LOUVAIN
School of Management

A Business-Driven Process for COTS-Based System Development Centered on Agents

by Sodany Kiv

A thesis submitted in fulfillment of the requirements for the degree of

Doctor in Economics and Management Sciences

of the Université catholique de Louvain

Examination Committee:

Prof. Manuel Kolp (UCL), Advisor

Prof. Yves Wautelet (HUB/KUL), Co-Advisor

Prof. Jean Vanderdonckt (UCL), Examiner

Prof. Stéphane Faulkner (FUNDP), Examiner

Prof. Christophe Schinckus (University of Leicester, UK), Reader

Prof. Ngoc Thanh Nguyen (Wroclaw University of Technology, Poland), Reader

Prof. Philippe Chevalier (UCL), President of the jury

To my grandpa NGO Kiv, you live forever in our hearts.

Acknowledgements

First of all, I would like to express my sincere thanks to my advisor Prof. Manuel Kolp for having accepted me to work as a teaching and research assistant within the ISYS research unit. I am very thankful to him for his support throughout my PhD.

Next, I would like to express my deepest gratitude to my co-advisor Prof. Yves Wautelet for always helping me and giving good advice. I would have been lost without him.

I also thank the members of my committee, Prof. Jean Vanderdonckt, Prof. Stéphane Faulkner, Prof. Christophe Schinckus, Prof. Ngoc Thanh Nguyen, and Prof. Philippe Chevalier for accepting to participate in the jury of this thesis and contributing with their knowledge and attention.

My appreciation and thanks go as well to my colleagues at the ISYS research unit at the Louvain School of Management. I especially want to thank Thi Ai Vi Tran and Kênia Soares Sousa for their friendship.

Many thanks go to my Cambodian friends in Belgium and Netherlands, who have helped me and have made my life more fun during all these years.

Last but not least, I would like to give very special thanks to my beloved family, particularly to my parents and husband, for their support, encouragement and, above all, their love.

Abstract

Enterprise information systems have progressively increased to become large-scaled and very complex inducing a high development time and cost, hard to manage software quality, poor understanding of user requirements leading to high risk when integrating new technology. As a consequence, there is a growing demand for efficient, manageable and cost-effective software development techniques.

The *COTS-based system development (CBSD)* approach is one of the solutions brought by research and industry within this particular context. This approach is based on the idea of building new systems by selecting appropriate *commercial off-the-shelf (COTS)* software components and assembling them within a well-defined software architecture. The potential benefits of this approach are mainly its reduced costs and shorter development time, while maintaining quality. However, the use of COTS software components in software system development presents new challenges tending to limit their use.

This thesis aims to contribute to the improvement of CBSD by proposing a methodology that addresses diverse issues related to CBSD. More precisely:

- For the characterization of COTS components, it provides a conceptual model that can be used by vendors to describe components to be published, and by users to specify selection criteria for required components and to properly integrate the selected ones into the system. This conceptual model can also be used to build a structured repository serving as yellow pages for components to be found and reused effectively;

- For COTS integration, it provides an architectural pattern for dynamic integration through the use of a wrapper-based multi-agent system. The originality of the proposed architecture is issued of its configurability with respect to the project specific business logic and its flexibility to adapt to the changing requirements and environment. An implementation model based on the JADE framework has been realized for validation purpose. This implementation indeed demonstrates feasibility of the proposed system architecture and constitutes a guidance for developers;

- For entire life cycle coverage, it provides a business-driven process for CBSD dealing with *business analysis*, *requirements analysis*, *COTS evaluation and selection*, and *COTS mismatches handling*. It leads to build a COTS-based system that meets the business needs and strategies, and that can be easily adapted to the business changes. The methodology does not only provide a high level description of the process but also some practical models and guidelines for accomplishing the activities defined in the process.

The methodology described on the latest bullet point is called *Rationale Incremental and Iterative Process for COTS-Based System Development* (*RecIProC*) and has been validated on a case study in the field of outbound logistics. In order to support the methodology application, computer-aided software engineering tools, which provide the assistance to perform defined activities, have also been developed.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Introduction

This chapter introduces the whole thesis. We describe, in Section 1.1, the research context of this thesis. Then, we explain the motivations of our research in Section 1.2. Next, we present the scope of this thesis in Section 1.3; Section 1.4 presents our research method. Finally, we provide a reading map for the rest of the dissertation in Section 1.5.

1.1 Research Context

The use of *commercial off-the-shelf (COTS)* components to develop large-scale information systems has become increasingly prominent over the past decade. With such an approach, there is no need to develop the system from scratch but rather to customize COTS components and integrate them into the system. A COTS component is a commercially available piece of software that other software projects can reuse and integrate into the system to-be. Due to the commercial potential of software reuse, the known similarities between the businesses; software packages at disposal tend to be more generic so that consumers can seek products meeting their requirements in a broader area. This gives COTS-based system development a tremendous amount of interests both in the research community and software industry.

COTS-based system development (CBSD) is based on the idea of building new systems by selecting appropriate COTS components and assembling them with well-defined software architecture. It has become a strategic field for building large-scale and complex systems due to potential benefits that are mainly its reduced costs and shorter development time, while maintaining quality [36].

Although CBSD approaches promise significant benefits, there are some technical problems that limit their use. Among those problems, we point out:

- how the business requirements must be captured and refined, based on a process that leads to the development of a COTS-based system;

- how to select COTS products that suit best organizational needs;
- how to handle the mismatches between the COTS capabilities and the system requirements;
- how the different COTS products must be put together and deployed using the latest technologies;
- how to manage the system upgrading.

In order to deal with such issues, adequate methods and techniques should be used for conducting the full life cycle of CBSD. Core steps that need to be included are:

- **Requirements engineering** which is involved with defining the desired capabilities and constraints, and helps establishing the COTS evaluation criteria;
- **COTS evaluation and selection** which is involved with evaluating existing products and selecting the one that best fits requirements;
- **COTS customization** which is involved with tailoring the selected COTS in order to cover unsatisfied or partially satisfied requirements;
- **COTS integration** which is involved with assembling a set of selected COTS components together to produce a desired system;
- **System evolution** which is concerned with maintenance issues such as updating the system with new COTS releases, adding new functionality to the system, and fixing errors.

This thesis aims to contribute to the improvement of CBSD by proposing a business-driven methodology covering the first four core steps aforementioned; i.e. *requirements engineering*, *COTS evaluation and selection*, *COTS customization*, and *COTS integration*. *System evolution* is out of the scope of this thesis. The methodology is indeed intended for the development of IT solutions with the use of COTS components that satisfy business requirements. This is achieved by establishing a development process that starts with business analysis phase in order to identify business needs, to define IT solutions to business problems, and to study the long-term strategic impacts of adopting the IT solutions. The result of the business analysis serves as the basis for the following phases of the development process. In addition, our methodology proposes a flexible system architecture allowing IT systems to be easily adapted to the business changes. The motivations of our work are described into the next section. It notably argues that software development methodologies for the custom system development are not entirely suitable for CBSD, and justifies the benefits of using the methodologies proposed in the agent-oriented software engineering for developing COTS-based systems.

1.2 Research Motivation

As illustrated in Figure 1.1, [120] identifies the specific activities of CBSD when compared to classical "from scratch" development. As evoked in [120], some of these activities are new, having no counterpart in traditional custom system development processes. Others are similar, yet the use of COTS components in the system development brings some pervasive changes to them. For instance, the requirements engineering of custom software development is essentially straightforward by describing a desired system through a set of specified functionalities and qualities that the system must meet. However, requirements engineering is very different when acquiring COTS-based systems since at least some software requirements must be flexible enough to accommodate the fluctuations of the commercial marketplace. In such cases, either the requirements will be adapted to the existing components or the developers will develop the software components in-house.

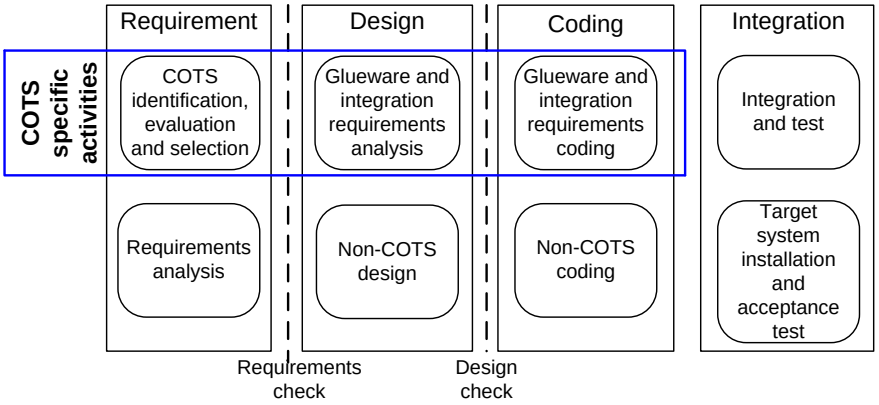


Figure 1.1: COTS-based system development process (from [120]).

Following [120], the activities that are specific to CBSD include *COTS components identification, evaluation and selection* at the requirement stage, *glueware design and integration requirements analysis* at the design stage, and *glueware and integration requirements coding* at the coding stage (see Figure 1.1). Indeed, after defining the system requirements, the system builders identify the products available in the marketplace that can be used to build the system. A set of components are preselected to be examined more closely in order to determine a few viable candidate products. The trade-off analysis between competing candidates will be done in term of technical and economical factors before making the selection decision. However, COTS components even the best-fitting ones will rarely perfectly meet system requirements. The project stakeholders will need to tailor the selected components in order to cover unsatisfied or partially satisfied requirements. These components will be then assembled together to build the desired system. Assembling COTS

components also presents unique difficulties because these COTS components are seldom built to plug into each other easily. The common technique to overcome this deficiency and to build an integrated system out of incompatible COTS components involves the development of *glueware*. Glueware is a type of software that can be used to "glue" or integrate software components to form a seamless integrated system.

Moreover, system maintenance and evolution are also very different when COTS components are used to build the system. Upgrading a COTS-based system means that there are new releases of COTS components used in the system. A system with several COTS components thus has a very heavy dependence on various release cycles of the COTS vendors. Upgrading a particular COTS component can result in several inconsistencies and expensive redesign of the system. Hence, the system architecture must be flexible and facilitate the components' substitution.

From an engineering point of view, custom system development is an act of creation. It starts by defining system requirements and creates a system that meets them. On the other hand, CBSD starts by defining a general set of requirements and then explores existing COTS components to see how they match the requirements. The best-matching components will be selected and customized if necessary before integrating them into the system under development. As depicted in the left of Figure 1.2, a custom system development approach involves specifying system requirements, defining architectural constraints, and undertaking system implementation. This approach is not applicable to CBSD because **the marketplace would not likely yield any products that perfectly meet the requirements and architectural constraints of the system under development**. Instead, with COTS-based systems, project teams must consider requirements, architecture and marketplace simultaneously, as depicted in the right of Figure 1.2. Any of the three circles might affect the other two, so none can proceed without knowledge and accommodation of the others. These arguments highlight that the process and activities of CBSD are different from those of custom development.

Based on these arguments, existing *software development methodologies (SDMs)* for custom system development must be adapted for CBSD [3]. As stated in [142], among the SDMs proposed in the literature, the SDMs inspired by system architectures and implementation rather than by organizational and enterprise ones lead to misunderstanding between business stakeholders and software developers. These misunderstandings result in failures to correctly implement requirements and mismatches between organizational needs and system modules that should automate them. These failures and mismatches are particularly sensitive when concerning large-scale information system development based on the use of COTS components. It is due to intrinsic organizational nature of large-scale information systems and the COTS component source codes are most often not available to the system developers. **Therefore, SDMs that do not provide organizational and business**

modeling phases and social-based design patterns fail to provide sufficient support to development of COTS-based large-scale information systems.

The *agent-oriented software engineering (AOSE)* that has emerged in recent years has contaminated positively information systems analysis and design processes [87]. The SDMs proposed in the AOSE research fields have contributed to reduce the mismatch between organizational requirements and systems design [97] that is strategic in COTS-based large-scale information system development projects. These SDMs offer modeling tools based on organizational and social concepts (*actors*, *agents*, *goals*, *objectives*, *responsibilities*, *social dependencies*, *process*, *qualities*, etc.) as fundamentals to conceive information systems [77]. This makes AOSE distinct from any other software engineering paradigm. The idea of modeling an information system in terms of autonomous entities with characteristics similar to human organizations introduces a close-to-real life system modeling, and therefore makes the system developments natural.

Based on these observations, the aim of our research is to build, apply and validate an original methodology, which is based on the SDMs proposed in the AOSE, for developing COTS-based system in the most efficient manner possible. An epistemological foundation of our work is presented in the appendix A.

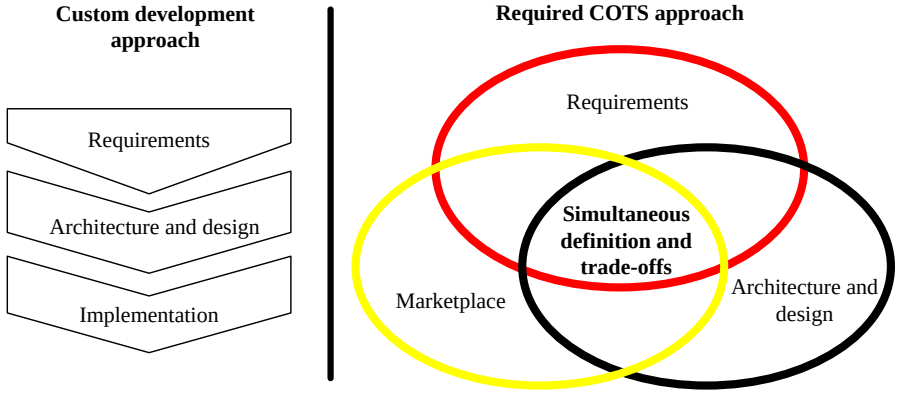


Figure 1.2: Custom vs. COTS-based approach (from [37]).

1.3 Scope of the Thesis

Before defining and validating our methodology, it is important to define the scope of our PhD research. Literature review reveals that numbers of researchers have identified the existence of different types of COTS-based systems, and have pointed out that the processes and activities involved to be followed in developing each type of system are largely different [29]. In order to clarify the scope of this thesis, it is essential to specify the category of COTS-based systems that we are focusing on.

The authors of [46] and [29] have classified COTS-based systems into three major categories, depending on the number of COTS products used to build the system:

- **Turnkey systems** are built around one or suite of COTS products that meet most of the desired functionalities. A turnkey system may be customized (e.g. through parameter adjustments) to better meet user's needs. For instance, Microsoft Office [1] is a turnkey system;
- **Intermediate systems** are built around one COTS product, and may further integrate customized elements, either developed in-house or acquired from a third party. The central COTS product dominates the system, and the amount of customization varies as needed. For instance, mySAP ERP [2] is an intermediate system;
- **COTS-intensive systems (the focus of this thesis)** are built by integrating several products or components, either from a third party or developed in-house. None of them dominates the system, but all are equally important. Such systems usually include glue-coding, i.e., writing custom code to facilitate communication between different system pieces. For instance, an e-collaboration system that is composed of an *Enterprise Resource Planning (ERP)*, a *Customer Relationship Management (CRM)* and a *Supply Chain Management (SCM).*

The existence of these defined types of COTS-based systems induces that there are three main alternatives for building up a large-scale distributed business system [78]:

1. Buy the whole system from a software vendor that provides all the functionality required;
2. Custom-build the system;
3. Buy a set of systems from multiple vendors, often with the objective of choosing the best-of-breed and integrate them together.

Given the unlikelihood of any single software vendor being able to meet all requirements, and given the cost of building a large-scale system from scratch, the third alternative is currently receiving a tremendous push toward becoming the only effective reality. This motivates us to focus our work on this approach of system development (i.e. COTS-intensive system development). We aim to addresses the issues associated with the development of such a system. More precisely, we propose a methodology for developing large-scale distributed business systems with the use of COTS components. This methodology covers the four first steps of the CBSD life cycle including Requirements engineering, COTS selection, COTS customization, and COTS integration.

This methodology provides *a business-driven process for CBSD*, *an integration architectural model*, and *a CASE-Tool for supporting the methodology*. As evoked earlier, this methodology is based on the SDMs proposed in the AOSE in order to reduce the mismatches between organizational requirements and systems design. More specifically:

- we propose using the requirements engineering frameworks proposed in the AOSE (i.e. i* [170] and NFR [53] frameworks) for analyzing the business needs and system requirements;
- we propose an agent-oriented approach for further analyzing the mismatches between the COTS features and the system requirements;
- we provide a generic multi-agent system (MAS) architecture modeling different software agents used for coordinating different COTS components to build the large-scale system under development;
- we have developed the proposed MAS using the JADE framework [20]. This constitutes an implementation model for developers.

1.4 Research Method

There are four main parts in the construction of this thesis, each of them was based on a different research method and approach. This sections summarizes these contributions and the approaches followed to build-up each of them, more precisely:

- The first part involved a literature study of the relevant approaches and techniques;
- The second part focused on defining key elements necessary and software architecture required to build up a methodology;
- The third part focused on building-up the methodology upon the defined concepts and architecture for full life cycle coverage;
- Finally, the fourth part consisted in the development of CASE tools for methodology support.

1.4.1 State of the Art

To successfully achieve this part, we proceeded to a review of scientific sources related to our main topics of interest. We indeed focused on research works related to CBSD, SDMs for custom system development, and requirements engineering frameworks.

We have found that there are many research works related to CBSD addressing diverse aspects of it (e.g. COTS selection, COTS integration, risk management, quality management, etc.). Nevertheless, there is a lack of established methodologies that cover the full life cycle of CBSD including *Requirements engineering*, *COTS evaluation and selection*, *COTS customization*, *COTS integration* and *System evolution*. In order to contribute to the improvement of CBSD, we aim to define an original methodology that covers the first four phases of the CBSD life cycle.

1.4.2 Architectural Design for CBSD

Based on the key elements composing COTS-components found in literature during the first part, we have build-up a custom ontology where syntax, semantic and linking of each relevant concept are depicted. We made use of a UML class diagram to accurately represent the concepts and their relationships.

Besides building up the ontology, we have defined a system architecture centered on agents for COTS integration. Indeed, through the literature review on COTS integration, we found that there is a lack of frameworks for integrating different components that can support frequent upgrades of COTS components. The architecture has been built by combining abilities of agent-technology with requirements for dynamic COTS allocation at runtime.

1.4.3 Methodology Definition

Rationale Incremental and Iterative Process for COTS-Based System Development (RecIProC) has been defined by analyzing existing research works related to the main objective of this thesis. We have firstly defined the conceptual foundations of our methodology by:

- studying the different types of COTS-based system development (i.e. *Turnkey systems*, *Intermediate systems*, and *COTS-intensive systems*) and specifying the focus of the methodology (i.e. *COTS-intensive systems*);
- aligning the methodology with organizational models for business-driven development;

Then, we have reviewed existing SDMs for custom system development and research works related to CBSD in order to define an adequate development process for CBSD. The approaches, methods, and techniques found in the literature have been analyzed and the most solid and widespread ones have been incorporated in order to appropriately construct our process.

RecIProC has been validated on a case study issued of the TransLogisTIC project (see [160] for an exhaustive description). It is an ambitious research project financed by the Walloon Region which is built around a long-term strategy aimed at developing a complete and efficient multi-modal transport system in Wallonia.

1.4.4 Tool Development

This part consisted in the development of *Computer-Aided Software Engineering (CASE)* tools providing the assistance to perform activities that we have defined in RecIProC. This includes:

- developing a component management system for managing components' information;
- defining a *request for proposal (RFP)* template;
- developing a VBA program in Ms Excel for analyzing the RFP answers from vendors;
- developing a business process modeling notation (BPMN) editor and adding it into DesCARTES, a previously developed CASE tool by our research unit (see [147, 171]). DesCARTES offers a set of functionalities that can be used for performing some activities of our methodology (e.g. the i* editor and the NFR editor);
- developing a hierarchy graph editor and a RFP generator from a hierarchy graph;
- developing the proposed MAS architecture for COTS integration using the JADE plateform.

1.5 Reading Map

This thesis is organized in the following chapters:

- **Part I: Introduction** has presented the research context, main motivations, scope of this thesis, and research method;
- **Part II: State of the Art** presents the state of the art related to our research context. *Chapter 2* provides a survey of relevant approaches in software engineering field. *Chapter 3* reviews the relevant research works on CBSD found in the literature;
- **Part III: Architectural Design for COTS-Based System Development** presents the proposed architectural design for dynamic COTS integration through the use of a wrapper-based multi-agent system. *Chapter 4* presents the conceptual foundations of our architectural design. *Chapter 5* presents the MAS architecture for COTS integration as well as an implementation model of the proposed MAS using JADE;

- **Part IV: RecIProC: Rationale Incremental and Iterative Process for COTS-Based System Development** presents the proposed methodology for CBSD. *Chapter 6* provides a description of RecIProC according to the specifications for RecIProC that we have defined with respects to the weaknesses of existing methods. *Chapter 7* introduces an application of our methodology on a case study and CASE tools that we have developed to support our methodology;
- **Part V: Conclusion** concludes this research work by presenting the major contributions and proposing future works.

Part II

State of the Art

Chapter 2

Software Engineering: A Survey of Relevant Approaches

This chapter presents the state of the art in the *Software Engineering (SE)*. It focuses on the relevant approaches to our research context. Section 2.1 presents different definition of SE and specifies the one that best matches our work. Section 2.2 overviews relevant requirements engineering frameworks to be used within our methodology. Section 2.3 introduces some representative software development life cycle models. Section 2.4 exposes the *Object-Oriented Software Engineering (OOSE)* which has been the most widely-used in the development of information systems. Section 2.5 describes the *Agent-Oriented Software Engineering (AOSE)* which is one of the most recent contributions to the field of software engineering.

2.1 Software Engineering: a Definition

The term *Software Engineering (SE)* was first introduced in 1968 during a NATO conference in Germany. It was defined as "*the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines*" [125]. Since, SE appeared under many definitions in literature. [111] defined it as "*the establishment and use of sound engineering principles and good management practice, and the evolution of applicable tools and methods and their use as appropriate, in order to obtain – within known and adequate resource provisions – software that is of high quality in an explicitly defined sense*". Later, it was defined in [134] as "*the use of our knowledge of computers and computing to help solve problems*".

The generic definition that best matches the work of this thesis is "*SE is an engineering approach to the software systems development that provides methodologies, tools and techniques to help software system developers in the analysis, design,*

implementation and testing of software system.", as defined by [124].

2.2 Requirements Engineering

SE usually begins with the identification of relevant requirements, so that in custom system development, the *requirements engineering (RE)* mainly consists of eliciting stakeholders' needs, refining the acquired goals into non-conflicting requirements statements and finally validating these requirements with stakeholders. The specified requirements will be translated into a software architecture and eventually implemented. Broadly speaking, requirements play a controlling role in a custom system development [7].

On the other hand, in CBSD, the requirements cannot neglect the availability of COTS products on the market, in the sense that some requirements may not be provided by the available COTS products. This leads to requirements adaptation taking into account the knowledge of the existing market acquired little by little. Therefore, requirements statements need to be *much more flexible* and *less specific* in order to support these adaptations [9]. In other words, requirements should be specified as desirable needs rather than as strict constraints. For instance, suppose that performance is a critical requirement for a database system but none of the evaluated products satisfies the desired response time. This is a typical situation to deal with the *buy versus build* decision. If the final resolution is buying a product, customers must accept product limitations and requirements that cannot be met by any available COTS products.

Due to the crucial role that RE plays in the software development life cycle, we have reviewed existing RE approach, methods and techniques in order to select the most appropriate to support our methodology. We found that the notion of goal has been increasingly used in requirements engineering methods and techniques. [172] explains why goals are useful in RE by providing a comprehensive attempt at understanding and clarifying the roles of goals across the different areas of RE including *Requirement acquisition*, *Relating requirements to organizational and business context*, *Clarifying requirements*, *Dealing with conflicts*, *Driving design*, and other RE tasks.

By definition, goals are the objectives and targets of achievement for a system. *Goal-oriented requirements engineering (GORE)* takes the view that requirements should initially focus on the *why* and *how* questions rather than on the question of *what* needs to be implemented. Traditional analysis and design methods focused on the functionality of the system to be built and its interactions with users. Instead of asking what the system needs to do, GORE methods ask why a certain functionality is needed and how it can be implemented. Therefore, goal-oriented methods give a rationale for system functionality by answering why a certain functionality is needed while also tracking different implementation alternatives and the criteria for

the selection among these alternatives.

As stated in [12], by focusing on goals instead of specific requirements, analysts enable stakeholders to communicate using a language based on concepts (e.g. goals) with which they are both comfortable and familiar. Moreover, since enterprise goals and system goals are typically more stable than the requirements, they are a beneficial source for requirements derivation.

In addition, GORE allows requirements to be represented at different levels of abstraction [155]. This provides a systematic process for refining high-level requirements into objectively measurable sub-requirements that can be matched with the COTS component features.

Based on these reasons, we propose to use GORE frameworks in our methodology to adequately define the requirements for guiding the COTS selection. Various GORE methods have been proposed. [92] compares fifteen different goal-oriented approaches based on a common framework for understanding goal-oriented approaches. As depicted in Figure 2.1, this framework is composed of four views:

- The *usage* view concerns the objectives of using goal modeling in RE;
- The *subject* view looks at the notion of a goal and its nature;
- The *representation* view concerns the way goals are expressed. Goals can be expressed in a variety of formats, using more or less formally defined notations. The authors differentiate between informal, semi-formal and formal approaches. Informal approaches generally use natural language text to express goals; semi-formal use mostly box and arrow diagrams; finally, in formal approaches goals are expressed as logical assertions in some formal specification language;
- The *development* view concerns the way that goal models are generated, and evolve. This view considers the dynamic aspects of goal driven approaches, i.e. the proposed way-of-working and the tool support provided for enacting this way-of-working.

Based on this framework, we have defined the criteria to select goal-oriented approaches to support our methodology as follows:

- We need goal-oriented approaches that can be used to understand the existing organizational situation and to describe how business goals relate to functional and non-functional system components;
- On the representation perspective, we prefer goal-oriented approaches wherein goals are expressed in formal approach;

- On the development perspective, we prefer goal-oriented approaches that provide guideline and tool support for constructing goal models.

Overview of the different goal-oriented approaches illustrated in Figure 2.2 helps us to understand and accordingly select the goal-oriented approaches that meet the criteria mentioned above. Two goal-oriented approaches were chosen to support our methodology: the i* modeling framework and NFR framework. These frameworks are described in the following sections.

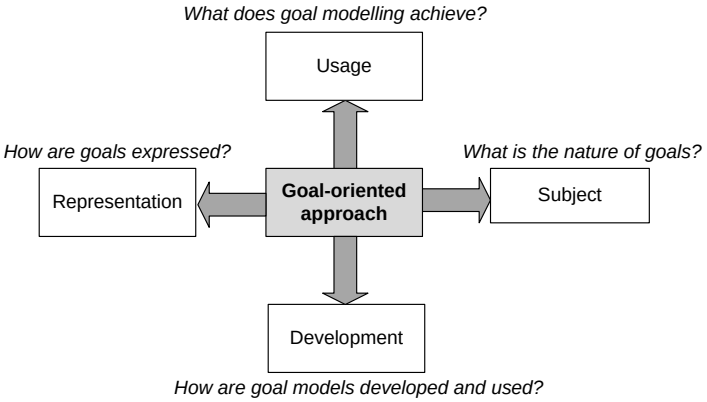


Figure 2.1: A framework for understanding goal-oriented approaches (from [92]).

2.2.1 The i* Framework

The i* framework is a goal-oriented and agent-oriented modelling framework proposed by Eric Yu [170]. It was originally developed to model information systems, and more particularly the dependencies between human and technological actors. It is composed of two types of models, each one corresponding to a different abstraction level: the *Strategic Dependency (SD)* and *Strategic Rationale (SR)* models.

A SD model represents the intentional level of a system. It consists of a set of actors (e.g. human or software system) and their dependencies including *Resource* dependency, *Task* dependency, *Goal* dependency, and *Softgoal* dependency.

- In a **Resource dependency**, one actor (*the depender*) depends on the other (*the dependee*) for the availability of an entity (physical or informational). By establishing this dependency, the *depender* gains the ability to use the entity as a resource. At the same time, the *depender* becomes vulnerable if the entity turns out to be unavailable;

- In a **Task dependency**, the *depender* depends on the *dependee* to carry out an activity. Tasks represent functional activities agents perform;

		Goal-Oriented Approaches														
Framework Components		**Cognitive Task Analysis**	**i* (Strategic Dependency Model)**	**Goal-Based Workflow**	**EKD**	**F^3 (OM)**	**ISAC**	**SIBYL**	**The reasoning loop model**	**REMAP**	**KAOS**	**GBRAM**	**Goal-Scenario Coupling**	**NFR Framework**	**GSN**	**GQM**
Usage	Understand current org. situation	✓	✓	✓	✓											
	Understand the need for change					✓	✓									
	Provide the deliberation context within which RE occurs							✓	✓	✓						
	Relate business goals to system components										✓	✓	✓	✓		
	Evaluate system specs. Against stakeholder goals														✓	✓
Subject	Enterprise goals	✓	✓	✓	✓						✓	✓	✓	✓		
	Process goals					✓	✓	✓	✓	✓						
	Evaluation goals														✓	✓
Representation	Formal		✓							✓	✓			✓		
	Semi-formal	✓		✓	✓	✓		✓	✓			✓	✓		✓	✓
	Informal						✓									
Development	Way-of-working		♦		♦		♦				♦	♦	♦	♦		
	Tool support	**M**	**MF**	**M**	**M**	**MG**	**MG**	**M**		**MF**	**MF**	**MG**	**MG**	**MF**	**M**	**MG**

✓= deal with the issue
♦ = suggest a number of steps and associated strategies
M = support for model construction, F = formal reasoning support, G = process guidance

Figure 2.2: Overview of some relevant goal-oriented approaches in RE (from [92]).

