier than the schedule.

$(RealEndTime - RealBeginningTime) > (EndTime - BeginningTime)$ means the FormalizedInteraction is later than the schedule.

Figure 3.11 represents the concept of FormalizedInteraction in K-CRIO and related concepts/relationships.

In summary, all the related concepts and their relationships in K-CRIO are presented in Figure 3.12.

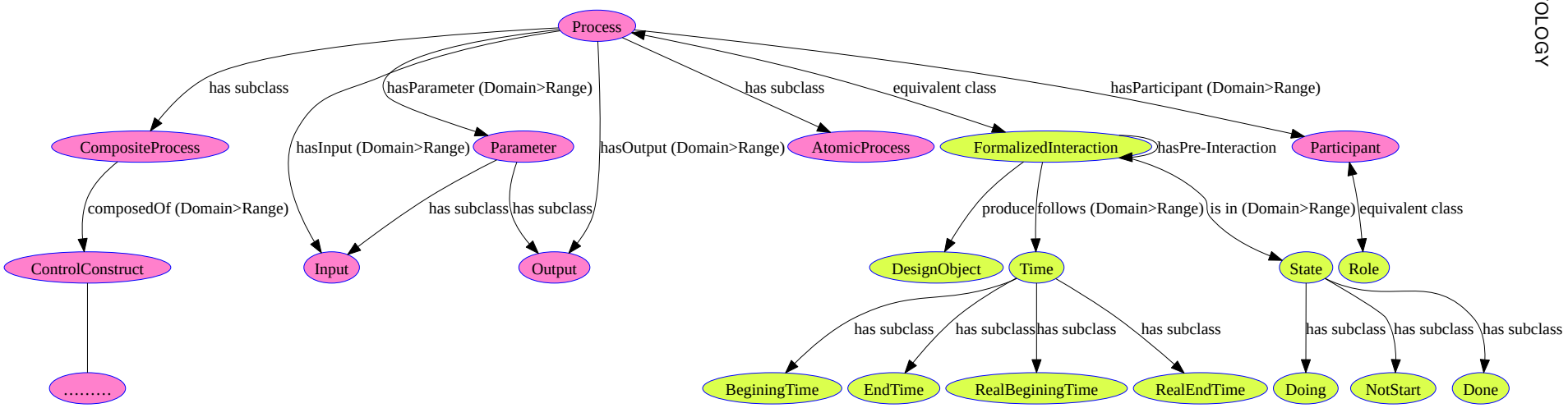

Figure 3.11: Interaction in K-CRIO

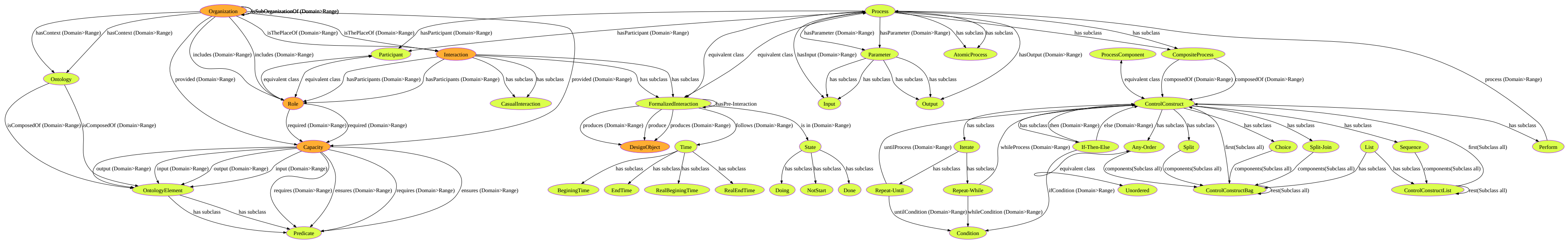

Figure 3.12: All related concepts and relationship in K-CRIO

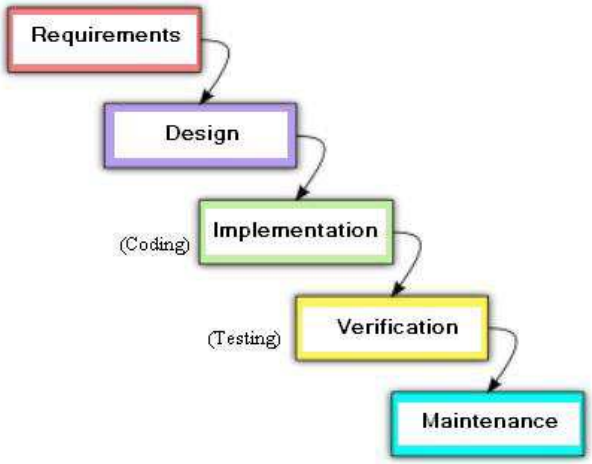

Figure 3.13: Software development process: the Waterfall Model

3.4/ A SIMPLIFIED SOFTWARE DEVELOPMENT PROCESS MODELED WITH K-CRIO

In the previous sections, we have presented the K-CRIO concepts and their relationships. In order to make the definition of K-CRIO more comprehensive and understandable, in this section, we will provide a brief case study to explain how to use K-CRIO to model business processes. The presented concepts and relationships should not be taken as a complete description.

More specifically, we use K-CRIO on a traditional software-development process, namely the Waterfall Model [Melonfire, 2006] illustrated by Figure 3.13. This model is one of the most famous for software processes and has been used widely in IT (Information Technology) organizations for project teams developing software.

The Waterfall Model is just a primer case study for K-CRIO that is used to illustrate the previous concepts and relationships. As such, the Waterfall model has been simplified in order to serve as an example. Other business processes or classical models of software-development processes could also be modeled by K-CRIO, like Spiral Model, Rapid Prototyping Model, etc.

As presented in Figure 3.13, one could see that the Waterfall Model defines five main phases. These phases are detailed by the following list.

- In the Requirement phase, the Requirement Analysts need to visit the customer and to study system requirements. They examine the need for possible software automation in the given software system. After feasibility study, the development team provides a document that holds the different specific recommendations for the candidate system. It also consists of personnel assignments, costs of the system, project schedule and target dates.
- In the Design phase, the overall software structure and its outlay are defined, like package architecture, database design if needed, data structure design, etc. All of these elements give birth to a design document and the main role that participates

to this phase is Designer.

- In the Implementation phase, the whole system design must be implemented into a machine-readable form by coders with programming languages like C, C++, Java and so on.
- After the code generation phase the software program testing activities begin in order to try detecting the bugs in the Verification phase.
- In the Maintenance phase, software will definitely go through change once when it is delivered to the customer.

As stated above, we could apply K-CRIO to model the Waterfall Model process. An IT Company should be seen as an Organization. The Software Developing Team is also an Organization and one of the IT Company sub-organizations. The Software Developing Team organizations includes Roles such as: Project Leader, Requirement Analyst, Designer, Coder, Tester (System Tester, and Unit Tester).

Speaking about actual Software-Development Project process, the Capacities required by the Roles are:

- managing project, organizing meeting, confirming the domain, supervising schedule, making decisions, examining and checking, etc.(required by Project Leader);
- communicating with clients or customers, analyzing the requirement of users, writing requirement document (required by Requirement Analyzer);
- designing the system (including system design, architecture design and database design, etc.), writing design document (required by Designer);
- coding (required by coder);
- test, writing test report (required by Tester, System Tester, Unit Tester);

Following the K-CRIO definitions of previous sections, the above description may be expressed as presented in Figure 3.14. Additionally, we take an instance of Project Leader to explain how to define these in OWL, it is depicted in Figure 3.6.

```
<Organization ref:ID="SoftwareDevelopmentTeam">
   <includes rdf:resource="#ProjectLeader"/>
   <provided rdf:resource="#SuperviseSchedule"/>
   <provided rdf:resource="#ComfirmSchedule"/>
   <provided rdf:resource="#OrganizeMeeting"/>
   <provided rdf:resource="#ConfirmDoman"/>
   <provided rdf:resource="#MakeDecision"/>
   <provided rdf:resource="#ExamineAndCheck"/>
</Organization>

<Role rdf:ID="ProjectLeader">
   <required>
      <Capacity rdf:ID="ConfirmSchedule"/>
   </required>
   <required>
      <Capacity rdf:ID="OrganizationMeeting"/>
   </required>
   <required>
      <Capacity rdf:ID="SuperviseSchedule"/>
   </required>
   <required>
      <Capacity rdf:ID="ConfirmDomain"/>
   </required>
```

```
    <required>
        <Capacity rdf:ID="ExamineAndCheck"/>
    </required>
    <required>
        <Capacity rdf:ID="MakeDecision"/>
    </required>
</Role>
```

Listing 3.6: Defining Project Leader Role in OWL

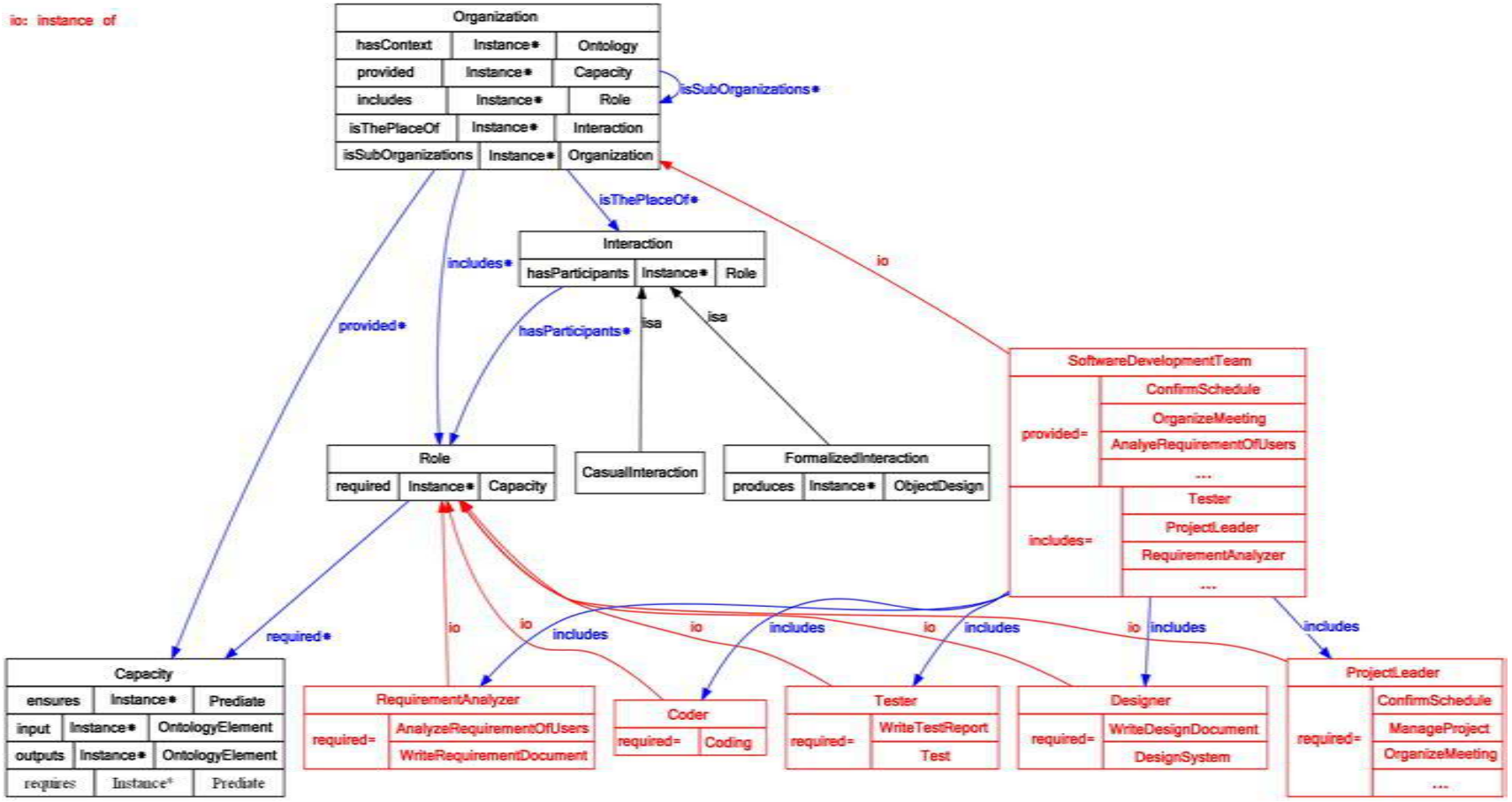

Figure 3.14: An Example of K-CRIO

The interactions in the SDT Organization (Software Developing Team) may be considered as Casual Interaction and Formalized Interaction separately. Chat is a kind of Casual Interaction between different persons (Roles), other examples may be Exchanging Mail or Joining Conference Meeting, etc. Furthermore, the whole Waterfall Software Developing process should be defined by Formalized Interaction as well as Composite Process to produce a software or tool defined as a specific DesignObject in K-CRIO. The corresponding Formalized Interaction is composed of a Sequence Control Construct. The components of the Control Construct Sequence are:

- CompositeProcess_RequirementAnalyzing has participants (Roles) of Project Leader, Requirement Analyzer, Designer to produce Requirement Document defined as DesignObject;
- CompositeProcess_SystemDesigning has participants (Roles) of Project Leader, Designer, Coder to produce System Design Documents defined as DesignObject;
- CompositeProcess_Implementation has participants (Roles) of Project Leader, Coder to produce coding source or some similar things defined as DesignObject;
- CompositeProcess_Verification has participants (Roles) of Project Leader, Coder, Tester, System Tester, Unit Tester to produce the software or tool defined as DesignObject;
- CompositeProcess_Maintenance has other participants (Roles), like servicing people to produce robust softwares or tools defined as DesignObject.

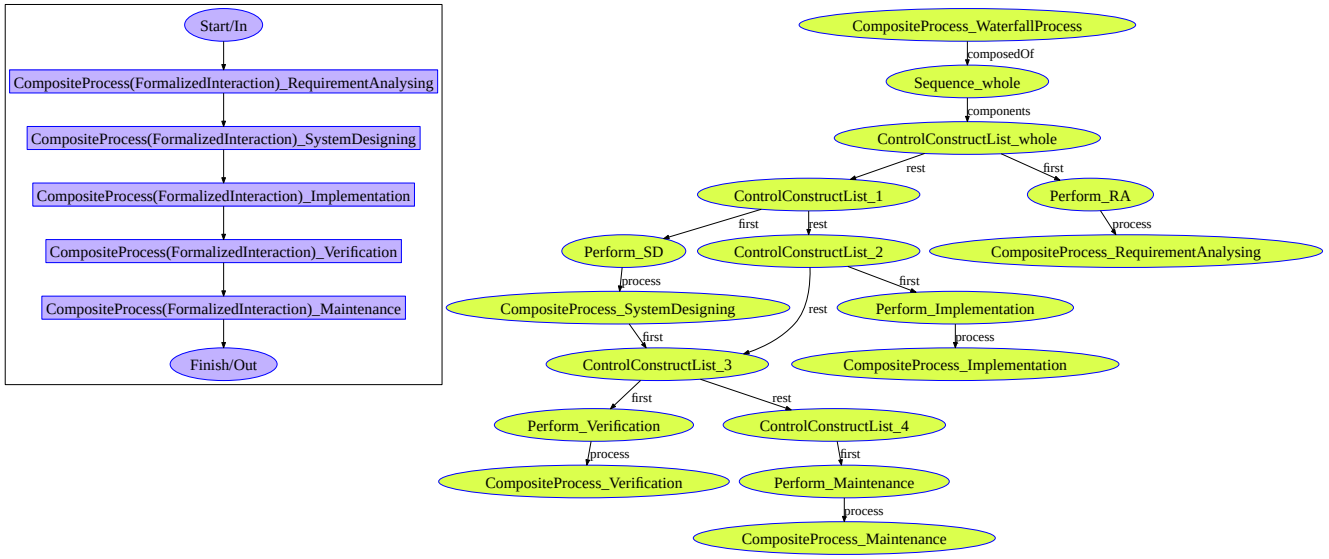

Figure 3.15: Waterfall Process by K-CRIO Ontology

The overall Waterfall process is defined by CompositeProcess_WaterfallProcess, which is composed of Sequence_whole. The single component of Sequence_whole is ControlConstructList_whole. The first element of ControlConstructList_whole is Perform_RA that

processes CompositeProcess_RequirementAnalysing. The rest of ControlConstructList_-Whole is ControlConstructList_1. Furthermore, the first of ControlConstructList_1 is Perform_SD that processes CompositeProcess_System Designing and the rest of that is ControlConstructList_2. Similarly, the first of ControlConstructList_2 is Perform_Implementation that processes CompositeProcess_Implementation and the rest of ControlConstructList_2 is ControlConstructList_3. The rest definition could be done in the same manner to finish the general description of Waterfall process.

On the contrary, if the document does not satisfy a given criterion (the condition Redo is true), it will be returned to Designers with some synthetic suggestions and then be checked again until the document is valid (the Composite Process *Review Design Document* is executed again and return a new value of *Redo*).

In sum, all the detail of the conceptualization of Waterfall Model with K-CRIO could be explained precisely in [Lin et al., 2011].

3.5/ CONCLUSION

In this chapter, we have defined the K-CRIO ontology, including each concept, related relationships and restrictions. More precisely, the K-CRIO Ontology is an organizational ontology, described with Ontology Web Language (OWL). The core concepts in K-CRIO are Organization, Role, Capacity and Interaction. We built K-CRIO Ontology in order to model and conceptualize business processes in enterprises. These business processes are dedicated to the design of products. Using the K-CRIO Ontology, we could not only conceptualize human roles and their relationships in enterprises, but also model the human activities and the scenario of how human roles' interactions happen.

The K-CRIO Ontology is our proposed approach trying to answer the concerns about enterprise modeling, especially about business processes. It is a core contribution of this thesis. The underlying assumption is that it should help developers to design and implement the system for supporting a special business process.

In our thesis, we want to design and implement an intelligent assistance system to support human activities within business processes, and as an example of such a process the Scrum process is detailed in the following chapter. This chapter is based upon the conceptualization of Scrum process with K-CRIO, as the primary attempt. Precisely, Scrum process is a famous agile software-development process, which is used widely in many project teams delivering their projects/products in software enterprises.

Therefore, following the definition of K-CRIO in this chapter, we will illustrate the use of K-CRIO to conceptualize the Scrum process in the following chapter. The content of next chapter also serves as a complete case study for explaining the usage of K-CRIO Ontology.

4

SCRUM PROCESS CONCEPTUALIZED WITH K-CRIO ONTOLOGY

4.1/ INTRODUCTION

Scrum is a well-known, iterative and agile software-development process, that involves different kinds of roles interacting with each other in order to achieve their purpose. It is currently widely used to develop software.

The Scrum process, as many other processes, involves several human actors interacting to produce deliverables and a final product under the form of a software. Even if it is an agile software-development process reputed of low ceremony, the sequence of interactions may appear more complex than the sequential Waterfall process, shortly described as an example in the previous chapter. For human actors involved in a Scrum process, assistance to support the process could help with many aspects such as respecting the schedule, managing the costs, delivering the right products, ...

In this chapter, we will conceptualize the different elements of Scrum Processes with the K-CRIO ontology that is presented in the Chapter 3. That is to say, we will use the K-CRIO ontology to describe for the Scrum process, the different involved organizations, human roles and interactions their capacities and how these roles interact in each phase of a Scrum process. For understanding the conceptualization Scrum with K-CRIO, we could also read related contents in [LIN et al., 2011a], [LIN et al., 2012].

The organization of this chapter is the following. In Section 4.2, we make an introduction to the Scrum Process. Based upon this general introduction to Scrum, we conceptualize all Scrum elements with the K-CRIO ontology in Section 4.3. Eventually, Section 4.4 concludes.

4.2/ A SHORT INTRODUCTION TO THE SCRUM PROCESS

Scrum is an iterative, incremental framework for projects and products or application developments and is often seen as an agile software-development process. Scrum structures development in cycles of work called Sprints (or iterations). These iterations last no more than one month each, and take place one after the other without pause. The people involved in Scrum could be divided into two groups. The first group denotes the Scrum team and has a figurative name: ”Pig”, where are three core components: The Product Owner, The Scrum Master and The Team. All these members are committed to

The Product Backlog (Example)					
ID	Name	Priority	Estimate Effort (hours)	Description	Note
1	Deposit	10	12	A user Log in, open deposit page, deposit money, go to my balance page and check that it has increased equal amount money.	Need a UML sequence diagram. No need to worry about encryption for now.
2	See your own transaction history	5	6	A user Log in, click on “transactions” and show the transaction record.	Use paging to avoid large DB queries. Design similar to view users page.
3	Edit your information	10	10	A user Log in, click on “personal information” and show the users' information	Change password need the confirmation of SMS
…	…	...	...	…	...

Figure 4.1: The Product Backlog

the project in the Scrum process. These components are the ones producing the product (objective of the project). The other group is composed by Ancillary people and is named ”Chicken”. Precisely, the ancillary people in Scrum are those with no formal role and infrequent involvements in the Scrum process that must, nonetheless, be taken into account. For example, one can cite some Stakeholders, Customers, Vendors or Managers (that represents a kind of person who will set up the environment for product development).

The Scrum process is presented in Figure 4.3 extracted from [Deemer et al., 2010]. In this figure, we could see that during the first step, the Product Owner needs to communicate with all the Stakeholders in order to transform each User Story into one item or one feature in the Product Backlog. The Product Backlog is the core document containing prioritized descriptions which will be referred by the Scrum Team during the Scrum Process. The product backlog is basically a prioritized list of functional requirements, non-functional requirement, issues, stories, features, or other related things. These elements in the Product Backlog describes what the customer wants. They are described using the customers’ terminology. We call these elements ” user stories”, or sometimes just backlog items. Figure 4.1 represents an example of Product Backlog about an e-bank project. Generally, the Product Owner could describe each item of the Product Backlog with the following fields:

- ID: a unique identification or an auto-incremented number;
- Name: a short descriptive name of a story;
- Priority: or alternatively named Importance, higher of which means the items is

more important to finish;

- Estimate effort: the initial assessment of how much work needed to implement this story. The unit is story point and usually corresponds to ”ideal man-days” or ”ideal man-hours”;
- Description: a high-level description of how this story will be demonstrated at the sprint demo;
- Note: any other information;
- Others: like Category (or Track, which is a rough categorization of this story), Components (which can identify which technical components will be involved in implementing this story, for example ”database, server, client”), Requester (showing which customer or stakeholder originally requested the item), Bug tracking ID (information for reporting bugs).

In a second step, before the beginning of each Sprint, a cross-functional team selects items (customer requirements) from the prioritized list (Product Backlog) [Deemer et al., 2010]. This phase occurs in the Sprint Planning Meeting. The team should commit to completing these items by the end of the Sprint. Each item represented in the Sprint Backlog, that can be called task, is decomposing one user story in Product Backlog down to small units. Precisely, in actual Scrum projects, teams could use different formats for the Sprint Backlog. These formats include specific Scrum software, Jira, Excel, or a physical task board screwed on the wall. However, no matter the selected format, the contents of one Sprint Backlog are generally represented by a list of tasks that defines a team’s work for a Sprint. The list emerges during the Sprint. Each task identifies those responsible for doing the work and the estimated amount of work remaining on the task on any given day during the Sprint. This estimated amount is updated with a given frequency. Figure 4.2, presents an example of Sprint Backlog which selects two user stories from the Product Backlog of Figure 4.1. More information about Scrum could be found in: http://www.scrummethodology.org/scrum-phases.html and [Deemer et al., 2010].

4.3/ Conceptualization of Scrum with K-CRIO

In this section, we present how to use the K-CRIO Ontology to conceptualize Scrum processes. Based on the general understanding derived from the previous description of Scrum, one can make the hypothesis that Scrum contains diverse kinds of people working together with specific interactive modes in order to achieve a common goal, such as delivering a product or exploring a project.

4.3.1/ Identification of Organizations, Roles and Capacities in Scrum Processes

Following the definition of K-CRIO, Scrum may be seen as one Organization that is called Scrum. Within this organization, there are smaller groups that can also be seen as Organizations (sub-organization of Scrum), like the groups ”Pig” and ”Chicken”. Among these

				New Estimate Effort Remaining at end of Day					
Product Backlog Item	Sprint Task	Assigned Worker	Initial Estimate Effort	1	2	3	4	5	...
Deposit	Modify database	Harry	5						
	Create webpage (UI)		6						
	Create webpage (Javascript logic)	Bob	8						
	...								
Edit your Information	Modify database	Harry	4						
	Create webpage (UI)		6						
	Update related user homepage (UI)		6						
	...								

Figure 4.2: The Sprint Backlog

organizations, the roles involved are Scrum Master, Product Owner,etc. These definitions expressed in a logical form are as follows.

Scrum:

$$Organization(Scrum) \wedge isSubOrganizationOf(Pig, Scrum)$$

$$\wedge isSubOrganizationOf(Chicken, Scrum)$$

Following K-CRIO definitions, Figure 4.4 describes all the related elements in Scrum Processes that are considered as Organization, Role and Capacity. Specifically, Organization, Role and Capacity are represented in purple, black and green respectively. In the following section, each of them will be detailed.

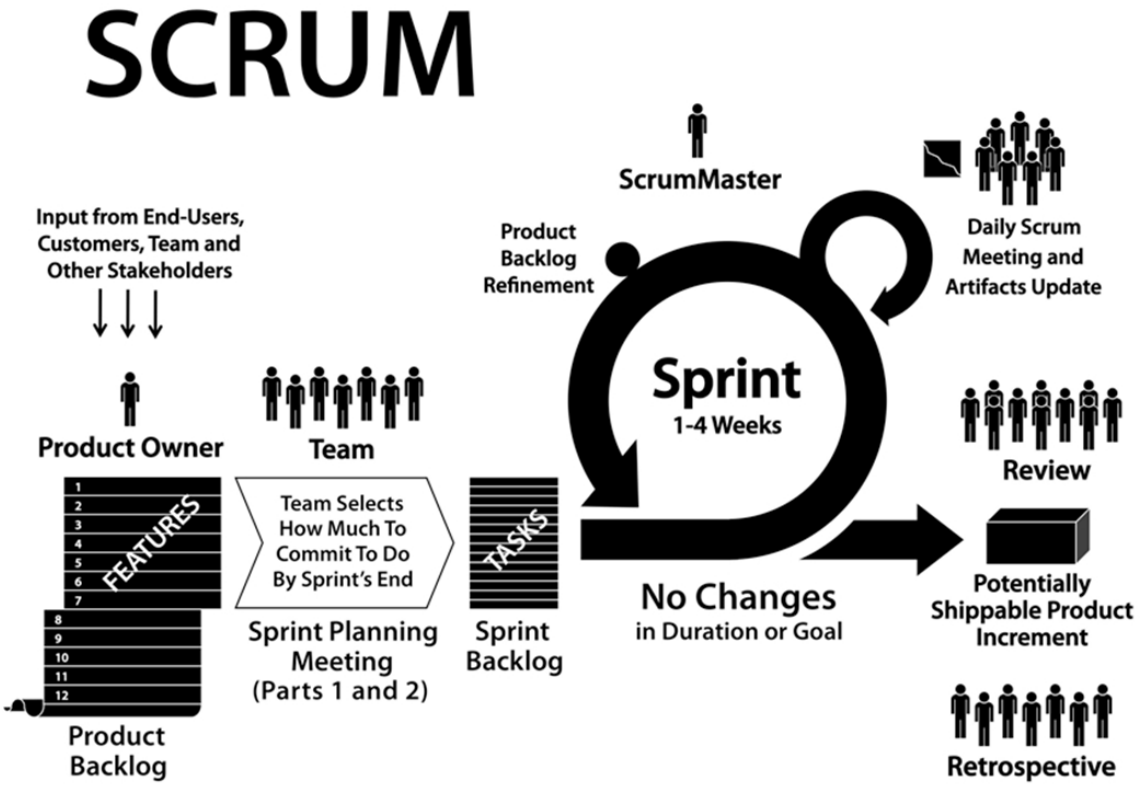

Figure 4.3: The process of Scrum

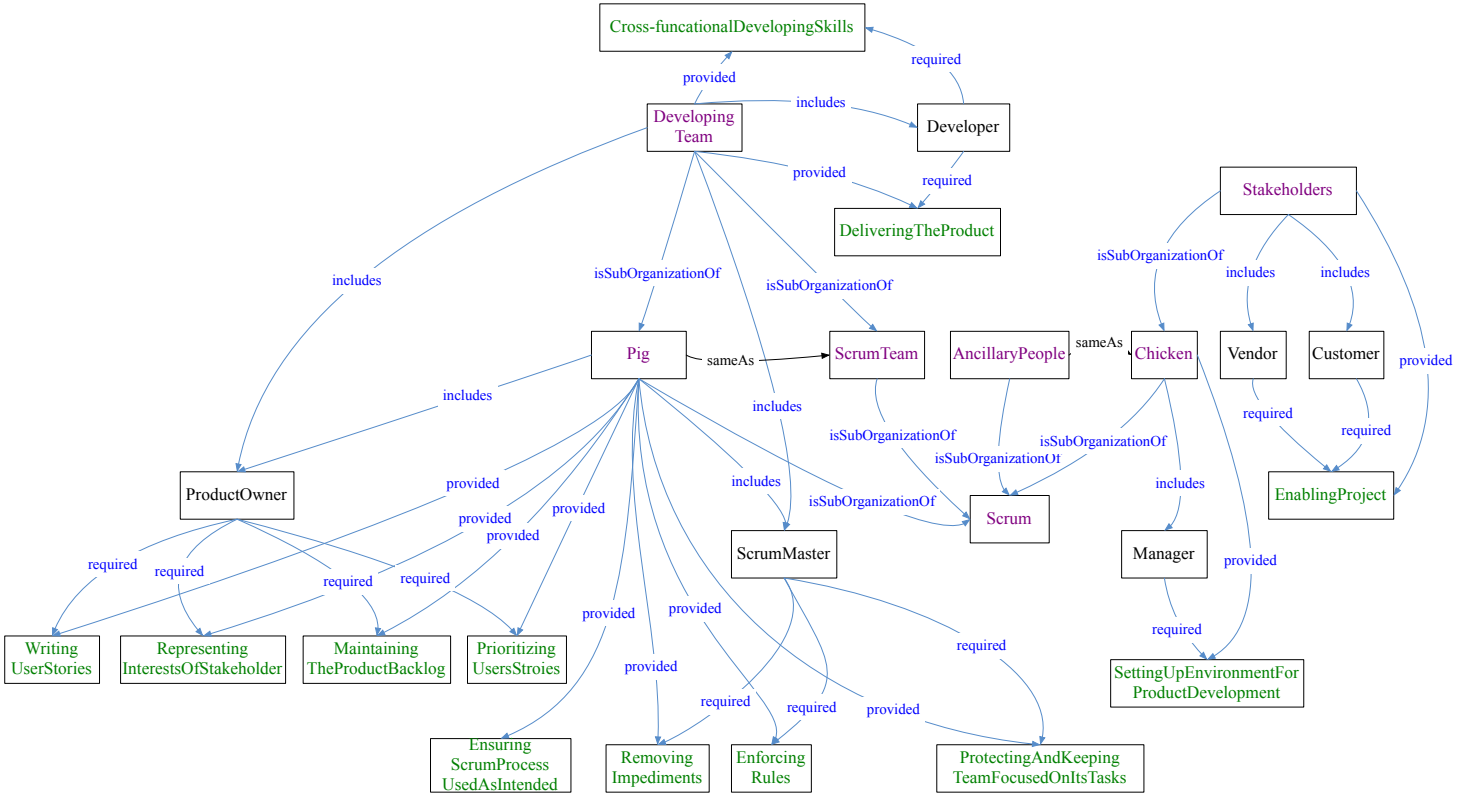

Figure 4.4: Scrum with K-CRIO

ORGANIZATION: "PIG"

In the *Organization: Scrum*, the group "Pig" (Scrum Team) aims at releasing the product following the requirement of customers and vendors through its members' contributions. Therefore, the group "Pig" (Scrum Team) may be seen as the *Organization: "Pig"*(*Organization: Scrum Team*) in K-CRIO, which is a sub-Organization of *Organization: Scrum*. As three significant components of the group "Pig", the Product Owner (PO), the Scrum Master (SM) and the Developing Team (DT) fulfill the objective of *Organization: "Pig"* through accomplishing their jobs. Deeply, the Product Owner and the Scrum Master could be both seen as a Role in K-CRIO, which is included in *Organization: "Pig"*. Furthermore, the Developing Team could be seen as Organization in K-CRIO, which is a sub organization of *Organization: "Pig"*. Therefore, we could define "Pig" as:

Pig (ScrumTeam):

$$Organization(Pig) \wedge isSubOrganizationOf(Pig, Scrum) \wedge isSubOrganizationOf(DT, Pig)$$

$$\wedge includes(Pig, PO) \wedge includes(Pig, SM) \wedge provided(Pig, CapacitiesforPO)$$

$$\wedge provided(Pig, CapacitiesforSM)$$

Because of the transitivity of isSubOrganizationOf, there is:

$$isSubOrganizationOf(Pig, Scrum), isSubOrganizationOf(DT, Pig)$$

$$\Rightarrow isSubOrganizationOf(DT, Scrum)$$

The Pig group is composed by two Roles : the Product Owner and the Scrum Master. The Product Owner represents the voice of the customer and writes customer-centric items (typically user stories), prioritizes them, and adds them to the product backlog. This role is also accountable for ensuring that the Team delivers value to the business:

Product Owner (PO):

$$Role(PO) \wedge includes(Pig, PO) \wedge provided(Pig, CapacitiesforPO)$$

As an enforcer of rules, the Scrum Master is accountable for removing the impediments to the ability of the team to deliver the sprint goal/deliverable and for ensuring that the Scrum process is used as intended:

Scrum Master (SM):

$$Role(SM) \wedge includes(Pig, SM) \wedge provided(Pig, CapacitiesforSM)$$

Differently, the Developing Team is typically made up of five to nine developers with cross-functional skills who do the actual work (analysis, design, development, testing, technical communication, documentation, etc.) for releasing the product. For this reason, the Developing Team is seen as Organization which is a sub-Organization of *Organization: "Pig"*. Additionally, in the Developing Team, the Developer may be seen as a Role in K-CRIO, which is included in *Organization: Developing Team*. Sometimes, Product Owner and Scrum Master may also be members of the Developing Team[1] [Deemer et al.,

[1] please refer to http://en.wikipedia.org/wiki/Scrum

2010]. This means that the *Organization: Developing Team* may include *Role: Product Owner* and *Role: Scrum Master* in certain situation. However, generally these two Roles are out of Developing Team.

Developing Team (DT):

$$Organization(DT) \wedge isSubOrganizationOf(DT, Pig) \wedge includes(DT, Developer)$$

$$\wedge provided(DT, CapacitiesforDeveloper)$$

Developer:

$$Role(Developer) \wedge includes(DT, Developer) \wedge required(Developer, CapacitiesforDeveloper)$$

Following the analysis above, *Organization: "Pig"* and *Organization: Developing Team* provide the following capacities required by the roles composing it.

- *Organization: "Pig"* provides:
  - *Capacity: Representing interests of stakeholder*, *Capacity: Writing User Stories*, *Capacity: Prioritizing Users Stories*, *Capacity: Maintaining the Product Backlog*, etc. (required by *Role: Product Owner*);
  - *Capacity: Protecting and keeping team focused on its tasks*, *Capacity: Enforcing rules,Removing impediments* (required by *Role: Scrum Master*);
  - *Capacity: Ensuring Scrum Process used as intended*,etc.
- *Organization: Developing Team* provides: *Capacity: Cross-functional developing skills*, *Capacity: Delivering the Product* (required by *Role: Developer*).

Additionally, in the former paragraphs, we described that Developing Team includes only one *Role: Developer*, who required cross-functional skills. This description of DT is following the official document of Scrum[2] [Deemer et al., 2010]. However, some software companies apply another different constitute of Developing Team. In these companies, the composed members of Developing Team are diverse roles. Each of them has its special ability, instead of the single *Role: Developer* required cross-functional skills. For this reason, the *Organization: Developing Team* may include various Roles, such as *Role: Designer*, *Role: System Designer*, *Role: UI Designer*, *Role: Coder*, *Role: Java Developer*,*Role: C++ Developer*, *Role: Tester Role: Unit Tester*, *Role:System Tester* and so on. Moreover, the capacity required by each of these roles are the same as the corresponding roles in the Waterfall process example of the previous chapter.

```
<Organization rdf:ID="Pig">
   <owl:sameAs rdf:resource="#ScrumTeam"/>
   <includes rdf:resource="#ProductOwner"/>
   <includes rdf:resource="#ScrumMaster"/>
   <Role rdf:ID="ScrumMaster">
      <owl:differentFrom>
         <Role rdf:ID="ProductOwner"/>
         </owl:differentFrom>
               <requird>
               <Capacity rdf:ID="ProtectingAndKeepingTeamFocusedOnItsTasks"/>
```

[2]SCRUM Phases: http://www.scrummethodology.org/scrum-phases.html

```
                </requird>
                <requird>
                <Capacity rdf:ID="EnforcingRules"/>
                </requird>
                <requird>
                <Capacity rdf:ID="RemovingImpediments"/>
                </requird>
    </Role>
    <provided rdf:resource="#EnforcingRules"/>
    <provided rdf:resource="#ProtectingAndKeepingTeamFocusedOnItsTasks"/>
    <provided rdf:resource="#EnsuringScrumProcessUsedAsIntended"/>
    <provided rdf:resource="#RemovingImpediments"/>
    <isSubOrganizationOf>
                <Organization rdf:ID="Scrum"/>
        </isSubOrganizationOf>
</Organization>
```

Listing 4.1: *Role: Scrum Master* in OWL

As an example describing how to define these elements with OWL, Figure 4.1 expresses the *Role: Scrum Master*.

ORGANIZATION: "CHICKEN"

In the *Organization: Scrum*, the group "Chicken" (Ancillary People) is a group of person including Stakeholder and Manager Roles. This group may be seen as the *Organization: "Chicken"* (*Organization: Ancillary People*). This Organization is a sub-Organization of *Organization: Scrum*. In this organization, the Stakeholder represents all people for whom the project will produce the agreed-upon benefits, which justify its production, such as the Customer and the Vendor. Furthermore, the Manager Role represents the people who will set up the environment for product development. Consequently, *Organization: "Chicken"* includes one *Role:Manager* and has one sub-Organization *Organization: Stakeholder*, which includes *Role: Customer* and *Role: Vendor*:

Chicken:

$$Organization(Chicken) \wedge includes(Chicken, Manager) \wedge isSubOrganizationOf(Stakeholder, Chicken)$$
$$\wedge provided(Chicken, CapacitiesforManager)$$

Stakeholder:

$$Organization(Stakeholder) \wedge isSubOrganizationOf(Stakeholder, Chicken) \wedge includes(Stakeholder, Customer)$$
$$\wedge includes(Stakeholder, Vendor) \wedge provided(Stakeholder, CapacitiesforCustomer)$$
$$\wedge provided(Stakeholder, CapacitiesforVendor)$$

Moreover, we could state:

$$isSubOrganizationOf(Chicken, Scrum), isSubOrganizationOf(Stakeholder, Chicken)$$
$$\Rightarrow isSubOrganizationOf(Stakeholder, Scrum)$$

Eventually, we denote various capacities supplied to corresponding roles.

- *Organization: "Chicken"* provides *Capacity: Setting up environment for product development*, which is required by *Role: Manager*.
- *Organization: "Stakeholder"* provides *Capacity: Enabling project* required by both *Role: Customer* and *Role: Vendor*.

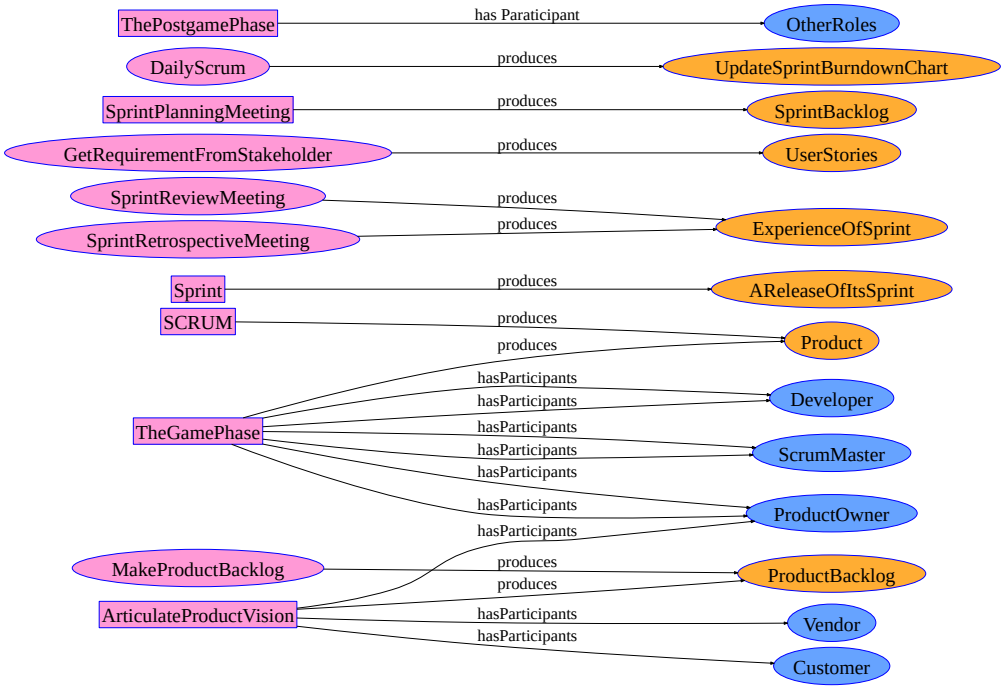

Figure 4.5: FormalizedInteraction in Scrum Process with K-CRIO

4.3.2/ Conceptualization of Interactions in Scrum Processes

So far, we have defined relevant Organizations, Roles and Capacities in Scrum. In order to describe Scrum processes, we still have to define the interactions occurring during the different Scrum phases. Interactions in K-CRIO may be considered as Casual Interaction or Formalized Interaction separately. Chat is a kind of Casual Interaction between different persons (Roles), other examples may be Exchanging Mail or Joining Conference Meeting, etc. It is Formalized Interactions of Scrum that produces relative *DesignObject*. In the Scrum processes, we have identified the following objects as *DesignObject*: *Product, Product Backlog, a sprint release, Sprint Backlog, Updated Burn-down Chart*.