- In a **Goal dependency**, the *depender* depends on the *dependee* to bring about a certain state in the world. The *dependee* is free to do any tasks necessary to

achieve the goal;

- In a **Softgoal dependency**, the *depender* depends on the *dependee* to perform some task that meets a softgoal. Softgoals represent non-functional requirements; they are thus not directly implemented but do have an influence on the way functional requirements are responded to.

In Figure 2.3, we show an example of the SD model for *Meeting Scheduling*, without computer-based scheduler support, adapted from [170]. There are two human actors in the model: the *Meeting initiator* and the *Meeting participant*. The *Meeting initiator* depends on the *Meeting participant* for the goal *Attend meeting*. In order to schedule the meeting, the *Meeting initiator* depends on the *Meeting participant* for the tasks *Send exclusion dates* and *Send preference dates*. These two elements are modelled as tasks because the *Meeting initiator*, as a *depender*, has made some decision on how the dates have to be sent (i.e. the dates have to be sent in certain format and belong within a certain data range). Once the dates are analysed, the *Meeting participant* depends on the *Meeting initiator* for the resource *Proposed date*. This element is modelled as a resource, because it is an informational entity. Finally, the *Meeting initiator* depends on the *Meeting participant* for the softgoal *Agreement achieved promptly*. It is important to note that promptly is relative to the criteria of the *Meeting initiator* and cannot be sharply defined.

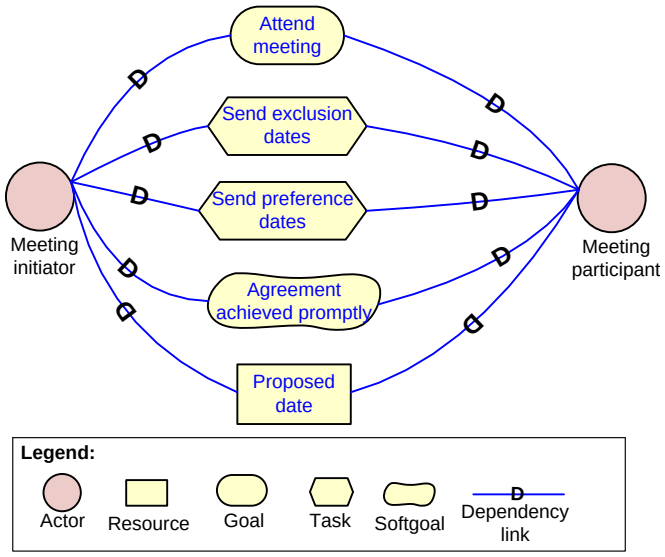


Figure 2.3: An example of Strategic Dependency model adapted (from [170]).

A SR model represents the rationale revel of a system. It allows visualizing the intentional elements into the boundary of an actor in order to refine the SD model to add reasoning ability. The dependencies of the SD model are linked to intentional

elements inside the actor boundary. The elements in the SR model are decomposed accordingly to the following links:

- **Means-end links** indicate a relationship between an end, and a means for attaining it. The "means" is expressed in the form of a task and with the "end" is expressed as a goal. In the graphical notation, the arrowhead points from the means to the end;

- **Contribution links** correspond to means-end links where the end is a softgoal, which allows stating explicitly if the contribution is negative or positive (+,-);

- **Task-decomposition links** state the decomposition of a task into different intentional elements (i.e., goal, task, resource and softgoal). There is a relation AND when a task is decomposed.

In Figure 2.4, we present an example of the SR model refined from the SD diagram depicted in Figure 2.3. We can see that the strategic dependencies are linked to the intentional elements. In Figure 2.4, we observe that the *Meeting initiator* has as a main task to *Organize the meeting*. This task is decomposed with a task-decomposition link into a goal *Meeting to be scheduled*, and two softgoals *Quick* and *Low effort*. The goal *Meeting to be scheduled* is the end to achieve by the task *Schedule meeting* and so, they are related with a means-end link. Likewise, the *Meeting participant* has the task *Find agreeable date*, which is decomposed into the task *Agree to date*. These two tasks contribute negatively to the softgoal *User friendly* and *Minimum interruption* of the *Meeting participant*, respectively. Nevertheless, if these two softgoals are achieved, they contribute positively to the softgoal *Low effort* for arranging the meeting.

2.2.2 The NFR Framework

The *Non-Functional Requirements (NFR)* framework is a goal-oriented approach proposed in [53] for addressing *non-functional requirements (NFRs)*. The main objective of the NFR framework is to represent, organize and analyse NFRs. It is goal-oriented since it treats NFRs as goals to be achieved. However, it remains different from the traditional goal-oriented approach as presented in [57] [12] since the framework uses the notion of softgoal which represents a goal that has no clear-cut criteria to determine whether they have been satisfied or not. In [53], Chung et al. use the term *to satisfice* to indicate that the goal satisfying is accomplished within acceptable limits. Therefore, a softgoal is considered satisficed when there is sufficient positive evidence and little negative evidence against it.

The NFR framework is established with the following key concepts:

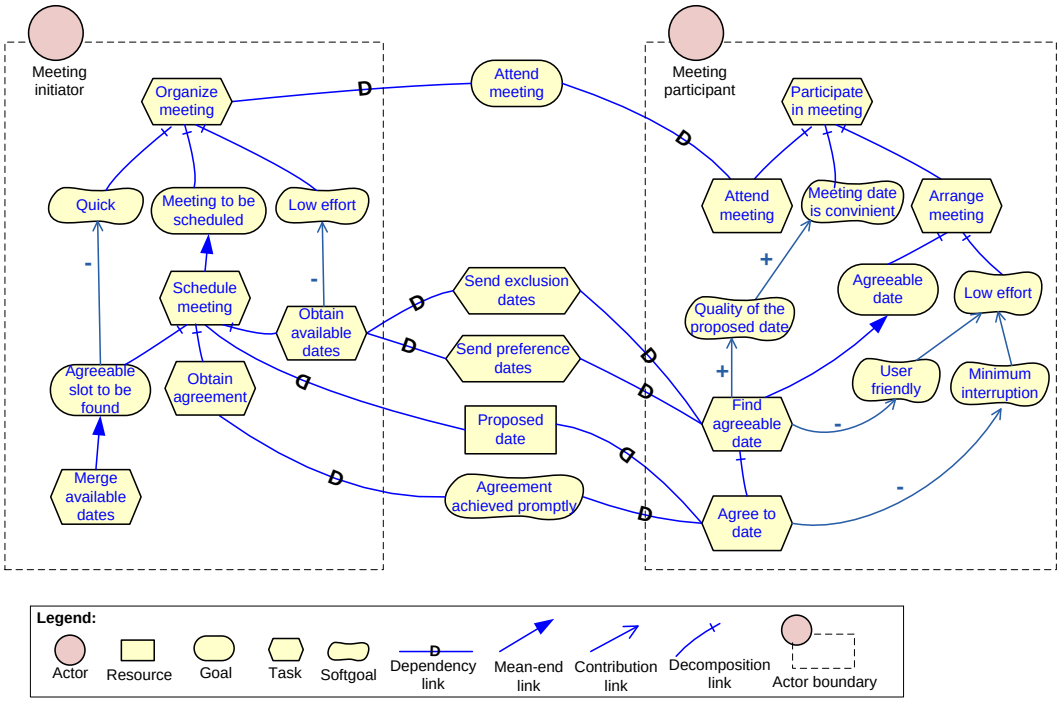


Figure 2.4: An example of Strategic Rational model adapted (from [170]).

- **Softgoal** is the basic unit for representing NFRs. There are three kinds of softgoals:
  - The NFR softgoals represent high-level non-functional requirements to be satisficed. The NFR softgoals have the following nomenclature: *Type [Topic1, Topic2, ...]*, where *Type* is a non-functional aspect (e.g. *Security* and *Topic* is a subject of the target system to which the softgoal is associated (e.g. *Accounts*). Topics can be further decomposed into attributes, indicated by a "." following the topic description (e.g. *Accounts.balance*);
  - Operationalizing softgoals are possible solutions (operations, processes, data representations, etc.) or design alternatives which help to achieve the NFR softgoal;
  - Lastly, claim softgoals justify the rationale and explain the context for a softgoal or interdependency link.
- **Interdependencies** indicate refinements of softgoals and the contributions of offspring softgoals toward the achievement of their parents. There are basically two types of contributions describing how the offspring contributes to satisfice its parent. The first one decomposes a softgoal into a set of sub-goals through

the use of *AND/OR* contributions. *AND* describes the relationship in which all the refined sub-goals must be met for the goal to be met. *OR* defines the relationship that at least one of the refined sub-goals must be met for the goal to be met. The other type of contribution relates one softgoal to another with the following values: *BREAK("- -")*, *HURT("-")*, *UNKNOW("?")*, *HELP("+")* and *MAKE("++")*;

- **Softgoal Interdependency Graph (SIG)** is the graph where softgoals and their interdependencies are represented. Figure 2.5 shows an example of a SIG adapted from [53];

- **Catalogues** group an organized body of design knowledge about NFRs types, development techniques and correlations among *operationalizing* and *NFR softgoals* that can be used in different application domains to compose the SIG.

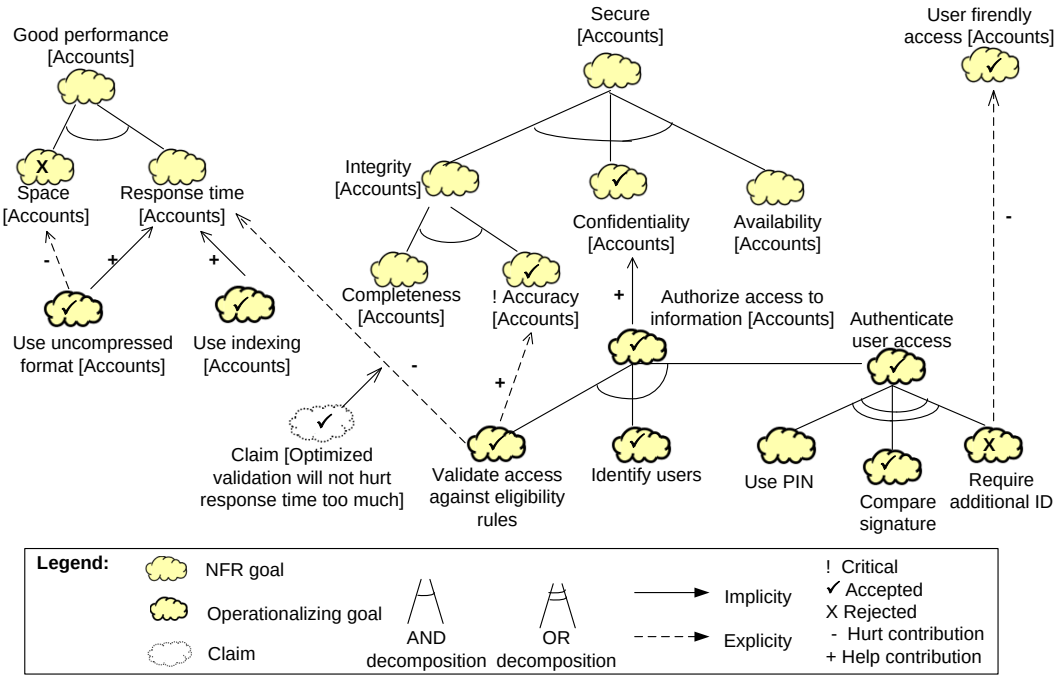


Figure 2.5: An example of SIG from ([53]).

2.3 System Development Life Cycle Models

According to [142], a *System Development Life Cycle (SDLC)* is "*a conceptual model used in project management to describe the stages involved in a system development project from an initial feasibility study through maintenance of the completed application*". Several SDLCs have been proposed over the years. The different SDLCs are all based on the same activities – *Analysis*, *Design*, *Implementation*, *Integration* and *Test* – and vary only in the way they are performed. In the section, we will overview different SDLCs.

2.3.1 The Sequential Model

The sequential model (e.g. a waterfall model) follows a systematic, sequential approach that begins at the system level and progresses successively from analysis to settlement. Each activity is regarded as concluded before the next activity begins. The output from one activity is the input to the next.

This model, represented in Figure 2.6, is the oldest project life cycle model in the software engineering history [56]. Nevertheless, there are some problems when applying this approach to software development.

- It is based on the assumption that it is possible to define and describe all system requirements and software features beforehand, or at least very early in the development process. However, many major problems in software engineering arise from the fact that it is difficult for the customer to state all requirements explicitly;

- It is difficult to add or change requirements during the development because, once performed, activities are regarded as completed. In practice, requirements will change and new features will be called for and a purely sequential approach to software development can be difficult and in many cases inadequate;

- Another problem of this approach is the late response to the customer. A working version of the software will not be available until late in the system development process and a major misunderstanding, if undetected until the working program is reviewed, can be disastrous.

Despite its disadvantages, the sequential model has an important place in software engineering. It constitutes the first attempt to build a framework for the proper incorporation of the software development activities. In practice, it is still in use today but it is never used in its pure form. For example, the waterfall model in combination with prototyping, or the V-model, has been widely used for entire life cycles or as a part of other models, covering particular phases of the entire process model [133].

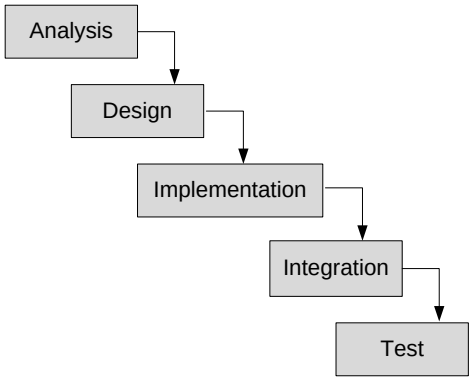


Figure 2.6: The sequential model (from [56]).

2.3.2 The V-Model

The V-model is a software development model which can be considered as an extension of the waterfall model. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape (see Figure 2.7). The main advance of this model is that it supports development with testing. It demonstrates the relationships between each phase of development and its associated phase of testing.

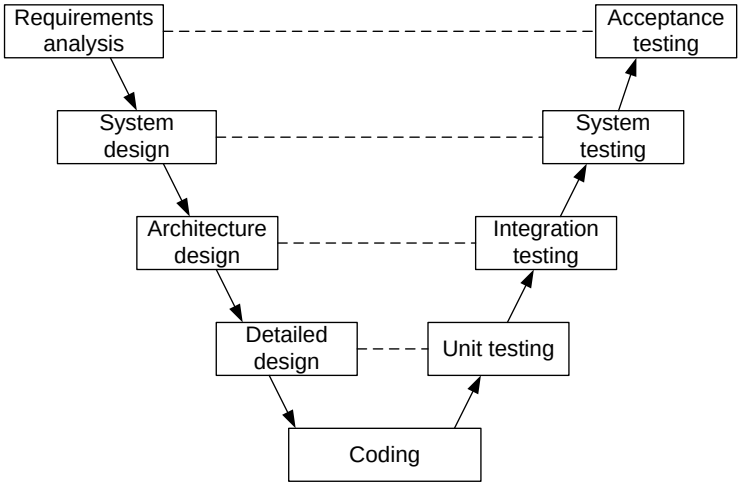


Figure 2.7: The V-model adapted (from [70]).

2.3.3 The Incremental Model

The incremental model combines elements of the sequential model with the iterative approach. Figure 2.8 shows, the incremental model applies the sequential model in

stages as calendar time progresses. Each sequence produces a deliverable "increment" of the software with new functionality added to the system [113].

When the incremental model is used, the first increment is often the core of the software system [140]. At this stage, the basic requirements are addressed but many supplementary features remain undelivered. As a result of the evaluation of the core software, a plan is developed for the next increment. The plan proposes the modification of the core software to satisfy requirements more effectively and the delivery of additional functionalities. This process is repeated following the delivery of each increment, until the complete software system has been developed.

The incremental model is useful for handling changes in requirements. Early increments can be implemented to satisfy current requirements and new and altered requirements can be addressed in later increments. If the core software is well received, the next increment can be added according to the plan. Increments can also be planned to manage technical risks.

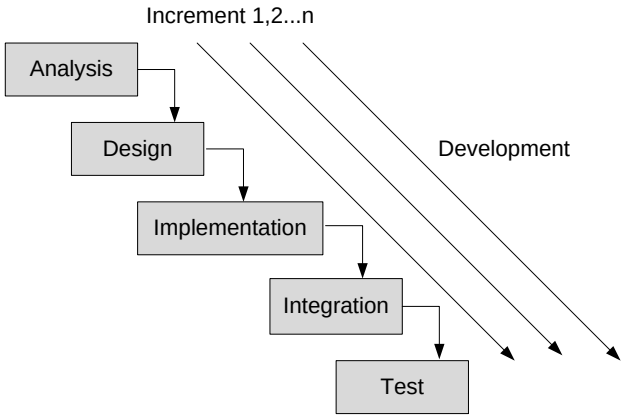


Figure 2.8: The incremental model (from [56]).

2.4 Object-Oriented Software Engineering

Object-Oriented Programming (OOP) has been the most widely-used programming paradigm in the development of information systems. It is claimed that the problem-solving techniques used in object-oriented programming more closely models the way humans solve day-to-day problems [38]. Following [38], in the older styles of programming, a programmer who is faced with some problem must identify a computing task that needs to be performed in order to solve the problem. Programming then consists of finding a sequence of instructions that will accomplish that task. But at the heart of OOP, instead of tasks we find *objects* - entities that have behaviors, that hold information, and that can interact with one another. Programming consists of designing a set of objects that model the problem at hand. Software objects in

the program can represent real or abstract entities in the problem domain. This is supposed to make the design of the program more natural and hence easier to get right and easier to understand.

Object-Oriented Software Engineering (OOSE) is a software design technique that is used in software design in OOP. OOSE is developed by Ivar Jacobson in 1992 [85]. It is made of a modeling language and a software development process. According to [14], the concepts and notation from OOSE have been incorporated into the *Unified Modeling Language (UML)*; and the software development process of OOSE has evolved into the *Rational Unified Process (RUP)*.

2.4.1 The Unified Modeling Language

The *Unified Modeling Language (UML)* [151] is a standardized modeling language in the field of object-oriented software engineering. UML is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software-intensive system under development. It offers a standard way to visualize a system's architectural blueprints, including elements such as activities, actors, business processes, database schema, (logical) components, programming language statements, and reusable software components.

UML has been normalized by the *Object Management Group (OMG)* in order to furnish a universal communication support because it is independent from application domain and programming languages. It is issued from the merging of the Booch(OOT) [35], Rumbaugh (OMT) [143] and Jacobson (OOSE) [85]. UML has significantly matured since its version 1.1. Several minor revisions (UML 1.3, 1.4, and 1.5) fixed shortcomings and bugs with the first version of UML, followed by the UML 2.0 major revision, which was adopted by the OMG in that was adopted by the OMG in 2005.

UML 2.0 has 13 types of diagrams that can be used to model the system in multiple views as well as multiple levels of abstraction. These diagrams are grouped into two categories (see Figure 2.9).

- **Structure diagrams**: emphasize the things that must be presented in the system being modeled. They are used to document the software architecture. Structure diagrams encompass:
  - **Class diagram**: describes the structure of a system by showing the system's classes, their attributes, and the relationships among the classes;
  - **Component diagram**: describes how a software system is split up into components and shows the dependencies among these components;
  - **Composite structure diagram**: describes the internal structure of a class and the collaborations that this structure makes possible;

  - **Deployment diagram**: describes the hardware used in system implementations and the execution environments and artifacts deployed on the hardware;
  - **Object diagram**: shows a complete or partial view of the structure of an example modeled system at a specific time;
  - **Package diagram**: describes how a system is split up into logical groupings by showing the dependencies among these groupings.

- **Behavior diagrams**: emphasize what must happen in the system being modeled. Since behavior diagrams illustrate the behavior of a system, they are used extensively to describe the functionality of software systems. Structure diagrams encompass:
  - **Activity diagram**: describes the business and operational step-by-step work-flows of components in a system. An activity diagram shows the overall flow of control;
  - **State machine diagram**: describes the states and state transitions of the system;
  - **Use case diagram**: describes the functionality provided by a system in terms of actors, their goals represented as use cases, and any dependencies among those use cases.
  - **Interaction diagram**: is a subset of behavior diagrams and emphasizes the flow of control and data among the things in the system being modeled. Interaction diagrams include:
    * **Communication diagram**: shows the interactions between objects or parts in terms of sequenced messages. They represent a combination of information taken from Class, Sequence, and Use Case Diagrams describing both the static structure and dynamic behavior of a system;
    * **Interaction overview diagram**: provides an overview in which the nodes represent communication diagrams;
    * **Sequence diagram**: shows how objects communicate with each other in terms of a sequence of messages. Also indicates the lifespans of objects relative to those messages. Timing diagrams: a specific type of interaction diagram where the focus is on timing constraints.

2.4.2 The Rational Unified Process

The *Rational Unified Process (RUP)* is a prescriptive, well-defined system development process, often used for object-oriented systems development [102]. The key aspects of RUP are:

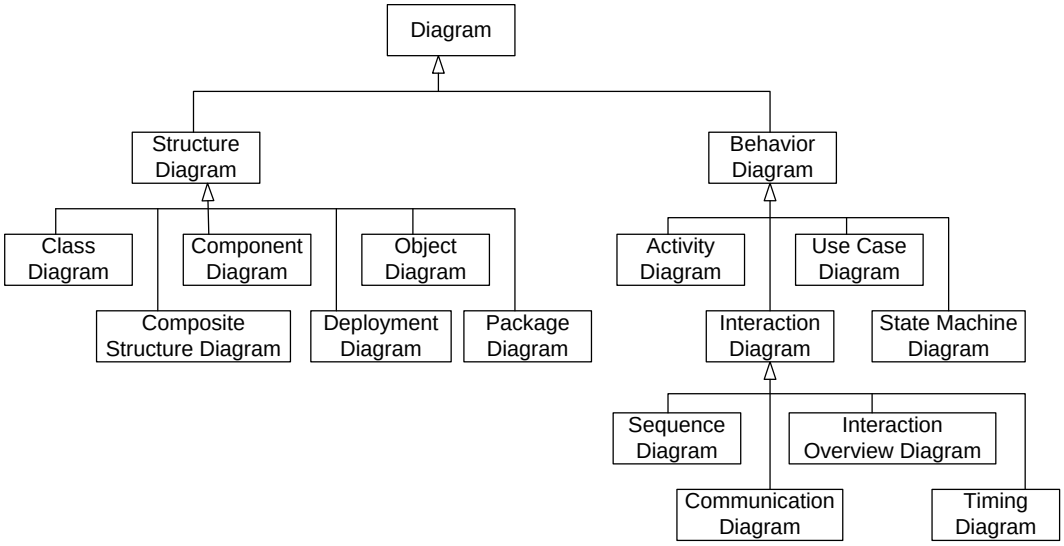


Figure 2.9: UML diagrams .

- **Iterative and incremental**. The process recognizes that it is practical to divide large projects into smaller ones or mini-projects. Each mini-project comprises an iteration that results in an increment. An iteration may encompass all the activities in the process; item **Risk driven**. Risk management is integrated into the development process and the iteration plan is developed based on high priority risks;

- **Use-case driven**. The process employs use-cases to drive the development from the beginning of the project to its end;

- **Architecture centric**. Architecture is the primary artifact to conceptualize, construct, manage and evolve the system. The process seeks to understand the most significant static and dynamic aspects in terms of software architecture.

RUP is structured in two dimensions: phases, which represent the four major stages that a project goes through over time, and disciplines, which are logical grouping of activities that take place throughout the project [102] (see Figure 2.10).

As depicted in Figure 2.10, RUP consists of four phases:

1. **Inception**. This phase is all about understanding the project scope and objectives and getting enough information to confirm that the project should be attempted. The main objectives of this phase are:

   - Understand what to build;
   - Identify key system functionality;

- Determine at least one possible solution;
- Understand the costs, schedule and risks associated with the project;
- Decide what process to follow and what tools to use.

2. **Elaboration**. During this phase, the system requirements are specified in greater detail and the architecture is defined and proved. The objectives of this phase are:

   - Get a more detailed understanding of the requirements;
   - Design, implement, validate and baseline the system architecture;
   - Mitigate essential risks and produce more accurate schedule and cost estimations;
   - Refine the development case and put the development environment in place.

3. **Construction**. During this phase, the system is developed to the point where it is ready for deployment. If necessary, early release of the system are deployed, either internally or externally, to obtain user feedback.

4. **Transition**. This phase focuses on delivering the system into production. The goal of this phase is to ensure that the requirements have been met to the satisfaction of the stakeholders. This phase is often initiated with a beta release of the application. Other activities of this phase include site preparation, user manual completion, and defect identification and correction.

RUP phases are divided into one or more iterations. Iterations address only a portion of the entire system being developed. Each iteration has a fine-grained plan with a specific goal and builds upon the work done by previous iterations. During each iteration we will alternate back and forth between the activities of the disciplines to achieve the goals of that iteration. The disciplines of RUP include:

1. **Business Modeling**. The goal of the business modeling is to understand the needs of the business. It involves working closely with the project stakeholders to assess the current status of the organization in which a system is to be deployed (the target organization) and identify improvement potentials.

2. **Requirements**. The goal of the requirements discipline is to elicit, document, and validate the scope of the project – what is and what is not to be built. This information is used by analysts, designers and programmers to build the system, by testers to verify the system, and by the project manager to plan and manage the project.

3. **Analysis and Design**. The goal of this discipline is to analyze the requirements for the system and to design a solution to be implemented, taking into consideration the requirements, constraints and all applicable standards and guidelines.

4. **Implementation**. The goal of the implementation discipline is to transform the design into executable code and to perform a basic level of testing, in particular unit testing.

5. **Test**. The goal of the test discipline is to perform an objective evaluation to ensure quality. This involves finding defects, validating that system works as designed and verifying that the requirements are met.

6. **Deployment**. The goal of the deployment discipline is to plan for the delivery of the system and to execute the plan to make the system available to end users.

7. **Configuration and Change Management**. The goal of this discipline is to manage access to the work products of the project. This involves tracking versions over time as well as controlling and managing changes to these versions.

8. **Project Management**. The goal of this discipline is to direct the activities that take place throughout the project. This involves managing risks, directing people and coordinating with people and systems outside the scope of the project to be sure that it is delivered on time and within budget.

9. **Environment**. The goal of this discipline is to support the development organization in terms of ensuring that the proper process, guidance (standards and guidelines), and tools(hardware or software) are available for the team as needed.

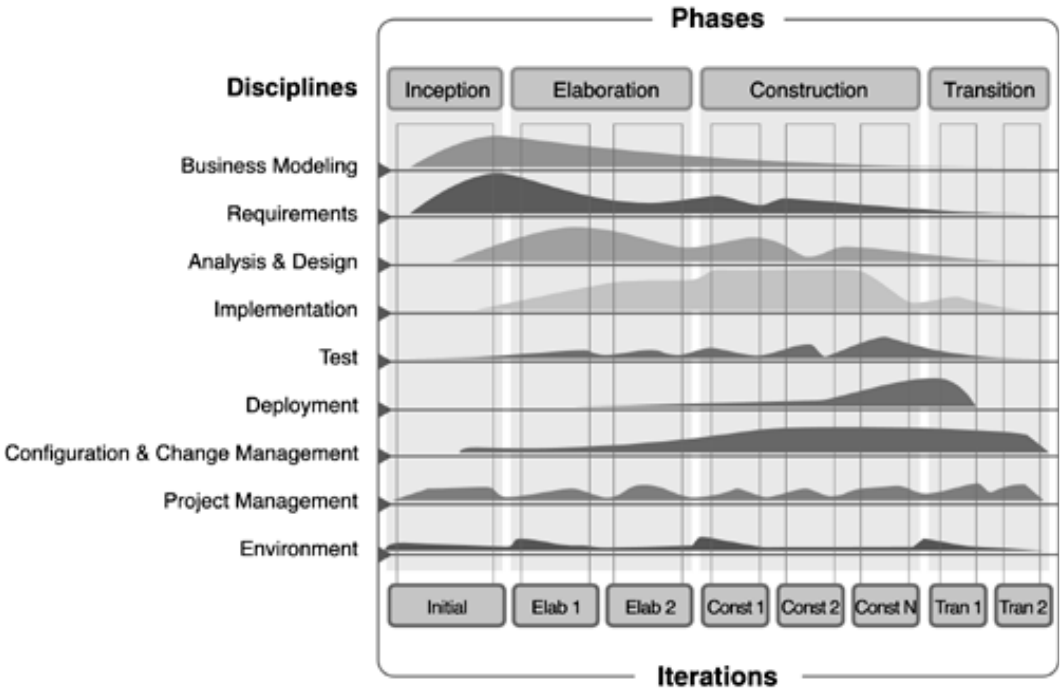


Figure 2.10: The rational unified process (from [102]).

2.5 Agent-Oriented Software Engineering

Agent-oriented software engineering is one of the most recent contributions to the field of software engineering. Its main concept is the *agent* but there is at present no a universal consensus on its definition. However, an increasing number of researchers find the following characterization useful [167]:

An agent is an encapsulated computer system that is situated in some environment, and that is capable of flexible, autonomous action in that environment in order to meet its design objectives.