Work-flow point of view

As shown in Figure 4.5, the complete Scrum Process could contains 11 formalized interactions (defined as FormalizedInteraction in K-CRIO):

- SCRUM (the whole process) produces the Product;
- ArticulateProductVision produces Product Backlog;
- GamePhase produces the same special Product with SCRUM;
- DailyScrum products Update Sprint Burndown Chart;
- SprintPlanningMeeting produces Sprint Backlog;
- GetRequirementFromStakeholer produces User Stories;

- SprintReviewMeeting and SprintRetrospectiveMeeting produces Experience Of Sprint;
- Sprint produces A Release Of Its Sprint;
- MakeProductBacklog produces Product Backlog;
- PostgamePhase produces other things, like client servicing, version control, etc.

In Figure 4.5, the concepts in pink are considered as FormalizedInteraction, in which, the ellipse shapes are considered as AtomicProcess and the box shapes are considered as CompositeProcess. Furthermore, the concepts in orange and the concepts in blue are respectively considered as DesignObject and Role.

In the following, we will detail each process. As sketched by the right block in purple of Figure 4.6, the whole Scrum process may be seen as a Composite Process that produces the *DesignObject: Product*. This Composite Process is composed by the Control Construct *Sequence* over the three following sub-processes. A Composite Process: *Articulate Product Vision* (also called *The Pregame Phase*), a Composite Process: *The Game Phase* and a Composite Process: *The Postgame Phase*[3].

In the left part of Figure 4.6, labels in pink are showing elements and their relationships in K-CRIO and labels in green are representing mapped concepts of Scrum Process conceptualized with K-CRIO. Precisely, the whole Scrum process is represented by SCRUM (CompositeProcess), which is composed of Sequence_4.

The components of Sequence_4 is ControlConstructList_6. The first of ControlConstructList_6 is the Perform_5 that processes ArticulateProductVision (CompositeProcess). The rest of ControlConstructList_6 is ControlConstructList_4.

Furthermore, the initial element of ControlConstructList_4 is Perform_TheGamePhase that processes TheGamePhase (CompositeProcess) and the rest is ControlConstructList_10. Similarly, the initial element of ControlConstructList_10 is Perform_7 that processes The Postgame Phase (CompositeProcess).

[3]SCRUM Phases: http://www.scrummethodology.org/scrum-phases.html

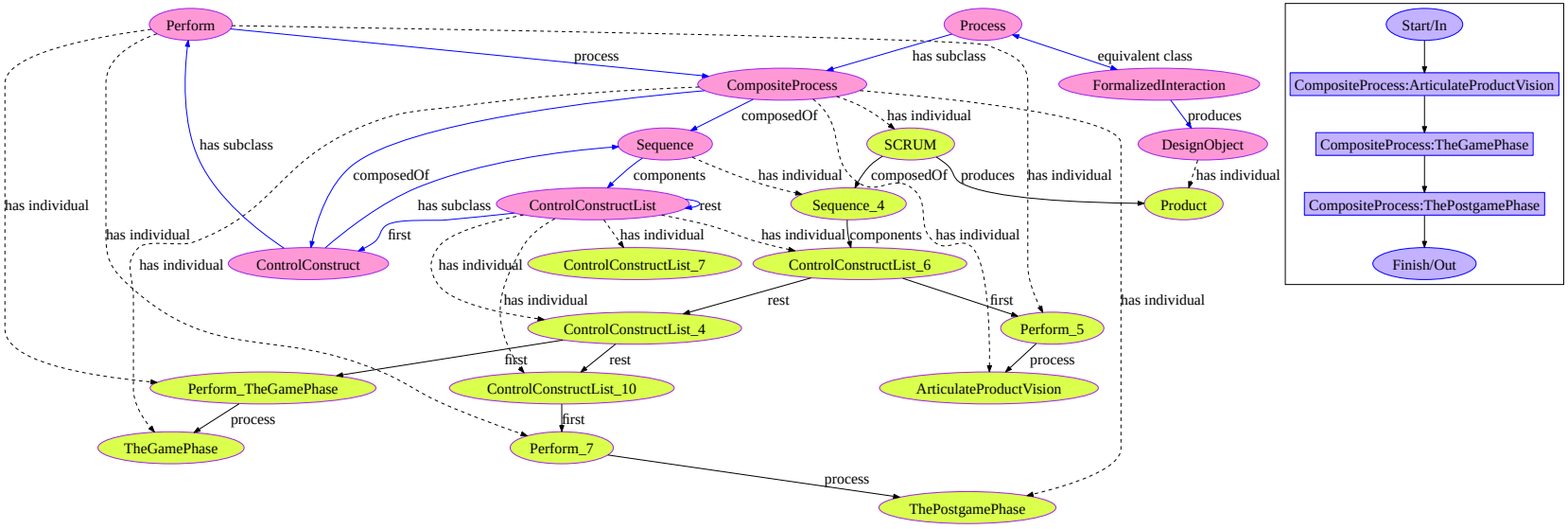

Figure 4.6: A Composite Process: *Scrum Process*

The right block in purple of Figure 4.7 generally states the *The Game Phase*, which may be seen as a composite process with the Control Construct *Repeat-While* for producing the *DesignObject: Product*. The Condition *Has Next Sprint* controls the loop and is initialized to true. While *Has Next Sprint* is true, the processes: *Sprint Planning Meeting*, *Sprint*, *Sprint Review Meeting* and *Sprint Retrospective Meeting* are executed orderly by the Control Construct *Sequence* and return a new value of *Has Next Sprint*. If *Has Next Sprint* is false, the Composite Process: *The Game Phase* is finished.

In more details, The GamePhase (CompositeProcess) is composed of Repeat-While_16. The condition (whileCondition) of Repeat-While_16 is HasNextSprint and the process (whileProcess) of the Repeat-While_16 is Sequence_19. The component of Sequence_19 is ControlConstructList_21. The first element of ControlConstructList_21 is Perform_20 that processes SprintPlanningMeeting (CompositeProcess). The rest of ControlConstructList_21 is ControlConstructList_23. Furthermore, the first element of ControlConstructList_23 is Perform_22 that processes Sprint (CompositeProcess) and the rest is ControlConstructList_7. The first element of ControlConstructList_7 is Perform_6 that processes SprintReviewMeeting (AtomicProcess). The rest of ControlConstructList_7 is ControlConstructList_9, the first of which is Perform_8 that processes SprintRetrospectiveMeeting (AtomicProcess).

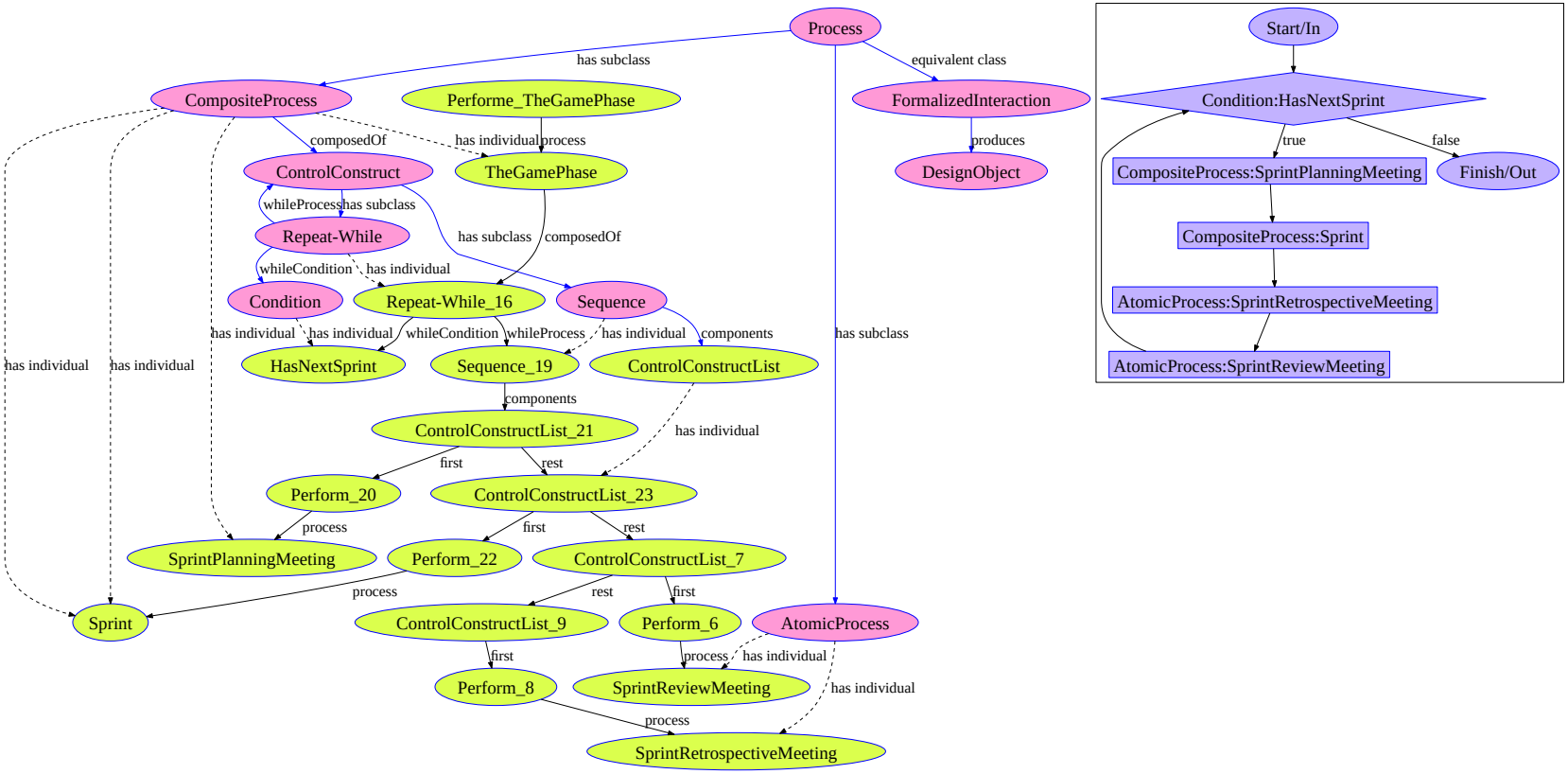

Figure 4.7: A Composite Process: *The Game Phase*

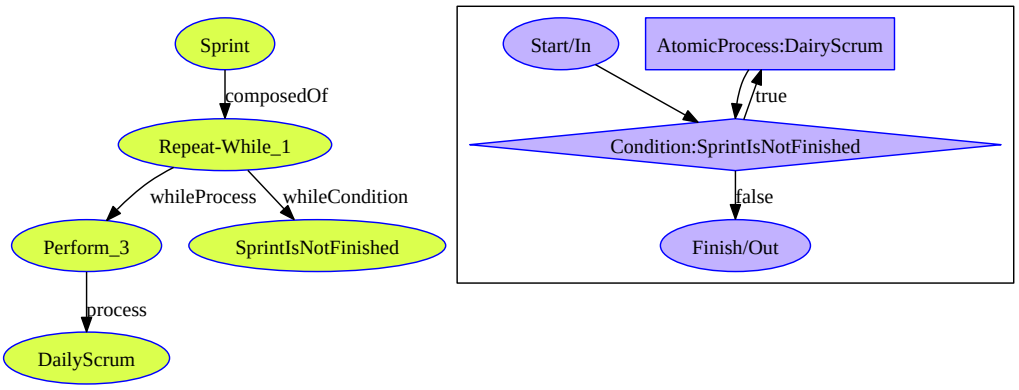

Figure 4.8: A Composite Process: Sprint

In the Scrum process Game Phase, the most important sub-process is *Sprint*, which may be seen as a composite process with the Control Construct *Repeat-While* as depicted in Figure 4.8. Every sprint is producing the *DesignObject: a release of its sprint*. The Condition *Sprint Is Not Finished* controls the loop and is initialized to be true. While *Sprint Is Not Finished* is true, the Atomic Process: *Daily Scrum* (producing *DesignObject: Updated Burndown Chart*) is executed and return a new value of *Sprint Is Not Finished*. If *Sprint Is Not Finished* is false, the Composite Process *Sprint* is finished and *Sprint Review Meeting* and *Sprint Retrospective Meeting* are executed sequentially. The mechanism for describing composite processes composed of Repeat-While is the same as what is stated above. Moreover, in Figure 4.2, we use this simple formalized interaction to explain how to express the Composite Process: *Sprint*.

```
<process: process>
   <process: CompositeProcess rdf:ID="Sprint">
      <process: composedOf>
         <process:Repeat-While rdf:ID="Repeat-While_1">
            <process:whileCondition>
               <expr:Condition rdf:ID="SprintNotBeFinished">
                        <expr:expressionLanguage rdf:resource="http://www.daml.org/services/owl-s/1.1/generic/Expression.owl#SWRL"/>
               </expr:Condition>
               </process:whileCondition>
            <process:whileProcess>
               <process:Perform rdf:ID="Perform_10">
                  <process:process>
                     <process:AtomicProcess rdf:ID="DailyScrum">
                     </process:AtomicProcess>
                  </process:process>
                        </process:Perform>
                </process:whileProcess>
         </process:Repeat-While>
      </process: composedOf>
   </process: CompositeProcess>
</process: process>
```

Listing 4.2: A Composite Process: *Sprint* by OWL-WS

Additionally, represented by the left part of Figure 4.9 *Articulate Product Vision* may be seen as a Composite Process with the Control Construct *Sequence*: an Atomic Process: *Get Requirement from Stakeholder*, in which the Product Owner communicates with the stakeholder of the product for collecting User Stories and an Atomic Process: *Make Prod-*

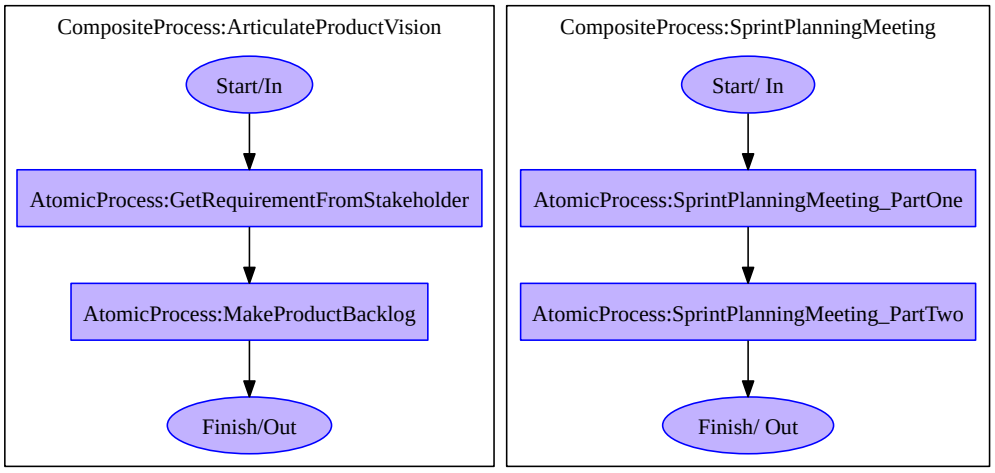

Figure 4.9: Composite Process: Articulate Product Vision and Composite Process: Sprint Planning Meeting

uct Backlog in order to produce as output the *DesignObject: Product Backlog*.

As the right part of Figure 4.9 *Sprint Planning Meeting* may be seen as a composite process aiming at the production of the *DesignObject: Sprint Backlog*. It is the same, respectively in Figure 4.9, which is composed of two sequential Atomic Processes: *Sprint Planning Meeting_PartOne* which is a discussion about what items are chosen for being realized in this Sprint, and *Sprint Planning Meeting_PartTwo* which is focusing on how to finish these tasks.

Eventually, the *Postgame Phase* may be seen as a composite process, which may be composed of the processes: *Integration Testing*, *Writing User Document*, *Training Users*, *Marketing Material Preparation* and so on.

SCHEDULING POINT OF VIEW

In this subsection, we will use the concepts related to State and Time in K-CRIO to conceptualize the states and time-table of each FormalizedInteraction. Some examples will illustrate queries and reasoning to address how the ontology could help people who pay attention about processes schedule and management of activities.

- we want to know the current state of one given project? That is to say, we want to know what are the current executing processes in this project?
  We suppose ***P*** is the process we want to find. Hence, P:

$$FormalizedInteraction(P) \wedge at(P, Doing) \wedge hasPre - Interaction(P, P') \wedge at(P', Done)$$

- we want to know which developers are late for the schedule during all the sprints?
  We suppose ***D*** is the Developer we want to find. Hence, D:

$$Developer(D) \wedge hasParticipants(Sprint, D) \wedge follows(Sprint, BeginningTime)$$
$$\wedge follows(Sprint, RealBeginningTime) \wedge follows(Sprint, EndTime) \wedge follows(Sprint, RealEndTime)$$
$$\wedge \{(RealEndTime - RealBeginningTime) > (EndTime - BeginningTime)\}$$

4.4/ CONCLUSION

In this Chapter, we have used the K-CRIO Ontology on an agile software-development process: Scrum. We have conceptualized the different elements appearing in Scrum processes and modeled how a Scrum project team could deliver a product following this specific business process.

The result of this conceptualization is expressed in OWL and corresponds to our vision of Scrum processes. This ontology helps to understand the different human behaviors involved in Scrum processes, how they interact and what are the deliverables. The content of this chapter may be a foundation for people who want to design a software system or tool to model actual Scrum Processes or manage assistance for Scrum actors.

In the next chapter, we will present our idea about designing such a software tool that is an assistant for Scrum project teams. This assistant aims at the automatization of Scrum processes, based on the conceptualization of Scrum with K-CRIO.

III

A Web Application with Multi-Agent System

5

A SCRUM WEB-BASED SYSTEM WITH MULTI-AGENT TECHNOLOGY

5.1/ INTRODUCTION

Nowadays, there exist many Agile Methodologies for Software Development such as: Scrum, XP[1], Crystal Methodology [Cockburn, 2004], Dynamic Systems Development Model (DSDM) Methodology [Stapleton, 1999], Feature-Driven Development (FDD) [Palmer and Felsing, 2001] and so on. These methodologies have gained widespread adoption and acceptance.

Therefore, many project management tools were designed for supporting such agile methodologies, including some non-free software, like ScrumWork, ThoughtWorks Mingle, etc. There also exist several open-source softwares, like IceScrum[2], Agilefant[3], XPlanner[4], etc. From the name of these existing softwares, we could deduce that some of them are particularly designed for a specific Agile method. ScrumWork and IceScrum, for example, are specific to the Scrum Method, and XPlanner is dedicated to XP. Inversely, some could be used for various Agile development methodologies. One of the hypotheses underlying this work is that, to be efficient, a tool should use a precise and semantic model of the methodology or business process being used. We thus have chosen to develop an assistance tool, specifically for the Scrum process.

Being an agile method, Scrum is inspired by their general principles: lightweight agile project management framework with broad applicability for managing and controlling iterative and incremental projects of all types based on cross-functional and self-organized teams.

Compared with existing Agile software, our contribution more specifically concerns the improvement of functions that monitor and estimate project costs and workers' efficiency. Precisely, we calculate how much money needs to be spent in order to finish one project. This computation uses the estimation time of each task or story in the project. The proposed tool also displays the estimated cost of each project timely and fire alarms if the budget is greater than a given maximum threshold. Furthermore, we calculate and display each workers' efficiency timely referenced by his/her historical situation concerning the implementation of tasks and user stories, on time, in advance or delayed.