An agent can be useful as a stand-alone entity that delegates particular tasks on behalf of a user (e.g. a personal digital assistant and e-mail filter). However, most of the cases, an agent exists in an environment inhabited by other agents, called a *Multi-Agent System (MAS)*. In a MAS, the global behavior derives from the interaction between the constituent agents: they cooperate, coordinate or negotiate with one another [159].

Similar to the research issues of other branches of SE, e.g. OOSE, the main purposes of AOSE are to create methodologies and tools that ease the development

of agent-based software which has to be flexible, easy-to-use, scalable and of high quality.

2.5.1 Benefits of AOSE

Comparing to other programming paradigms, *Agent-oriented programming (AOP)* can be seen as an extension of OOP [158]. As described in section 2.4, the main concept of OOP is the *object*. An object is a logical combination of data structures and their corresponding methods (functions). Agents are similar to objects, but they also support structures for representing mental components, i.e. beliefs and goals. As opposed to objects, agents are able to act without external intervention in order meet theirs goals. In addition, agents support high-level interaction using agent communication languages (e.g. FIPA ACL and KQML) between agents based on the "speech act" theory as opposed to ad-hoc messages frequently used between objects [107].

Indeed, there is no evidence yet to suggest that AOSE will become broadly used in enterprise system developments. However, there are convincing arguments that analyzing, designing and implementing a complex software system as a collection of interacting autonomous agents represents a promising point of departure for software engineering [86]. In [86], the authors show that:

- the agent-oriented decompositions are an effective way of partitioning the problem space of a complex system. A complex software system can be decomposed into subsystems, the same way that a MAS can be decomposed into the elements that constitute the system (software agents);
- the key abstractions of the agent-oriented mindset, such as agents, (social) interactions and organizations, are a natural mean of modeling complex systems;
- the agent-oriented philosophy for identifying and managing organizational relationships is appropriate for dealing with the dependencies and interactions that exist in a complex system.

2.5.2 MAS Development Methodologies

MAS has become one of the most interesting research fields in the computer science community. Many MAS software development methodologies have been proposed such as *GAIA* [168], *MASE* [59], *MESSAGE* [42], *Tropos* [48], *INGENIAS* [132], *PASSI* [40] and *ADELFE* [22].

Tropos is typically representative of these methodologies. It is a requirements and organizational driven software development methodology that is founded on intentional and social concepts. Tropos adopts the i* organizational modeling framework

[171], which offers the notions of actor, goal and (actor) dependency, and use these as a foundation to model early and late requirements, architectural and detailed design. Tropos and other MAS software development methodologies use the Waterfall Software Development Life Cycle. These methodologies typically do not cover all the aspects of the software engineering life cycle depicted in the Spiral Model as some object-oriented development methodologies do such as the Unified Process [103]. It is obvious that the advantages of spiral development, such as efficient acquisition, early implementation, continuous testing and modularity should be applied in the development of Agent-Oriented software.

I-Tropos [159, 163, 164] is an interesting representation of spiral SDLC methodologies since it is an original spiral software development process for agent-oriented software development. It extends the waterfall SDLC of the Tropos methodology to a spiral one in order to include the iterative use of several software engineering and project management disciplines during the different phases of the process. These phases are typically inspired by the UP project management processes family [103]; they concern the inception of the project, more specifically the requirements gathering and business case definition, the elaboration of the logical architecture of the system-to-be, the construction of the system and finally the transition to put the system in production and operation.

I-Tropos is made of disciplines repeated iteratively during these phases while the effort spent on each discipline is variable from one iteration to another. The core disciplines of I-Tropos are *Organizational Modeling*, *Requirements Engineering*, *Architectural Design*, *Detailed Design*, *Implementation*, *Test* and *Deployment*. The Organizational Modeling and Requirements Engineering disciplines are respectively strongly inspired from *Early and Late Requirements phases* of Tropos. Moreover, I-Tropos includes support disciplines to handle the project development i.e. *Risk Management*, *Time Management*, *Quality Management* and *Software Process Management*.

2.6 Chapter Summary

This chapter has presented the state of the art in the SE. It focuses on the relevant approaches to our research context. Firstly, it presented different definitions of SE and specifies the one that best match our work – "*SE is an engineering approach to the software systems development that provides methodologies, tools and techniques to help software system developers in the analysis, design, implementation and testing of software system.*".

Next, it has presented our review on relevant RE frameworks and explained our selection of them to be used within our methodology. The selected RE frameworks, i.e. the i* and NFR frameworks, are also presented.

Then, it has introduced some representative software development life cycle models including the sequential model, the V model and the incremental model. It also exposed OOSE which has been the most widely-used in the development of information systems. It introduced unified modeling language (UML) and the software development process (RUP) that constitute the OOSE.

Finally, this chapter described AOSE. It pointed out its benefits and overviewed some relevant MAS methodologies, specifically Tropos and I-Tropos.

Chapter 3

Component Based Software Development

This chapter provides a state of the art review on *COTS-Based Software Development (CBSD)*. Section 3.1 presents the characteristics of COTS components. Section 3.2 overviews some relevant CBSD life cycle models. Section 3.3 introduces representative COTS selection approaches. Section 3.4 describes different evaluation strategies that have been used during COTS selection. Section 3.5 reviews three main decision making techniques for COTS selection. Section 3.6 exposes software project management for CBSD.

3.1 COTS Component

This section presents the characteristics of COTS components. It begins with the COTS definitions and followed by the component granularities.

3.1.1 COTS Definition

The literature about COTS addresses several development issues and is very heterogeneous in terminology. A consensus about the COTS characteristics and their definition does not exist. However, authors agree that COTS are a special class of reusable components.

COTS can be either software or hardware or a mixture of both. In this dissertation, **we only focus on software COTS**, however some of the issues and advices are equally applicable to hardware.

A survey about the different meaning and coverage of the COTS is presented in [121], as summarized in Table 3.1. In addition, more recently, [153] gives the detailed, empirically based definition: *"A COTS product is a commercially available*

or open source piece of software that other software projects can reuse and integrate into their own products."

From these definitions we can see that off-the-shelf products coverage can be very different. More likely, researchers and practitioners could use the same word with different meanings. Some of them use the term COTS covering freeware and Open Source Software as well as other kinds of components (e.g.,[117][24]).

3.1.2 COTS Component Granularity

The authors of [78] define different levels of COTS component granularity that define quite different categories of component as depicted in Figure 3.1. They are presented bellow from the finest-grained to the most coarse-grained.

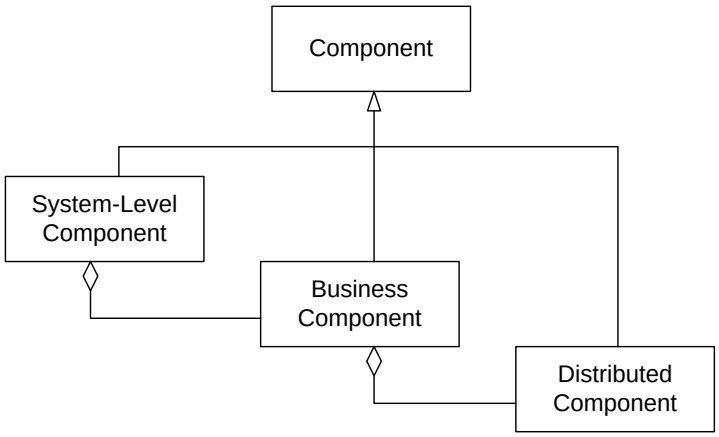


Figure 3.1: Different levels of component granularity (from [78]).

3.1.2.1 Distributed Component

The lowest granularity of software component is called a *"distributed component."* It is normally realized using a component implementation technology such as COM [58], Enterprise Java Beans [6], or a CORBA-Component product [68]. The distributed component has the following implementation characteristics:

- It has a well-defined build-time and run-time interface;
- It can be independently plugged into a run-time environment;
- It is *network addressable,* meaning that it can be addressed over the network at run-time.

A distributed component is normally composed of a number of classes, as illustrated in Figure 3.2.

Table 3.1: Different definitions of COTS (from [121]).

Source	Definition
Vigder and Dean [157]	Define COTS as pre-existing software products, sold in many copies with minimal changes; whose customers have no control over specification, schedule, and evolution; access to source code as well as internal documentation is usually unavailable; complete and correct behavioral specifications are not available.
Carney and Leng [47]	This approach considers Origin and Modifiability as attributes to define COTS. The possible values for these attributes are: • **Origin**: Independent Commercial Item, Special Version of Commercial Item, Component Produced by Contract, Existing Components from External Sources, Component Produced In-house. • **Modification**: Extensive Reworking of Code, Internal Code Revision, Necessary Tailoring and Customization, Simple Parameterization, Very Little or no Modification.
Basili and Boehm [18]	Specify that COTS has the following characteristics: 1. the buyer has no access to the source code; 2. the vendor controls its development, and; 3. it has a non-trivial installed base. This definition is more restrictive and does not take into account some types of software products like software products developed for special purposes and not widely deployed, special version of commercial software products and open source software.
Software Engineering Institute (SEI) [37]	A COTS product is: sold, leased, or licensed to the general public; offered by a vendor trying to profit from it; supported and evolved by the vendor, who retains the intellectual property rights; available in multiple, identical copies; and used without source code modification.

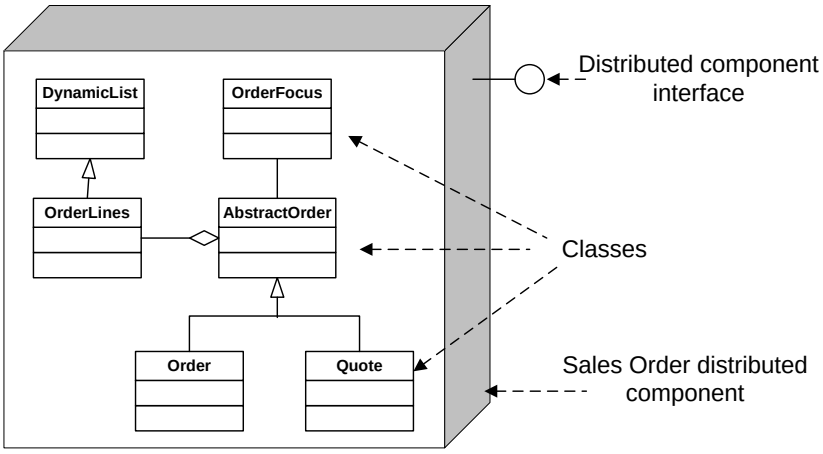


Figure 3.2: Example of a distributed component and classes (from [78]).

In [78], the nature of the distributed component is defined from the point of view of the functional designer. The functional designer should not be concerned with technical complexities such as *locating objects on the network*, *how data is communicated from a client to a server*, *transactional boundaries*, *concurrency*, *event management*, etc. The definition of *distributed component* abstracts these and other aspects and is largely technology independent.

3.1.2.2 Business Component

Following [78], a *business component* is the software implementation of an autonomous business concept or business process. It consists of all the software artifacts necessary to represent, implement, and deploy a given business concept as an autonomous reusable element of a larger distributed information system.

A *business component* does not represent just any business concept, but those concepts that are relatively autonomous in the problem space. For example, the concept *customer* would normally be a good candidate, whereas *date* would probably not. It is implemented using software component technology which is as a composition of distributed components. "Autonomous" does not mean isolated. As business concepts in the problem space relate to other concepts, for example, an *order* is by a certain *customer* for one or more *items*, so business components mainly provide useful functionalities through the *collaboration* with other business components.

3.1.2.3 System-Level Component

A *system-level component*, corresponds to an information system, e.g. an invoice management system or a payroll system. [78] defines it as follows:

A system-level component is a set of cooperating business components assembled together to deliver a solution to a business problem.

According to this definition, a system-level component is simply a set of cooperating business components able to provide a business solution, that is, to address a specific business problem in the domain space. An example of a system-level component that manages vendor invoices is depicted in Figure 3.3 where each box represents an individual business component.

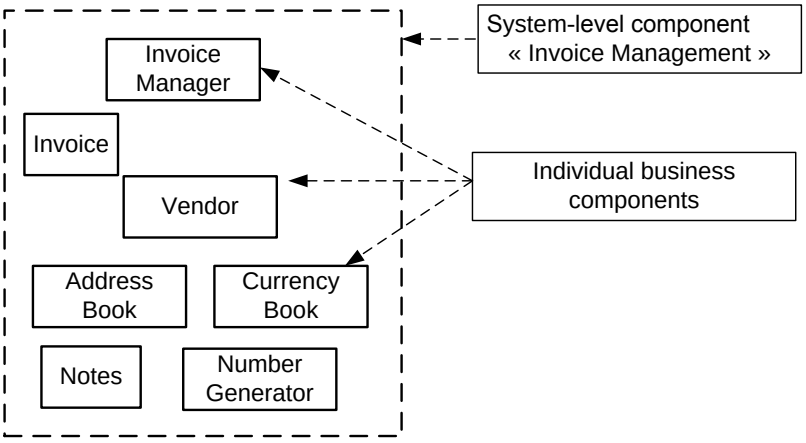


Figure 3.3: Example of a system-level component (from [78]).

3.2 CBSD Life Cycle Models

There is a consensus that the use of a COTS component implies changes in the software process [37]. Most studies on CBSD found in literature can be positioned on one of the two following levels: at the level of the whole CBSD life cycle or at the level of one of its phases e.g. COTS selection. This section presents different life cycle models for CBSD that have been proposed in the literature.

3.2.1 The Sequential Model

A CBSD process that follows the sequential approach is proposed by Sommerville in [152]. It is composed of six phases illustrated in Figure 3.4:

- **Outline System Requirements.** The user requirements are outlined briefly – rather than developed in detail, as specific requirements limit the number of components that might be used;
- **Identify Components.** A complete outlined set of requirements are used to identify as many components as possible for reuse;

- **Negotiate Requirements.** Requirements are refined and modified so that they can comply with available components;
- **Architectural Design.** Architectural design is developed;
- **Search for reusable components.** Requirements can be negotiated to incorporate architecture compatible components. After system architecture is designed, steps 2 and 3 may be repeated;
- **Integration.** Finally the chosen components are integrated to build the system.

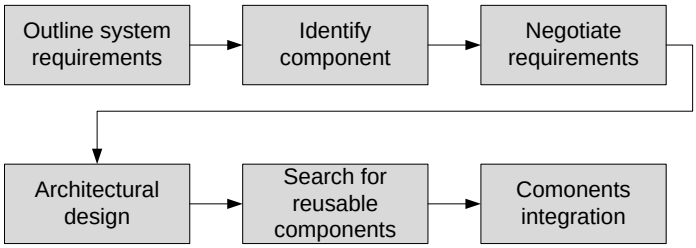


Figure 3.4: The CBSD process model proposed in [152].

On the other hand, Morision et al. [120] have investigated the various processes that were used across fifteen CBSD projects at the National Aeronautic and Space Administration (NASA). Figure 3.5 illustrates the process that is the most representative for the processes used in the projects. This process also follows the sequential approach.

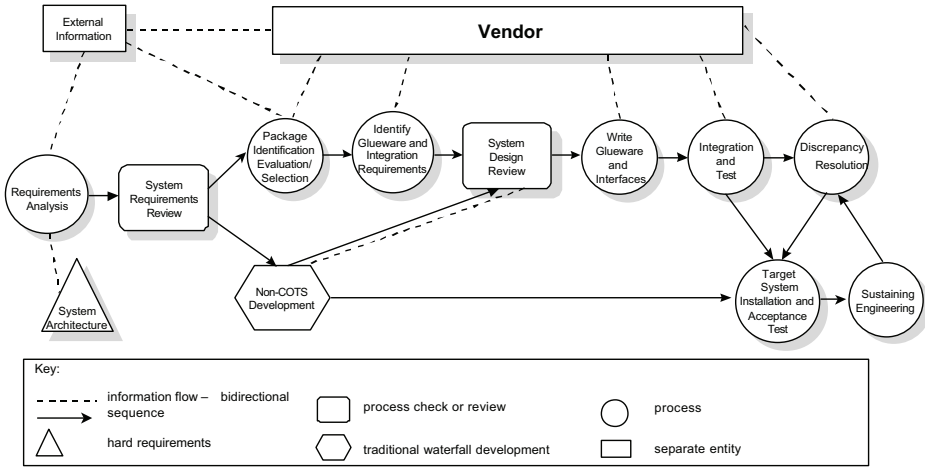


Figure 3.5: NASA's CBSD process (from [120]).

3.2.2 The V-Model

In [56], Crnkovic proposes a process model for CBSD by adapting the V-model for traditional software development. The activities in different phases of this development process is illustrated in Figure 3.6.

- **Requirements Analysis and Specification.** Requirements are analyzed and specified keeping in mind the available components. If possible, requirements can be negotiated as well in order to make use of the available components;
- **System and Software Design.** In this phase again, component pool is available. Components selected in the previous phase may have to be rejected if they do not fit into the overall design of the software. Alternatives can be selected from the component pool;
- **Implementation and Unit Testing.** An ideal case to build an application is the direct integration or connection of component interfaces. But in practice, glue code/component wrappers need to be written to bridge component interface mismatches. Sometimes new functions have to be written to fill the gaps in system requirements and component capabilities. These modules are then tested separately;
- **System Integration.** The integration process includes integration of standard infrastructure components such as database management and communication software components that build a components framework and the application components;
- **System Verification and Validation.** Standard tests and verification techniques are used here. Error identification is more difficult in case of black box components, which are procured from various vendors;
- **Operation Support and Maintenance.** In the operational system, new components may have to be added or existing components have to be modified/removed in order to support the changing requirements. The support and maintenance process includes deployment of new or modified components in the system, change in glue-code, or replacement of troubling components.

3.2.3 The Y-Model

Capretz proposed in [44] a CBSD life cycle named the Y model. In this model, basic activities of a software development process such as System Analysis, Design, Implementation, Testing, Deployment and Maintenance are available as such. Some

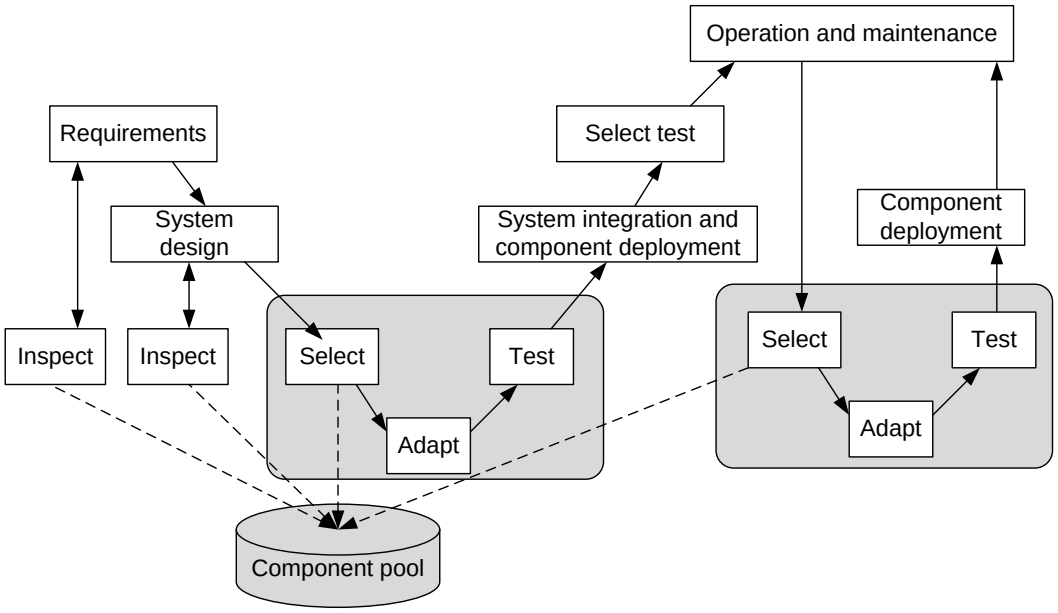


Figure 3.6: The V-model adapted for component-based system development (from [56]).

additional activities are added to support the component-based software development as depicted in Figure 3.7. New phases introduced in this process are as follows:

- **Domain Engineering.** Domain engineering is concerned with identification of potentially reusable components for a particular domain. For example in the banking domain, the components such as *CreateAccount*, *CheckBalance*, *DepositAmount* and *WithdawAmount* can be reused across multiple banking applications. Domain engineering is a process of analysing an application domain in order to identify area of commonality and ways to describe it using a uniform vocabulary;

- **Frameworking.** A framework is a skeleton or a template used for producing software in an application domain. Frameworks capture the semantic relationships between the components of a particular domain. The main purpose of frameworking phase is to reuse the software components already developed and then classify them further to form new frameworks.

- **Assembling.** In this phase, the software application is composed from the existing reusable software components;

- **Archiving.** The components developed for a particular software application

are archived for future use in other related applications. This involves activities such as cataloging and storage.

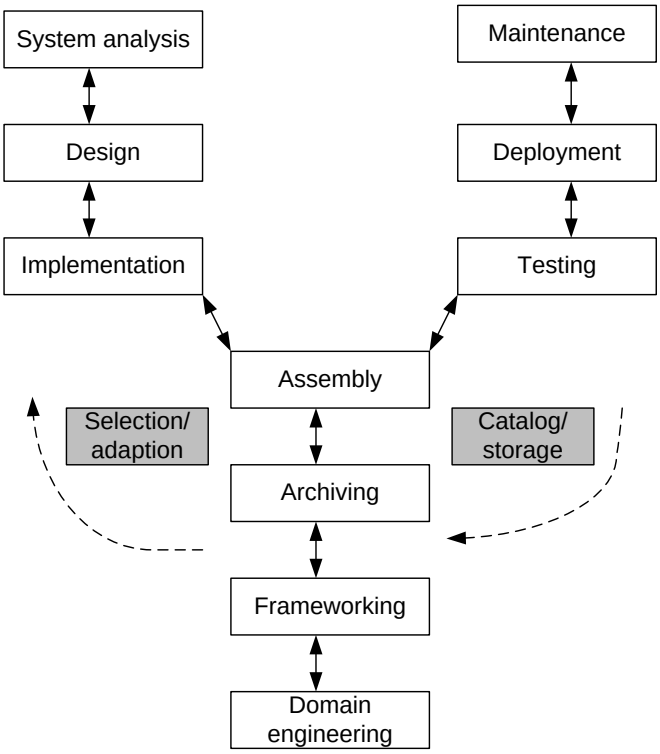


Figure 3.7: The Y-model (from [44]).

3.2.4 The Evolutionary Process for Integrating COTS

The *Evolutionary Process for Integrating COTS (EPIC)* [4] is a modified form of RUP [102]. The essence of the process lies in a "simultaneous definition and trade-offs" among four *spheres of influence*, as depicted in Figure 3.8[4].

- **Stakeholder needs and business processes** denotes requirements (including quality attributes such as performance, security, and reliability), end-user business processes, business drivers, and operational environment;
- **Marketplace** denotes available and emerging COTS technology and products, non-development items and relevant standards;
- **Architecture and Design** denotes the essential elements of the system and the relationship between them. The elements include structure behavior, us-

age, functionality, performance, resilience, reuse, comprehensibility, economic and technology constraints and tradeoffs, and aesthetic issues;

- **Programmatics and risk** denotes the management aspects of the project. These aspects consider the cost, schedule, and risk of building, fielding, and supporting the solution to include the cost, schedule, and risk for changing the necessary business processes.

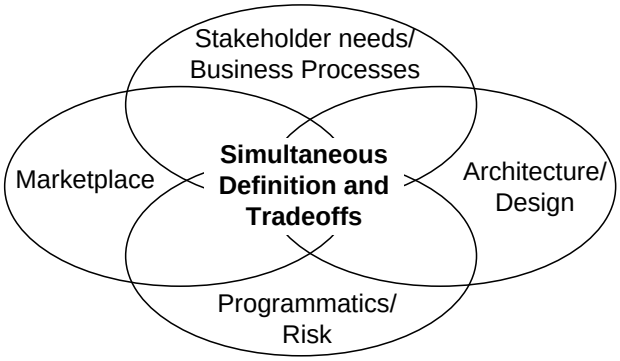


Figure 3.8: EPIC's spheres of influence (from [4]).

These four spheres are simultaneously defined and traded through the life of the project because a decision in one sphere will inform and likely constrain the decision that can be made in another sphere. For instance, a stakeholder need may be stated in a way that cannot be satisfied by any pre-existing component. Similarly, a potential pre-existing component may not be compatible with the organization's existing infrastructure or use a licensing strategy that would be cost prohibitive. Furthermore, the new release of an already selected component may change the behavior of the solution.

In order to maintain balance between the four spheres, EPIC creates an environment that supports the iterative definition of the four spheres over time while systematically reducing the trade space within each. This allows a decision in one sphere to influence, and be influenced by, decisions in other spheres. Each iteration in EPIC, as depicted in Figure 3.9 consists of:

1. Planning the iteration;

2. Gathering information and refining the solution set;

3. Assembling an executable system;

4. Assessing how closely the iterations objectives were (or were not) met.

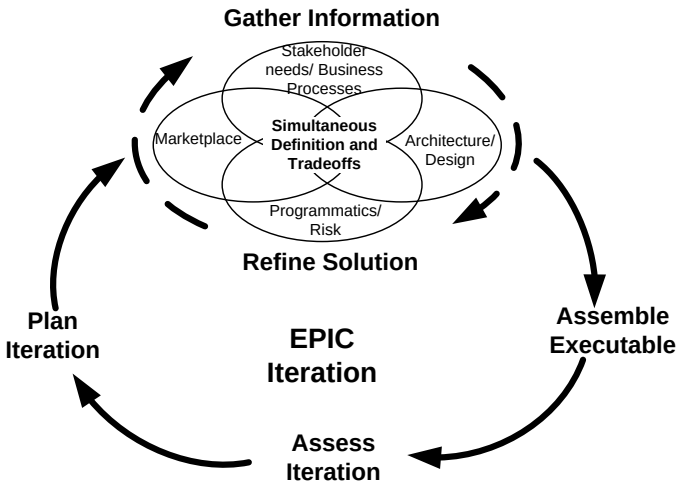


Figure 3.9: An iteration in EPIC (from [4]).

While these activities are the same for every iteration, the focus, depth, and breadth of the activities within an iteration are adjusted to meet specific iteration objectives.

3.3 COTS Selection Processes

While progressing, COTS research has proposed several approaches for addressing diverse aspects of COTS selection [116]; a summary is depicted in Table 3.2.

Table 3.2: COTS selection evolution (adapted from [116]).

Year	Name	Evolution
1999	OTSO [98]	The basic structure of COTS selection process
1996	OTSO [99] [100]	Further elaboration of OTSO
1997	IusWare [122]	Formalization of of the COTS selection activities (mainly evaluation)
1997	PRISM [109]	Generic architecture to be used during the COTS evaluation process
1997	CISD [154]	Multiple COTS selection
1998	PORE [110]	Requirements engineering process for COTS selection process
1999	STACE [105]	Studying the effects of social factors
2000	CAP [130]	Tailorability of the evaluation process
2001	CRE [8]	Emphasis on non-functional requirements
2001	CARE [49]	Further refinement for the requirements engineering process (ongoing project)
2002	PECA [54]	Detailed tailorable process
2002	StoryBoard [74]	Use of screenshots and use-cases for requirements
2002	CS [39]	Multiple COTS selection
2003	WinWin [28]	Risk-driven evaluation
2004	DesCOTS [73]	Using quality models during the evaluation
2007	MiHOS [117]	Systematic handling for the mismatches between COTS attributes and requirements
2008	GOThIC [15]	Goal-oriented method for structuring COTS market-place

This section does not aim at describing all the existing COTS selection approaches in literature but only some relevant ones on the basis of the different aspects of COTS selection that they have addressed.

3.3.1 Basic Structure of COTS Selection Process

Following [116], the *Off-the-Shelf Option (OTSO)* method has been considered as the first widespread COTS selection method. It was firstly proposed by Kontio in [98] and was then further elaborated in [99] and [100]. OTSO is considered as an

important milestone in the evolution of COTS selection practices as it served as a basis for other approaches. It defined the basic structure of the COTS selection process.

This method compares COTS products based on two factors: *value* and *cost*. The value is estimated based on hierarchical criteria which consist of functionalities, qualities, strategic concerns, and architectural constraints. The cost is estimated based on: *acquisition cost*, *further development costs*, and *integration cost*.

The OTSO method is composed of the following main steps:

1. **Evaluation criteria**: defining evaluation criteria;
2. **Search**: searching the market for possible COTS;
3. **Screening**: filtering out the COTS that do not comply with the must-have requirements;
4. **Evaluation**: evaluating the benefit and cost of each COTS candidate;
5. **Analysis of results**: using the analytic hierarchy process (AHP) [144] to consolidate the evaluation results and select a COTS product.

The OTSO method provides the basic structure of COTS selection methods and serves as a basis for other approaches. However, the main limitation of this approach is the lack of attention to requirements. It assumes that the requirements have been already defined and that they are fixed. Consequently, the method does not provide or suggest any effective mean to acquire requirements. Moreover, the OTSO just mentions the possibility to have mismatches between the system requirements and COTS features but do not provide any strategy on how to deal with them. Note that such situations are very common in CBSD and need to be properly examined.