[1] eXtreme Programming: http://www.extremeprogramming.org/index.html
[2] http://www.icescrum.org
[3] http://www.agilefant.org
[4] http://xplanner.codehaus.org

These functionalities are a real contribution compared to existing tools. Indeed, a lot of Scrum Masters expect of a Scrum tool (i) that they could provide support for the elements of a Project following the Scrum Method (building project with iterated Sprints, releases and some special charts showing the project proceeding), but also, (ii) support for indicating financial updated estimations of every Scrum Project. These estimations should help Scrum Masters in order to monitor and manage project estimated costs to adjust the project plan as soon as possible. In the meanwhile, Scrum Masters ask for a scrum tool that could give them some advises and help them take decisions, such as selecting team members, estimating task effort, etc.

Our Scrum tool is based upon a typical web-based tool written in Java. Underlying, this web-based application there is a Scrum ontology, written with K-CRIO, and a Multi-Agent System (MAS). The agents are used for their autonomy, re-activity and pro-activity capabilities to implement the above cited functionalities.

The chapter is organized as follows. Section 5.2 introduces some of the existing agile tools and their features. Section 5.3 provides an overview of the proposed Scrum Tool, by describing its targeted goals and provided features. Then, Section 5.4 presents the architecture of the Scrum Tool and details the Multi-Agent System contributing to update the estimate cost of project and workers' efficiency indicators. Finally, it draws some conclusion in Section 5.5.

5.2/ BACKGROUND: EXISTING AGILE TOOLS

The market for agile project management tools is vigorous, with dozens of offerings from both small and large vendors. Moreover, more and more products are hitting the market regularly. Market leading commercial offerings include Rally, VersionOne, Thoughtworks Mingle, and Danube Scrum Works. A few open source tools also have been around for some time and several have been used widely, like Agilefant, IceScrum, Agilo[5], eXPlainPMT[6] and so on. In this section, we will make a brief summary and analysis of the open-source alternatives, which have demonstrated significant usage measured by a combination of download volume.

It is to be noted that most of these softwares could indicate general Scrum elements and model the Scrum process. One can cite Project with its Backlog composed of User Stories or tasks, Sprints as iterations, Release and so on. Some of them propose several User roles. IceScrum has User roles: Product Owner (PO), Scrum Master(SM), Team Member, Stakeholder, Customer. Agilo has User roles: SM, PO, Team Member. The others didn't have User roles. For a special Scrum User, IceScrum is the most well suited for the Scrum Method, but not suitable for large projects with multiple teams working on a single product; only a single release and single sprint can be active at a time. Considering this aspect, Agilefant avoids this weakness and is supporting larger organizations and projects, but it has disadvantages in the "task board" and "white board" views. Agilo has excellent task board and white board views but like Trac[7] it does not support agile concepts very well. eXPlainPMT owns an intuitive interface but has serious shortcoming such as no support for the tasks duration estimation.

[5] http://www.agilosoftware.com/
[6] http://www.userstories.com/products/
[7] http://trac.edgewall.org

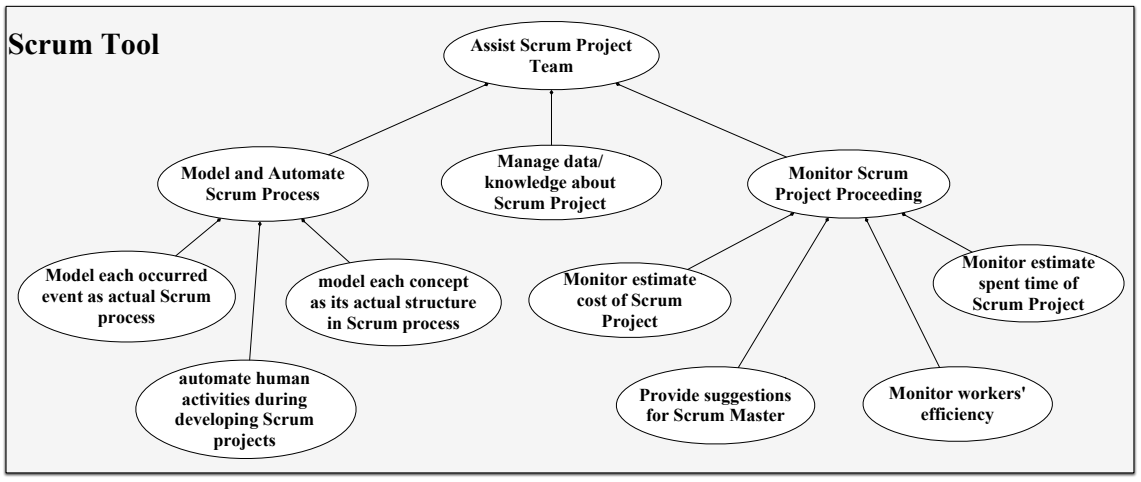

Figure 5.1: Goal Diagram of our Scrum Tool

5.3/ Overview of our Scrum Tool

5.3.1/ The goals of our Scrum Tool

The goal of our Scrum Tool is to provide an assistance platform for distributed Scrum users. On the one hand, the system aims at modeling and automating human activities and concepts during development projects following the Scrum method. More specifically, it aims at managing data/knowledge about elements and events appearing in the Scrum process. On the other hand, the system monitors projects by estimating both the cost and the time of each task whatever it granularity. Moreover, the system can monitor each human worker in order to compute it efficiency and provide suggestions for Scrum Masters for helping them take decisions.

Considering the distribution of the users of our Scrum Tool and the distribution of projects data we have deployed the system as a web-based system. Figure 5.1 presents the goal hierarchy analysis for the system . The general goal of our Scrum Tool is to assist Scrum project teams. In order to achieve this goal, we have identified three sub goals:

- modeling and automating Scrum process, which is in turn divided into three sub goals:
  - modeling each occurred event as actual Scrum process, for example, as actual human activities occurring in Scrum process, the project is released as iterative sprints;
  - modeling each concept as its actual structure in Scrum process, for example, Product Backlog is composed of user stories for representing requirements;
  - automating human activities during developing Scrum projects.
- managing data/knowledge of Scrum projects, that is to say, accompanying Scrum projects life cycle, related data/knowledge are reported and updated.
- monitoring Scrum projects, which is divided into four sub goals as
  - monitoring estimated spent time of Scrum projects;

- monitoring estimated costs of Scrum projects, that is to say, estimated cost (spent money) of each project is computed and monitored in real time automatically, according to the users actions and their task execution duration;
- monitoring workers' efficiency, that is to say, monitor the rate of finishing tasks on time for each worker;
- providing suggestions for Scrum Masters.

5.3.2/ FUNCTIONS OF OUR SCRUM TOOL

Based on the conventional functions of Scrum software, in our Scrum Tool, a user should register firstly by filling required information. Once identified the user will have a role assigned within the Scrum project. With the confirmation of identifier and password, a logged user can create new project as Scrum Master or operate all invoked projects. Each role has specifics allowed operations. The possible roles were described in the chapter 4. Among these roles one can cite: Scrum Master, Product Owner or Developer, and, for example, each developer could update it spent hours on his/her assigned tasks.

In the meanwhile, each project owns the following specific elements:

- a dashboard (for representing last activities about this project),
- a sandbox (for building temporary user stories or tasks before being valid in Backlog),
- a backlog
- a time line
- a release plan
- a sprint plan
- a team

Each element is mapped onto one button in the menu bar of system interface. We could see this menu bar in the top of system interface, like Figure 5.2.

Users could build and edit user stories, update the hierarchy of tasks with estimated effort (hours) in the Sandbox and Backlog, and could build and edit iterated sprints and releases to implement user stories or tasks selected from Backlog for one project. In order to precisely monitor each project, each project owns specifics burn-down charts of sprints and releases. Generally, all the basic performances necessary for building projects as Scrum method are implemented in our Scrum Tool. However, these functions described above are synthesizing basic functions in existing Agile software for Scrum users, which are not the main contribution.

As presented in the introduction, one of the contributions of our system is to display how much money needs to be spent for finishing one project. The idea is to help Scrum Masters understanding the estimated cost of each project in a timely manner and control the money spent for each project. The result of this function could be shown in the left of Figure 5.2. A second contribution is to compute and record each workers' efficiency. In other words, the system records all the tasks (user stories) assigned to the workers

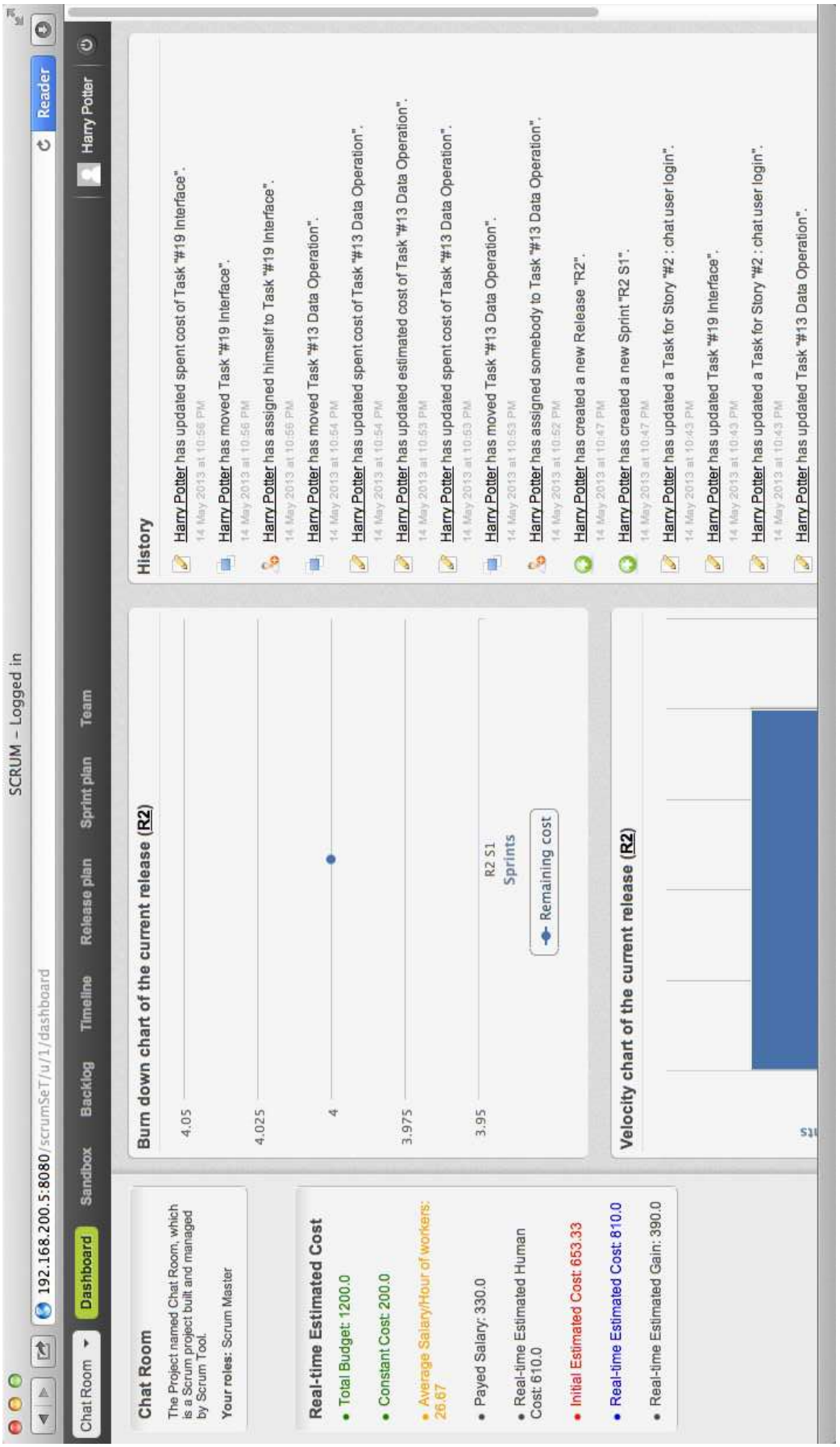

Figure 5.2: Elements for each project

and calculate the rate of workers' finishing tasks on time with the initial estimated efforts (hours) and real-time (maybe final) actual spent time of all the related tasks. The result of this function is represented in Figure 5.3 .This recorded efficiency can help Scrum Masters in order to take decision about team members and specifically help promoting the accuracy of estimation of a task assigned to a specific worker involvement.

Last but not least, our last contribution consists in providing suggestions to Scrum Master, including suggestions for assigning team members to tasks (as Figure 5.4), or suggestions for re-estimating tasks effort, because of its assigned worker, etc. All these improvements are really needed by scrum masters as it allow to accurately monitor scrum projects in real-time.

5.4/ Description Architecture of Scrum Web-based System

To resume, the Scrum tool presented in this chapter is a traditional web-based tool developed in Java. This tool relies on the modeling of Scrum Method by one Organizational Ontology, K-CRIO, and on a Multi-Agent System (MAS) animating the functionalities. Agents are used for updating estimated costs of each project and each workers' efficiency, when any change happens in the project. In the following sections, we will detail the architecture of the system.

5.4.1/ Multi-Agent System

The majority of the existing tools for Scrum Method are based on a typical web-based architecture. Their main functionalities are to build the project and monitor project lifecycle. Our Scrum Tool is trying to help it users to monitor and control cost, time and workers' efficiency in Scrum Projects so as to make projects not only follow the Scrum Method, as for example, using iterating sprints and daily meetings, but also being understood and controlled by its Scrum Master. This enhanced control is reached owing to the accurate estimation of time and cost, when any change is happening during the development. In order to achieve this target, we use a Multi-Agent System (MAS) for improve the original web-based System. Indeed, MAS have proven to be a fitted paradigm to provide solutions in situations where resources are spatially and temporally distributed. Additionally, it could enhance overall system performance, specifically along the dimensions of computational efficiency, reliability, extensibility, robustness, maintainability, responsiveness, flexibility, and reuse. The MAS in our scrum tool is implemented with the platform Janus[8] [Gaud et al., 2009].

As stated in Figure 5.5, our Scrum Tool uses the popular framework Struts 2[9]. The left part Figure 5.5 represents how a user could get the related result operating from a web-browser. User actions trigger effects on the MAS, various agents will be launched accompanying with the successful log-in of any user.

In the following, we brought elements from the analysis of the MAS in our Scrum Tool with the ASPECS methodology [Cossentino et al., 2010a] as the guide, which is a step-by-