3.3.2 Requirements-Driven COTS Selection Approaches

The importance of a suitable requirements engineering process for CBSD has been more and more evidenced since 1998. In this context, the *Procurement-Oriented Requirement Engineering (PORE)* [110] approach represented a key milestone. It suggests that requirements engineering and the elicitation of COTS features be conducted in parallel, as depicted in Figure 3.10. This means that the defined requirements inform the selection process and vice versa, which is more realistic than a fixed set of requirements.

Moreover, the PORE approach applies the progressive filtering evaluation strategy (i.e., the products that do not meet core requirements are selectively and iteratively rejected and removed for the candidate list). At the beginning of the process there are few requirements specified and a large number of candidate products. Through several iterations, it is possible to refine the product list until the

most suitable product is selected. According to the method, the compliance between features of COTS and system requirements is a fundamental step for effective product selection. However, it does not describe in detail how the matching between requirements and COTS features is performed and how products are eliminated.

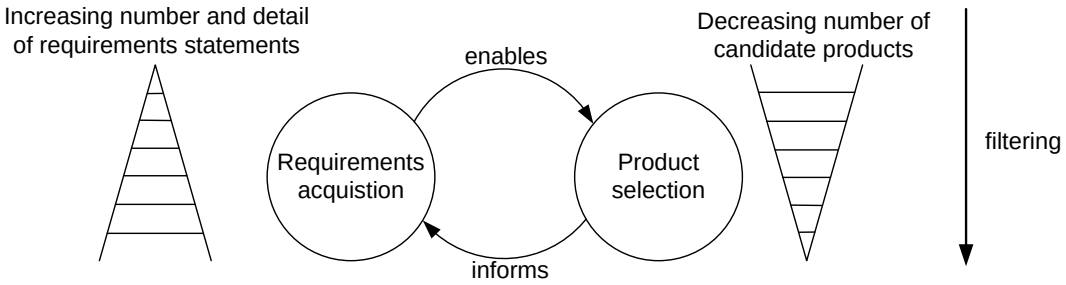


Figure 3.10: Overview of the PORE's iterative process (from [127]).

The *COTS-Based Requirements Engineering (CRE)* [8] is another requirements-driven COTS selection approach. It emphasizes the important of non-functional requirements as decisive criteria to evaluate alternative COTS components. A key issue addressed by this method is the definition and analysis of NFRs during the COTS selection process. It adopts the NFR framework proposed in [53] (see Section 2.2.2) for gathering the user NFRs. Similarly to the PORE , the CRE approach suggests iterations between requirements acquisition and product selection applying the progressive filtering evaluation strategy. The CRE approach is composed of four iterative phases: *Identification*, *Description*, *Evaluation*, and *Acceptance*. The identification phase is based on a careful analysis of influencing factors; there are five groups of factors: user requirements, application architecture, project objective and restrictions, and product availability and organizational infrastructure. The core requirements and COTS candidates are identified in this phase. During the specification phase, further requirements are defined. The NFR framework is used to model the NFRs. In the evaluation phase, COTS candidates are evaluated. During the acceptance phase, the evaluation team has to resolve legal issues pertaining to the purchasing of the product and licensing. The highest ranked COTS is selected if it passes some legal acceptance tests; if not the next COTS is taken, and so on. One of the main drawbacks of this methods is it does not address issues of quality testing and it is not clear how the product's quality issues are verified with regard to the system non-functional requirements. Another problem with the method is concerned to the lack of support in cases when NFRs are not properly satisfied.

In 2001, a project was started by Chung et al. to define a more complete COTS selection approach called *COTS-Aware Requirements Engineering (CARE)* approach [49][50][51][52]. The CARE approach draws upon the ideas of existing methodologies including *Rational Unified Process (RUP)* [84], *Model-Based Archi-*

tecting and Software Engineering (MBASE) [27], and PORE [127]. The goal is to complement and extend these methodologies in order to provide a requirements engineering methodology that is agent- and goal-oriented, and explicitly support the definition and selection of COTS from a technical view. It assists the requirements engineer with the challenging tasks of defining goals, matching, ranking, and selecting potential COTS components, and negotiating changes to the components and/or the system under development. The CARE approach emphasizes the importance to keep requirements flexible since they have to be constrained by the capabilities of available COTS components.

Figure 3.11 shows an overview of the CARE process. When defining the system goals, the CARE approach considers the identification, definition, and interactions among agents, or stakeholders for a system. Once the agents are identified, their goals for the system need to be defined; these goals are used to drive the system development. Goals are high-level objectives of the system. They may be functional (hardgoals) or non-functional (softgoals). The CARE approach uses the i* framework [170] (see Section 2.2.1) for modelling agents, hardgoal and softgoal dependencies among agents, and how agents accomplish a goal. When defining the system goals, the requirements engineer would usually go through successive goal refinements. Such refinements can take the form of decomposition of goals into subgoals, identification of the conflicting and synergistic relationships among goals, and negotiation of the conflicts discovered. The CARE approach adopts the NFR framework for supporting the goal refinements. Moreover, the CARE approach uses a knowledge base (repository) that is populated with description of components. The descriptions of the COTS components are stored and maintained at two levels of abstraction: their goals and their detailed specifications. The goals provide high-level description of the functional and non-functional capabilities of a component. On the basis of the component descriptions, searches are enabled to determine which components appear to be potentially useful. Although the approach points out the importance of mapping system requirements and components specification, it does not provide or suggest any systematic solution to support the possible mismatching between them.

3.3.3 Mismatch-Handling Aware COTS Selection

The *Mismatch-Handling Aware COTS Selection (MiHOS)* [118] proposes a structured way to handle the COTS mismatches during and after the selection process. This approach supports decision makers in dealing with two key issues:

- Handling COTS mismatches during and after the selection process. This means analysing the mismatches, and then making appropriate decision about their resolutions;

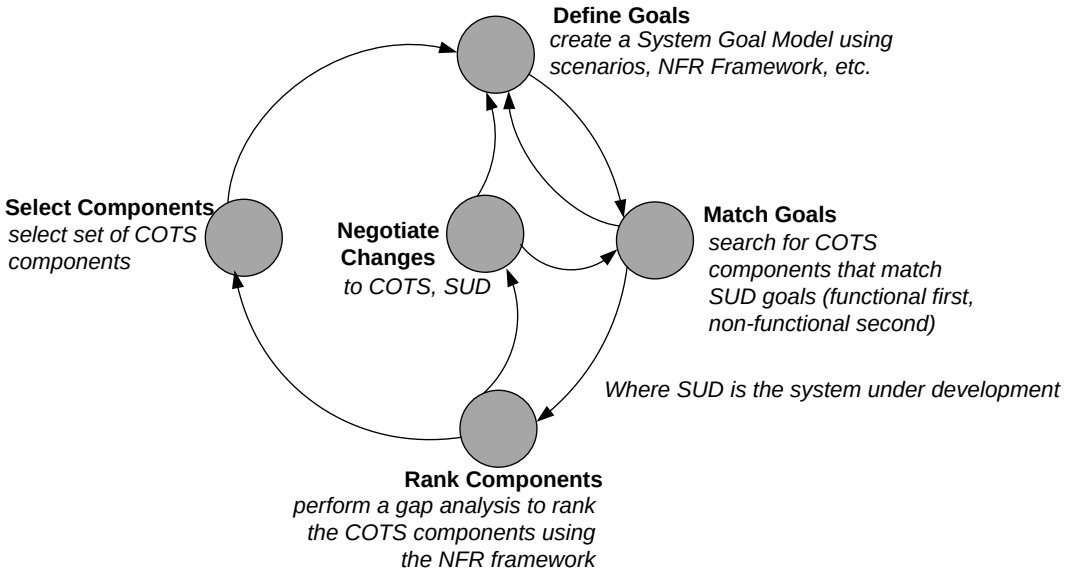


Figure 3.11: An overview of the CARE process (from [52]).

- Estimating the anticipated fitness of COTS products if their mismatches are resolved under limited resources. This helps to select COTS products based on the fitness they will eventually have, if they are selected.

The MiHOS approach is composed of two main parts:

- **MiHOS.Select** refers to the actual COTS selection process. It includes six steps: Steps 1 to 4 constitute the *DuringEvaluation* phase, and Steps 5 and 6 constitute the *AfterEvaluation* phase. Like most other COTS selection approaches, the MiHOS approach starts by defining a set of evaluation criteria. It uses a goal graph to represent the requirements whereby high-level requirements (*Strategic goals*) are decomposed into less abstract ones until reaching a level (*Technical goals*) at which their satisfaction can be directly judged in a COTS product. The MiHOS approach also suggests iterations between requirements acquisition and product selection;

- **MiHOS.Handle** refers the mismatch handling component. It includes two main parts: Part (A) which should be considered in the *DuringEvaluation* phase, and Part (B) which should be considered in the *AfterEvaluation* phase. For mismatches that are considered in the *DuringEvaluation* phase, MiHOS suggests three possible actions to handle them: Resolve by requirements adjustment, Tolerate, or Postpone. Postponed mismatches are stored in a mismatches repository. In the *AfterEvalution* phase, the postponed mismatches

are analysed by a special component called IDS (Iterative Decision Support for mismatch handling). The output from IDS includes: 1) the anticipated fitness of COTS candidates, which is used at Step 5 of MiHOS.Select to select the best-fit COTS, and 2) mismatch resolution plans, which are used at Step 6 of MiHOS.Select to resolve the right set of mismatches using the right resolution actions.

The main limitation of this method is that it focuses only on selecting a single COTS software product since it does not address the interoperability problem between COTS products as well as architectural mismatches. Furthermore, it focuses more on functional requirements than the non-functional ones.

3.3.4 Multiple COTS Selection

The COTS selection approaches that we have described above are suitable only for single COTS selection. However, as argued in [154], COTS components are designed to meet the needs of a marketplace rather than to satisfy the requirements of a particular organization. Consequently, there is no single component satisfying all the defined requirements. Hence, it is necessary to have adequate methodologies for multiple COTS selection. In this context, the *Combined-Selection (CS)* [39] is a COTS selection approach aiming to support the multiple COTS selection. It suggests to perform the activities related to COTS selection at two levels: global and local. The global level is responsible of selecting the set of COTS products that will be used to build a COTS-intensive system. It supervises the evolution, and synchronizes the results, of the different individual selection processes for each area. Its main objective is to find the best overall combination of products. The local level uses existing COTS evaluation and selection techniques (e.g OTSO [98] or PORE [127]) to select individual COTS products that are combined at the global level.

The Combined-Selection approach composes of the following main steps:

- Plan the combined selection process and enactment of the individual selection processes (global level);
- Identify COTS candidates for individual areas (local level);
- Identify global COTS integration scenarios (global level);
- Evaluate individual scenarios at each individual area (local level);
- Evaluate integration scenarios (global level);
- Select the COTS products (global level).

This approach contributes to the improvement of the COTS-based system development by proposing a structured way for managing the multiple COTS selections. However, it proposes only a high-level process without recommending techniques for use in the process such as identifying different COTS components needed, and conflicts between the selected COTS components. Moreover, by using the existing COTS selection approaches for the local level, the approach inherits the weaknesses of the method used at this level.

3.3.5 Social-Technical Approach to COTS Evaluation

Among the existing COTS selection approaches, the *Social-Technical Approach to COTS Evaluation (STACE)* [105] has been the first approach to emphasize the important of non-technical factors such as costs, business issues, vendor performance and reliability during the component evaluation process. Costs include direct costs, such as the price of the COTS components, and indirect costs, such as the cost of adapting to local needs as well as training costs. Business issues include people and process problems that must be overcome before successfully implementing the COTS-based system, such as as management support and internal organizational politics, staff skills and attitudes. Vendor performance and reliability includes vendor infrastructure and stability, vendor reputation, references, customer base and track record.

3.4 COTS Evaluation Strategies

COTS evaluation is one of the main steps of the COTS selection process; it determines the fitness of COTS products. The output of this step provides necessary information for supporting the COTS selection decision making. In literature, there are three strategies that can be followed to conduct COTS evaluation [106][129]:

1. **Progressive filtering** represents a strategy whereby a COTS product is selected from a relatively large number of COTS products. Then, progressively more discriminating evaluation mechanisms are applied in order to eliminate less fitting products;

2. **Puzzle assembly** represents a strategy which begins with the promise that a COTS-based system requires to assemble various components together as pieces of puzzles (e.g. COTS-intensive systems). This implies that a product that fits in isolation might not be acceptable when combined with other products. Therefore, this strategy suggests simultaneously considering the requirements of all products in the puzzle;

3. **Keystone identification** represents a strategy which starts by identifying a set of key requirements and after, searches for products that satisfy these

requirements. This allows quick elimination of a large number of products that do not satisfy the key requirements.

As stated in [129], more than one of these three strategies may be employed in a COTS selection process. For example, keystone identification may be used first, then, at the second stage, progressive filtering. **In our research work, we apply these three strategies jointly.** Our methodology begins with puzzle assembly, then followed by keystone identification and progressive filtering. More precisely, our COTS selection process begins with the identification of all the system-level components that need to be integrated into the puzzle and the non-functional requirements and architectural constraints that concern every part of the system (puzzle assembly). These non-functional requirements and architectural constraints will be used to evaluate the COTS component candidates in order to ensure that the selected components can be easily integrated into the system. Then for each required system-level component we define first the key requirements in order to quickly eliminate a large number of COTS component candidates that do not fulfill the key requirements (keystone identification). The pre-selected candidates will be progressively filtered until we get a COTS component that best meet the requirements (progressive filtering).

3.5 Decision Making Techniques for COTS Selection

In order to select a suitable COTS component, each candidate should be ranked on how well it fits users' requirements. Decision making techniques have been used in existing COTS selection methods for this purpose. In this section, we overview some decision making techniques that are commonly used in COTS selection processes.

3.5.1 Weighted Score Method or Weighted Average Sum

The *Weighted Score Method (WSM)* or *Weighted Average Sum (WAS)* is an aggregation technique and the most commonly used technique in many decision making situations. The WSM/WAS technique calculates the overall fitness for each component against the evaluation criteria using the formula:

$$OverallScore_i = \sum_{j=1}^{n}(weight_j * score_{ij}) \text{ for } i = 1, 2, \ldots, m$$

Where m is the number of alternatives, n is the number of criteria, $weight_j$ is the weight of the j^{th} criterion, and $score_{ij}$ is the fitness score of i^{th} COTS component in terms of the j^{th} criterion.

The weight of each criterion is assigned by decision makers in accordance to its importance and the fitness score represents the compliance of the component candidate with a specific criterion. Table 3.3 shows an example of the application of

the method for the COTS selection. In the example, three components are evaluated against five criteria. The weights of criteria are rated on a 9-point scale:

- 1 indicates a criterion that is unimportant;
- 9 indicates a criterion that is extremely important;
- Any value between 1 and 9 indicates intermediate levels of importance.

In addition, the fitness scores are estimated and assigned value from 0 to 1; 0 indicates no satisfaction and 1 indicates full satisfaction of the criterion by the component candidate. The overall scores are calculated using the equation above. In the WSM/WAS method, the overall scores represent the ranking of the alternatives but the differences between these scores do not indicate the relative superiority of these alternatives.

Table 3.3: An example of WSM/WAS.

Criteria	**Weight**	**Comp. A**	**Comp. B**	**Comp. C**
C1	7	0.6	0.8	1
C2	9	0.7	0.8	0.5
C3	5	0.9	0.7	0.4
C4	7	0.6	0.8	0.6
Total score		**19.2**	**21.9**	**17.7**

The application of the WSM/WAS is straight-forward and presents results that intuitively make sense. However, as stated in [126], the WSM/WAS has several limitations. First, as the WSM/WAS produces real numbers as results, these are easily interpreted as if they represented truly the differences between alternatives in ratio or distance scales. This would be true only if all the criteria weights have been given using distance or ration scale, and this is rarely the case [99]. Moreover, consolidating evaluation results into a single score is sometimes misleading because a high score in one criterion will hide a poor performance in another. Second, estimating the weight is difficult when the number of criteria is high.

3.5.2 Analytic Hierarchy Process (AHP)

The AHP method [144] is based on the idea of decomposing a multiple criteria decision making problem into a criteria hierarchy as depicted in Figure 3.12. At each level in the hierarchy the relative importance of criteria is assessed by comparing them in pairs. [144] introduced a 9-point intensity scale for indicating the level of relative importance which can be used in this comparison. The results of the comparison are then converted into normalized rankings using an *eigenvalue* technique

(see [144]) on the comparison matrix. The normalized rankings represent the weights of the compared criteria. Finally, the alternatives are similarly compared in pairs with respect to the criteria. The technique suggests that comparing criteria in pairs results in more reliable comparison results so that it is possible to avoid the problem of having to assign absolute value to alternatives. Only their relative preferences or values are in that way compared.

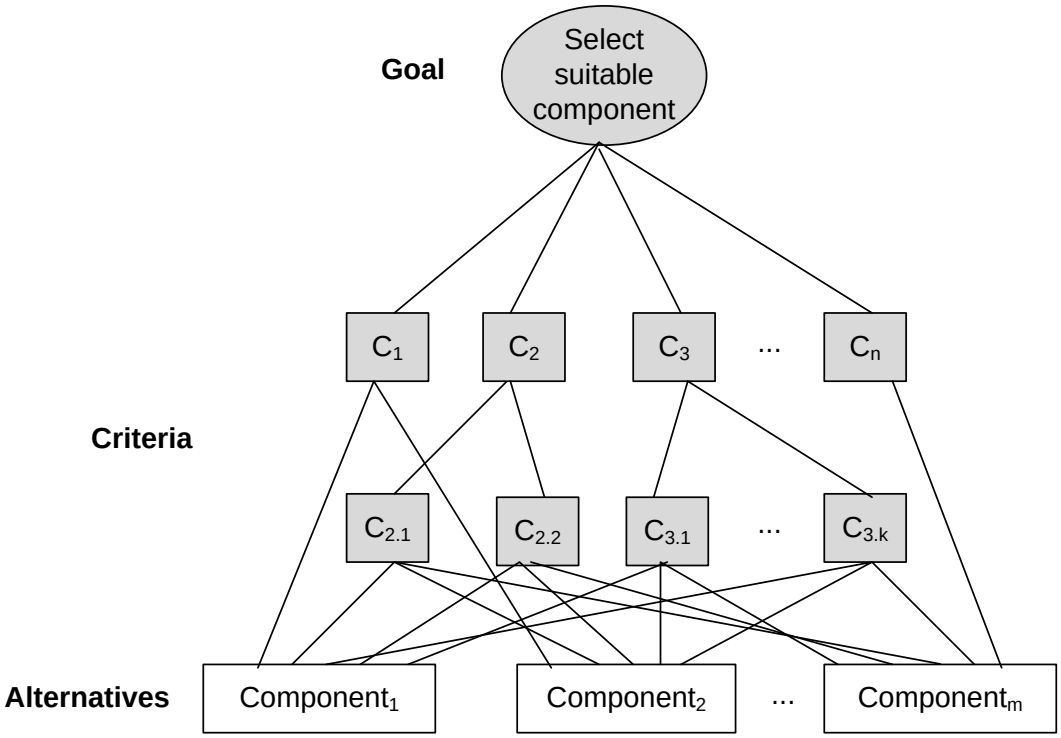


Figure 3.12: An example of AHP.

Table 3.4 shows an example of applying AHP to weight four criteria at one level of the hierarchy. A pair-wise comparison is performed among the criteria, and the results are represented using the 9-points scale. The resultant normalized ranking (i.e. weight) of each criterion is listed in the Priority column of Table 3.4. In this example, C2 is the most important criterion among the four criteria.

Saaty argues that hierarchies are a natural way for humans to organize their view of the world and they represent real world phenomena [144]. Criteria and alternative are compared in pairs, which results in more reliable comparisons results. This way it avoids the problem of having to assign absolute values to alternatives.

However, according to [126], the AHP method has two key limitations. First, AHP assumes independent criteria, which is rarely the case in real situations. Sec-

Table 3.4: An example of AHP.

	C1	**C2**	**C3**	**C4**	**Total Scores**	**Priority**
C1	1	4	5	4	14	0.357
C2	1/4	1	8	8	17.25	0.44
C3	1/5	1/8	1	5	6.325	0.161
C4	1/4	1/8	1/5	1	1.575	0.04
					39.15	**1**

ond, AHP involves many pair-wise comparisons, which would require a large amount of effort and time for a large number of criteria.

3.5.3 Gap Analysis Approach

Ncube and Dean present in [126] the limitations of current decision-making techniques including the two commonly used techniques described above in the evaluation of COTS software components. Consequently, they proposed an alternative approach that can be considered as requirements-driven and that migrates the loss of detail encountered when employing a weighted aggregation approach. The approach applies the principle of gap analysis to evaluation and allows selection based on the cost of bridging the gap. This requires a novel viewpoint where we examine products, not for the purpose of eliminating them if they do not meet the requirements, but for the purpose to attempting to determine what capabilities the components lack in terms of requirements. We then analyze these capabilities deficiencies to determine the cost of implementing supplementary functionalities for each component (or component set under evaluation). These are the fulfillment costs and there will be a set of these for each evaluation that we undertake. The *fulfillment costs* are used as a basis for selection of an appropriate product set.

Table 3.5 shows an example of gap analysis evaluation. In the example, three components are evaluated against five requirements. The information about the gaps between the evaluated components and requirements are recorded in the table.

During the gap analysis, there are three potential results:

1. the capabilities of the product and the requirements match exactly;
2. the component partially fulfills the requirements and does not provide any inherent capabilities that exceed the requirements;
3. the component fulfills some or all of the requirements but also incorporates capabilities that fall outside the boundaries of the original system's needs.

Table 3.5: An example of gap analysis evaluation matrix (from [126]).

Requirement	Component 1	Component 2	Component 3
R1	Limited Java support	Complete solution	Complete solution
R2	Inaccurate math	Precision only to 2 decimals	No math engine
R3	10% less than required reliability	No reliability figures available	Complete solution
R4	Complete solution	Vendor out of country	Vendor Canadian
R5	Complete solution	Linux platform required	Windows only

Once the gaps between components and requirements have been determined, a further step is required to establish the cost of reducing the gap to an acceptable solution which depends on then nature of the gap:

1. In the case of an exact match between requirements and products' capabilities, the cost of gap reduction is null;

2. In the case where a product does not fully meet a requirement and that requirement is firm – i.e. it cannot be restated or relaxed – then the solution is to determine the cost of adding functionally to cover the component's deficiencies with respect to the requirement;

3. In the case the component does not fully meet a requirement but that requirement is less rigid, one might negotiate a change in the requirement so that the product's capabilities and the requirement match more closely. The most common situation is that by combining requirements' adjustment and adding functionality the deficiency can be addressed. The cost of employing this product would then be calculated on the basis of both the cost of negotiation and the cost of adding functionality;

4. In the case where the component's capabilities fall outside the boundaries established by the known requirements, we have the situation where (i) we either accept the excess capability, (i.e. the capability is a benefit) and provide it as a part of the system, or (ii) we must attempt to inhibit access to that capability (i.e. the capability is a liability) from within the system. Indeed, there are various degrees of acceptance or rejection that can be negotiated as with the previous case. The costs in this case are calculated from the cost of custom coding for hiding the unwanted capabilities, the cost of adding beneficial functionality and the cost of adjusting the requirements.

3.6 Software Project Management for CBSD

A software project has two main activity dimensions: engineering and project management. The engineering dimension deals with building the system and focuses on issues such as *how to design*, *test*, *code*, and so on. The project management dimension deals with properly planning and controlling the engineering activities to meet project goals for cost, schedule, and quality. Software project management can be cut into a series of disciplines [135], among those we focus on: *Effort estimation*, *Quality management*, *Risk management* and *Change management*.

3.6.1 Effort Estimation Model

Accurate estimates of the cost and time for software projects are crucial for better project planning, monitoring, and control [114]. Project managers usually stress the importance of improving estimation accuracy and techniques to support better estimates.

Classically, the principal components of project costs are: *total cost of ownership*, *expertise and training costs*, and *effort costs* (the costs of paying the project team) [119]. The dominant one is the effort cost and unfortunately it is the most difficult to estimate and control [34]. Moreover it has the most significant effect on the overall costs since effort cost models are concerned with the estimation of effort in staff/month, size of the systems, duration, productivity and resources to assign to the project.

Typically, effort estimations are made up at minimum to get an idea of how long the project will last and how much resources are needed to complete it in time. There are several ways of estimating the effort. One way is to make a "guesstimation" by making qualified people doing it following techniques such as expert panels, Delphi or wideband Delphi [41].

Another way is to use models such as regression models to help calculate it. Several estimating models have been developed over the years such as the *Use Case Points (UCP)* [11], the Function Point Analysis [5], the *Constructive Cost Model (COCOMO)* [30], and the *SLIM* model [137].

The COCOMO model is one of the mostly used model and it was originally published in 1981 (COCOMO 81). By the mid-1990s, software engineering practices had changed sufficiently to motivate a new version called COCOMO II, plus a number of complementary models addressing special needs of the software estimation community [34]. COCOTS is an extension of COCOMO model for CBSD.

One of the key features of COCOTS is that estimates are done based upon the *classes* of COTS components being examined. COTS components can usually be sorted by basic function such as GUI builders, operating system, database, etc. The authors found that grouping COTS components into classes is the most effective way to gather calibration data [34].

COCOTS is composed of four related sub-models, each one designed to capture a different element of the total cost of using COTS components in building new systems. These sub-models are:

1. Assessment sub-model is for estimating the effort on candidate COTS component assessment;

2. Tailoring sub-model is for estimating the effort on COTS component tailoring;

3. Glue Code sub-model is for estimating the effort on the development and testing any integration or "glue" code needed to plug a COTS component into a larger system;

4. Volatility sub-model is for estimating the effort on increased system level programming due to volatility in incorporated COTS components.

3.6.2 Quality Management

The aim of quality management is to ensure that the quality expected and contracted with clients is achieved throughout the project. For this purpose, software quality models have been widely used for assessing software quality during software system developments.

According to [82], *"a quality model is the set of characteristics and the relationships between them which provide the basis for specifying quality requirements and evaluating quality"*. Several quality models have been proposed in the literature, e.g. McCall's Quality Model [112], ISO 9126 Quality Model [81], Boehm's Quality Model [33, 32], Dromey's Quality Model [62] and FURPS Quality Model [72]. These models are intended to evaluate the quality of software in general; none of them is dedicated to COTS-based systems. [23] argued that it is because they include characteristics that are not necessarily applicable to COTS components. In this sense, [23, 145, 139, 94] proposed quality models specific for COTS components.

[23] provide a quality model that can be used by the software architecture and designers to evaluate the available COTS components to be integrated into the system they are developing. It follows the ISO 9126 quality model [81] which is composed of two layers; characteristic layer and sub-characteristic layer. A characteristic is further refined into multiple sub-characteristics, and each sub-characteristic has a set of associated *metrics*, where a *metric* has a formula used to compute the metric value.

Table 3.7 shows the quality model proposed in [23]. It is basically the ISO 9126 quality model (see Table 3.6), where the *Portability* characteristic disappears as well as the *Fault tolerance*, *Stability* and *Analyzability* sub-characteristics. Two new sub-characteristics are added: *Compatibility* and *Complexity*. Moreover, other sub-characteristics (shown in bold) have changed their meaning in this new context. The

Table 3.6: ISO 9126 quality characteristics (from [81]).

Characteristics	**Sub-characteristics**
Functionality	Suitability Accuracy Interoperability Compliance Security
Reliability	Maturity Recoverability Fault tolerance
Usability	Learnability Understandability Operability
Efficiency	Time behavior Resource behaviour
Maintainability	Stability Analyzability Changeability Testability
Portability	Installability Conformance Replaceability Adaptability

authors also stated that it is important to specify the moment in which a characteristic can be observed or measured. According to [23], there are some characteristics observable at runtime (e.g. Performance) and others observable during the product life cycle (e.g. Maintainability). They also proposed the quality *attributes* for measuring these characteristics, where an *attribute* is a quality property to which a metric can be assigned, as shown in Table 3.8 and Table 3.9). The metrics that will be used for measuring these quality attributes are:

- **Presence.** This metric describes if an attribute is present in a component or not;
- **Time.** This metric is used to measure time intervals;
- **Level.** This metric is used to indicate a degree of effort, ability, etc. It is described by an integer variable that can take any of the following values: 0 (Very low), 1 (Low), 2 (Medium), 3 (High), 4 (Very high);

Table 3.7: Quality model for COTS components (from [23]).

Characteristics	**Sub-characteristics (Runtime)**	**Sub-characteristics (Life cycle)**
Functionality	Accuracy Security	Suitability Interoperability Compliance *Compatibility*
Reliability	Recoverability	Maturity
Usability		**Learnability** **Understandability** **Operability** *Complexity*
Efficiency	Time behavior Resource behavior	
Maintainability		Changeability Testability

- **Ratio.** This metric is used to describe percentages.

Similarly, [145, 139, 89] also proposed quality models for COTS components on the basis of the ISO 9126 quality model. However, one of the main drawbacks of these models is that they are based on ISO 9126 which is now out of date. The IS0 25000 (SQuaRE) [83] quality model is the successor of the ISO 9126 which is being broadly used. It has broadened the system quality from 6 characteristics (in ISO 9126) to 8, incorporating the same quality characteristics with some amendments.

Since the ISO 25000 is the current quality standard and it has quality factors and quality measures that do more justice to end user quality evaluation requirements, the authors of [89] has upgraded their quality model "Q'Facto 10" (based the ISO 9126) to "Q'Facto 12" [90] on the basis of the ISO 25000. Table 3.10 shows the Q'Facto 12 COTS component quality model.

Table 3.10: The Q'Facto 12 COTS component quality model (from [90]).

Quality factor	**Quality criteria**
QF1 - Functionality	Presence of preconditions and postconditions Modularity
	continued on next page

continued from previous page

Quality factor	**Quality criteria**
	Presence of standardization
	Presence of certification
	Correctness
QF2 - Security	Access controllability
	Data encryption
QF3 - Interoperability	Data compatibility
	Version compatibility
	Software compatibility
QF4 - Reliability	Persistence
	Presence of fault tolerant mechanism
QF5 - Efficiency	Disk capacity
	Response time
QF6 - Maintainability	Component stability
	Migration ease level
QF7 - Portability	Installability documentation
	Installability complexity
	Deployability documentation
	Deployability complexity
	Mobility
	Replacement ease level
QF8 - Testability	Test suit documentation
	Proofs of previous tests
	Component execution control
	Component environment control
	Component function feature control
	Performance trace
	Error trace
QF9 - Reusability	Presence of domain abstraction
	History of reuse
	Presence of hardware independence
	Presence of software independence
	Accessibility
QF10 - Usability	Training
	Presence of demonstration
	Presence of context sensitive help
	Effort to configure
	Understandability

continued on next page

continued from previous page

Quality factor	**Quality criteria**
QF11 - Usability in use	Customer satisfaction level Effectiveness level
QF12 - Safety in use	Software risk level Commercial risk level Operator risk level Public risk level

On the other hand, [94] define a quality model called *C-QM* for evaluating COTS components based on the characteristics of COTS components. C-QM consists of four factors as high level quality attributes and each factor has a set of quality criteria (see Table 3.11).

These quality models can be used to help defining the non-functional requirements during the COTS selection phase.

3.6.3 Risk Management

Risks are factors that may adversely affect a project, unless project managers take appropriate countermeasures. According [114], risk is a major killer of projects and as such it needs to be managed and controlled during the project life cycle.

Following [114], there are five basic phases within the risk management cycle:

1. Identify that a risk exists;
2. Analyze the severity of the risk;
3. Plan to handle the risk based on its impact and occurrence probability;
4. Mitigate the risk;
5. Track the risk. Once the risk has been mitigated to an acceptable severity level, the risk should be tracked to ensure the continued control of the risk.

As stated in [102], risk management must be a continuous activity because it is seldom that we can identify all risks at the beginning of the project. New risks can be identified continuously. Moreover, the impact and occurrence probability of risks can be evolved over time so that they need to be re-evaluated continuously.

In addition to the classical risks that exist with developing large scale systems, the use of COTS components requires managers to modify their typical mitigation strategies for some of the classic risks and develop new mitigation strategies for risks that are particular to CBSD [141].

Table 3.8: Quality attributes for COTS components measurable at runtime (from [23]).

Sub-characteristics	**Attribute**	**Type**
Accuracy	Precision	Ratio
	Computational accuracy	Ratio
Security	Data encryption	Presence
	Controllability	Presence
	Auditability	Presence
Recoverability	Serializable	Presence
	Persistent	Presence
	Transactional	Presence
	Error handling	Presence
Time behavior	Response time	Time
	Throughput	Integer
	Capacity	Integer
Resource behavior	Memory utilization	Integer
	Disk utilization	Integer

Several studies have been conducted on risk management of CBSD [138, 141, 29, 101]. Notably, [29] summarized lessons learned from CBSD university projects and proposed corresponding risk-management strategies (see Table 3.12).

Table 3.9: Quality attributes for COTS components measurable during life cycle (from [23]).

Sub-characteristics	Attribute	Type
Suitability	Coverage	Ratio
	Excess	Ratio
	Service implementation coverage	Ratio
Interoperability	Data compatibility	Presence
Compliance	Standardization	Presence
	Certification	Presence
Compatibility	Backwards compatibility	Presence
Maturity	Volatility	Time
	Evolvability	Integer
	Failure removal	Integer
Learnability	Time to use	Time
	Time to configure	Time
	Time to admin	Time
	Time to expertise	Time
Understandability	User documentation	Level
	Help system	Level
	Computer documentation	Presence
	Training	Presence
	Demonstration coverage	Ratio
Operability	Effort for operating	Level
	Tailorability	Level
	Administrability	Level
Complexity	Provided interface	Integer
	Required interface	Integer
	Complexity ratio	Ratio
Changeability	Customizability	Integer
	Customizability ratio	Ratio
	Change control capability	Level
Testability	Start-up self-test	Presence
	Test suite provided	Presence

Table 3.12: Risks in CBSD (from [29]).

Risk	Common Mitigation Plan
R1. Requirement changes and mismatches.	Prototyping and business case analysis can help to estimate the effect of change and corresponding team effort and schedule needed. Win Win negotiation among all stakeholders must be maintained in each development phase.

continued on next page

continued from previous page

Risk	Common Mitigation Plan
R2. Many new non technical activities are introduced in the Software Engineering Process.	Stakeholders with Domain expertise, Evaluation expertise must be included in the evaluation process.
R3. Miss possible COTS candidates within the COTS selection process.	Stay as broad as possible when doing the initial searching for candidates.
R4. Too much time spent in assessment due to too many requirements and too many COTS candidates.	Identify the "showstopper" requirements and filter all the COTS candidates that do not meet these during the initial assessment and then proceed for a more detailed assessment with the remaining COTS candidates.
R5. Might not include all key aspects for establishing evaluation criteria set.	Involve experienced, knowledgeable stakeholders for reviewing evaluation criteria and weight distribution judgements.
R6. Introducing new COTS candidates is likely and require replanning.	Develop a contingency plan in cases of addition of a new COTS product. Identify the limits on schedule and budget while making the introduction.
R7. Faulty vendor claims may result in feature loss and/or significant delays.	Detailed analysis provides greater assurance of COTS characteristics with respect to vendor documentation. Detailed assessment beyond literature review or vendor provided documentation should be performed in the form of hands-on experiments and prototyping.
R8. Ability or willingness of the organization to accept the impact of COTS requirements.	The project operational concept must identify such risks and they must be conveyed to the higher management.
R9. Difficulty in coordinating meeting with key personnel may result in significant delays.	The key decision making personnel must be well accounted for the project life cycles. The project manager must make them aware of the approximate time required to be spent with them during the process of assessment etc. The decision making personnel must be kept as minimal as possible.

continued on next page

continued from previous page

Risk	Common Mitigation Plan
R10. Inadequate vendor support may result in significant project delays.	The licensing of COTS products must account for vendor support details. In case of contracting labour the developers with experience in using the COTS must be selected.
R11. COTS package incompatibilities may result in feature loss and significant project delays.	COTS integration issues must be considered during assessment. The number of COTS products must be kept as minimal as possible.
R12. Added complexity of unused COTS features.	The number of unused features could be identified and the added complexity because of the presence of such features must be calculated during COTS assessment.
R13. Overly optimistic expectations of COTS quality attributes.	Significant quality features must be tested before selecting COTS products. Special testing packages may be used. Evaluation could be carried out at sites where the COTS is actually being used.
R14. Overly optimistic COTS package learning curve.	A most likely COTS package learning curve must be accounted for during planning the schedule.
R15. A version upgrade may result in re-tailoring of COTS package.	Ensure that the features used to implement the capabilities still exist in the new version before the version upgrade.

3.6.4 Organizational Change Management

Organizational change is inevitable when working on information system development, especially with COTS-based system development. Organizational change management is another critical factor that must be addressed by the project manager [173].

According to [123], organizational change management is the process of developing a planned approach to change in an organization. The objective is typically to maximize the collective benefits for all people involved int the change and to minimize the risk of failure of implementing the change. This involves identifying organizational units and employees within the company that will be affected by the intended system, and ensuring that those employees understand and manage the company's perception of and responses to the changes that will occur after the implementation.

Affected employees may resist change introduced by the new system for the

Table 3.11: C-QM quality model (from [94]).

Factors	**Criteria**
Functionality	Commonalty Suitability Completeness
Reusability	Commonality Modularity Customizability Comprehensiveness
Maintainability	Modularity Interface abstractness Changeability
Conformance	Standard conformance Reference model conformance

several reasons [43]:

- The change is not communicated to the affected people;
- They do not understand the reasons;
- They do not see the benefit to them;
- They do not have the skills and knowledge to handle the change.

Preparing for organizational change will facilitate a smoother implementation of the system and for a more effective use of the system once in use. By establishing an organizational change management plan as part of the project management, we can expect the following benefits [43]:

- Less resistance to the new system and the new business processes;
- Fewer complaints and negative attitudes toward the new system;
- More positive attitudes from the end-users;
- More effective use of the new system.

Creating an effective organizational change management plan involves [123]:

1. Identifying the groups of institutional stakeholders that will be affected by the changes introduced by the new system;

2. Identifying how each group will be affected (e.g. new roles, structures, processes);

3. Identifying the new skills and behaviors that will be required by each group;

4. Determining the barriers, issues, and types of resistance that we could face with each group;

5. Defining change management interventions to handle these barriers and issues of each group and regularly reviewing and updating the plan.

3.7 Chapter Summary

This chapter has reviewed the state of the art on CBSD. Firstly, it presented the characteristics of COTS components by providing an overview of the different definitions of COTS and the different levels of component granularity.

Next, it described some relevant CBSD life cycle models including the sequential model, the V-Model, the Y-Model and the Evolutionary Process for Integrating COTS which is a modified from of RUP for CBSD.

Then, it introduced some representative COTS selection approaches on the basis of the different aspects of COTS selection that they have addressed. More precisely, it presented the OTSO COTS selection approach that has been considerer the first widespread COTS selection and that provides the basic structure of COTS selection methods; some relevant requirements-driven COTS selection approaches including PORE, CRE, and CARE; mismatch-handling aware COTS selection approach; multiple COTS selection approach; and social-technical approach to COTS evaluation.

This chapter also exposed some relevant approaches that have been used in CBSD including different evaluation strategies that have been used during the COTS selection, i.e. progressively filtering, puzzle assemble and keystone identification; the three main decision making techniques for COTS selection, i.e. weighted score method, analytic hierarchy process, and gap analysis approach; software project management for CBSD, i.e. risk management, quality management, effort estimation models, and organizational change management.

Part III

Architectural Design for COTS-Based System Development

Chapter 4

Architectural Foundations

In this chapter, we present the conceptual foundations of the proposed architectural design for CBSD. We firstly introduce the *commercial off-the-shelf (COTS)* component specification that we adopt in this thesis and that serves as a basis for our methodology. Then, we present the different characteristics of components integration and specify the characteristics adopted by our methodology. We demonstrate the advantages of using of agent-oriented approach for systems integration.

4.1 A Characterization for COTS Components

Among the different COTS definitions that we can find in the literature (see Section 3.1.1), in this thesis we follow the SEI definition [115]:

"A COTS product is a [software] product that is:

1. *sold, leased, or licensed to the general public;*
2. *offered by a vendor trying to profit from it;*
3. *supported and evolved by the vendor, who retains the intellectual property rights;*
4. *available in multiple, identical copies; and*
5. *used without source code modification by a consumer."*

Therefore, we consider a *COTS-Based System (CBS)* as a **computer based application that integrates one or more COTS products**, while CBSD as **the processes that lead to the development of a CBS**. Specifically, our research focuses on the development of large scale distributed business systems with

the use of COTS software components. The systems aimed to be built with the methodology developed in this thesis target **coarse-grained COTS components**. **In this sense, for the rest of this thesis, when referring to a COTS component, we mean a COTS system-level component, which corresponds to an information system** (see Section 3.1.2 for component granularity).

Since COTS components are bought from the marketplace and not developed in-house, COTS components' users normally do not know how each of the system-level components are built. Therefore, in our methodology, COTS components are not described from the point of view of their internal compositions but rather from their externals. According to [78], there are two main ways in which a system-level component can be connected to its externals: a black-box approach or a white-box approach.

- **Black box.** An interface to the system-level component is provided through a specific access artefact, of which there are two flavors: a *gateway* (see Figure 4.1) or an *interoperability adapter* (see Figure 4.2). The term *gateway* is used to identify an interface implemented at development-time, while an *interoperability adapter* is the implementation of an interface after having deployed the system. An interoperability adapter may be specified and implemented by an organization different from the one that developed the original system. A gateway will totally hide internals of the system, and so a client of the gateway will be unable to determine whether the system is composed from business components or built as a monolithic system;

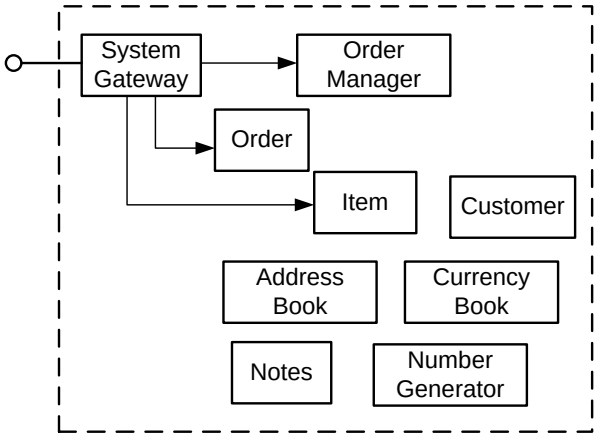


Figure 4.1: Example of a system interface provided by a gateway (from [78]).

- **White box.** An effective interface to the system-level component is provided by allowing direct access to its constituent lower level components (business component). This is done by exporting a selected set of business component

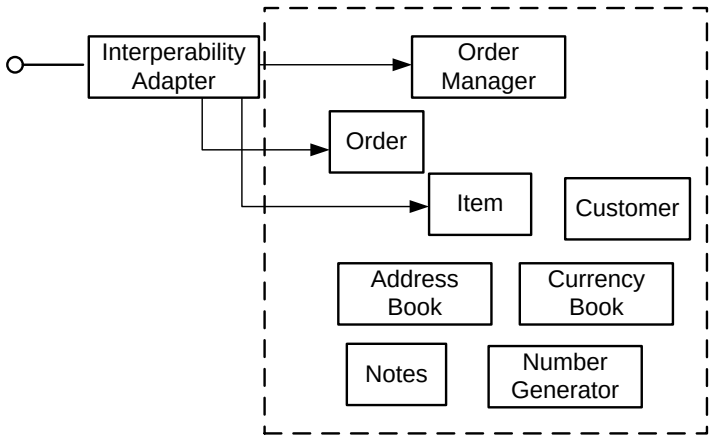


Figure 4.2: Example of a system interface provided by an interoperability adapter (from [78]).

interfaces so as to give them system-external visibility. Such interface will normally require a set of additional constraints to be imposed on them for security and other reasons. This may, in turn, demand specific system-external business component interfaces to be specified and implemented. In Figure 4.3 such constrained interfaces are indicated by the letter "C".

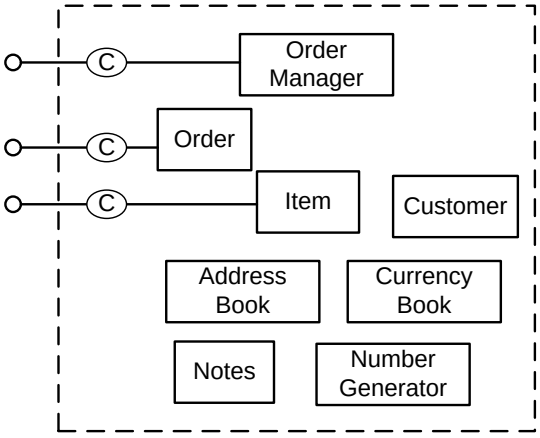


Figure 4.3: Example of exporting constrained business component interfaces (from [78]).

4.2 An Ontology for COTS Component Representation

We have defined a component metat-model that illustrates the relevant elements of a COTS component and their relationships using a UML class diagram. This

model is adapted from the component model defined in [56] for representing the fine-grained component (i.e. distributed component), which focuses only on the technical descriptions of a component (see Figure 4.4).

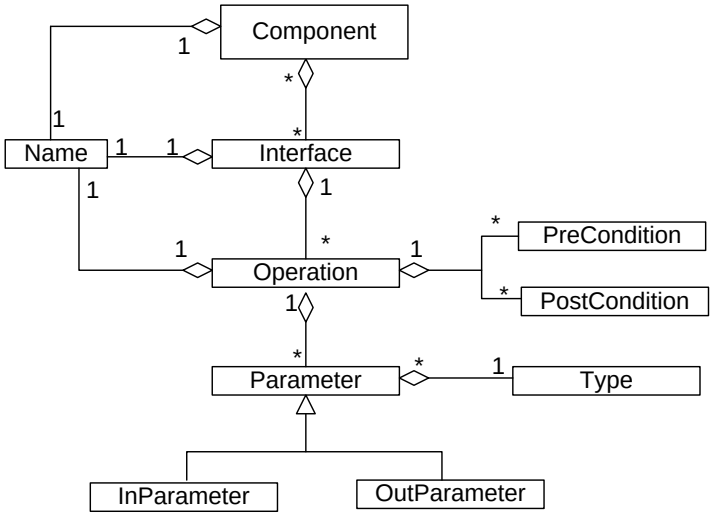


Figure 4.4: The component meta-model (from [56]).

As evoked earlier, we focus on the coarse-grained COTS components (i.e. system-level component) and the ontology for describing the COTS component is defined from the point of view of its external. We suppose **each component to be provided with its gateway**. Figure 4.5 depicts our component meta-model which is structured as follows:

- Each component possesses *general information* that allows the customers to identify and make a selection of the available components in the marketplace. This general information includes *information attributes* like the *product name*, *category*, *domain*, *type*, *version*, *vendor*, *cost*, and *programming language*;
- Each component is associated with a set of *features* that it provides. *Features*, based on the definition found in [91], are any *prominent and distinctive characteristics of a system that are visible to users.* A component user normally selects a component according to how well the features provided by the component meet the requirements;
- Each component will be also evaluated according to its *quality* which is described through a set of *quality characteristics*;
- Each quality characteristic is measured through its *quality attributes*;
- Each component is connected to the external environment through a *system gateway*;

- A *system gateway* consists of a set of named *interfaces*;
- An *interface* is composed of a set of named *operations*;
- Each *operation* has zero or more *input* and *output parameters* and a syntactic specification associates a *data type* with each of these parameters;
- Each *operation* is also associated with a set of *preconditions* and *postconditions*. *Preconditions* are assertions that the component assumes to be fulfilled before an operation is invoked. *Postconditions* are assertions that the component guarantees will hold just after an operation has been invoked, provided the operation's preconditions were true when it was invoked.

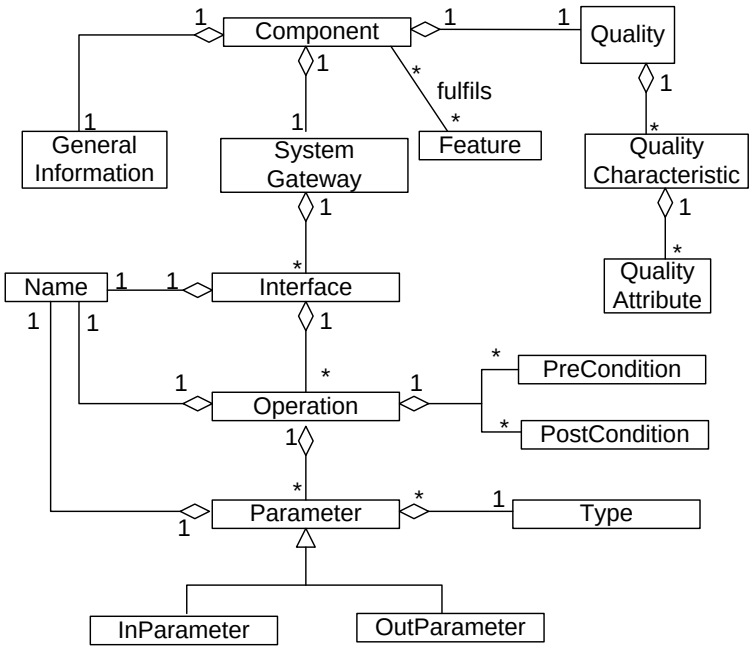


Figure 4.5: Our component meta-model.

In our methodology the COTS components are described at two levels of abstraction. *General Information*, *Feature* and *Quality* provide high level descriptions of a component, which are necessary during the COTS selection phase. On the other hand, *System Gateway* provides technical descriptions of a component, which are useful during the COTS integration phase. During the development process, the project team will seek for information about each COTS component according to this ontology.

4.3 Components Integration: Definition and Characteristics

According to [78], system-level components to be integrated can be:

1. Two independently developed systems that do not allow any internal modification and that must have a minimal set of contact points. In this case, whether the internal of the two systems are implemented in exactly the same way or in totally different languages and technology should not matter;

2. Two independently developed systems built using a common set of techniques, infrastructure, business components, application architecture, and also addressing common reduction in development costs, while still aiming to have a minimal set of contact points.

These two meanings for integration are two different models that achieve similar ends. **Our methodology is nevertheless based on the first model in which system-level components constituting the system under development may be bought from different vendors. These components interact with each other if necessary to address the information processing needs of the multiple end users.**

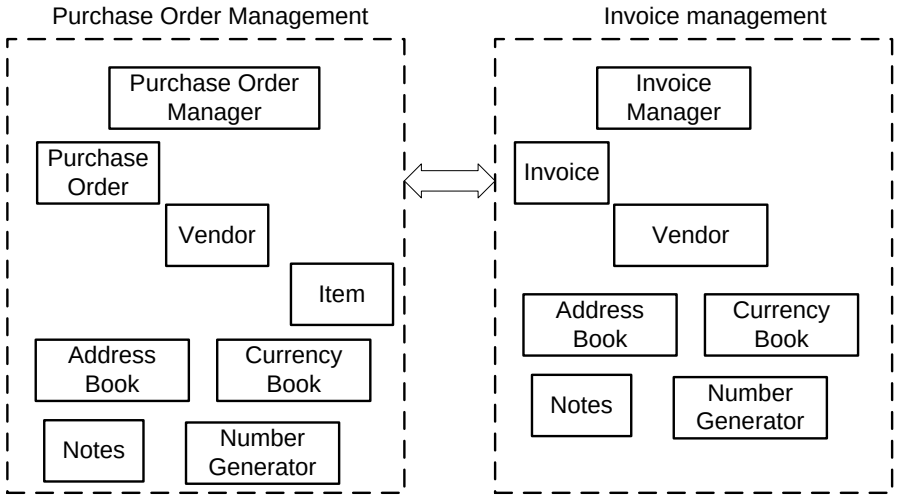


Figure 4.6: Example of an integrated system (from [78]).

The components constituting the system interact with each other according to a small number of styles. The three main interaction styles between system-level components are as follows [78]:

- **Master-slave collaboration.** In this style of collaboration, one system– the master– is clearly the initiator of all communication and the only active

partner, while the other system–the slave–is a passive partner and a service provider. The master initiates all communications and all requests to the slave. In Figure 4.7, the exchange between master-slave system-level components is shown as a unidirectional exchange. In reality, in most cases, there is a requirement for both systems to be able to initiate a communication;

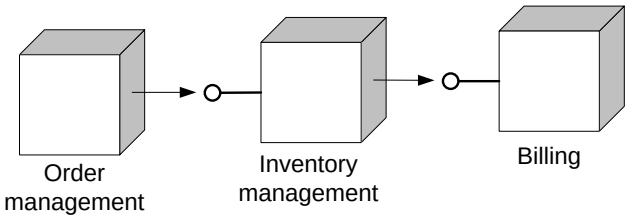


Figure 4.7: Example of a master-slave collaboration (from [78]).

- **Coordinated collaboration.** In this style of collaboration, illustrated in Figure 4.8, all exchange of information between two or more system-level components is done through a coordinating entity. The coordinating entity is called `Component Manager` in the figure;

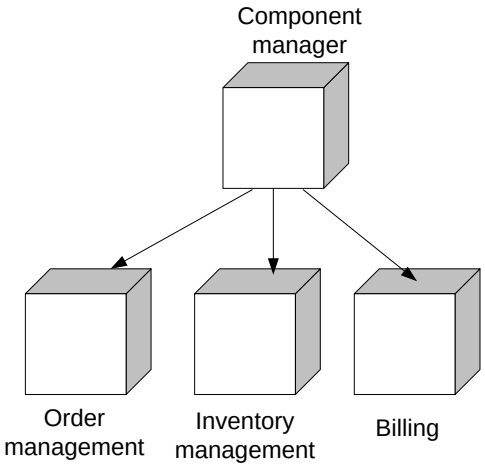


Figure 4.8: Example of a coordinated collaboration (from [78]).

- **Peer-to-peer collaboration.** In this style of collaboration, the participating system-level components can freely and directly invoke each other and initiate an exchange of information, as illustrated in Figure 4.9. This style enables maximum collaboration flexibility, but it is also the most complex and expensive to develop, test, deploy, manage at run-time, and maintain. Peer-to-peer collaboration is very similar to human collaboration and it is intellectually very appealing. By analyzing an ideal interaction between two systems, we

can deduce that the communication will often need to be bidirectional. Any of the two systems may need to receive information or ask for information from the other.

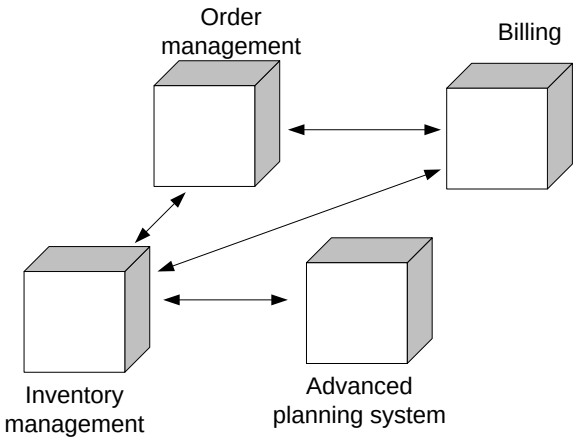


Figure 4.9: Example of a peer-to-peer collaboration (from [78]).

These three main interaction styles between system-level components may be used jointly in a complex system. Figure 4.10 shows an example where the three interaction styles are used:

1. The user may want to perceive the system constituting of a set of collaborating system-level components as a whole system through a single user interface, which is sometimes called a *federation desktop*. This is an example of coordinated communication;

2. At the same time, behind the scenes, the different systems may need to communicate directly to each other. This may be done using a peer-to-peer interaction style;

3. One or more systems may also need to contact a remote system for some point-to-point communication. In the figure, the order management system connects to a bank in order to update the organization's own accounts. This is an example of a master-slave interaction style.

Building such a complex system with the use of COTS components is the focus of our thesis. Hence, our methodology proposes a system architecture for implementing a large and complex system where the conditions mentioned above do exist. The proposed system architecture is described in the following chapter.

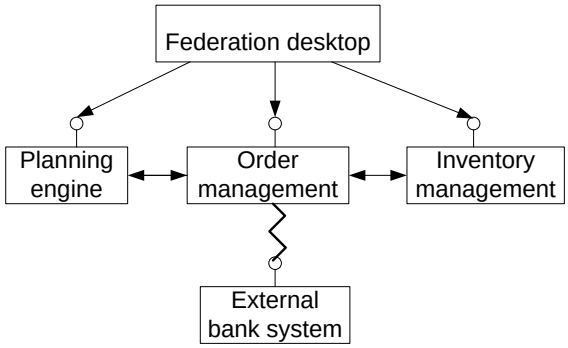


Figure 4.10: Example of a mixing styles (from [78]).

4.4 An Agent-Oriented Approach to Systems Integration

A popular integration approach is based on the object-oriented (OO) programming paradigm that can be traced back to the 1960s and has been used in various application domains for about two decades [13, 65, 66, 67, 88]. It emphasizes programming efficiency by stressing the modularity of data structures and code sharing. It uses a centralized or tightly- coupled integration approach. It has been widely used for systems integration, particularly after the development and deployment of three major distributed object standards: *Common Object Request Broker Architecture (CORBA)* by the Object Management Group [68], *Distributed Component Object Model (DCOM)* by Microsoft [58], and *Java/Remote Method Invocation (Java/RMI)* [61]. These OO frameworks provide well-designed component/object models, and integration mechanisms supporting interfaces to link components/objects together. However, one of the main drawbacks of these frameworks is the absence of dynamic allocation during the functional execution. Indeed, in these frameworks, the integrated components are statically bound, and the collaboration mode among them is fixed so that it cannot be adjusted and modified especially when the system is running. Therefore, these frameworks are unable to adapt to frequently changing requirements and environments.

With the emergence of agent technology, many researchers have been probing into agent-based solutions for systems integration and some reached the conclusion that the agent technology provides a natural way to realize integrative business information systems [95]. Moreover, as argued in [136], the use of agent technology within middle-ware has several advantages, i.e.:

1. A software agent has social ability [55]. An agent could communicate with human users and accept the delegated tasks. Furthermore, it is also a communicative program that interacts with other programs/agents in speech-acts

[156], which means communication similar to human talk. A complex task could be completed through the cooperation of software agents;

2. If necessary an agent can be mobile, with the ability to migrate from one host to another during its execution. From the distributed systems point of view, a mobile agent is a program with a unique identity that can move its code, data and state between networked machines. To achieve this, mobile agents are able to suspend their execution at any time and to continue once residing in another location [20]. Some studies [169] have shown that mobile agents could reduce the network load. The main reason for this is that mobile agents communicate with the applications locally and the communication links can be disconnected after migration;

3. A software agent with intelligent abilities is potentially suitable for handling sophisticated distributed computations. As argued in [86], it will be of benefit for engineering complex software systems using agent technology.