[8] http://www.janus-project.org
[9] http://struts.apache.org/2.x/

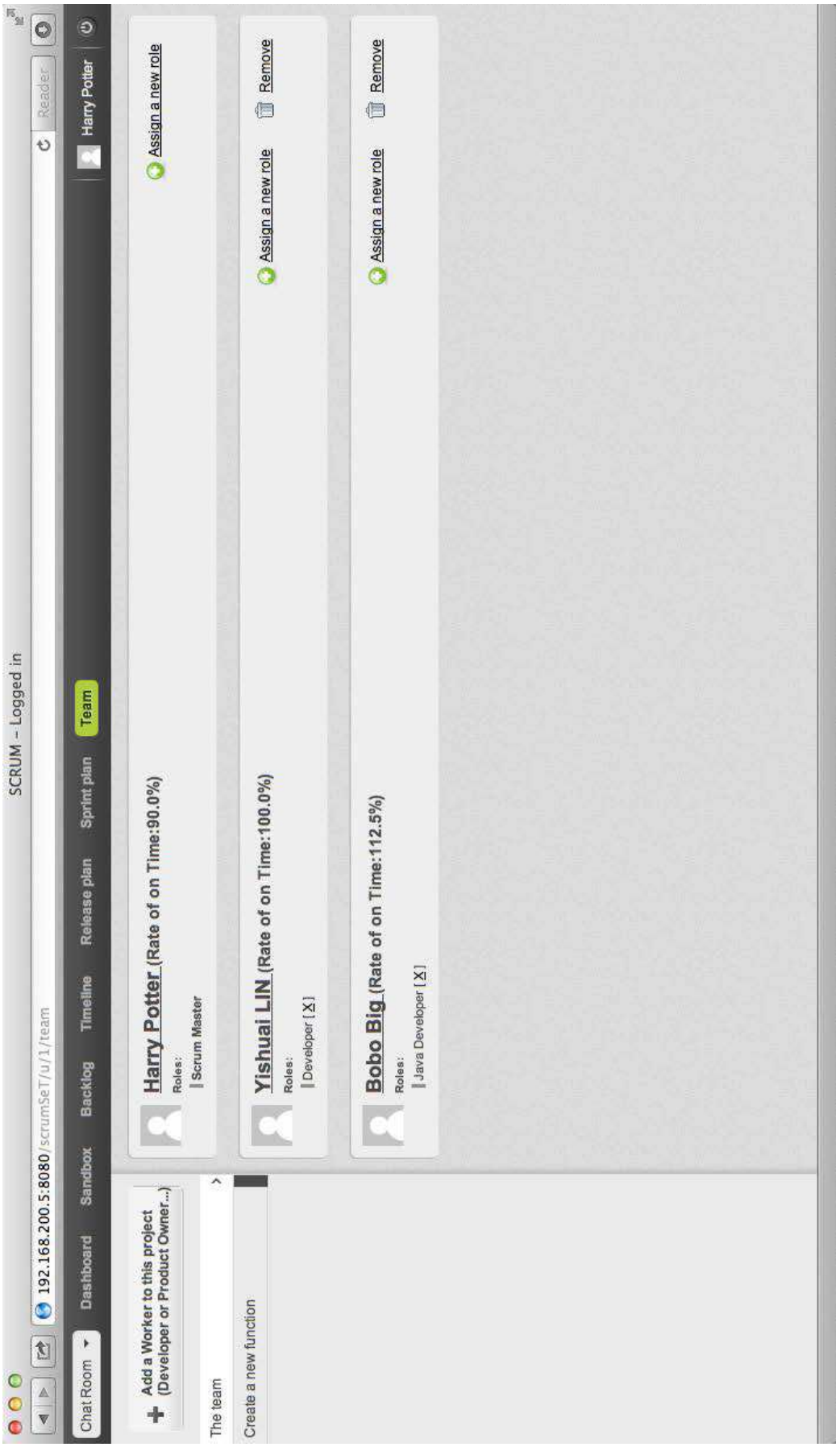

Figure 5.3: Monitor workers' efficiency

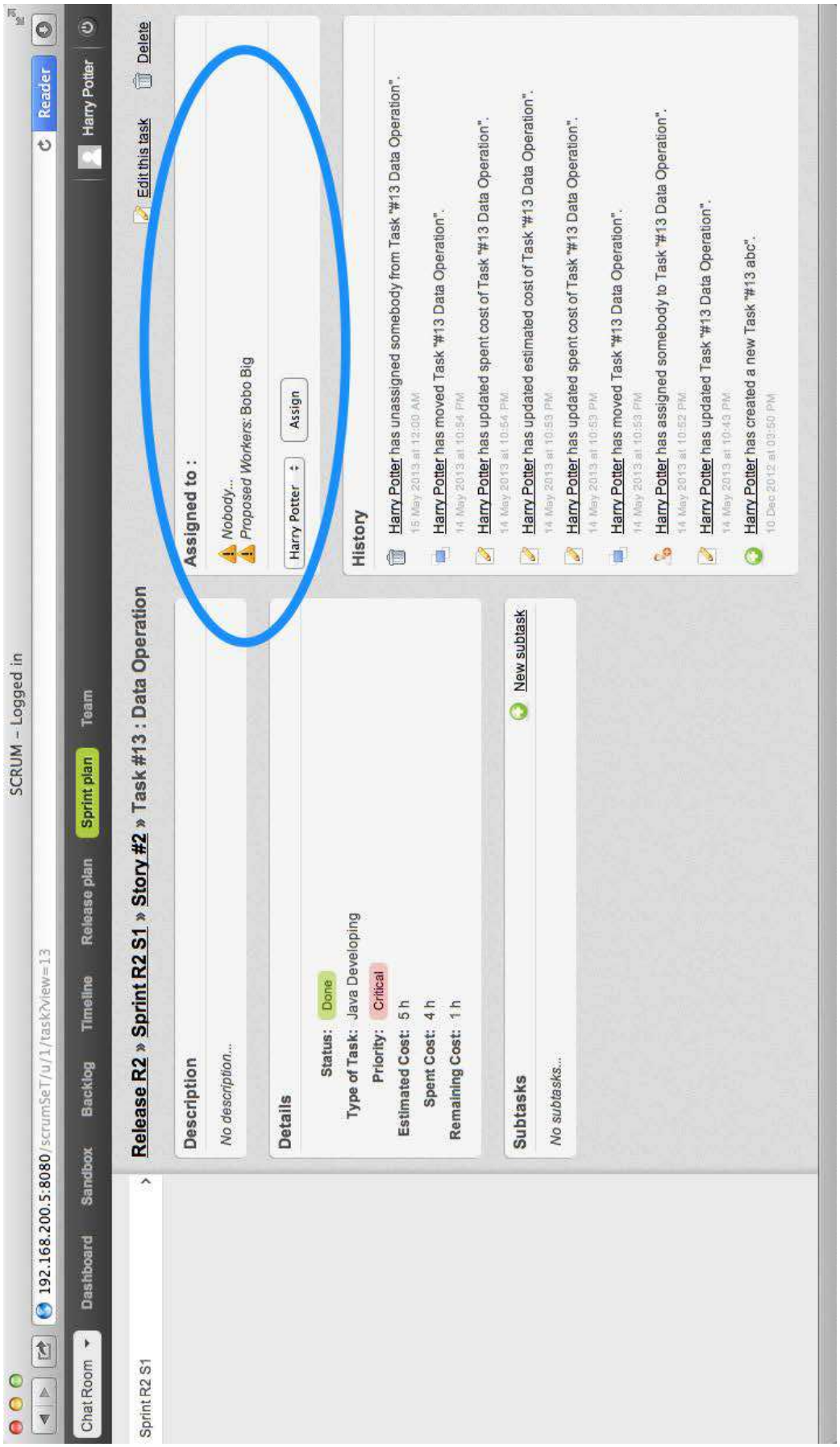

Figure 5.4: Suggestions for assigning team members to tasks

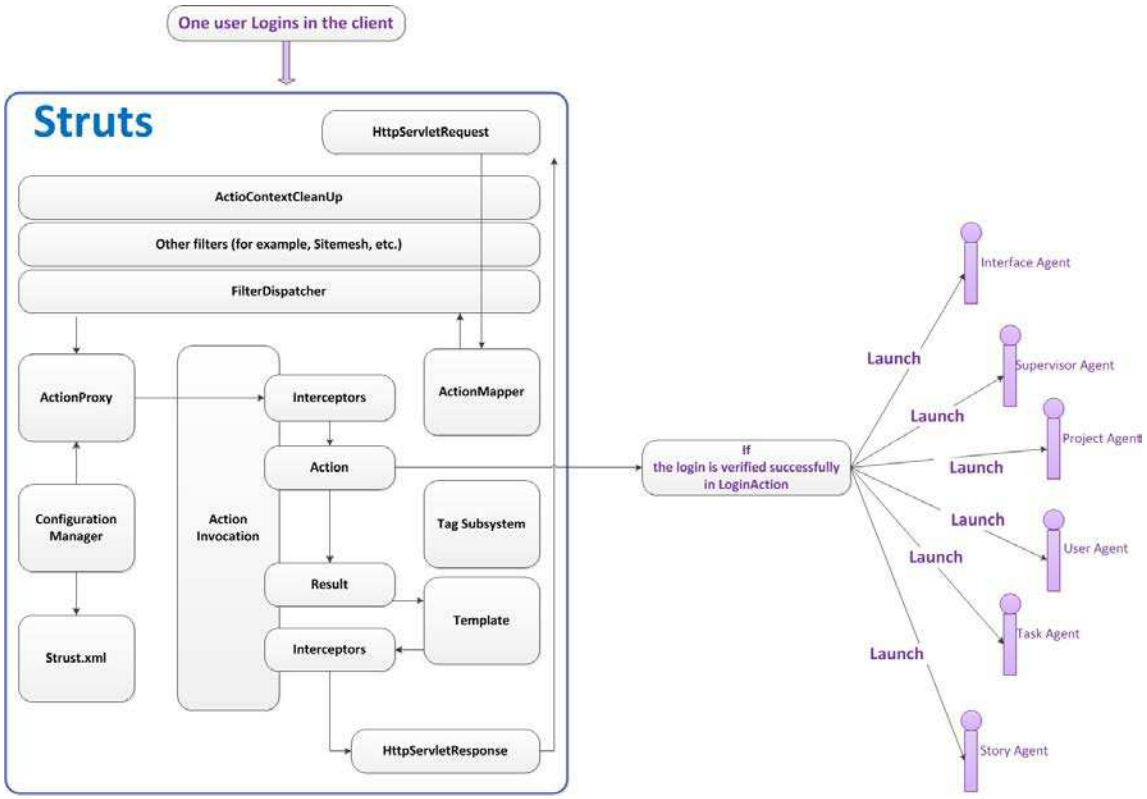

Figure 5.5: Import MAS to typical web-based System following Struts 2

step to code software engineering process dedicated for the analysis and design of MAS. Based on the organizational ontology, the functionalities to be realized are assigned to organizations that accomplish them also by means of the hierarchical decomposition of the organization structure in sub-organizations. The principle of the analysis is as that each Organization owns its goal. This goal is fulfilled by the interactions of the different roles that define the organization. Each role may require special capacities, and is played by an agent. The behavior of each role can be seen as a contribution towards the achievement of the goal of its Organization. The structure of organizations, roles and agents in our Scrum Tool is described in Figure 5.6.

5.4.2/ IS ORGANIZATION

IS Organization: The goal (g0 in Figure5.6) of the Organization is getting the actions of users in the client part (always in the browser), in which there are two roles played by two related agents:

- **Interface Role**: it is played by the *InterfaceAgent*, which will be activated when the *InterfaceAgent* is launched. The required abilities of *Interface Role* are
  - getting the message about users' operations in the client part,
  - transferring received messages to *Supervisor Role*.

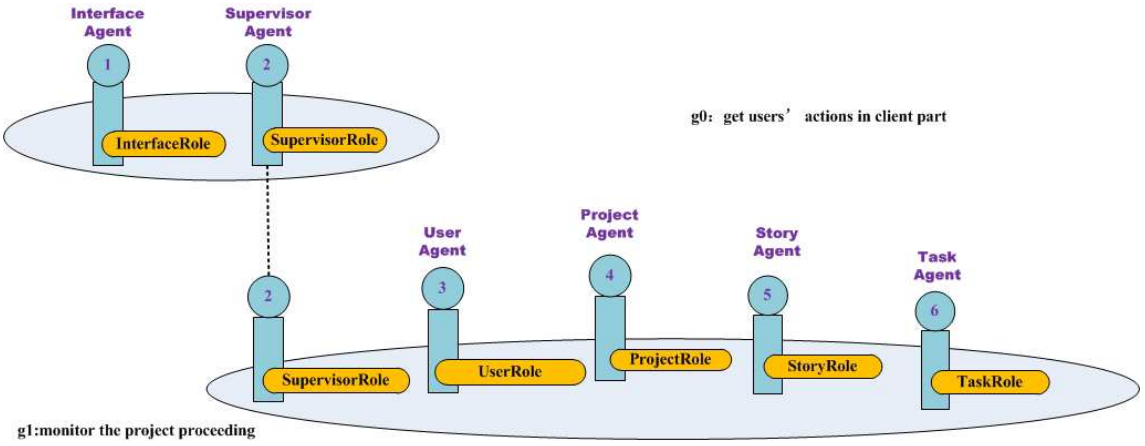

Figure 5.6: The structure of MAS

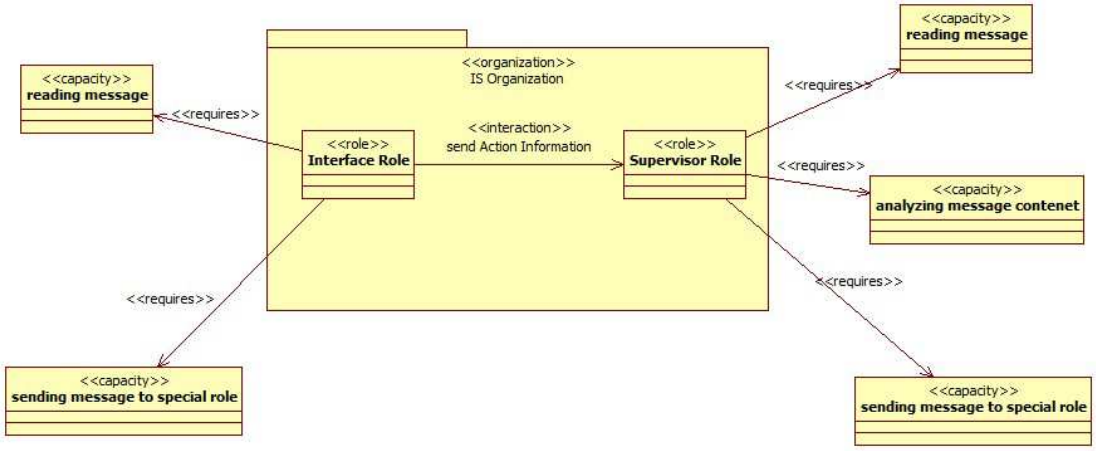

Figure 5.7: Interactions, Role and Capacity Identification in *IS Organization*

- **Supervisor Role**: it is played by the *SupervisorAgent*, which will be activated when the *SupervisorAgent* is launched. In order to get the common target of IS Organization, the abilities required by Supervisor Role are
  - reading received messages and sending messages to a special role,
  - analyzing the content of messages for making decision about who is the receiver of the id message sent by *Supervisor Role*. The receiver could be one of the roles included in Monitor Organization, except Supervisor Role, Project Role, Task Role, Story Role or User Role.

Figure 5.7 describes the fragment of Interactions, Role and Capacity Identification in *IS Organization*. Exclusively, interactions between *InterfaceRole* and *SupervisorRole* in *IS Organization* are as that *InterfaceRole* sent the information of users' actions in the client to *SupervisorRole*, including the type of actions and id of related action, for example, the type of action maybe " update project" and Id of related action is Id of this updated project.

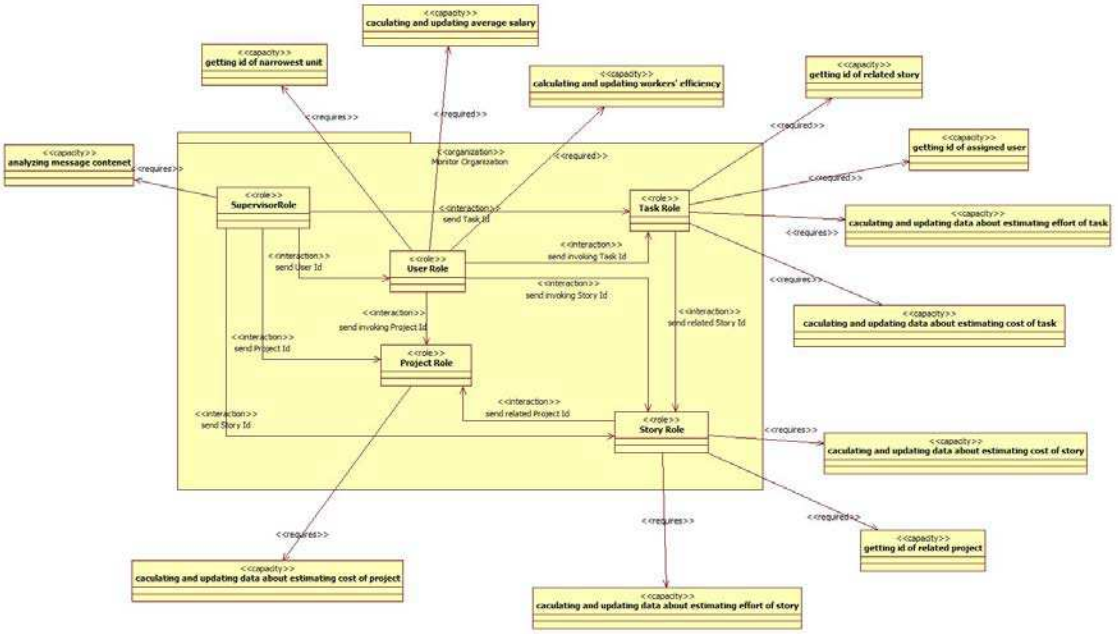

Figure 5.8: Interactions, Role and Capacity Identification in *Monitor Organization*

5.4.3/ MONITOR ORGANIZATION

Monitor Organization: The goal (g1 in Figure 5.6) of this Organization is to monitor the time effort and cost of projects, workers' efficiency (rate of finishing tasks on time) and help Scrum Master make a decision during the scrum project execution. In this organization, there are five roles played by five agents respectively:

- **Supervisor Role**: it is included by two Organizations: *IS Organization* and *Monitor Organization*. The detail of the role is described in the IS Organization details.
- **Project Role**: it is played by the *ProjectAgent*, which will be activated when the *ProjectAgent* is launched. For fulfilling the goal project monitoring, the abilities required are:
  - reading the id of the message to know which project changed and needs to be monitored,
  - calculating and updating all the data related to cost estimation of this project, including the cost spent for paying salaries, the estimated cost for finishing the rest of the stories and tasks in the product backlog, etc..
- **User Role**: it is played by the *UserAgent*, which will be activated when the *UserAgent* is launched. Similarly, the abilities required are
  - reading received messages and sending id messages to special role,
  - getting related project and updating the average salary of workers' of this project
  - for this project, getting id of the narrower unit between task and user story. Precisely, task is narrower than user story.

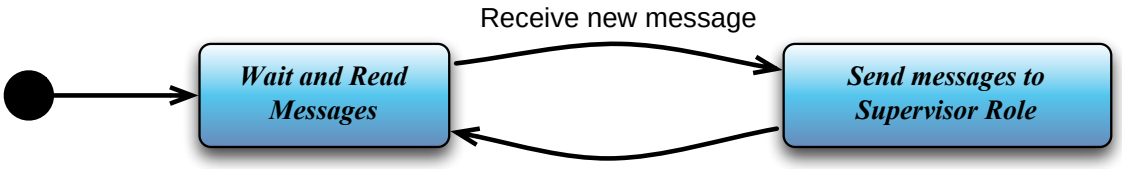

Figure 5.9: Role behavior Description of InterfaceRole

  - calculating and updating workers' efficiency (rate of finishing tasks on time).
- **Story Role**: it is played by the *StoryAgent*, which will be activated as the *StoryAgent* is launched. With the common aim of monitoring project, the Story Role required abilities are
  - reading messages and sending messages to a special role,
  - getting id of project, which contains the story with special id.
  - calculating and updating all the data related to the effort estimation of the story identified by the received id, including the spent time, the estimated effort for finishing the rest of the tasks in the story, etc..
  - calculating and updating all the data about cost estimation of the story identified by the received id, including the cost spent for paying salaries, the estimated cost for finishing the rest of the tasks in the story, etc..
- **Task Role**: it is played by the *TaskAgent*, which will be activated when the *TaskAgent* is launched. It is the finest grain unit of completion in a Scrum project. The required abilities about the role are
  - reading received message and sending message to special role,
  - getting id of story, which contains the task with special id,
  - getting id of user, who is working on the task with special id,
  - calculating and updating all the data related to effort estimation of the task defined by the received id, including the time spent, the estimated effort for finishing the rest of the work of the task etc..
  - calculating and updating all the data related to cost estimation of this task defined by the received id, including the cost spent for paying salaries, the estimated cost for finishing the rest of the work in the task, etc..

Figure 5.8 describes the fragment of Interactions, Role and Capacity Identification in *Monitor Organization*. Additionally, in Figure 5.8, the capacities of reading message and sending message to special role for each role are omitted for highlighting particular capacities for each role. Accompanying the Figure 5.15, 5.16 and 5.17, the pattern of the interactions and the scenarios description of all roles will be presented in the following.

- In the lifecycle of the InterfaceRole, there are two sequence states circularly as
  - the default state is reading messages about user actions in the client part repeatedly until a new message is received and then go to the next state.
  - send all the new actions messages to SupervisorRole and then go back to the state of reading messages.

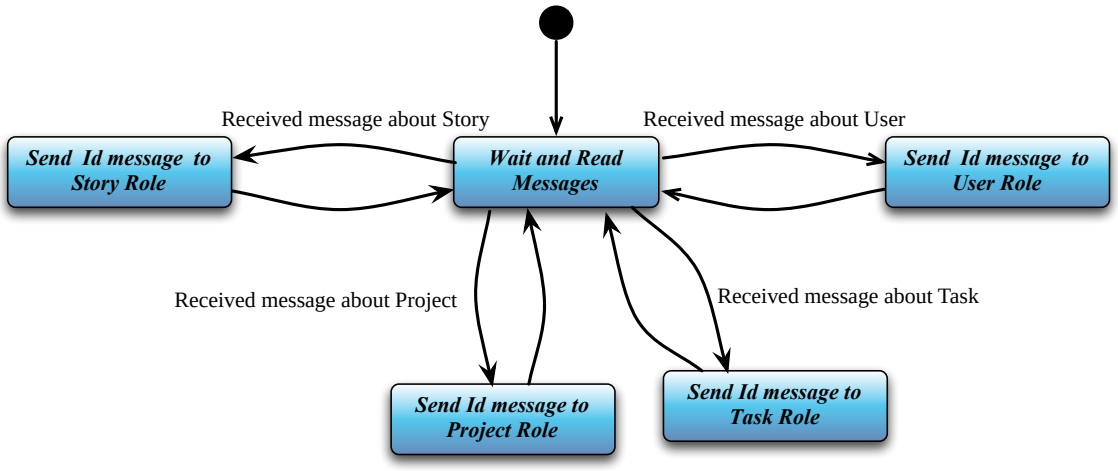

Figure 5.10: Role behavior Description of SupervisorRole

The State Diagram of InterfaceRole is presented in Figure 5.9.

- In the lifecycle of SupervisorRole, there are two sequence states circularly as
  - the default state is reading messages received from InterfaceRole repeatedly until a new message is received, it then goes to the sending message state.
  - send related Id to right kind Role after judging the content of received message, which is about User, Task, Story or Project. For Example, if the message is " update project", the SupervisorRole will send id of the project (ProjectId) to ProjectRole. Similarly, if the message is " user", the SupervisorRole will send id of the user (UserId) to UserRole and so on. After that, it goes back to the reading messages state.

The State Diagram of SupervisorRole is presented in Figure 5.10.

- In the cycle life of UserRole, there are three sequence states circularly as
  - the default state is reading messages (the content of message is id of User) repeatedly until a new message is received and then goes to a next state depending on the sender of the message.
  - if the sender of the message is SupervisorRole, it gets the id of related projects and goes to the next state.
    * calculates and updates the average salary of all the projects got above and goes to next state.
    * gets the the id of narrower unit between task and story composed of the project above (task is a narrower unit than story), and then goes to the state of sending message. That is to say, if the id of task composed of the project is not null, then *UserRole* goes to the state of sending message. Inversely, if id of task composed of the project is null, *UserRole* continues to get the id of story composed of the project, then goes to the state of sending message.

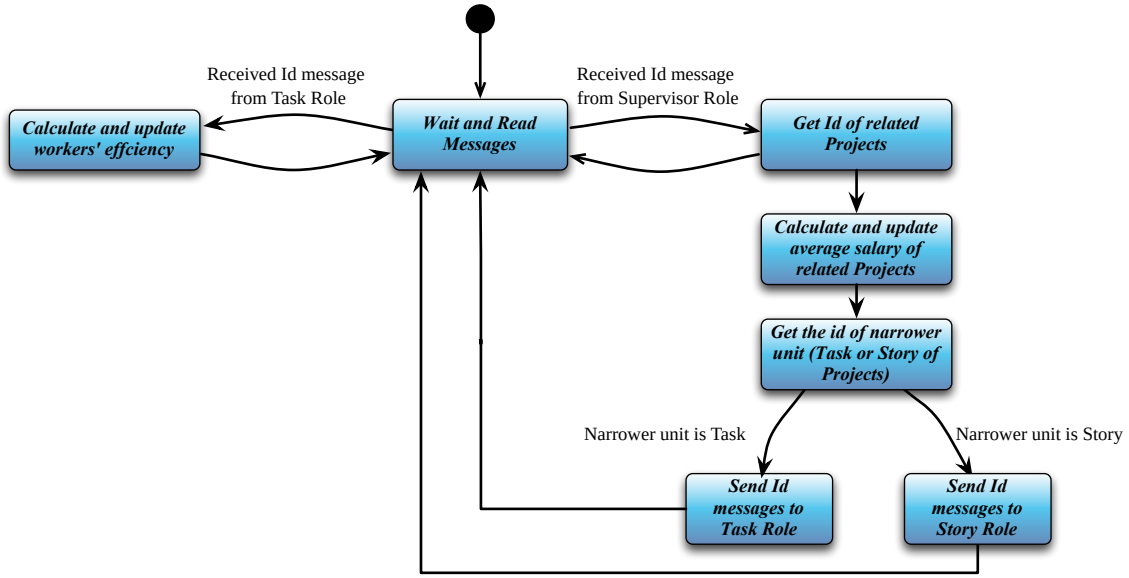

Figure 5.11: Role behavior Description of UserRole

  - sends the id obtained (if existing) in the upper state to the related Role. Specifically, if the id obtained is about Task, the UserRole will send a message including the id of the task to TaskRole. After that it goes back to the reading messages state .
- if the sender of the message is TaskRole, it calculates and updates the workers' efficiency, then it goes back to the reading message state.