OO development is thus not the only efficient paradigm for constructing large-scale software systems. Software agents, which are actually intelligent software objects, could have better interaction ability than traditional objects and therefore suit for building distributed systems. This motivates us to apply agent technology for systems integration within our methodology.

4.5 Chapter Summary

This chapter has presented the conceptual foundations of the architectural design for CBSD proposed in this thesis. It introduced the COTS component specification that we adopt in this thesis. It serves as a basis for our methodology in the sense that each COTS component has to meet this specification. Then, it presented the different characteristics of systems integration and specified the characteristics adopted by our methodology. Finally, it described the reasons behind the use of agent-oriented approach in our methodology for COTS integration.

Chapter 5

An Agent-Driven Integration Architecture

This chapter focuses on our agent-based system architecture. Section 5.1 describes the proposed system architecture. Section 5.2 shows the possibility of implementing our proposed MAS using the JADE agent framework.

5.1 Integration Architecture

This section presents first the architectural layers of our integration architecture as well as the middleware composition. An architectural description of the MAS layer is then given.

5.1.1 Vertical Architectural Layers and Middleware Composition

Figure 5.1 presents the proposed system architecture. Logically, it composes of three vertical architectural[1] layers:

- *The Graphical User Interface (GUI) layer* is the top layer that provides users with a means to interact with the system. This GUI layer constitutes the *federation desktop* of the system. We propose to use the web technology to build it so that it can be accessed via browsers in a standardized way from anywhere;
- *The MAS layer* is the middle layer that is in charge of the functional decomposition of the client/user requests and manages the different interfacing aspects with COTS components;

[1]Note that in the context of this thesis unless explicitly specified, when we use to the term "architecture" we refer to the architecture of the MAS located in the middle layer not to the three layered vertical architecture.

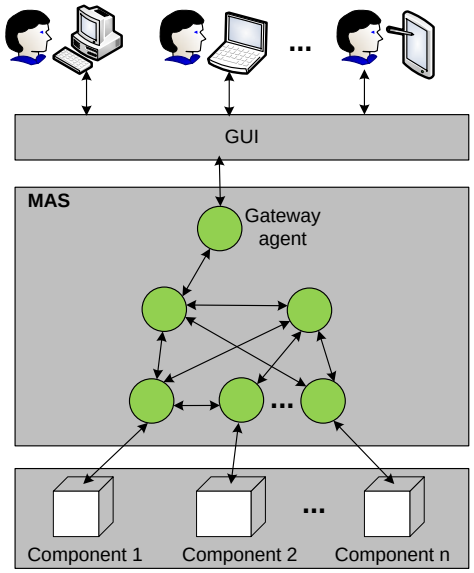


Figure 5.1: Vertical system architecture.

- *The Component layer* is the lowest-level layer that composes of a set of components capable to execute operations requested by agents. In the context of this thesis, agents can interface with COTS components through the components' gateways.

Specifically considering the MAS layer, we propose a MAS design compliant with the FIPA specification [149]. Figure 5.2 defines a meta-model of the main parts constituting our MAS in the form of a UML-based class diagram [151], those parts are:

- **Agent Platform (AP)**: Following FIPA specifications [149], the MAS needs a platform for efficient and stable interaction between intelligent agents. An AP provides the physical infrastructure in which agents are deployed. It consists of FIPA agent management components including the *Agent Management System(AMS)* and the *Directory Facilitator (DF)*, the agent themselves and any additional support software. A single AP may be spread across multiple computers, thus resident agents do not have to be co-located on the same host;
- **Agent Management System (AMS)**: The AMS is a FIPA agent management component. It is a mandatory component of an AP and is responsible for managing the operation of an AP, such as the creation and deletion of agents, and overseeing the migration of agents to and from the AP. Each agent must register with an AMS in order to obtain an *Agent Identifier (AID)* which is

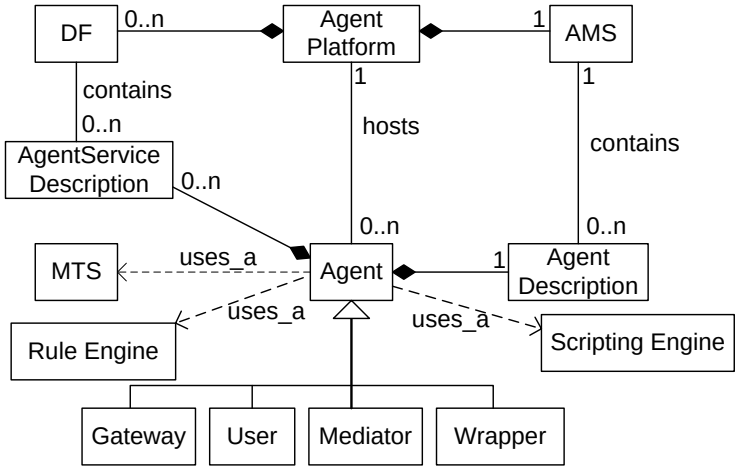


Figure 5.2: Meta-model of the main MAS parts.

then retained by the AMS as a directory of all agents present within the AP and their current state (e.g. *active*, *suspended* or *waiting*);

- **Directory Facilitator (DF)**: The DF is an optional component of an AP providing yellow pages services to other agents. It maintains an accurate, complete and timely list of agents and must provide the most current information about agents in its directory on a non-discriminatory basis to all authorized agents. An AP may support any number of DFs which may register with one another to form federations. The main functions of DF are to manage the service subscription and unsubscription of agents and respond to the service search requests;

- **Message Transport Service (MTS)**: The MTS is a service provided by an AP to transport FIPA *Agent Communication Language (ACL)* [149] messages between agents on any given AP and between agents on different APs;

- **Rule Engine**: The rule engine is used to evaluate and execute the rules associated with agents' adaptive capabilities. It is integrated into the AP in order to build adaptive agents;

- **Scripting Engine**: In our proposal, some parts of agents' capabilities especially the integration logics can be written in scripting language so that they can be changed dynamically. The scripting engine is integrated into the AP to execute these scripts;

- **Agent**: An agent is a computational process that inhabits an AP and typically offers one or more computational services that can be published as a service description. In our MAS, there are four types of agents:

- `Gateway` agent. This agent is the access point of the GUI layer to the MAS;
- `User` agents. Each user of the system has an user agent that represents him/her. `User` agents can autonomously perform actions that the users should do;
- `Wrapper` agents. Each component that constitutes the system is agentified into a `Wrapper` agent. The `Wrapper` agents manage the states of the components they are wrapped around, invoking them when necessary. The `Wrapper` agents are thus in charge of managing the different interfacing aspects with COTS components;
- `Mediator` agent. This agent is designated to accomplish the user requests sent from the `Gateway` agent. It realizes the request by invoking the relevant services offered by the agents which play the `Service provider` role and sends the response back to the `Gateway`.

The agents' internal structure follows an agent model (see Section 5.1.2) in which the agents' intelligence resides. This intelligence is issued of the following properties:

- *Autonomy.* Our agents can decide whether to perform an action on request from another agent. When an agent receives a request, it will indeed check whether it can reply to this. If it cannot, it will send a negative answer to the request sender;
- *Sociability.* Our agents can interact with users or other agents. They communicate with each other through message exchanges;
- *Pro-activity.* Our agents do not simply act in response to the received messages from the other agents. Each agent can take the initiative to perform a given task in order to respond to the changes that occur in its environment;
- *Adaptivity.* In our proposal, it is possible to create agents' capabilities which can be changed while the system is running. This allows them to adapt to its environment which is continuously evolving.

Conceptual foundations for implementing these properties are presented in the following section.

5.1.2 Agent Model

Figure 5.3 depicts the relevant elements of an agent and their dependencies using a UML class diagram. The model is structured as follows: an agent has an associated *MessageQueue*, a *BeliefBase*, a *ServiceBase*, and a set of *Capabilities*.

- *MessageQueue* and *ACLMessage*. The message queue is a sort of mailbox associated to every agent: it stores all the ACL messages sent by other agents and that have to be read by the agent. This allows agents to be sociable;
- *BeliefBase* and *Belief*. A belief base is a set of beliefs. Beliefs represent the agent's perception of the world, what the agent know about itself and the external environment. Agents can take actions according to their beliefs. This allows agents to be pro-active;
- *ServiceBase* and *Service*. A service base contains a set of services offered by the agent to the others. A service is an act or performance offered by one party to another. It is described in term of its properties, i.e. *type*, *name*, etc. With the *ServiceBase* and *BeliefBase*, agents can know and decide what to do with other agents' requests. This allows agents to be autonomous;
- *Capability*, *Operation* and *Component*. A capability represents a task that an agent can carry out. It composes of a set of operations to be executed with the aim of achieving a specific goal. Some operations are furnished by components;
- *Service* and *Capability*. Each agent service is associated with an agent capability but not vise-versa. Some agent capabilities do not need to be exposed to other agents, so they are not associated with any agent services;
- *Capability* and *Rule*. Some capabilities are based on the activation of different logical rules. These logical rules are written in a scripting language. Working with rules helps keeping the logic separated from the application code; it can be moved outside the code. Moreover, these rules can be adjusted and modified when the system is running. This allows agents to be adaptive.

In the system integration process, the communication and collaboration among agents enables the integration of components in the MAS whose architectural description is described into the next section.

5.1.3 MAS Architectural Description

This section aims to provide an architectural description of the MAS using i* [171] and *Agent-UML (AUML)* [69] models. Those will be structured into four complementary dimensions:

- The *social dimension* identifies the relevant agents in the system and their intentional interdependencies;
- The *rationale dimension* identifies the capabilities provided by agents to realize the intentions identified by the social dimension. This dimension describes what each capability does;

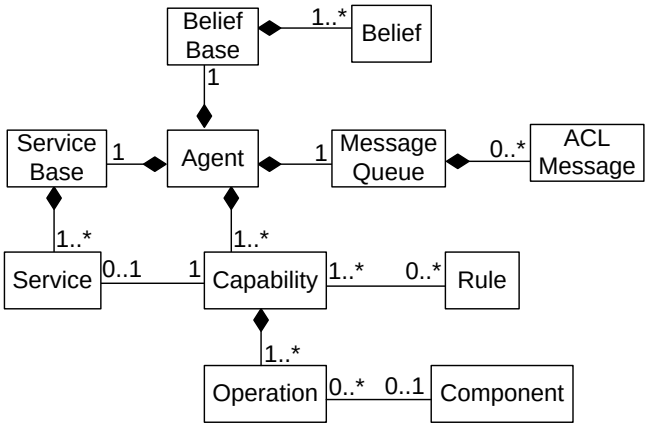


Figure 5.3: Agent model.

- The *communicational dimension* models the temporal message exchange between agents;

- The *dynamic dimension* models the operational (step-by-step) workflows of agents in the system.

The architectural description goes beyond a pure system design by incorporating a description over multiple complementary dimensions illustrating the different aspects of the MAS architecture for components integration. The description of the MAS architecture presented in the following sections is generic but can be adapted to a particular business logic issued of a project specific domain model.

5.1.3.1 Social Dimension

Figure 5.4 illustrates the social dimension of the proposed MAS through an high level view of its architecture using the i* framework. The `Gateway` agent depends on the `Mediator` agent to accomplish user requests. The `Gateway` agent will send user requests to the `Mediator` agent and wait for the responses from it.

In addition, as depicted in Figure 5.4, there are two roles of agents: `Service consumer` and `Service provider`. In our case, the `Mediator` and the `User` agents play the `Service consumer` role while the `Wrapper` agent can play both roles. This is because a `Wrapper` agent may need to invoke services provided by other agents. The `Service consumer` agents depend on the `Service provider` agents to accomplish the services they offer. The `Service consumer` agents will send requests to the `Service provider` agents and wait for the responses from them.

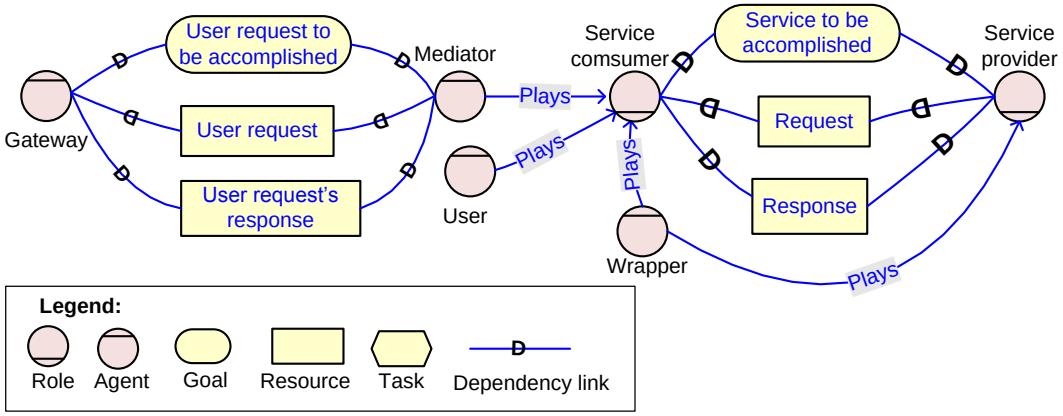


Figure 5.4: Social dimension of our MAS architecture.

5.1.3.2 Rationale Dimension

The rationale dimension aims at modeling agent reasoning. In this dimension, we identify capabilities of each agent that can be used to achieve the intentional dependencies. Capabilities are listed in Table 5.1. Each row presents a capability: its *name*, its *informal definition*, and the *agents* the capability belong to.

When receiving a user request from the GUI layer, the `Gateway` agent converts it into a ACL message with the `GetUserRequest` capability and sends it to the `Mediator` agent with `SendUserRequest` capability. The `Gateway` agent waits for the user request's response from the `Mediator` agent and send it to the GUI layer through the `GetUserRequestResult` and `SendUserRequestResult` capabilities.

`Mediator`, `User` and `Service provider` use the `GetRequest` capability to get the request from other agents. Upon the reception of a request, these agents analyze whether they can realize it with the `AnalyzeRequest` capability. If they can, they will load and execute the relevant integration script through the `RealizeRequest` capability. During the request realization, they may need to search for the relevant `Service provider` agents offering the services needed to accomplish the request with the `SearchServiceProvider` capability. If there is not any `Service provider` agent providing a service needed, they handle this negative answer. Otherwise, they send request to the `Service provider` agents and wait for their responses with the `SendRequest` and `GetSubResult` capabilities. They use the `SendResult` capability to send the request's response to the request sender agent.

`ServicesRegister`, `ServicesDeregister` and `ServicesUpdate` are three specific capabilities that belong to `Service provider` agent. Each `Service provider` agent uses these capabilities for registering and deregistering descriptions of services it offers with the `DF`.

Table 5.1: Capabilities of agents.

Capability Name	Informal Definition	Agent
GetUserRequest	Get the user request from the GUI layer	Gateway
SendUserRequest	Send the user request to the Mediator agent	Gateway
GetUserRequest-Result	Get the user request's result sent from the Mediator agent	Gateway
SendUserRequest-Result	Send the user request's result to the GUI layer	Gateway
GetRequest	Get the request sent from the other agents	Mediator, Service provider, User
AnalyzeRequest	Analyze the received request	Mediator, Service provider, User
RealizeRequest	Realize the request by loading and executing the relevant integration script	Mediator, Service provider, User
SearchService-Provider	Search for the corresponding Service provider agents	Mediator, Service provider, User
SendRequest	Send request to the Service provider agents	Mediator, Service provider, User
GetSubResult	Record result provided by the Service provider agents	Mediator, Service provider, User
SendResult	Send the request's response to the Gateway agent	Mediator, Service provider, User
ServicesRegister	Register descriptions of services that it offers with the DF	Service provider
ServicesDeregister	Deregister descriptions of services that it offers from DF	Service provider
ServicesUpdate	Update descriptions of services that it offers with the DF	Service provider
ProactiveAction	It is an abstract capability to represent the autonomous actions of the agent	Service provider, User

In addition, User and Service provider agents can have autonomous actions. We define ProactiveAction capability to represent the capability that implements these actions. It is to be implemented by the developers according to the system under development context.

5.1.3.3 Communicational Dimension

The communicational dimension models, in a temporal manner, the dynamic behavior of the software system, depicting how agents interact by passing messages. Graphically, a AUML sequence diagram is used to represent the message exchange between agents.

Figure 5.5 shows the communication diagram of our MAS architecture for the user request realization. When the `Gateway` agent forwards a user request (`UserRequest`) to the `Mediator` agent, the `Mediator` agent receives and analyses the request. In case that the `Mediator` agent cannot answer the request, it sends a failure message (`UnknownRequest`) to the `Gateway` agent. Otherwise it sends a message (`FindServiceProviderRequest`) to the `DF` to find the relevant `Service provider` agents for each service needed to realize the request. The `DF` will answer with (`Null`) if there is not any `Service provider` agent for the requested service. Otherwise, the `DF` will answer with information of the relevant `Service provider` agents (`ServiceProviderInformation`). Respectively, the `Mediator` agent will send a failure message (`ServiceNotFound`) to the `Gateway` agent or send a (sub)request (`(Sub)Request`) to the relevant `Service provider` agent. There are two possible answers from the `Service provider` agent: a negative (`Failure`) or a positive (`(Sub)Result`). If the `Mediator` agent gets a negative answer from the `Service provider` agent, it will send a failure message (`RequestFailure`) to the `Gateway` agent. Otherwise, when the `Mediator` agent completes the user request realization, it will send the result (`UserRequestResponse`) to the `Gateway` agent.

Figure 5.6 illustrates the communication diagram of our MAS architecture for the interaction between the `Service consumer` and `Service provider` agents. It begins with the `Service consumer` agent sending a message (`FindServiceProviderRequest`) to ask the `DF` to find the relevant `Service provider` agents for the service needed. The `DF` will answer with (`Null`) if there is not any `Service provider` agent found for the requested service. Otherwise, the `DF` will answer with information of the relevant `Service provider` agents (`ServiceProviderInformation`). In the second case, the `Service consumer` agent will send the request (`Request`) to the relevant `Service provider` agents. There are two possible answers from a `Service provider` agent: a negative (`Failure`) or a positive (`(RequestResponse`).

5.1.3.4 Dynamic Dimension

The dynamic dimension models the internal logic of a complex operation. Graphically we use the dynamic diagram [96] which is an extended version of a UML activity diagram for agent-oriented systems, to model the process.

In a dynamic diagram, each agent constitutes a swimlane of the diagram. The capability is represented in a round-corner box and placed in the swimlane corresponding to the agent that it belongs to. An internal event is represented by a

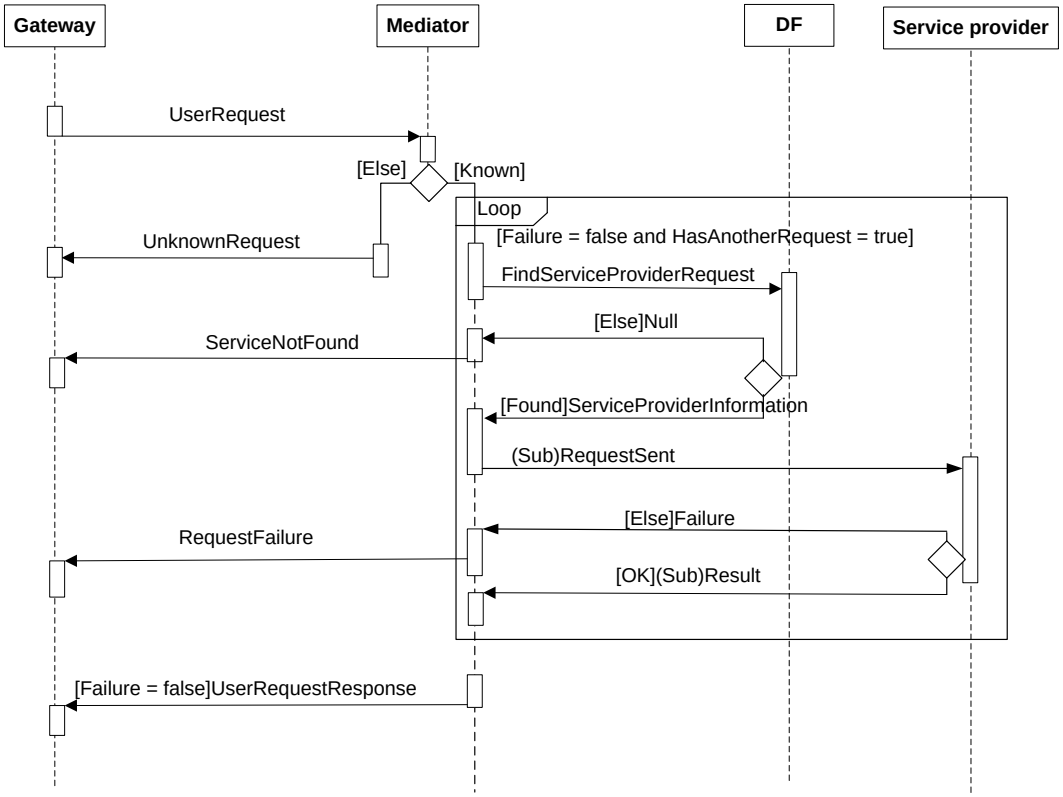


Figure 5.5: Communication diagram for the user request realization.

dashed arrow and an external event by a solid arrow. An event may be handled by alternative capabilities that are enclosed in a box. Synchronization and branching are represented as usual.

The dynamic diagram relating to the user request realization is depicted in Figure 5.7. When the `Gateway` agent gets a user request from the GUI layer (`UserRequestReceived`), it converts the user request into a ACL message and forwards it to the `Mediator` agent (`UserRequestSent`). Upon the reception of the request (`RequestReceived`), the `Mediator` agent analyses it in order to define whether it can answer the request or not. In case that the `Mediator` agent cannot answer the request, it sends a failure message to the `Gateway` agent. Otherwise it asks the `DF` for the relevant `Service provider` agents for each service needed to realize the request. If there is not any `Service provider` agent for the requested service, the `Mediator` agent will send a failure message to the `Gateway` agent. Otherwise, it sends a (sub)request to the relevant `Service provider` agents (`(Sub)RequestSent`). When receiving a request from the `Mediator` agent (`RequestReceived`), the

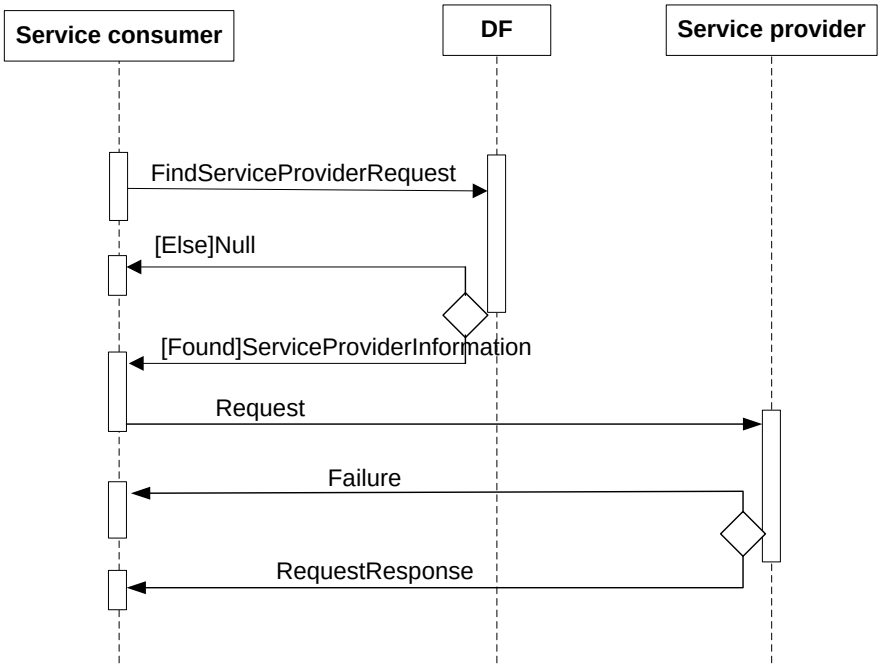


Figure 5.6: Communication diagram for the interaction between the `Service consumer` and `Service provider` agents.

`Service provider` analyzes whether it can response the received request or not. In case that it cannot, it sends a failure message to the `Mediator` agent. Otherwise, it realizes the request and sends the answer to the `Mediator` agent. There are two possible answers from the `Service provider` agent: a negative if there is a failure occurring during the request realization or a positive. If the `Mediator` agent gets a negative answer from the `Service provider` agent, it will send a failure message to the `Gateway` agent. Otherwise, the `Mediator` agent sends the result to the `Gateway` agent.

Figure 5.8 illustrates the dynamic diagram for the interaction between the `Service consumer` and `Service provider` agents. Each time that a `Service consumer` wants to invoke a service offered by other agents, it asks the `DF` for the relevant `Service provider` agents for each service needed. If there is not any `Service provider` agent for the requested service, the `Service consumer` agent handles this negative answer. Otherwise, it sends a (sub)request to the relevant `Service provider` agents (`(Sub)RequestSent`). When receiving a request from the `Service consumer` agent (`RequestReceived`), the `Service provider` analyzes whether it can response the received request or not. In case that it cannot, it sends a failure message to the `Service consumer` agent. Otherwise, it realizes the request and

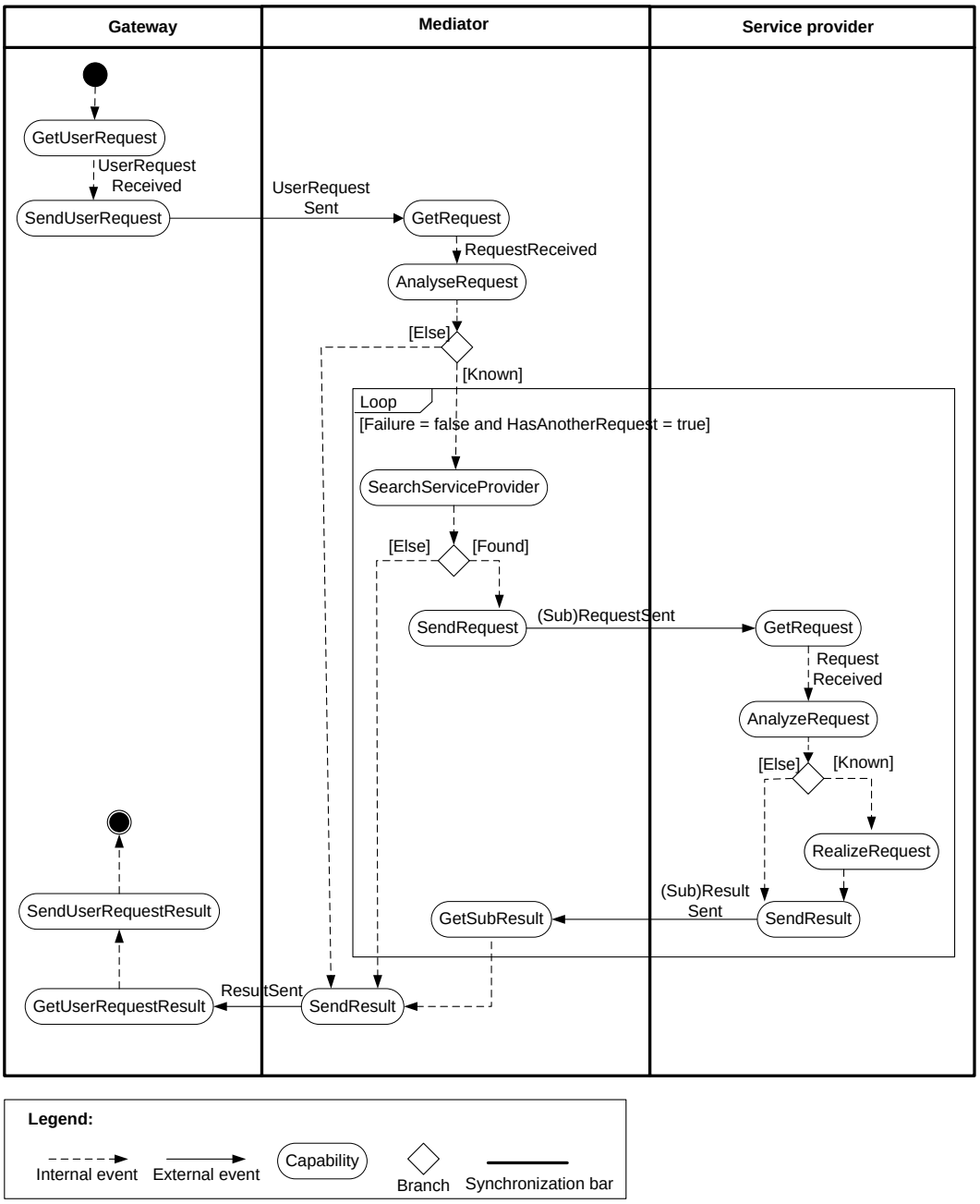


Figure 5.7: Dynamic diagram of the user request realization.

sends the answer to the `Service consumer` agent. A negative answer is sent if there is a failure occurring during the request realization.

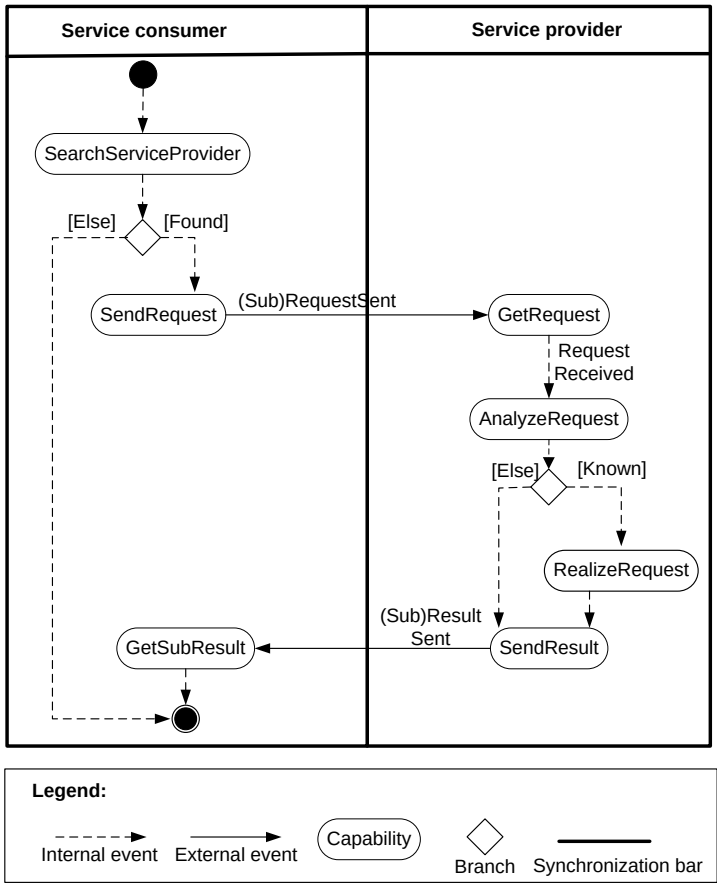


Figure 5.8: Dynamic diagram of the interaction between the `Service consumer` and `Service provider` agents.

5.2 Implementation Model

In order to ground our work, we present in this section the implementation view of the proposed architecture using the *Java Agent DEvelopment Framework* (*JADE*) [20].

5.2.1 Overview of the JADE Framework

JADE is a framework used for implementing MAS, which conforms to the FIPA standard. JADE simplifies the MAS development while ensuring standard compliance through a comprehensive set of system services and agents. It provides MAS developers with a number of features that allow to simplify the development process [19]. We point out here some features that eases the implementation of our proposed MAS:

- JADE is a FIPA-compliant agent platform, which includes the *Agent Management System* (*AMS*), the *Directory Facilitator* (*DF*), and the *Agent Communication Channel* (*ACC*) which provides a *Message Transport System* (*MTS*) and is responsible for sending and receiving messages on an agent platform;
- JADE can be integrated with script and rule engines which are essential to enable the COTS dynamic integration;
- A distributed agent platform that can be split on several hosts provided that there is no firewall between them. Only one Java application, and accordingly only one *Java Virtual Machine*, is executed on each host. Agents are implemented as one Java thread and Java events are used for effective and light-weight communication between agents on the same host. Parallel tasks can be executed by one agent, and JADE schedules these tasks in a more efficient way than the Java Virtual Machine does for threads.

The rest of this section presents some notions of programming with JADE that we need to know in order to implement our proposed MAS.

5.2.1.1 Agents Creation

Creating an agent in JADE is as simple as defining a class that extends the `jade.core.Agent` class and overriding the default implementation of the methods that are automatically invoked by the agent platform during the agent life cycle, including `setup()` for agent initialization and `takeDown()` for agent clean-up. Consistent with the FIPA specification, each agent instance is identified by an *Agent Identifier* (*AID*). In JADE, an AID is represented as an instance of the `jade.core.AID` class.

5.2.1.2 Agent Communication

Agent communication is probably one of the most fundamental features of JADE and is implemented in accordance with the FIPA specification. The JADE communication paradigm is based on asynchronous message passing. Each agent is equipped with an incoming message box and message polling can be blocking or non-blocking. A message in JADE is implemented as an object of the `jade.lang.acl.ACLMessage` object and the message is sent by calling the `send` method of the `Agent` class. By calling the `receive` method of the `Agent` class, the message next in the queue is fetched. The `receive` method can be provided with a message template to return only a message that matches a pattern defined by the template. A simple example of how a message is composed and sent is show in Code extract 5.1.

```
ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
msg.setLanguage("A-Language");
msg.setOntology("An-Ontology");
```

```
msg.setContent("VerifyUserAccountServiceInformation");
msg.setReceiver(new AID("AgentName", AID.ISLOCALNAME));
send(msg);
```

Code extract 5.1: Composing and sending a message.

5.2.1.3 Defining Agents' Capabilities

In JADE, an agent capability can be represented as a behavior which is implemented as an object of a class that extends `jade.core.behaviors.Behavior` class. Each of such behavior class must implement two abstract methods, including `action()` and `done()`. The `action()` method defines the operations to be performed when the behavior is in execution. The `done()` method returns a boolean value to indicate whether or not a behavior has completed and is to be removed from the pool of behaviors of an executing agent. The `addBehavior` method is used to add a capability to an agent and every behavior has a member variable called `myAgent` that points to the agent that is executing the behavior. This provides an easy way to access an agent's resource from within the behavior.

Behavior is specialized in `SimpleBehavior` and `CompositeBehavior`. `SimpleBehavior` represents simple atomic tasks. It is in turn specialized into `OneShotBehavior`, `CyclicBehavior`, `WakerBehavior`, and `TickerBehavior`. `OneShotBehavior` is used to represent tasks to be executed only once; `CyclicBehavior` models cyclic tasks that are restarted after finishing their execution cycle; `WakerBehavior` is used for tasks to be executed after a given timeout; `TickerBehavior` is used to represent tasks to be repetitively executed after waiting a given period. `CompositeBehavior` represents complex tasks, that are made up by composing a number of other tasks. `CompositeBehavior` is specialized into `SerialBehavior` and `ParallelBehavior`, where `SerialBehavior` is specialized into `FSMBehavior` and `SequentialBehavior`. `FSMBehavior` is used when the complex task is composed by tasks corresponding to the states of a *Finite State Machine (FMS)*; `SequentialBehavior` is a classical sequential composition of sub-tasks; `ParallelBehavior` allows the definition of concurrency, where tasks are executed in virtual parallelism.

In our case, the MAS developers need to implement behaviors that process message received from other agents. Such behaviors must be continuously running (`CyclicBehavior`) and, act at each execution of their `action()` method, must check if a message matching the specified message template has been received and process it (see Code extract 5.2 as example).

```
private class B1 extends CyclicBehavior {
public void action(){
 ACLMessage msg;
 msg = myAgent.receive(messageTemplate);
 if (msg != null) {
```

```
    // Message received. Process it
    ...
  }
  else{
    block();
  }
}
```

Code extract 5.2: Blocking a behavior waiting for a message.

5.2.1.4 Agent Discovery: The Yellow Page Service

JADE provides the yellow page service allowing agents to publish descriptions of one or more services they provide in order that other agents can easily discover and exploit them. Service subscriptions, unsubscriptions, modifications and searches can be performed at any time by any agents during its lifetime. A simple example of how an agent subscribes its service is shown in Code extract 5.3 and Code extract 5.4 shows an example of how an agent asks the DF for the relevant `Service provider` agents.

```
DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID());
ServiceDescription sd = new ServiceDescription();
sd.setType("ServiceName");
sd.setName(getLocalName()+"ServiceName");
dfd.addServices(sd);
try {
 DFService.register(this, dfd);
}
catch (FIPAException fe) {
 fe.printStackTrace();
}
```

Code extract 5.3: Subscripting a service to the `DF` agent.

```
Vector ServiceProviderAgents = new Vector();
DFAgentDescription template = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("ServiceName");
template.addServices(sd);
try {
 DFAgentDescription[] result;
 result= DFService.search(myAgent,template);
 for (int i = 0; i < result.length; ++i) {
  ServiceProviderAgents.addElement(result[i].getName());
 }
}
catch (FIPAException fe) {
```

```
  fe.printStackTrace();
}
```

Code extract 5.4: Searching for `Service provider` agents.

5.2.1.5 Integrating JADE with a Rule Engine

In our case, we need to integrate the rule engine into some agent capabilities. This can be done by integrating the rule-engine into an agent as a JADE behavior.

Currently, a number of different rule engines are available. Probably the best know of them is *Java Expert System Shell (JESS)*. JESS is a rule engine and scripting environment written entirely in Java [71] and has always been widely adopted by the JADE community to realize rule-based agent systems. [17] examines the use of both JADE and JESS for the development of intelligent agent systems. Examples of the integration of JADE and JESS can be also found in [150]. However, since JESS is no more licensed as a free open-source package, the necessity to have low cost alternatives is becoming more and more impelling. [21] proposes *Drools4JADE (D4J)* for integrating JADE agents with the Drools rule engine which is an open source rule engine and also written in Java [148].

Code extract 5.5 shows an example of how business rules can be implemented with Jess. These rules can be used for implementing a pricing engine for on-line sales. The engine is supposed to look at each order, together with a customer's purchasing history, and apply various discounts and offers to the order. Rules can be added, removed and modified easily and dynamically according to the pricing policies of the company which can be changed often.

```
(defrule 10%-volume-discount
  "Give a 10% discount to everybody who spends more than $100."
  ?o <- (Order {total > 100})
  =>
  (add (new Offer "10% volume discount" (/ ?o.total 10))))

(defrule 25%-multi-item-discount
  "Give a 25% discount on items the customer buys three or more of."
  (OrderItem {quantity >= 3} (price ?price))
  =>
  (add (new Offer "25% multi-item discount" (/ ?price 4))))
```

Code extract 5.5: Rules written in Jess.

5.2.1.6 Integrating JADE with a Scripting Engine

In our proposed architecture, some agents' capabilities especially the integration logics are written in scripting language. BeanShell is a new type of scripting language and it allows to use Java as a scripting language [146]. Code extract 5.6 shows how to integrate a BeanShell script inside the agent behavior.

```
public void action() {
        Interpreter i = new Interpreter();
        try {
              i.set("myAgent", myAgent);
                  i.set("behaviour",this);
                  i.source("beanShell.bsh");
                } catch (Exception e) {
                       e.printStackTrace();
                }
  }
```

Code extract 5.6: Integrating a BeanShell script inside an agent behavior.

5.2.2 MAS Implementation with JADE

In this section, we present the implementation of the proposed MAS using JADE. Figure 5.9 illustrates the class diagram containing different classes that implement the proposed MAS. This implementation is described in detail in the rest of this section.

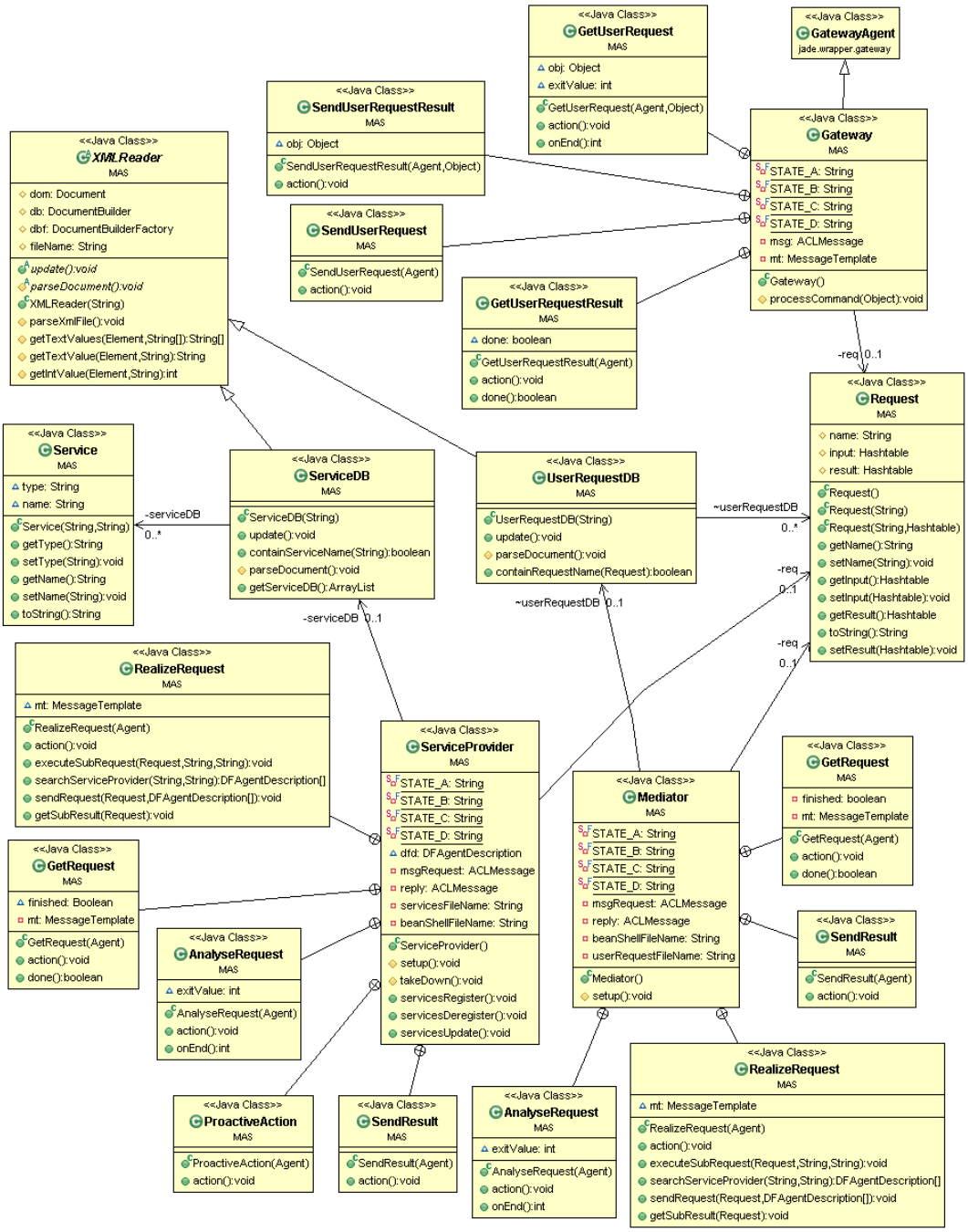


Figure 5.9: Class diagram of our MAS implementation.

5.2.2.1 Connecting GUI and MAS layers

Figure 5.1 depicts the vertical architectural layers of our proposed architecture. In our case, developers need to implement the `Gateway` agent which is a gateway between GUI and MAS layers. To this end, JADE offers some utility classes in the `jade.wrapper.gateway` package that could help developers. This package contains `JadeGateway` and `GatewayAgent` classes. The `JadeGateway` class provides a simple yet powerful gateway between some non-JADE code and a JADE based multi agent system. It is particularly suited to be used inside a *Servlet* or a *JSP*.

Code extract 5.7 shows a simple example of a `Servlet` that connects with `Gateway` agent. The `JadeGateway` is initialized and `Gateway` agent is responsible for processing all requests from the `Servlet`. A user request is created to be sent to the `Gateway` agent.

A user request is an instant of the `Request` class. Each request has a *name*, *a set of input parameters* and *a set of output parameters*. The specification of each request has to be documented and shared among developers. Each request has to be created with the correct name and sets of input and output parameters. The created request is sent to the `Gateway` agent by calling the method `JadeGateway.execute(Object request)`.

The method `JadeGateway.execute(Object request)` will return only after the method `GatewayAgent.releaseCommand(request)` has been called by the `Gateway` agent. Then, the `Servlet` gets the reply from the `Gateway` agent.

```
public class MyServlet extends HttpServlet {
 private final static String HOST = "localhost";
 private final static String PORT = "8888";

 public void init() throws ServletException {
  super.init();

  // Setting which class will be the GateWayAgent
  Properties pp = new Properties();
  pp.setProperty(Profile.MAIN_HOST, HOST);
  pp.setProperty(Profile.MAIN_PORT, PORT);
  JadeGateway.init("MAS.Gateway", pp);)
 }

 protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
   throws ServletException, IOException {

  // Reading the user input
  String bookTitle = request.getParameter("bookTitle");

  // Createing a request to send to the Gateway agent
  Hashtable parameters = new Hashtable();
  parameters.put("title", bookTitle);
  Request req = new Request("BookPrice");
  req.setInput(parameters);
  try {
```

```
    // Sending request to Gateway agent
    JadeGateway.execute(req);

   } catch (Exception e) {
    e.printStackTrace();
   }

   // Getting reply from the Gateway agent
   if (req.getResult() == null)
    request.setAttribute("Reply", "Unknown request.");
   else {
    Hashtable result = req.getResult();
    int price = (Integer) result.get("price");
    request.setAttribute("Reply", Integer.toString(price));
   }

   // Forwarding result to a JSP
   this.getServletContext().getRequestDispatcher("myjsp.jsp")
     .forward(request, response);
 }
}
```

Code extract 5.7: A simple example of a Servlet connecting with `Gateway` agent.

5.2.2.2 `Gateway` Agent Implementation

The `Gateway` agent is a class that extends the `GatewayAgent` class and redefines its method `processCommand`. When the method `JadeGateway.execute(Object request)` is called, it causes the callback of the method `processCommand` of the `Gateway` agent. In our case, we create a FSM behavior to handle the user request (see Code extract 5.8).

In JADE, a FSM behavior is used when the complex task is composed by tasks corresponding to the states of a Finite State Machine (FMS). As we can see in the Code extract 5.8, the FSM behavior is composed of `GetUserRequest`, `SendUserRequest`, `GetUserRequestResult` and `SendUserRequestResult` behaviors. These behaviors are the implementation of the corresponding `Gateway` agent's capabilities described in Section 5.1.3.2. The implementations of these behaviors are presented respectively in Code extract 5.9, 5.10, 5.11 and 5.12. They are private classes inside the `Gateway` class.

```
public class Gateway extends GatewayAgent {

 private static final String STATE_A = "A";
 private static final String STATE_B = "B";
 private static final String STATE_C = "C";
 private static final String STATE_D = "D";

 private Request req;
 private ACLMessage msg;
 private MessageTemplate mt;

 @Override
 protected void processCommand(Object obj) {
```

```
  // Creating a FMS behavior for handling the user request
  FSMBehaviour fsm = new FSMBehaviour(this);

  // Registering states
  fsm.registerFirstState(new GetUserRequest(this, obj), STATE_A);
  fsm.registerState(new SendUserRequest(this), STATE_B);
  fsm.registerState(new GetUserRequestResult(this), STATE_C);
  fsm.registerLastState(new SendUserRequestResult(this, obj), STATE_D);

  // Registering the transitions
  //Getting to the state B if the request is known
        fsm.registerTransition(STATE_A, STATE_B, 1);
  //Getting to the final state if the request is unknown
  fsm.registerTransition(STATE_A, STATE_D, 0);

  fsm.registerDefaultTransition(STATE_B, STATE_C);
  fsm.registerDefaultTransition(STATE_C, STATE_D);

  // Adding the FSM to the Gateway agent
  addBehaviour(fsm);
 }
```

Code extract 5.8: Extract code of the `Gateway` agent.

```
 private class GetUserRequest extends OneShotBehaviour {

  Object obj;
  int exitValue;

  public GetUserRequest(Agent agent, Object obj) {
   super(agent);
   this.obj = obj;

  }

  public void action() {
   if (obj instanceof Request) {

    req = (Request) obj;
    exitValue = 1;

   } else {
    exitValue = 0;

   }

  }

  @Override
  public int onEnd() {

   return exitValue;
  }

 }// End of inner class
```

Code extract 5.9: Extract code of the `GetUserRequest` behavior.

```
private class SendUserRequest extends OneShotBehaviour {

 public SendUserRequest(Agent agent) {
  super(agent);
 }

 public void action() {
  msg = new ACLMessage(ACLMessage.REQUEST);
  msg.addReceiver(new AID("Mediator", AID.ISLOCALNAME));
  try {
   msg.setContentObject(req);
  } catch (IOException e) {

   e.printStackTrace();
  }
  msg.setReplyWith("request" + System.currentTimeMillis());
  send(msg);
 }

}// End of inner class
```

Code extract 5.10: Extract code of the `SendUserRequest` behavior.

```
private class GetUserRequestResult extends Behaviour {
 boolean done = false;

 public GetUserRequestResult(Agent agent) {
  super(agent);
 }

 public void action() {
  mt = MessageTemplate.and(
    MessageTemplate.MatchSender(new AID("Mediator", AID.ISLOCALNAME)),
    MessageTemplate.MatchInReplyTo(msg.getReplyWith()));
  ACLMessage reply = receive(mt);
  if (reply != null) {
   try {
    req.setResult((Hashtable) reply.getContentObject());
    System.out.println(req.getResult());
   } catch (UnreadableException e) {

    e.printStackTrace();
   }
   done = true;

  } else {
   block();
  }

 }

 public boolean done() {

  return done;
 }
}// End of inner class
```

Code extract 5.11: Extract code of the `GetUserRequestResult` behavior.

```
 private class SendUserRequestResult extends OneShotBehaviour {

  Object obj;

  public SendUserRequestResult(Agent agent, Object obj) {
   super(agent);
   this.obj = obj;
  }

  @Override
  public void action() {

   releaseCommand(obj);

  }

 }

}
```

Code extract 5.12: Extract code of the `SendUserRequestResult` behavior.

5.2.2.3 `Mediator` Agent Implementation

Code extract 5.13 shows how the `Mediator` is implemented. It consists of a FSM behavior which is composed of `GetRequest`, `AnalyzeRequest`, `RealizeRequest` and `SendResult` behaviors. These behaviors are the implementation of the corresponding `Mediator` agent's capabilities described in Section 5.1.3.2.

```
public class Mediator extends Agent {

 protected void setup() {

  Object[] args = getArguments();
  if (args != null && args.length > 1) {
  /*Getting the name of file that stores
   the BeanShell script used in RealizeRequest
   behavior.*/
   beanShellFileName = (String) args[0];

   /*Getting the name of XML file that stores
   the description of user requests that can
   be handled by the Mediator.*/
   userRequestFileName = (String) args[1];
  }

  FSMBehaviour fsm = new FSMBehaviour(this);

  fsm.registerFirstState(new GetRequest(this), STATE_A);
  fsm.registerState(new AnalyzeRequest(this), STATE_B);
  fsm.registerState(new RealizeRequest(this), STATE_C);
  fsm.registerState(new SendResult(this), STATE_D);

  //State transition.
  fsm.registerDefaultTransition(STATE_A, STATE_B);
  fsm.registerTransition(STATE_B, STATE_C, 1);
```

```
  fsm.registerTransition(STATE_B, STATE_A, 0);
  fsm.registerDefaultTransition(STATE_C, STATE_D);
  fsm.registerDefaultTransition(STATE_D, STATE_A);

  addBehaviour(fsm);
 }
```

Code extract 5.13: Extract code of the `Mediator` agent.

Code extract 5.14 shows the implementation of the `GetRequest` capability. It is done when it gets a request. Otherwise, it is blocked.

```
 private class GetRequest extends Behaviour {
  Boolean done = false;
  private MessageTemplate mt;

  public GetRequest(Agent agent) {
   super(agent);

  }

  public void action() {
   mt = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
   msgRequest = receive(mt);
   if (msgRequest != null) {
    reply = msgRequest.createReply();
    try {
     Object contentObject = msgRequest.getContentObject();
     if (contentObject instanceof Request) {
      req = (Request) contentObject;
      done = true;
     }
    } catch (UnreadableException e) {
     e.printStackTrace();
    }

   } else {
    block();
   }
  }

  public boolean done() {

   return (done);
  }
 }// End of inner class
```

Code extract 5.14: Extract code of the implementation of the `GetRequest` capability.

Code extract 5.15 shows the implementation of the `AnalyzeRequest` capability allowing the `Mediator` agent to check if it can answer the request received. To simplify, this is done for the moment by checking the request name. All the user requests that can be handled by the `Mediator` agent are stored in an XML file. Code extract 5.16 shows a simple example of such file. The `UserRequestDB` class (Code extract B.8) is used to read such XML file. It extends the `XMLReader` class (Code extract B.5) which is an abstract class offering the methods to read XML files.

```
private class AnalyzeRequest extends OneShotBehaviour {
 int exitValue = 0;

 public AnalyzeRequest(Agent agent) {
  super(agent);
 }
 public void action() {
  if (req != null) {
   UserRequestDB userRequestDB=new UserRequestDB(userRequestFileName);
   if (userRequestDB.contains(req))
    exitValue = 1;
  }
 }

 @Override
 public int onEnd() {
  return exitValue;
 }

}// End of inner class
```

Code extract 5.15: Extract code of the implementation of the `AnalyzeRequest` capability.

```
<?xml version="1.0" encoding="UTF-8"?>
<UserRequest>
  <Request>
        <Name>BookPrice</Name>
  </Request>
  <Request>
        <Name>BookCatalogue</Name>
  </Request>
</UserRequest>
```

Code extract 5.16: Code extract of the `UserRequest` XML file.

Code extract 5.17 shows the implementation of the `RealizeRequest` capability. In order to get the dynamic integration among component, the realization of the request is written in BeanShell script. The use of BeanShell script inside the JADE behavior is shown in Code extract 5.17. Code extract 5.18 provides a simple example of BeanShell script for the `RealizeRequest` behavior. The `SearchServiceProvider`, `SendRequest` and `GetSubResult` capabilities that are used during the request realization are implemented in form of three corresponding methods inside the `RealizeRequest` behavior. The implementations of these capabilities are presented respectively in Code extracts 5.19, 5.20 and 5.21.

```
public class RealizeRequest extends OneShotBehaviour {

 public RealizeRequest(Agent agent) {
  super(agent);

 }
```

```
  public void action() {

   Interpreter i = new Interpreter();
   try {
    i.set("myAgent", myAgent);
    i.set("behaviour", this);
    i.set("req", req);
    i.source(beanShellFileName);
   } catch (FileNotFoundException e) {
    e.printStackTrace();
   } catch (IOException e) {
    e.printStackTrace();

   } catch (EvalError e) {
    e.printStackTrace();
   }

  }
```

Code extract 5.17: Extract code of the implementation of the `RealizeRequest` capability.

```
import java.util.Hashtable;
import jade.core.Agent;
import jade.core.behaviours.OneShotBehaviour;
import MAS.Request;
import MAS.Mediator;


public void executeSubRequest(Request subReq, String serviceType,
    String serviceName) {
    behaviour.searchServiceProvider(serviceType, serviceName);
    behaviour.sendRequest(subReq);
    behaviour.getSubResult(subReq);
  }

if (req.getName().equals("BookPrice")) {

        executeSubRequest(req,"Book","BookPrice");

} else if (req.getName().equals("BookCatalogue")){
        executeSubRequest(req,"Book","BookCatalogue");
}
```

Code extract 5.18: Code extract of Bean shell script for the `Mediator`.

```
  public void searchServiceProvider(String type, String name) {
   try {
    // Build the description used as template for the search
    DFAgentDescription template = new DFAgentDescription();
    ServiceDescription templateSd = new ServiceDescription();
    System.out.println("Service␣provider␣search:␣" + serviceType + ":"
      + serviceName);
    templateSd.setType(type);
    templateSd.setName(name);
    template.addServices(templateSd);
```

```
    DFAgentDescription[] results = DFService.search(myAgent, template);
    System.out.println(results.length);
    serviceProviders = results;

   } catch (FIPAException fe) {
    fe.printStackTrace();
    serviceProviders = null;
   }

  }
```

Code extract 5.19: Extract code of the implementation `SearchServiceProvider` capability.

```
  public void sendRequest(Request request) {

   if (serviceProviders.length > 0) {

    // Send the cfp to all sellers
    ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
    for (int i = 0; i < serviceProviders.length; ++i) {
     msg.addReceiver(serviceProviders[i].getName());
    }
    try {
     msg.setContentObject(request);
    } catch (IOException e1) {

     e1.printStackTrace();
    }
    msg.setReplyWith("request" + System.currentTimeMillis());
    myAgent.send(msg);
    // Prepare the template to get proposals
    mt = MessageTemplate.and(
      MessageTemplate.MatchConversationId("conversationId"),
      MessageTemplate.MatchInReplyTo(msg.getReplyWith()));

   } else {
    System.out.println("Service provider not found.");

   }
  }
```

Code extract 5.20: Extract code of the implementation `SendRequest` capability.

```
  public void getSubResult(Request request) {
   if (serviceProviders.length > 0) {
    ACLMessage reply = myAgent.receive(mt);
    while (reply == null)
     reply = myAgent.receive(mt);
    try {
     request.setResult((Hashtable) reply.getContentObject());

    } catch (UnreadableException e) {
     // TODO Auto-generated catch block
     e.printStackTrace();
    }
   }
```

```
  }
 }// End of inner class
```

Code extract 5.21: Extract code of the implementation `GetSubResult` capability.

5.2.2.4 `Service provider` Agent Implementation

Code extract 5.22 illustrates the implementation of a `Service provider` agent. It composes of two main behaviors which are executed in parallel. The first handles the request from other agents and the second handles its proactive actions. The proactive actions are specific to each agent. Consequently, we do not focus on the implementation of the second behavior.

Similarly to the `Mediator` agent, the first behavior of `Service provider` agent is complex behavior implemented through a FSM behavior composed of `GetRequest`, `AnalyzeRequest`, `RealizeRequest` and `SendResult` behaviors wherein their implementations are almost the same except the used of `ServiceDB` class (Code extract B.6) instead of `UserRequestDB` in the `AnalyzeRequest` behavior.

Code extract 5.23 shows the implementation of the `AnalyzeRequest` capability of a `Service provider` agent. To simplify, this is done for the moment by checking the request name which corresponds to the service name offered by the `Service provider` agent. The `ServiceDB` class is used to read the XML file that stores the description of services offered by the `Service provider` agent. Code extract 5.24 shows a simple example of such file. A service is an instant of the `Service` class. To simplify, each service is defined through its type and name.