The State Diagram of UserRole is presented in Figure 5.11.

- In the cycle life of TaskRole, there are six sequence states circularly as
  - the default state is reading messages (the content of the message is the id of the Task) repeatedly until a new message is received and then it goes to the monitoring task state.
  - monitors the task referenced by the received id, specifically it calculates and updates the data about real-time estimation of cost of the task. After that, it goes to the next state.
  - gets id of related story containing the task with special id, and then goes to the next state.
  - gets id of users involved in the task with special id, and then goes to the next state.
  - sends the obtained id of the user to UserRole and then goes to the next state.
  - sends the obtained id of the story to StoryRole and then goes back to the reading messages state.

The State Diagram of TaskRole is presented in Figure 5.12.

- In the StoryRole lifecycle, there are four sequence states circularly as

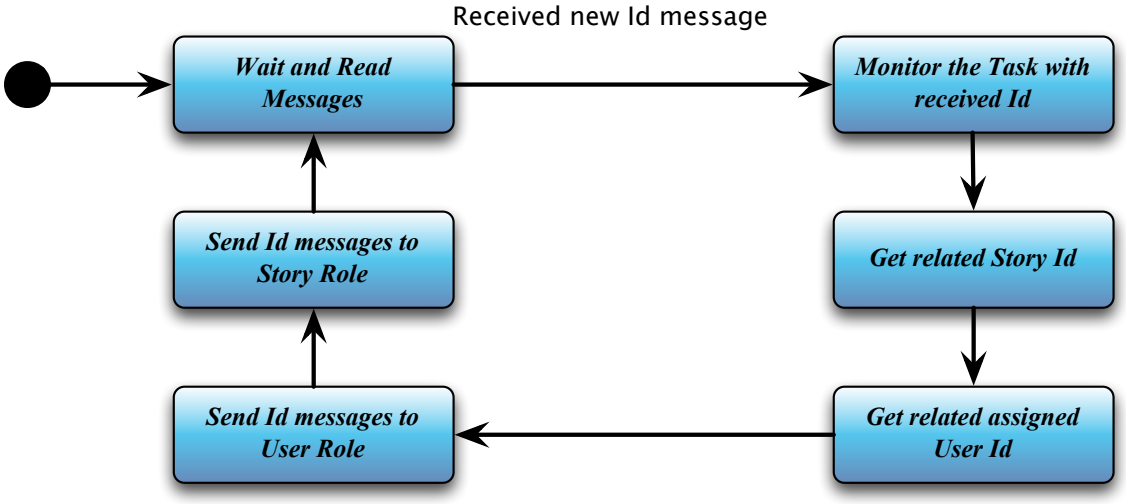

Figure 5.12: Role behavior Description of TaskRole

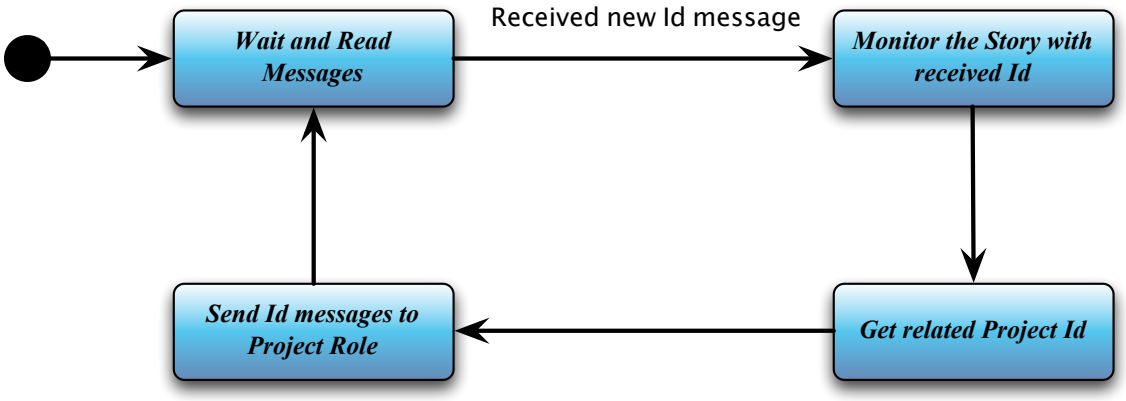

Figure 5.13: Role behavior Description of StoryRole

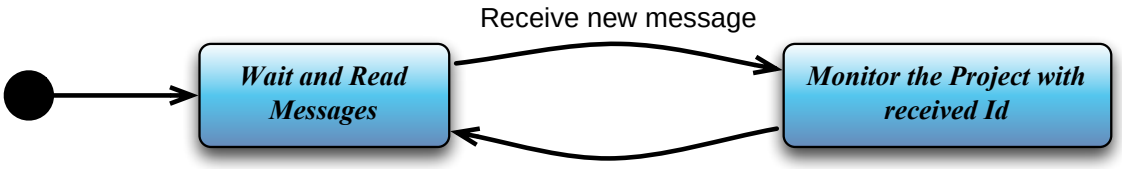

Figure 5.14: Role behavior Description of ProjectRole

  - the default state is reading messages (the content of the message is the Story id) repeatedly until a new message is received and goes to the next state of monitoring story.
  - monitors the story referenced by the received id, specifically calculates and updates the data related to the real-time estimation of the cost of the story. Then goes to the next state.
  - gets id of related project containing the story with special id, and then goes to the next state.
  - sends the id of the project (if existing) obtained from the upper state to ProjectRole. Then goes back to the reading messages state.

  The State Diagram of StoryRole is presented in Figure 5.13.

- In the ProjectRole lifecycle, there are two sequence states circularly as

  - the default state is reading messages (the content of the message is the id of the Project) repeatedly until a new message is received and then goes to the monitoring project state.
  - monitors the project referenced by the received id, specifically calculates and updates the data related to real-time estimation of the cost of the project. Then goes back to the reading messages state.

  The State Diagram of ProjectRole is presented in Figure 5.14.

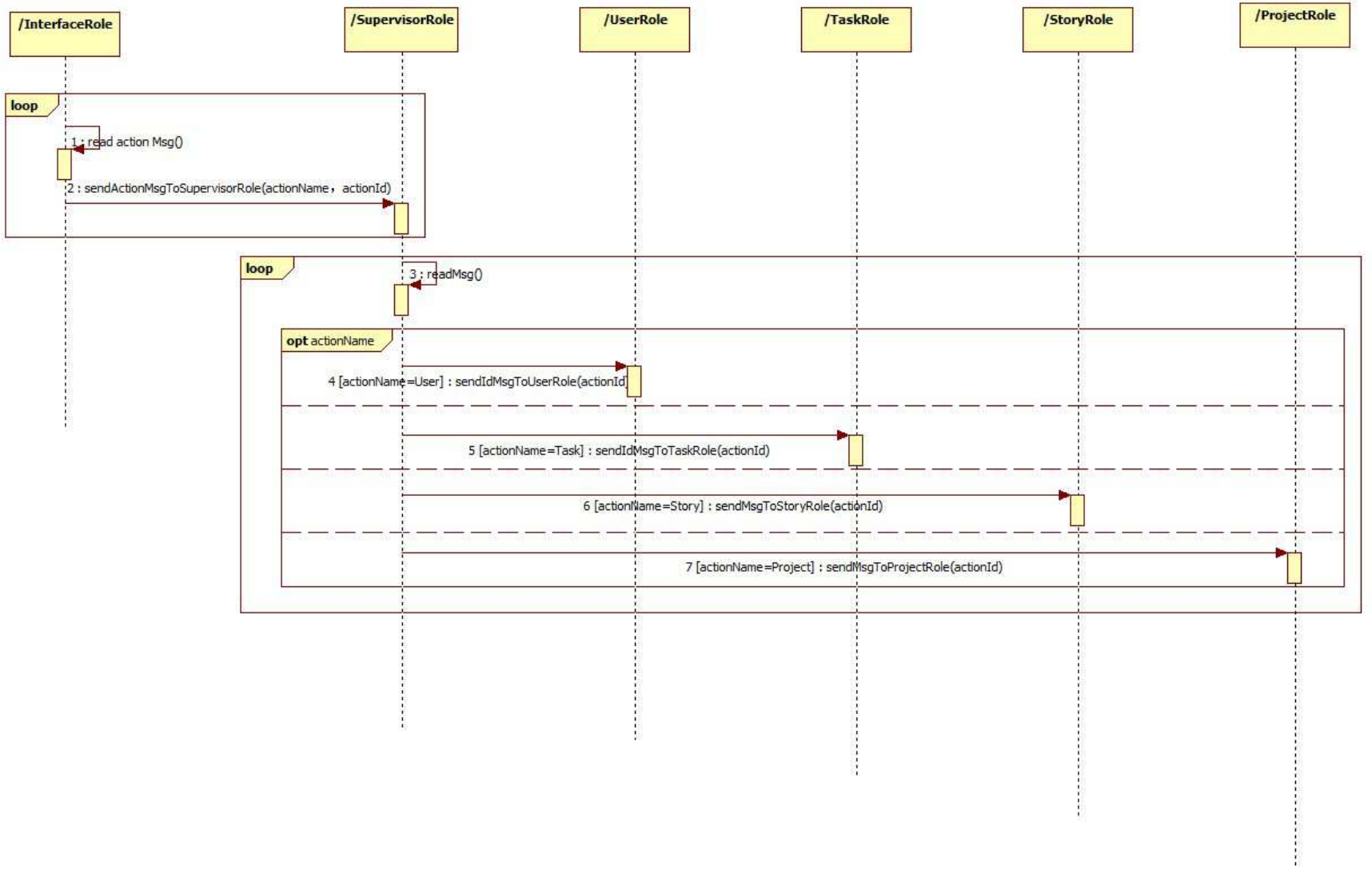

Figure 5.15: Scenarios Description: the part of `InterfaceRole` and `SupervisorRole`

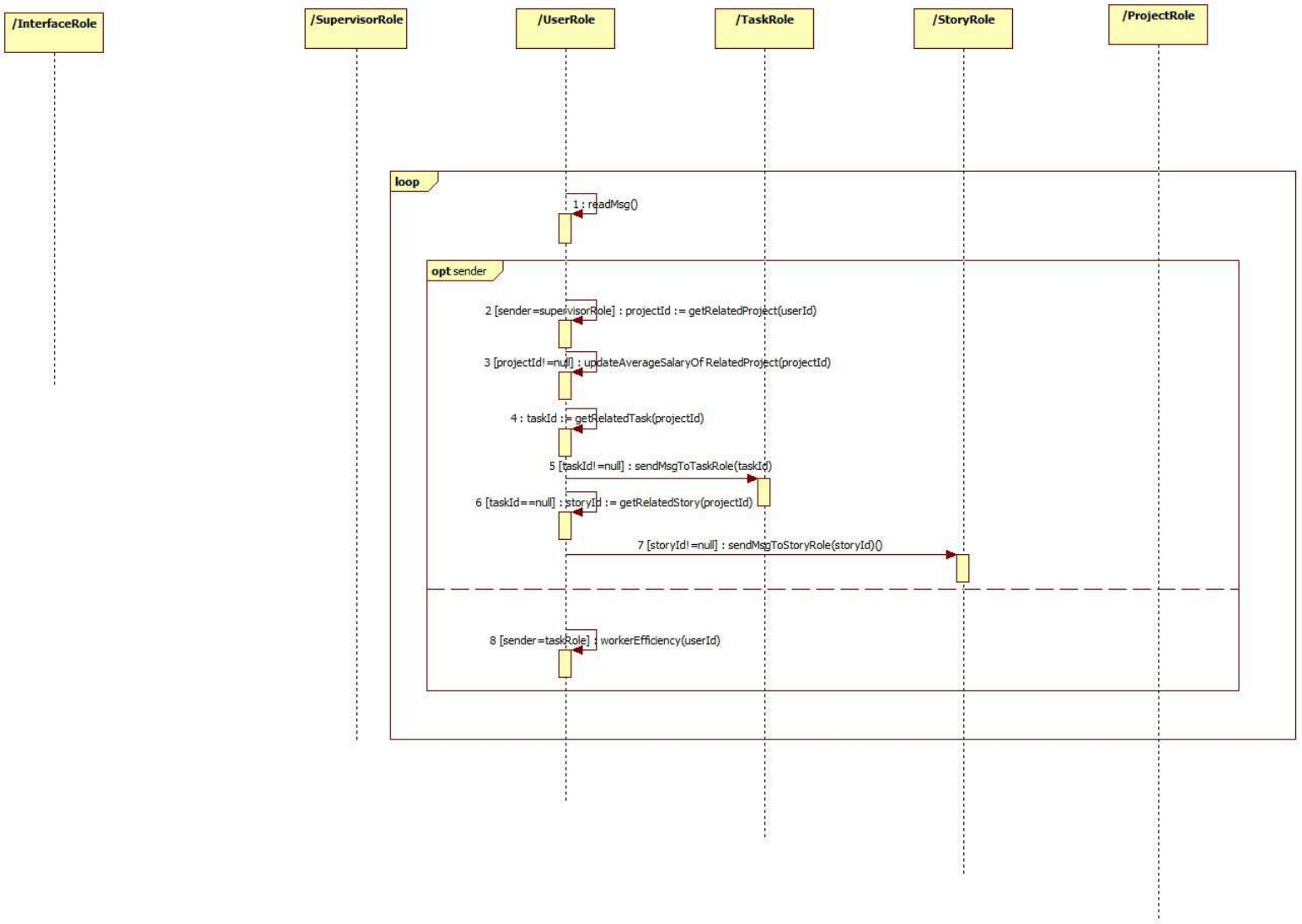

Figure 5.16: Scenarios Description: the part of `UserRole` (following the Figure 5.15)

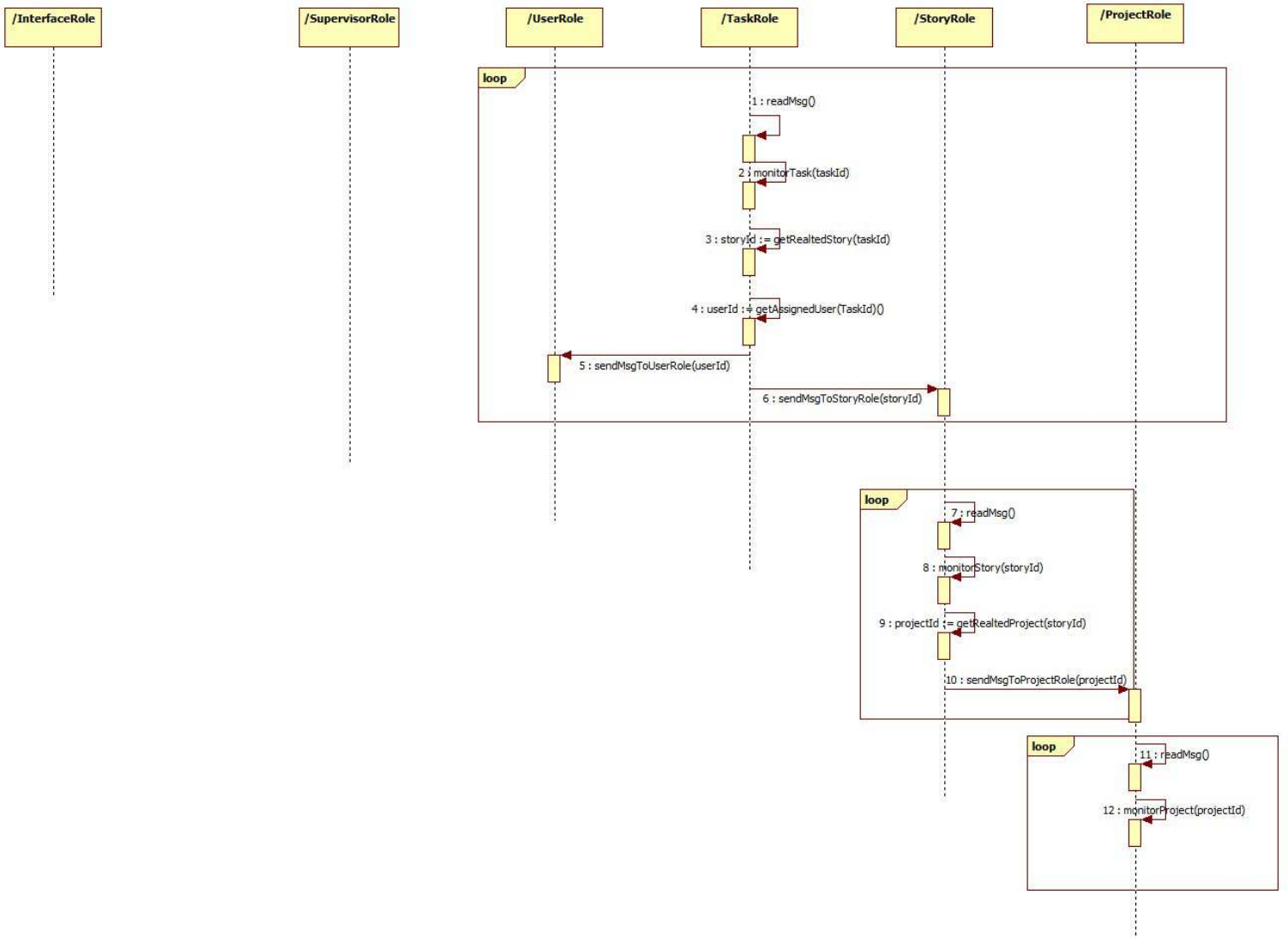

Figure 5.17: Scenarios Description: the part of `TaskRole`, `StoryRole` and `ProjectRole` (following the Figure 5.16)

5.4.3.1/ How to monitor estimate cost of projects ?

The enumeration of all the capacities required by various roles and provided by respective organizations is represented by the concept of capacity in Figures 5.7 and 5.8. These capacities are: reading message, sending message to a special role, calculating and updating data about the estimated cost of the project and so on. In this subsection, we didn't detail all of them but the important capacities instructing how to monitor the estimate cost of one project and workers' efficiency (rate of finishing tasks on time) via the interactions of Project Role, Task Role, Story Role, User Role included in their Monitor Organization.

For each Scrum Project, we suppose one project cost is composed of the constant cost and the real-time estimated personal salary cost.

- The former is the constant cost that Scrum Master could generally ensure at the beginning of the project. In our Scrum Tool, each Scrum Master must provide the constant cost of project when creating the project. That is to say, when the estimated cost of the project is over or below the budget of the project, it is difficult for Scrum Masters to get expected profits of the project by adjusting this part cost.
- Considering about the estimated salary cost, it is containing the paid part and the estimation part for finish the rest of the project. Furthermore, the cost of the project is associated with all the user stories composing the project, except the stories still in Sandbox. More precisely,
  - the paid part cost is the sum of the paid cost of all the stories in Sprints,
  - the paying part cost is the sum of the estimated cost of stories in the Backlog,
  - the paying part cost for finishing the rest work of stories in the Sprints.

  Similarly, each user story in Backlog or Sprint may be composed by several tasks. Therefore, the cost of one user story is the sum of the costs of composed tasks, in which, the paid part of the story is the sum of the spent costs of composed tasks and the paying part is the sum of estimated costs for finishing the rest work of all the composed tasks.

Hence, the two different costs of each project is updated by Project Role required Capacity "Monitoring Project" as:

$$\textbf{real-time project cost} = \textit{constant cost} + \textit{real-time personal estimated cost of project} \tag{5.1}$$

$$\textbf{real-time personal estimated cost of project} = \textit{paid cost of project}+\textit{paying cost for rest work of project} \tag{5.2}$$

The two different costs of each story is updated by Story Role required Capacity "Monitoring Story" as:

$$\textbf{paid cost of project} = \sum \textit{paid cost of user story in the project} \tag{5.3}$$

$$\textbf{paying cost for rest work of project} = \sum \textit{paying cost for remain work of user story in the project} \tag{5.4}$$

Two different costs of each task is updated by Task Role required Capacity "Monitoring Task" (see Figure 5.18) as:

$$\textbf{paid cost of user story} = \sum \textit{paid cost of tasks composing the user story} \quad (5.5)$$

$$\textbf{paying cost of user story} = \sum \textit{paying cost of tasks composing the user story} \quad (5.6)$$

Based on the hierarchical analysis, the question turns to how to calculate two kinds of cost of each task (paid cost and estimate rest cost). Figure 5.18 represents the capacity of updating task cost, required by Task Role. At beginning, it needs to set the salary unit for the task via judging whether the task was assigned to one special user.

- If the current task was assigned to one user, set the salary unit as the salary of this user:

$$\textbf{\textit{salary unit}} \longleftarrow \textit{salary of assigned user} \quad (5.7)$$

- Otherwise, if the current task was not assigned to any user, set the average salary of invoked project as salary unit for the current task:

$$\textbf{\textit{salary unit}} \longleftarrow \textit{average salary of invoked project} \quad (5.8)$$

Following, update paid cost of the task as spent hours of the task timing salary unit got above.

$$\textit{paid cost = spent hours} \times \textbf{\textit{salary unit}} \quad (5.9)$$

Update remaining paying cost as estimate remaining hours timing salary unit, and real-time estimate cost of the task as the sum of paid cost and remaining paying cost.

$$\textit{remaining paying cost = estimate remaining hours} \times \textbf{\textit{salary unit}} \quad (5.10)$$

$$\textit{real-time estimate cost = paid cost + remaining paying cost} \quad (5.11)$$

The estimated remaining hours for one task could be updated by its assigned user at any time. According to the role scenario, described in the Figure 5.15, 5.16 and 5.17, any change done, by a user, on any project information (e.g. Time or Cost of Task, Story, Project) is monitored by the Interface Role. This change is then forwarded to the Supervisor Role which identifies the impacted level in the project decomposition. This level is then informed (e.g. Project, Story or Task). Knowing this level, the changed information is propagated by message to the upper level (e.g. from Task to Story, or from Story to Project). At each level, a dedicated role is in charge of updating the impacted informations (e.g. Time or Cost).

5.4.3.2/ HOW TO MONITOR WORKERS' EFFICIENCY?

In this section, we present how to monitor users' efficiency, the rate of finishing tasks on time. One users' efficiency is defined initially as 100% . As the user contributing to some project, User Role will update the rate as the situation of the user working on tasks with the formula 5.12. For User Role, the signal of re-calculating one users' efficiency is that any task contributed by the user is changed. For example, one developer updated the remaining hours for one task or the Scrum Master updated the initial estimated hours of

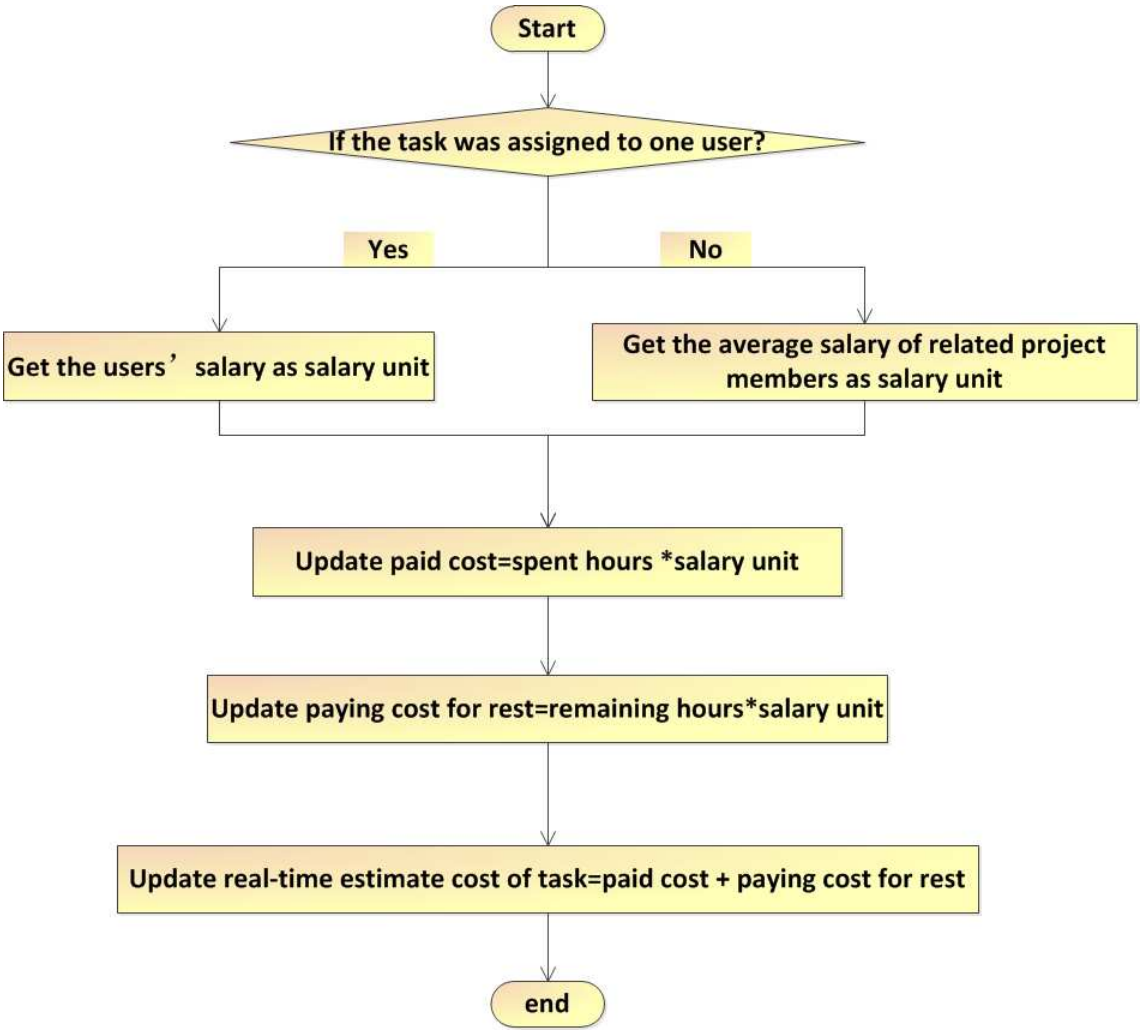

Figure 5.18: How to calculate task cost

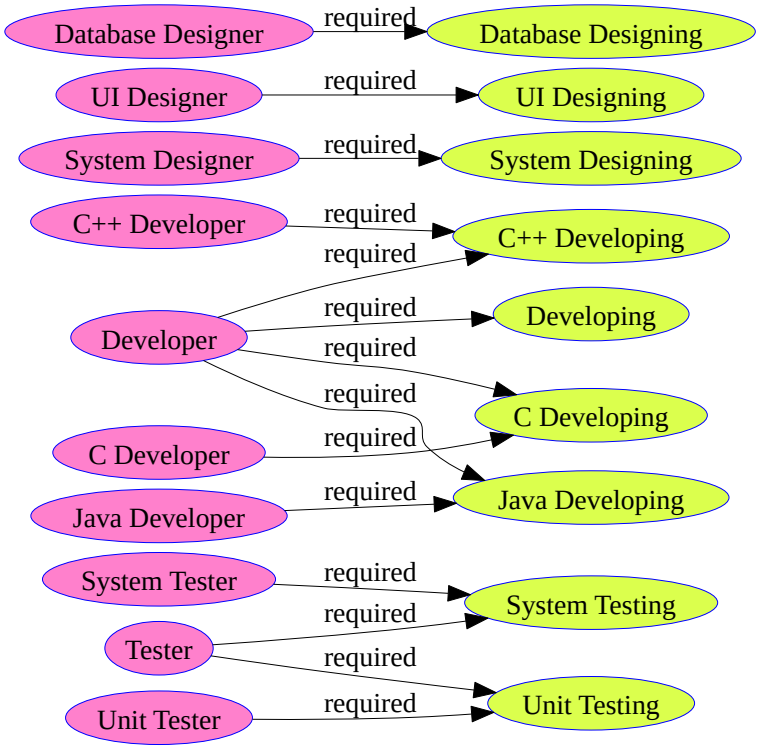

Figure 5.19: Roles and their required Capacities in Developing Team

one task. From the view of interactions between roles, it is when Task Role receives a message and got id of assigned users, it sends these ids to User Role who will update related users' efficiency as defined in equation 5.12.

$$\textbf{Rate of finishing tasks on time} = \frac{\sum \textit{Initial Estimated Time of Task}}{\sum \textit{Real-time Estimated Time of Task}} \times \textbf{100\%} \quad (5.12)$$

5.4.3.3/ How to provide suggestions to Scrum Masters ?

During Scrum projects, our Scrum Tool wants to give Scrum Master some suggestions in order to take decisions, such as recommending workers for one selected task from the current project team and help the Scrum Master estimating one task assigned to a specific person. The general idea for the first suggestion is based on the conceptualization of Developing Team with K-CRIO. If the user as the member of one Scrum Project owns its particular role, like Tester, who is the defined role required special defined capacities (abilities) in the Organization: Developing Team. Hence, our Scrum Tool will recommend this user (Tester) to tasks described about testing, unit testing and system testing. All the roles and their required capacities are represented in Figure 5.19, in which roles are in pink and capacities are in green.

Furthermore, for helping Scrum Masters with the estimation of one task assigned to a specific person, our Scrum Tool estimates with the initial estimated time of the task and the rate of finishing tasks on time for the task assigned user/worker (assigned workers'

efficiency), that as defined in equation 5.13:

$$\textbf{Estimated spent hours for one Task} = \frac{\textit{Initial Estimated Time of the Task}}{\textit{Rate of finishing tasks on time for Assigned User}} \tag{5.13}$$

5.5/ CONCLUSION

The content of this chapter is our first attempt to design and implement an intelligent assistance system to support human activities within business processes, based on the result of modeling and conceptualization of business processes with K-CRIO.

Scrum is our selected business process. Therefore, in this chapter, we have presented a Scrum Tool for assisting Scrum project teams to deliver products/projects. It is an intelligent system, which is based on a web-based architecture, including a Multi-Agent System, and based on a conceptualization of the Scrum process with the K-CRIO ontology presented in the previous chapters.

Precisely, in this chapter, we have analyzed the goal of our Scrum Tool. This tool, relying on a riche semantic model of the scrum process can not only manage related data appearing during Scrum processes, but also help user to monitor and control the cost/time, workers' efficiency, and provide suggestions to Scrum Masters. In order to achieve these goals, we have compared our specific contributions with some existing agile tools. After that, we have given details of the analysis and design of the system.

Finally, it is the last chapter for describing our contributions to this thesis. Following this chapter, we will make a common conclusion to sum up all contents of our work presented in this thesis. Then, we will discuss possible extension directions for this work in the future.

IV

Conclusions and Perspectives

6

CONCLUSION

6.1/ GENERAL CONCLUSION

Along this thesis, we have worked on the definition of an organizational ontology K-CRIO for modeling and conceptualizing various business processes in enterprises. Our approach was then applied to the specific field of software engineering processes. Moreover, we have used K-CRIO to model and conceptualize an agile software-development process, namely the Scrum process. Based on the result of the modeling and conceptualization of Scrum, we have designed and implemented an intelligent assistance system for Scrum project teams. This tool aims to support human activities within Scrum process. Precisely, the system is a kind of web-based system that uses a Multi-Agent System to implement it functionalities.

In order to understand the works presented in this thesis, we briefly review the path that we followed throughout this work.

In Chapter 2, we discuss related works. Firstly, we describe the statements of business processes in enterprises. After that, we present and analyze some existing models/methods to support business processes in enterprises. For representing elements and knowledge appearing in processes in enterprises, we discuss the features of ontologies used for modeling enterprises. Moreover, we present definition of agents and advantages of Multi-Agent Systems.

Based on the analysis of the observations of Chapter 2, we present in Chapter 3 our approach, an organizational ontology named K-CRIO, used to model enterprise business processes, and especially to model and conceptualize human relationships and activities within these processes. The main characteristics of K-CRIO are summarized hereafter:

- K-CRIO Ontology is defined with the Ontology Web Language.
- The core concepts in K-CRIO are: Organization, Role, Capacity and Interaction. These four concepts are all defined by an `owl:class`. The relationships among these concepts are described by an `owl:ObjectProperty`.
- An organization could have sub-organizations, and include one or more roles.
- A role may identify a person, status or generic behavior. A role is a necessary part to achieve social objectives (goals of its organization). Different roles in the same organization could interact with each other.

- Capacity is an ability required by each role to finish its job or contributing to the common goals of its organization.
- Interaction could be understood as a work-flow, in order to achieve a goal or to contribute to a goal of one organization.

Based on the definition of K-CRIO Ontology of Chapter 3, we apply it to the modeling of various enterprise business processes. Chapter 4 is dedicated to the description of the Scrum software development process, widely used in IT enterprises. There were two major reasons to provide this process as a complete case study of K-CRIO Ontology. Firstly, it demonstrates the ability of K-CRIO to model enterprise business process. Secondly, it is the basic building block of the design and the implementation of an intelligent assistant Scrum Tool, described in Chapter 5. Hence, the K-CRIO Ontology is used to conceptualize each item appearing in the Scrum process and to model how a Scrum project team could deliver its products following this business process. Chapter 5 thus consists of the analysis and descriptions of this Scrum Tool, including:

- the goal is to provide a platform for distributed Scrum users to model actual human activities and workflow within Scrum process, manager/monitor Scrum projects and provide suggestions to Scrum users.
- the architecture of the scrum tool is a traditional web-based tool, including a Multi-Agent System developed with the MAS platform Janus. Agents were used for monitoring and updating estimated costs of each project and each workers' efficiency.

6.2/ Perspective and Further Research Directions

The three main axis presented here, can be considered as extensions of this work. The first one aims at improving the advantages of the K-CRIO Ontology in terms of semantic reasoning. The second axe concerns the creation of K-CRIO sub-ontologies for specific processes. This task is not easy and a tool to visualize the work of modeling business processes with K-CRIO should enhance the usability of K-CRIO. The last axis deal with the improvement of our Scrum Tool in terms of the functionalities, mainly to enhance its usability and level of intelligence.

6.2.1/ Implementation of semantic application based on the K-CRIO Ontology

Ontologies can be used to formally specify concepts and relationships within a domain. The resulting logic based representation form a conceptual model that can help with storage, management and sharing of data among different research groups. Relational databases can effectively store and retrieve extensional data, but Ontologies have advantages in queries with semantic reasoning.

The amount of works in knowledge engineering domain and the knowledge management fields shows that such approaches can be used to go a step further and add knowledge for support business processes as already presented in [Gomes et al., 2009, Monticolo et al., 2006, 2007]. Therefore, based upon the K-CRIO Ontology, we may design and build

a semantic database to gather, save and update data in enterprises. With this semantic database, we should easily query to answer questions like

- “What interactions are in progress currently in one given organization?”
- “One worker has played which roles lastly?” and so on.

6.2.2/ Implementing an editor for K-CRIO

Although, enterprises require methods to define complex business processes, actually, most people in enterprises are not acquainted with what is Ontology and OWL, and they also are not used to reason about method for modeling their business processes.

Therefore, K-CRIO may need a matched assistance tool. This tool may provide operations and a graphical representation for visualizing the work of modeling business processes by K-CRIO, in order to expand users of K-CRIO to people in enterprises who are not familiar with ontologies. Moreover, automatic or semi-automatic assistance and guidelines such as the ones presented in [Miled et al., 2009] may be used to support the process of ontology creation.

6.2.3/ Methodology based upon K-CRIO for guiding the design and implementation of intelligent assistance tools to support business processes

The Scrum Tool designed and implemented in this work, relies on the conceptualization of the Scrum process with K-CRIO. Actually, conceptualization of other processes with K-CRIO is a mandatory first step for people that aim to design and develop intelligent software.

However, in this PhD, we have not followed a principled approach for the conceptualization and the analysis and design of the tool. There may be a lot of works if one wants to develop a tool for another process. Thus, a methodology leading from user (process user in this case) requirements to an assistance tool should be helpful to minimize the amount of works. Such methodology, if the developed system is still based on MAS as it its the case in this PhD, can be based on ASPECS. Indeed, with ASPECS the integration of ontological analysis during the preliminary phases is already done. What is lacking is some kind of activity that links a process description to intelligent agent support.

6.2.4/ Improving Scrum Tool

As an intelligent assistance system for Scrum project teams to support and automate their working, the current Scrum Tool is just the primary version. It has a lot of functions that could be improved, such as:

- reusing of released user stories/ tasks in new projects;
- helping Scrum Master to find problems leading to project delays/ project deficit;
- providing solutions to Scrum project teams, after understanding problems.

Bibliography

Readings in distributed artificial intelligence. ed by a. h. bond and l. gasser (morgan kaufmann, 1988). *SIGART Bull.*, (110):25–26, October 1989. ISSN 0163-5719. doi: 10.1145/74664.1059787. URL http://doi.acm.org/10.1145/74664.1059787. Reviewer-Van Dyke Parunak, H.

BPMN 2.0 by Example Version 1.0. Technical Report 3, Object Management Group, June 2010.

E. Adam, C. Kolski, R. Mandiau, and E. Vergison. A software engineering workbench for modeling groupware activities. In *Proceedings of the Tenth International Conference on Human-Computer Interaction*, volume 4 of *Universal access in HCI : inclusive design in the information society*, pages 1499–1503, 2003.

Emmanuel Adam and Rene Mandiau. A hierarchical and by role multi-agent organization: Application to the information retrieval. In Felix F. Ramos Corchado, Victor Larios-Rosillo, and Herwig Unger, editors, *Advanced Distributed Systems (5th ISSADS'05)*, volume 3563 of *Lecture Notes in Computer Science (LNCS)*, pages 291–300. Springer-Verlag (New York), Guadalajara, Mexico, January 2005, Revised Selected Paper.

Michael Ashburner, Catherine Ball, Judith Blake, David Botstein, Heather Butler, Michael Cherry, Allan Davis, Kara Dolinski, Selina Dwight, Janan Eppig, Midori Harris, David Hill, Laurie Issel-Tarver, Andrew Kasarskis, Suzanna Lewis, John Matese, Joel Richardson, Martin Ringwald, Gerald Rubin, and Gavin Sherlock. Gene ontology: tool for the unification of biology. *Nat Genet*, 25(1):25–29, 2000. ISSN 1061-4036. doi: 10.1038/75556.

Nattapat Attiratanasunthron and Jittat Fakcharoenphol. A running time analysis of an ant colony optimization algorithm for shortest paths in directed acyclic graphs. *Inf. Process. Lett.*, 105(3):88–92, January 2008. ISSN 0020-0190. doi: 10.1016/j.ipl.2007.08.013. URL http://dx.doi.org/10.1016/j.ipl.2007.08.013.

Stefano Beco, Barbara Cantalupo, Ludovico Giammarino, Nikolaos Matskanis, and Mike Surridge. OWL-WS: A workflow ontology for dynamic grid service composition. In *eScience*, pages 148–155. IEEE Computer Society, 2005. ISBN 0-7695-2448-6.

Fabio Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*. Wiley, 2007. ISBN 0470057475.

A. Bernaras, I. Laresgoiti, N. Bartolome, and J. Corera. An ontology for fault diagnosis in electrical networks. In *Intelligent Systems Applications to Power Systems*, 1996.

W. N. Borst. Construction of engineering ontologies for knowledge sharing and reuse. 1997.

E. Bottazzi and R. Ferrario. Preliminaries to a dolce ontology of organizations. *International Journal of Business Process Integration and Management*, 4(4):225–238, 2009.

Emanuele Bottazzi, Emanuele Bottazzi, and Roberta" Ferrario. R.: A path to an ontology of organizations. *IN: PROCS. OF EDOC INT. WORKSHOP ON VOCABULARIES, ONTOLOGIES AND RULES FOR THE ENTERPRISE (VORTE*, 2005. doi: 10.1.1.100.2130.

Christine Bourjot, Vincent Chevrier, and Vincent Thomas. A new swarm mechanism based on social spiders colonies: From web weaving to region detection. *Web Intelligence and Agent Systems*, 1(1):47–64, March 2003. ISSN 1570-1263.

M. Bunge. *Treatise on Basic Philosophy: Volume 3: Ontology I: The Furniture of the World*. Springer, 1 edition, 1977. ISBN 9027707804.

A. Gangemi N. Guarino A. Oltramari C. Masolo, S. Borgo and L. Schneider. Wonderweb deliverable d17. the wonderweb library of foundational ontologies and the dolce ontology. Technical report, November 2002.

Giovanni Caire, Wim Coulier, Francisco J. Garijo, Jorge Gomez, Juan Pavón, Francisco Leal, Paulo Chainho, Paul E. Kearney, Jamie Stark, Richard Evans, and Philippe Massonet. Agent oriented analysis using message/uml. In Michael Wooldridge, Gerhard Weiß, and Paolo Ciancarini, editors, *Agent-Oriented Software Engineering II, Second International Workshop, AOSE 2001, Montreal, Canada, May 29, 2001, Revised Papers and Invited Contributions*, volume 2222 of *Lecture Notes in Computer Science*, pages 119–135. Springer, 2002. ISBN 3-540-43282-5.

Coral Calero, Francisco Ruiz, and Mario Piattini. *Ontologies for Software Engineering and Software Technology*. 2006.

Cristiano Castelfranchi and Jean-Pierre Müller, editors. *From Reaction to Cognition*, volume 957 of *LNCS*, Neuchâtel, Switzerland, August 1993. 5th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW, Springer.

Lawrence Chung, Brian Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering (the Kluwer International series in Software Engineering, Volume 5)*. Springer, 1st edition, 1999. ISBN 0792386663.

Alistair Cockburn. *Crystal Clear: A Human-Powered Methodology for Small Teams*. Addison-Wesley Professional, 1 edition, 2004. ISBN 0201699478.

Massimo Cossentino, Nicolas Gaud, Stéphane Galland, Vincent Hilaire, and Abderrafiaa Koukam. A holonic metamodel for agent-oriented analysis and design. In *LNAI 4659 "Holonic and Multi-Agent Systems for Manufacturing" (HoloMAS'07)*, pages 237–246, September 2007.

Massimo Cossentino, Nicolas Gaud, Vincent Hilaire, Stephane Galland, and Abderrafaa Koukam. Aspecs: an agent-oriented software process for engineering complex systems. *Autonomous Agents and Multi-Agent Systems*, 20:260–304, 2010a. ISSN 1387-2532. URL http://dx.doi.org/10.1007/s10458-009-9099-4. 10.1007/s10458-009-9099-4.

Massimo Cossentino, Nicolas Gaud, Vincent Hilaire, Stéphane Galland, and Abderrafiaa Koukam. ASPECS: an agent-oriented software process for engineering complex systems. *Autonomous Agents and Multi-Agent Systems*, 20(2):260–304, march 2010b.

Massimo Cossentino, Vincent Hilaire, Nicolas Gaud, Stephane Galland, and Abderrafiaa Koukam. *The ASPECS process*. Springer, 2013.

Pete Deemer, Gabrielle Benefield, Craig Larman, and Bas Vodde. The scrum primer. Technical report, www.goodagile.com, 2010.

Arnaud Dury, Florence Le Ber, and Vincent Chevrier. A reactive approach for solving constraint satisfaction problems. In Jörg Müller, Munindar P. Singh, and Anand S. Rao, editors, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555 of *LNAI*, pages 397–412. Springer-Verlag: Heidelberg, Germany, July 1999.

Jacques Ferber. *Multi-Agent Systems—An Introduction to Distributed Artifical Intelligence*. Addison-Wesley, 1999. ISBN 0-201-36048-9.

Jacques Ferber and Olivier Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In Y. Demazeau, E. Durfee, and N.R. Jennings, editors, *ICMAS'98*, july 1998.

Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizations: An organizational view of multi-agent systems. In *In LNCS n. 2935, Procs. of AOSE'03*, pages 214–230. Springer Verlag, 2003.

Mark S. Fox. An organizational view of distributed systems. *IEEE Trans. on System, Man, and Cybernetics*, SMC-11(1):70–80, January 1981.

Mark S. Fox, Mihai Barbuceanu, and Michael Gruninger. An organisation ontology for enterprise modeling: Preliminary concepts for linking structure and behaviour. *Computers in Industry*, 29(1-2):123 – 134, 1996. ISSN 0166-3615. WET ICE '95.

Mark S. Fox, Mihai Barbuceanu, Michael Gruninger, and Jinxin Lin. An organization ontology for enterprise modelling. In *Modeling, International Conference on Enterprise Integration Modelling Technology 97*. Springer, 1997.

M.S. Fox and M. Grüninger. Ontologies for enterprise modelling. In Kurt Kosanke and JamesG. Nell, editors, *Enterprise Engineering and Integration*, Research Reports Esprit, pages 190–200. Springer Berlin Heidelberg, 1997. ISBN 978-3-540-63402-7. doi: 10.1007/978-3-642-60889-6_22.

Lars Marius Garshol. Metadata ? thesauri ? taxonomies ? topic maps ! making sense of it all. *Information Science*, pages 378–391, february 2004.

N. Gaud, S. Galland, F. Gechter, V. Hilaire, and A. Koukam. Holonic multilevel simulation of complex systems. application to real-time pedestrians simulation in virtual urban environment. *Simulation Modelling Practice and Theory*, 16(10):1659–1676, 2008.

Nicolas Gaud, Stéphane Galland, Vincent Hilaire, and Abderrafiaa Koukam. An organizational platform for holonic and multiagent systems. In *6th International Workshop ProMAS 2008, Lecture Notes in Computer Science 5442*, pages 104–119, Estoril, Portugal, May 2009. Springer.

Franck Gechter, Vincent Chevrier, and François Charpillet. A reactive agent-based problem-solving model : Application to localization and tracking. *ACM Transactions on Autonomous and Adaptive Systems (ACM TAAS)*, 1(2):189–222, November 2006.

M. R. Genesereth. Knowledge interchange format. principles of knowledge representation and reasoning. In *Proceeding of the Second Interational Conference, Cambridge*, pages 599–600, 1991.

J. H. Gennari, D. E. Oliver, W. Pratt, J. Rice, and M. A. Musen. A web-based architecture for a medical vocabulary server. *Proceedings / the ... Annual Symposium on Computer Application [sic] in Medical Care. Symposium on Computer Applications in Medical Care*, pages 275–279, 1995. ISSN 0195-4210.

Godfray. Challenges for taxonomy. *Nature*, 417(6884):17–19, 2002. ISSN 0028-0836. doi: 10.1038/417017a.

S. Gomes, D. Monticolo, V. Hilaire, and B. Eynard. Content management based on multi agent systems for collaborative design. *International Journal of Product Development*, 8(2):178–192, 2009.

Thomas Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, 1993. ISSN 1042-8143. doi: 10.1006/knac.1993.1008.

Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.*, 43:907–928, December 1995. ISSN 1071-5819.

Nicola Guarino and Nicola" Guarino. Semantic matching: Formal ontological distinctions for information organization, extraction, and integration. *INFORMATION TECHNOLOGY, INTERNATIONAL SUMMER SCHOOL, SCIE-97*, pages 139–170, 1997. doi: 10.1.1.20.7515.

Alexander Helleboogh, Giuseppe Vizzari, Adelinde Uhrmacher, and Fabien Michel. Modeling dynamic environments in multi-agent simulation. *Autonomous Agents and Multi-Agent Systems*, 14(1):87–116, 2007. URL http://dx.doi.org/10.1007/s10458-006-0014-y.

Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. A model for the structural, functional, and deontic specification of organizations in multiagent systems. In *Proceedings of the 16th Brazilian Symposium on Artificial Intelligence: Advances in Artificial Intelligence*, SBIA '02, pages 118–128, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-00124-7.

B. L. Humphreys, D. A. Lindberg, H. M. Schoolman, and G. O. Barnett. The unified medical language system: an informatics research collaboration. *Journal of the American Medical Informatics Association : JAMIA*, 5(1):1–11, 1998. ISSN 1067-5027.

David Isern, David Sánchez, and Antonio Moreno. Organizational structures supported by agent-oriented methodologies. *Journal of Systems and Software*, 84(2):169–184, 2011. URL http://dx.doi.org/10.1016/j.jss.2010.09.005.

Igor Jurisica, John Mylopoulos, and Eric Yu. Ontologies for knowledge management: An information systems perspective. *Knowledge and Information Systems*, 6(4):380–401, 2004. ISSN 0219-1377. doi: 10.1007/s10115-003-0135-4.

B. Kiepuszewski, A. H. M. Ter Hofstede, and W.M.P. van der Aalst. Fundamentals of control flow in workflows. *Acta Informatica*, 39:143–209, 2002.

Malte Kiesel, Sven Schwarz, Ludger van Elst, and Georg Buscher. Mymory: Enhancing a semantic wiki with context annotations. In Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis, editors, *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1-5, 2008, Proceedings*, volume 5021 of *Lecture Notes in*

Computer Science, pages 817–821. Springer, 2008. ISBN 978-3-540-68233-2. URL http://dx.doi.org/10.1007/978-3-540-68234-9_65.

D.B. Lenat and R.V. Guha. *Building Large Knowledge Bases*. Addison Wesley, 1990.

Freddy Limpens, Fabien L. Gandon, and Michel Buffa. Bridging ontologies and folksonomies to leverage knowledge sharing on the social web: A brief survey. In *ASE Workshops*, pages 13–18. IEEE, 2008. ISBN 978-1-4244-2776-5. URL http://dx.doi.org/10.1109/ASEW.2008.4686305.

YISHUAI LIN, Vincent HILAIRE, Nicolas GAUD, and Abderrafiaa KOUKAM. Towards an ontological approach for the description of design processes: the scrum example. In *First International Symposium on Data-Driven Process Discovery and Analysis (SIMPDA)*, jun 2011a.

YISHUAI LIN, Vincent HILAIRE, Nicolas GAUD, and Abderrafiaa KOUKAM. A conceptualization of organizations involved in product design: a first step towards reasoning and knowledge management. *International Journal of Digital Information and Wireless Communications*, 1(1):141–153, nov 2011b. ISSN 2225-658X. URL http://www.sdiwc.net/noahjohn/web-admin/upload-pdf/00000103.pdf.

Yishuai Lin, Vincent Hilaire, Nicolas Gaud, and Abderrafiaa Koukam. K-crio: an ontology for organizations involved in product design. In *Proceedings of the DICTAP'11 conference*, number 167 in Communications in Computer and Information Science series. Springer, 2011.

YISHUAI LIN, Vincent HILAIRE, Nicolas GAUD, and Abderrafiaa KOUKAM. *Scrum conceptualization using K-CRIO ontology*, volume 116 of *Lecture Notes in Business Information Processing*, pages 1–19. Springer, jul 2012.

Pattie Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):31–40, July 1994.

David Martin, Mark Burstein, Drew McDermott, Sheila McIlraith, Massimo Paolucci, Katia Sycara, Deborah L. McGuinness, Evren Sirin, and Naveen Srinivasan. Bringing semantics to web services with OWL-S. In *First International Workshop on Semantic Web Services and Web Process Composition*, pages 243–277, 2004.

Nada Matta, Hassan Atifi, Mohammed Sediri, and Mohammed Sagdal. Analysis of interactions on coordination for design projects. In Kokou Yétongnon, Albert Dipanda, and Richard Chbeir, editors, *Sixth International Conference on Signal-Image Technology and Internet-Based Systems, SITIS 2010, Kuala Lumpur, Malaysia, December 15-18, 2010*, pages 344–347. IEEE Computer Society, 2010. ISBN 978-0-7695-4319-2. URL http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5714190.

Deborah L. McGuinness and Frank Van Harmelen. http://www.w3.org/tr/owl-features/.

Raul Medina-mora, Raul Medina-mora, Terry Winograd, Rodrigo Flores, and O" Flores. The action workflow approach to workflow management technology. *IN PROCEEDINGS OF ACM CSCW'92*, pages 281–288, 1992. doi: 10.1.1.151.3701.

Melonfire. Understanding the pros and cons of the waterfall model of software development, september 2006. URL http://www.techrepublic.com/article/understanding-the-pros-and-cons-of-the-waterfall-model-of-software-development/6118423.

Peter Mika. Ontologies are us: A unified model of social networks and semantics. *J. Web Sem*, 5(1):5–15, 2007. URL http://dx.doi.org/10.1016/j.websem.2006.11.002.

Achraf Ben Miled, Davy Monticolo, Vincent Hilaire, and Abderrafiaa Koukam. An approach for building holonic organizational models of design processes for knowledge management. In *International Workshop on Organizational Modeling*, 2009.

Davy Monticolo, Vincent Hilaire, Abder Koukam, and Sébastian Meunier. An approach for building project memories to facilitate design process in a concurrent engineering context. In *ISPE CE*, pages 279–287, 2006.

Davy Monticolo, Vincent Hilaire, Abder Koukam, and Samuel Gomes. A multi agent model to support the knowledge management process inside professional activities. In *ICDIM*, pages 799–804, 2007.

John Mylopoulos. Information modeling in the time of the revolution. *Inf. Syst.*, 23(3-4):127–155, May 1998. ISSN 0306-4379. doi: 10.1016/S0306-4379(98)00005-2. URL http://dx.doi.org/10.1016/S0306-4379(98)00005-2.

Robert Neches, Fikes Richard, and Finin Tim. Enabling technology for knowledge sharing. *AI Magazine*, 12(3), 1991.

Antonio De Nicola, Mario Lezoche, and Michele Missikoff. An ontological approach to business process modeling. In *IICAI*, pages 1794–1813, 2007.

OMG. Unified Modeling Language (UML), Infrastructure, V2.1.2. Technical report, Object Management Group, 2007.

Steve R. Palmer and Mac Felsing. *A Practical Guide to Feature-Driven Development*. Pearson Education, 1st edition, 2001. ISBN 0130676152.

Daniel Rodríguez, Elena García, Salvador Sánchez, and Carlos Rodríguez-Solano Nuzzi. Defining software process model constraints with rules using owl and swrl. *International Journal of Software Engineering and Knowledge Engineering*, 20(04):533–548, 2010. doi: 10.1142/S0218194010004876. URL http://www.worldscientific.com/doi/abs/10.1142/S0218194010004876.

S. Rodriguez, V. Hilaire, and A. Koukam. Towards a holonic multiple aspect analysis and modeling approach for complex systems: Application to the simulation of industrial plants. *Simulation Modelling Practice and Theory*, 15(5):521–543, May 2007.

Craig Schlenoff, Craig Schlenoff, Michael Gruninger, Florence Tissot, John Valois, Taddle Creek Road, Steptools Inc, Josh Lubell, and Jintae" Lee. The process specification language (psl) overview and version 1.0 specification. 1999a. doi: 10.1.1.34.1404.

Craig Schlenoff, Craig Schlenoff, Don Libes, Mihai Ciocoiu, and Michael" Gruninger. Process specification language (psl): Results of the first pilot implementation. *In proceedings of IMECE*, pages 14–19, 1999b. doi: 10.1.1.37.3741.

Richard Scott and Gerald Davis. *Organizations and Organizing: Rational, Natural and Open Systems Perspectives*. Prentice Hall, 1 edition, 2006. ISBN 0131958933.

Carles Sierra and Liz Sonenberg. A real-time negotiation model and A multi-agent sensor network implementation. *Autonomous Agents and Multi-Agent Systems*, 11(1):5–6, 2005. URL http://dx.doi.org/10.1007/s10458-005-1281-8.

J. Stapleton. DSDM: Dynamic systems development method. In *Technology of Object-Oriented Languages and Systems, 1999. Proceedings of*, pages 406–406, 1999. doi: 10.1109/TOOLS.1999.779095.

Rudi Studer, Rudi Studer, V. Richard Benjamins, and Dieter" Fensel. Knowledge engineering: Principles and methods. 1998. doi: 10.1.1.41.1007.

Bill Swartout, Ramesh Patil, Kevin Knight, and Tom Russ. Towards distributed use of large-scale ontologies. In *Proceedings of the 10th. Knowledge Acquisition for Knowledge-Based Systems Workshop*, 1996.

Katia P. Sycara. Multiagent systems. *AI Magazine*, 19(2):79–92, 1998.

W. M. P. Van der Aalst. Three good reasons for using a petri-net-based workflow management system. doi: 10.1.1.147.3781.

F. Vernadat. *Enterprise Modeling and Integration: Principles and Applications*. Springer, 1st edition, 1996. ISBN 0412605503.

Y. Wand and R. Weber. An ontological model of an information system. *IEEE Transactions on Software Engineering*, 16(11):1282–1292, 1990. ISSN 0098-5589. doi: 10.1109/32.60316.

Dirk Wodtke and Gerhard Weikum. A formal foundation for distributed workflow execution based on state charts. In *Proceedings of the 6th International Conference on Database Theory*, ICDT '97, pages 230–246, London, UK, UK, 1997. Springer-Verlag. ISBN 3-540-62222-5. URL http://dl.acm.org/citation.cfm?id=645502.757730.

Ye Yao, Cai Wandong, Abderrafiaa Koukam, and Vincent Hilaire. A multi-hierarchical group mobility model for tactical mobile wireless networks. *International Journal of computational information system*, 5(1), 2009.

F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: the gaia methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3), July 2003.

Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Organizational abstractions for the analysis and design of multi-agent systems. In Paolo Ciancarini and Michael Wooldridge, editors, *AOSE*, volume 1957 of *Lecture Notes in Computer Science*, pages 235–251. Springer, 2000. ISBN 3-540-41594-7.

Abstract:

The work presented in this PhD thesis defines a new approach for the modeling and the conceptualization of enterprise business processes in the perspective of building intelligent assistance software tools to support these processes. The proposed approach defines an organizational ontology, named K-CRIO. Its description is based on the Ontology Web Language. To illustrate our work, an intelligent assistance system has been designed and implemented according to the result from the modeling and conceptualization of a specific business process with the K-CRIO Ontology. It is a web-based application that integrates and takes full advantage of multi-agent systems. The K-CRIO Ontology is an Ontology dedicated to the study of organizations and the analysis of business processes adopting an organizational point of view. Specifically, it is used to understand, analyze and reason about organizations and the processes they implement. The targeted organizations are those composed of entities involved throughout products' design and, to do so, following a defined business process. The range of this type of organizations is quite wide. We have thus limited our study to organizations that produce software as the final process goal, specifically IT enterprises delivering software products or services. In this context, the K-CRIO ontology could be used to model structure of the considered organizations (entity relationships) and model human activities appearing in their business processes. This ontology could be used to support process assistance within the described organizations. More specifically, the ontology could provide means for reasoning, annotating resources, monitoring design processes, enabling searches and pro-actively proposing tips and proper content. In order to illustrate the usage of K-CRIO, we apply K-CRIO on two different processes: the Waterfall Model and the Scrum methodology. These examples are both classical software-development processes. Moreover, for Scrum, the famous agile software-development process widely used in software enterprises, we have designed and developed an intelligent assistance tool. This tool mainly helps Scrum Masters to make decision by monitoring Scrum project teams' activities within their various projects and collecting knowledge about these activities.

Résumé :

Le travail présenté dans cette thèse définit une nouvelle approche pour la modélisation et la conceptualisation des processus métiers dans les entreprises afin de construire des outils logiciels d'assistance intelligents qui prennent en charge ces processus. L'approche proposée définit une ontologie dédiée à l'étude des organisations, nommée K-CRIO. Elle est décrite à l'aide du langage de représentation des connaissances OWL. Afin d'illustrer nos travaux, un système d'assistance a été implanté sur la base des résultats issus de la modélisation et de la conceptualisation d'un processus métier spécifique avec l'ontologie K-CRIO. Ce système prend la forme d'une application Web qui intègre et exploite pleinement les avantages des systèmes multiagents. L'ontologie K-CRIO est une ontologie dédiée à l'étude des organisations et à l'analyse organisationnelle des processus métiers qu'elles mettent en œuvre. Plus précisément, elle est utilisée pour comprendre, analyser et raisonner sur ces organisations. Les organisations visées sont celles composées d'acteurs humains impliqués tout au long de la conception de produits et, pour ce faire, organisés selon un processus métier. L'éventail de ce type d'organisations est assez large. Nous avons donc limité notre étude aux organisations qui produisent des logiciels comme objectif final du processus. Dans ce contexte, l'ontologie K-CRIO peut être utilisée pour modéliser la structure organisationnelle du processus (acteurs et leurs relations) et les activités qui en résultent. Cette ontologie peut ensuite être exploitée afin de concevoir des outils d'assistance à la mise en œuvre des processus ciblés au sein des organisations décrites. Plus précisément, l'ontologie fournit des moyens de raisonnement, d'annotation des ressources, et de suivi des processus de conception, permettant des recherches et de proposer pro-activement des conseils et des contenus appropriés. Afin d'illustrer l'utilisation de K-CRIO, nous appliquons K-CRIO sur deux processus différents: le modèle en cascade et la méthodologie Scrum. Ces exemples sont des processus de développement de logiciels classiques. En outre, pour le processus Scrum, qui est un processus agile de développement de logiciel, largement utilisé dans les entreprises de logiciels, nous avons conçu et développé un outil d'assistance intelligent. Cet outil contribue principalement à aider les *Scrum Masters* en leur fournissant des indicateurs pour les assister dans leurs prises de décisions ainsi que par la constitution d'une base de connaissances sur les activités des membres de leur équipe projet.

■ École doctorale SPIM - Université de Technologie Belfort-Montbéliard
F - 90010 Belfort Cedex ■ tél. +33 (0)3 84 58 31 39
■ ed-spim@univ-fcomte.fr ■ www.ed-spim.univ-fcomte.fr