When a `Service provider` is started up, it registers the services that it offers to the yellow page service of JADE. It deregisters its services from the yellow page when it is off. The `ServiceRegister`, `ServicesDeregister` and `ServicesUpdate` capabilities are implemented in form of three corresponding methods inside the `Service provider` agent. The implementations of these capabilities are presented respectively in code extracts 5.25, 5.26 and 5.27.

```
public class ServiceProvider extends Agent {
 private static final String STATE_A = "A";
 private static final String STATE_B = "B";
 private static final String STATE_C = "C";
 private static final String STATE_D = "D";

 DFAgentDescription dfd;

 private Request req;
 private ACLMessage msgRequest, reply;
 private String servicesFileName;
 private String beanShellFileName;
 private ServiceDB serviceDB;

 @Override
```

```
protected void setup() {

 dfd = new DFAgentDescription();
 dfd.setName(getAID());
 Object[] args = getArguments();
 if (args != null && args.length > 0) {
  servicesFileName = (String) args[0];
  beanShellFileName = (String) args[1];
  servicesRegister();

 }

 FSMBehaviour fsm = new FSMBehaviour(this);

 fsm.registerFirstState(new GetRequest(this), STATE_A);
 fsm.registerState(new AnalyzeRequest(this), STATE_B);
 fsm.registerState(new RealizeRequest(this), STATE_C);
 fsm.registerState(new SendResult(this), STATE_D);

 // State transitions
 fsm.registerDefaultTransition(STATE_A, STATE_B);
 fsm.registerTransition(STATE_B, STATE_C, 1);
 fsm.registerTransition(STATE_B, STATE_A, 0);
 fsm.registerDefaultTransition(STATE_C, STATE_D);
 fsm.registerDefaultTransition(STATE_D, STATE_A);

 //The fsm behavior for handling the request
 //from other agent and another behavior for proactive
 //actions are executed in parallel
 ParallelBehaviour b = new ParallelBehaviour();
 b.addSubBehaviour(fsm);
 b.addSubBehaviour(new ProactiveAction(this));
 addBehaviour(b);

}
protected void takeDown() {
 // Deregister from the yellow pages
 servicesDeregister();
}
```

Code extract 5.22: Extract code of the `Service provider` agent.

```
 private class AnalyzeRequest extends OneShotBehaviour {
 int exitValue = 0;

 public AnalyzeRequest(Agent agent) {
  super(agent);

 }

 public void action() {

  if (req != null) {
   if (serviceDB.containServiceName(req.getName()))
    exitValue = 1;
  }
 }

 @Override
```

```
  public int onEnd() {
   return exitValue;
  }

 }// End of inner class
```

Code extract 5.23: Extract code of the implementation `AnalyzeRequest` capability of a `Service provider` agent.

```
<?xml version="1.0" encoding="UTF-8"?>
<Services>
 <Service>
                <Type>Book</Type>
        <Name>BookPrice</Name>
  </Service>
  <Service>
                <Type>Book</Type>
        <Name>BookCatalogue</Name>
  </Service>

</Services>
```

Code extract 5.24: Code extract of `Services` XML file.

```

 public void servicesRegister() {
  serviceDB = new ServiceDB(servicesFileName);
  Iterator it = serviceDB.getServiceDB().iterator();
  Service service;
  while (it.hasNext()) {
   service = (Service) it.next();
   ServiceDescription sd = new ServiceDescription();
   sd.setType(service.getType());
   sd.setName(service.getName());
   System.out.println(service.getType() + "␣" + service.getName());
   dfd.addServices(sd);
  }
  try {
   DFService.register(this, dfd);
  } catch (FIPAException fe) {
   System.out.println("Service␣registrering␣exception.");
   fe.printStackTrace();
  }

 }
```

Code extract 5.25: Extract code of the implementation of the `ServiceRegister` capability.

```
 public void servicesDeregister() {
  try {
   DFService.deregister(this);
  } catch (FIPAException fe) {
   fe.printStackTrace();
  }

```

```
}
```

Code extract 5.26: Extract code of the implementation of the `ServiceDeregister` capability.

```
public void servicesUpdate() {
  servicesDeregister();
  dfd.clearAllServices();
  servicesRegister();
  System.gc();
}
```

Code extract 5.27: Extract code of the implementation of the `ServiceUpdate` capability.

5.3 Chapter Summary

In this chapter, we have presented our proposed integration architecture through the use of a wrapper-based multi-agent (MAS) architecture. More precisely, we propose a MAS architecture designed to wrap the different components used in the software system acting as an abstraction layer between the user's (high level) requests, their functional decomposition and the coordination of execution by the components.

Our architectural description goes beyond a pure system design by incorporating a description over multiple complementary dimensions illustrating the different aspects of the MAS architecture for components integration. It is generic but can be adapted to a particular business logic issued of a project specific domain model. This allows the developers and designers to focus on requirements models on the one hand and on (standardized) interfacing with the MAS on the other hand to adequately tune and configure the architectural level.

We also show the possibility of implementing our proposed MAS using the JADE agent framework. This constitutes an implementation model which facilitates developers to implement our proposed MAS.

Part IV

RecIProC: Rationale Incremental and Iterative Process for COTS-Based System Development

Chapter 6

Towards an Agent-Oriented Methodology for CBSD

In this chapter, we firstly present the shortcomings we can draw from our literature review on the CBSD and the specifications for our methodology. Then, we present the adaption of agent-oriented techniques for system analysis and modeling for our methodology. Finally, our methodology for CBSD called *Rationale Incremental and Iterative Process for Building COTS-based Systems (RecIProC)* is described.

6.1 Weaknesses of Existing CBSD Methods

In this section, we present the conclusion of our review on the existing works on CBSD. We firstly point out a list of shortcomings. Then, on the basis of these shortcomings we define a set of requirements to be fulfilled by our methodology.

6.1.1 Shortcomings

The main shortcomings we can draw from our literature review on the CBSD (see Chapter 3) are the following:

1. methodologies that address the life cycle of CBSD provide only high-level processes without providing any practical models and guidelines for use in the process;

2. there is not any approach that deals with the strategic reasoning of a CBSD project;

3. there is a lack of approaches that deal with all dimensions (i.e, functional requirements, NFRs, non-technical, etc.) concerning COTS selection;

4. most approaches are suitable only for single COTS selection and few approaches addressing multiple COTS selection provide only high-level processes.

6.1.2 Specifications for RecIProC

The aim of this thesis is to contribute to the improvement of CBSD. We seek to address the above shortcomings by proposing a methodology that meet the following specifications for RecIProC:

1. providing an adequate CBSD process with some practical models and guidelines for use in the process;
2. covering business analysis, requirements analysis, COTS evaluation and selection, COTS mismatches handling, and COTS integration;
3. dealing with all dimensions (i.e. functional requirements, NFRs and non-technical aspect) concerning COTS selection;
4. suitable for both single and multiple COTS selections;
5. providing tool support.

6.2 Adopting the i* Framework for Our Methodology

The aim of our work is to propose a methodology for developing large scale distributed business systems with the use of COTS components. Such a system allows a company to automate and integrate the majority of its operations; this makes the company more competitive in the market place. It is thus important to have a good understanding of the target organizational environment.

According to the comparative study for goal modeling techniques presented in [92], there are a few modeling techniques that have been proposed to describe the current organizational situation during requirements elicitation. They are *GOMS* [45], *Goal-based Workflow* [63], *i* framework* [170] and *EKD* [93]. Among them, only the i* framework provides the mechanism to model complex social and organizational relationships and to help reasoning about them. Indeed, it allows intention, relationship and motivation modeling among organizational members. From the SD and SR models, we can have a better understanding of how the organizational environment is working, human and work relations among the organizational actors which can be humans or software systems.

Despite the well-known advantages of the i* modeling framework, there are some issues that still need to be improved to ensure its effectiveness in practice [64] [161]. [161] argued that one of the main problems is the lack of an adequate *scope element*. A scope element is a high level model element identified in the analysis phase and

used to drive the software development process. Indeed, in the i* model, the goal is the only element that can be considered as a scope element since it represents the most abstract functional issue. However, its use as a scope element has the following drawbacks [161]:

- The atomicity of the goal is not clearly defined. A goal decomposition can involve atomic goal part of higher level ones. Consequently, there is a large number of goals presented in i* diagrams of huge software projects – too many for keeping an easily understandable vision of the software project. Moreover, goals are often modeled at different abstraction levels so that they can be totally or partially overlapping which is a major drawback to keep the simple goal-based vision;
- The distinction between a task and a goal is not always trivial in the sense that different persons can model similar behavior by a goal or by a task.

To address this problem, the authors of [64] propose to extend the i* model with a new element/diagram at higher abstraction level allowing to represent a service the agent should provide in a non redundant way. The main idea of this proposal is the representation of an organizational model as a composition of business services where these services represent the functionalities that the organization offers to potential customers. This approach is taken as the starting point of the work presented in [161]. The authors of [161] use the idea of a service level diagram with non-overlapping services. More precisely, they extend the i* modeling framework with a new model called the *Strategic Service Model (SSM)* to present on the highest aggregation level the services provided by the system with the actors involved as well as the potential threats and opportunities they can face.

In this thesis, we propose to use the SSM proposed in [161] for defining the scope of the project and modeling the big picture of the intended integrated system. For clarity reason, instead of using all the elements of the SSM (see [161]), the SSM used in our methodology consists of a set of actors and links connecting actors. Actors are intentional entities used to model people, physical devices or software systems that use or offer services. Each link represents a service dependency between two actors whereby one actor (*the depender*) depends on the other (*the dependee*) to offer a service. There are typically two types of services in the SSM: *Business services* – business processes accomplished in the business domain of the company – and *User services* – services provided by the software system to the end user. The use of the SSM allows all the project stakeholders to share a common aggregate view of what the COTS components should offer as well as their dependency relationships. In addition, it also allows to analyze the strategic impacts in term of threats and opportunities that the project can face.

Furthermore, in our methodology, the SD and SR models will be used to provide a more detailed description of organizational environment and functional requirements

of each application package. While an SD model representing the intentional level of an application package will be used to drive the search for corresponding COTS components in the marketplace, an SR model representing the rationale level of the application package will be used to lead the COTS candidates evaluation at high-level.

In [170], the non-functional requirements of the system can be represented as softgoals due to the similarity between non-functional requirements and softgoals which lies in the fact that they are both subjective and there are no clear-cut criteria for achievement. However, in our methodology, we distinguish softgoals from non-functional requirements. As argued in [76], non-functional requirements or quality requirements define constraints but do not define system functions while softgoals characterize states that the system should attain and thus can describe directly or indirectly a system function. According to [76], **a softgoal is a goal with quality constraints**. For instance, "*All banking transactions must be treated in a secure manner*" is a softgoal. It represents a goal "*All banking transactions must be treated*" constrained by the quality requirement "*in a secure manner*". This distinction provides a clearer definition of softgoal. The author of [76] suggests to add a new element representing the quality requirement to the i* model. However, in our methodology, for defining the non-functional requirements of the system, we propose to use the NFR framework [53]. In this way, we reduce the complexity of the resulting SD and SR models.

In addition, it is also important to clarify the distinction between a task and a goal. For this clarification, we apply a constraint defined in [159] that a task cannot be decomposed into sub-goals. This constraint ensures that goals are more abstract than tasks. In our research context, a goal represents a high-level requirement of which satisfaction cannot be directly measured or judged in a COTS component while a task represents a technical goal which can be matched with the application features.

In order to prioritize the requirements, we associate each goal, softgoal and task with a degree of desirability:

- **Very High (VH)**: very critical goal that must be fulfilled otherwise the success of the project will be strongly compromised;
- **High (H)**: critical goal that must be fulfilled otherwise the success of the project will be compromised;
- **Medium (M)**: important goal that should be fulfilled in order to ensure that significant goals will be satisfied;
- **Low (L)**: desirable goal that could be interesting to have but that does not affect the success of the system;

- **Very Low (VL)**: slightly desirable goal that does not affect the success of the system.

Following the comparative study of the i* variants presented in [16], Table 6.1 summarizes the specification of the i* variant used in our methodology. It shows the comparison between Yu's i* and our i* variant according to the following criteria:

- *Types of models.* As stated earlier, we propose to use the SS model in addition to the two existing SD and SR models for modeling the most aggregate static view of the project for adequate identification of different individual systems;
- *Types of actors.* In our methodology, the specialization of actors is not used in order to reduce the complexity of the models;
- *Intentional elements.* Service is a new intentional element in addition to the existing ones (i.e. goal, softgoal, task and resource);
- *Relationships among intentional elements.* Since the specialization of actors is not used, only dependencies among actors by means of intentional elements is used;
- *Relationships among intentional elements.* We propose to use the four types of relationships: *dependencies*, *means-end*,*decompositions* and *contributions*;
- *Degree of desirability.* As evoked above, for prioritizing the requirements, a degree of desirability is assigned to each goal, softgoal and task.

The following section describes the development process of our methodology. It will provide guidelines to build the i* models with respect to the above specification.

Table 6.1: Specification of our i* variant.

Criteria	Yu's i*	**Our i* variant**
Types of models	SD and SR	SS, SD and SR
Types of actors	1 generic 3 specific: role, position and agent	1 generic
Intentional elements	Goal, softgoal, task, resource	Service, goal, softgoal, task, resource
Relationship among actors	Dependencies among actors by means of intentional elements Relationships among specific types of actors "occupies", "cover" and "play" Relationship "is-part-of"	Dependencies among actors by means of intentional elements
Relationship among intentional elements	Dependencies among actors Means-end relationships Decomposition relationships Contribution relationships	Dependencies among actors Means-end relationships Decomposition relationships Contribution relationships
Degree of desirability	N/A	5 degrees of desirability are associated with goal, softgoal and task:Very high, High, Medium, Low, Very low

6.3 Rationale Incremental and Iterative Process for CBSD

This section describes our methodology for CBSD called *Rationale Incremental and Iterative Process for Building COTS-based Systems (RecIProC)*. To present an overview of the process, we use the Software Process Engineering Meta-Model (SPEM) [131].

6.3.1 The Software Process Engineering Meta-Model (SPEM)

The Software Process Engineering Meta-Model (SPEM), specified by the Object Management Group (OMG), is used to define and describe software development processes and their components [131]. It constitutes a sort of ontology of software development processes. A full description of SPEM can be found in [131]. Figure

6.1 depicts the relevant SPEM concepts used in this thesis.

- *Phase.* It is used to express significant segments of the complete life cycle;
- *WorkDefinition.* It constitutes a kind of operation that describes the work performed in the process;
- *Activity.* It is described as a unit of work that must be engaged in order to produce products;
- *WorkProduct.* It is anything produced, consumed or modified by a process. Information, documents and models constitute different kinds of WorkProduct.

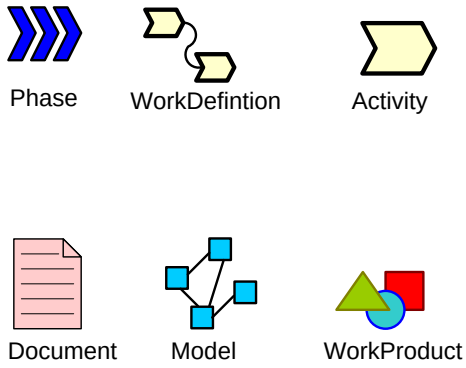


Figure 6.1: Some relevant SPEM elements.

6.3.2 Process Model

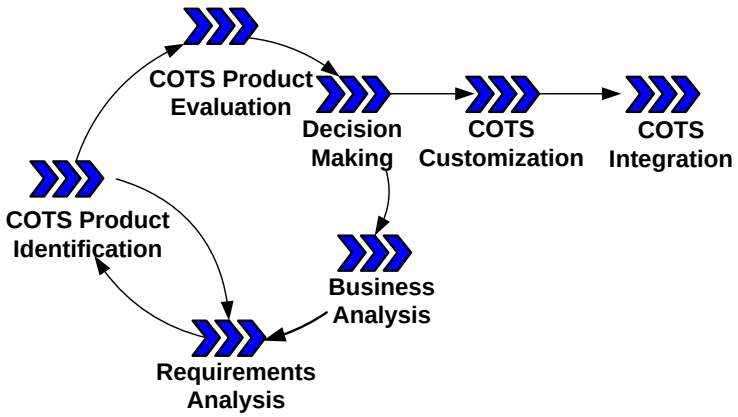


Figure 6.2: A RecIProC development cycle.

RecIProC is an iterative cyclic approach for incrementally integrating COTS components into the system. Each cycle produces a deliverable "increment" of the

software with new COTS added to the system. As depicted in Figure 6.2, a RecIProC cycle consists of seven iterative phases including *Business Analysis*, *Requirements Analysis*, *COTS Product Identification*, *COTS Product Evaluation*, *Decision Making*, *COTS Customization* and *COTS integration*. An overview of the RecIProC cycle is depicted in Figure 6.3. The different phases of the RecIProC cycle will be described in the following sections.

6.3.3 Business Analysis Phase

This phase consists of three main WorkDefinitions[1] including *Studying the business context*, *Defining the scope of the project*, and *Analyzing strategic impacts*. These WorkDefinitions are mainly performed during the first cycle of the project. During the following cycles, they are performed to handle the business changes. The result from the business analysis phase serves as a basis for the following phases.

6.3.3.1 Studying the Business Context

The aim of this WorkDefinition is to understand the business context in order to define the business problems to be solved or the business opportunities to be addressed by the information system. The understanding is based on business plans, annual reports, and interviews with executives.

6.3.3.2 Defining the Scope of the Project

The aim of this WorkDefinition is to identify the different systems that will be integrated together to build the intended system. As stated in Section 6.2, we suggest to use the *Strategic Service Model* (SSM) proposed in [161] to help accomplishing this WorkDefinition.

A *Strategic Service Diagram (SSD)* is built-up iteratively. The process of building a SSD (see Figure 6.4) begins with the identification of an initial set of organizational stakeholders involved in the problem addressed. Next, the business service dependencies among these stakeholders should be identified. Then, the software systems that the stakeholders need in order to fulfill the services they offer are included as well as the user services that each system needs to offer. At this point, a first version of the SSD is obtained. If new actors or new service dependencies have to be incorporated in the diagram, the process iterates. The created SSD needs to be validated with the stakeholders since it defines the scope of the project.

[1] In the sense defined by the Software Process Engineering Metat-Model (SPEM)[131]

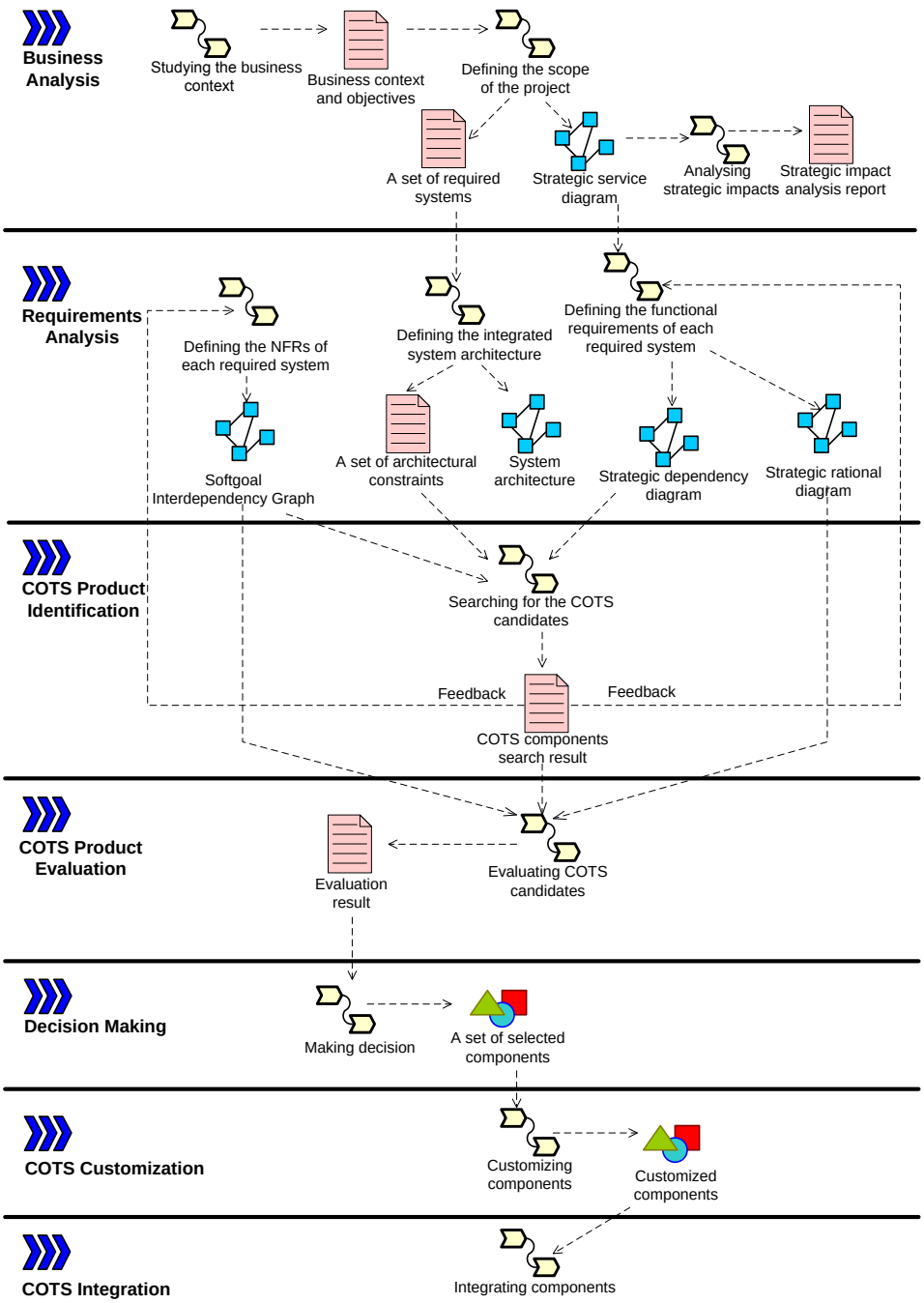


Figure 6.3: Overview of a RecIProC cycle.

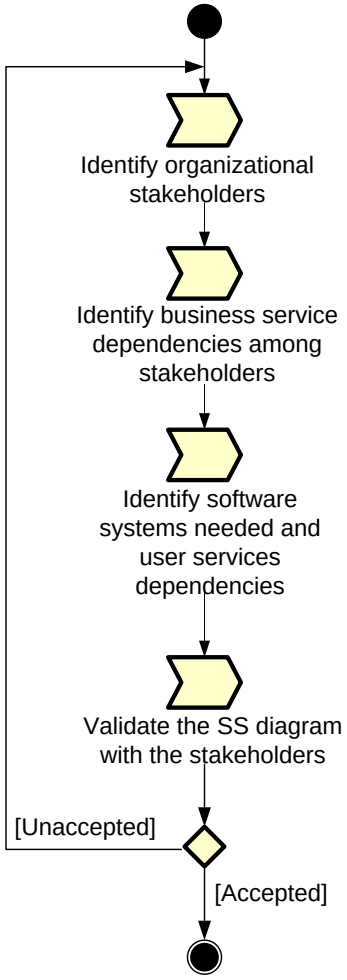


Figure 6.4: An SSD building process.

6.3.3.3 Analyzing Strategic Impacts

This WorkDefinition aims to study the long-term strategic impacts of adopting the software project. In our methodology, the SSM is used to model the project scope and according to the theoretical model of the SSM [161], it allows business analysts to analyze the strategic impacts in term of threats and opportunities that the project can face. To this end, the added value/risk exposure for each particular service to each particular opportunity/threat is evaluated on the basis of a *Low/Medium/High/None* scale. The opportunity/threat hierarchy is established through an "*Opportunity/Threat Table*" assigning to each service an opportunity added value/risk exposure probability. We basically define four values for opportunities added value/threats exposure for a particular service:

- **Low**: "L" in a yellow filled cell, has a weight of 1;
- **Medium**: "M" in a orange filled cell, has a weight of 2;
- **High**: "H" in a red filled cell, has a weight of 4;
- **Non-existing**: an empty cell, this service is not concerned by this opportunity/threat, has a weight of 0.

The process of determining the opportunity and threats as well as assigning the values just mentioned is done through interview with the stakeholders. These values are not "guessed" by software analysts but obtained by an interview/survey process.

The *Overall Risk Exposure* of a service is computed as follows:

$$OverallRiskExposure_s = \sum_i tw_i.re_i^s$$

where tw_i is the weight of the threat i and re_i^s is the exposure of the service s to the threat i. As we can see, re_i^s can thus take the value 0 (Non-existing), 1 (Low), 2 (Medium), or 4 (High).

Similarly, the *Overall Opportunity Added Value* of a service is computed as follows:

$$OverallOpportunityAddedValue_s = \sum_i ow_i.av_i^s$$

where ow_i is the weight of the opportunity i and av_i^s is the exposure of the service s to the opportunity i. As we can see, av_i^s can thus take the value 0 (Non-existing), 1 (Low), 2 (Medium), or 4 (High).

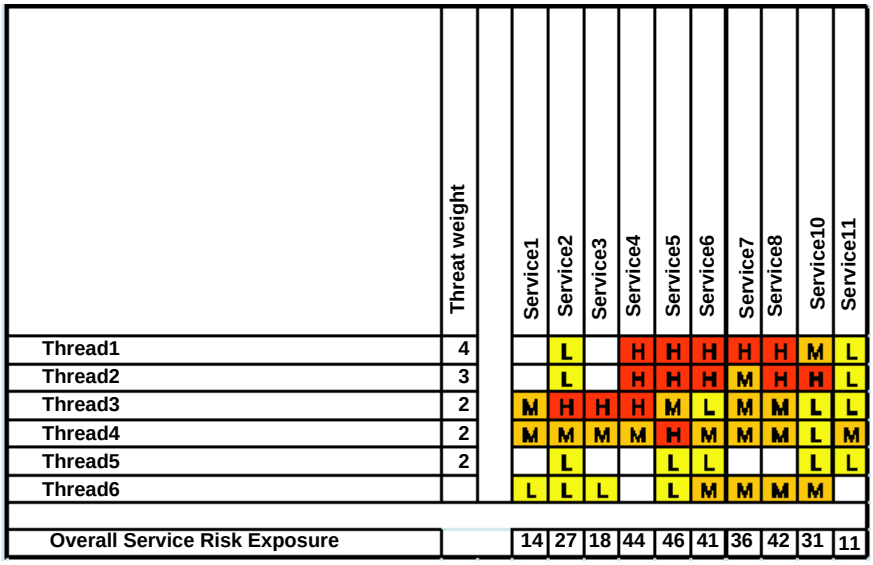
Figure 6.5 illustrates an examples of threat/service matrix (see Figure 6.6 for an example of opportunity/service matrix).

6.3.4 Requirements Analysis Phase

This phase focuses on defining the system requirements. It consists of three main WorkDefinitions including *Defining the integrated system architecture*, *Defining the functional requirements of each required system*, and *Defining the NFRs of each required system.*

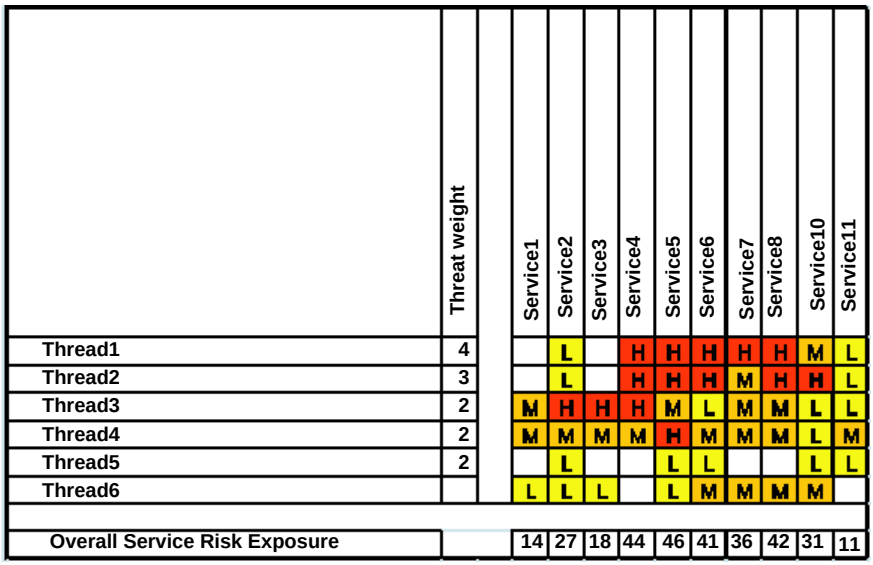
6.3.4.1 Defining the Integrated System Architecture

This WorkDefinition aims to define the architecture of the integrated system. The output of this work is a system architecture model and a set of architectural constraints that will be used in the evaluation of COTS components. In our methodology, we propose a system architecture model for integrating COTS components (see chapter 5 for the detail).



	Threat weight		Service1	Service2	Service3	Service4	Service5	Service6	Service7	Service8	Service10	Service11
Thread1	4			L		H	H	H	H	H	M	L
Thread2	3			L		H	H	H	M	H	H	L
Thread3	2		M	H	H	H	M	L	M	M	L	L
Thread4	2		M	M	M	M	H	M	M	M	L	M
Thread5	2			L			L	L			L	L
Thread6			L	L	L		L	M	M	M	M	
Overall Service Risk Exposure			14	27	18	44	46	41	36	42	31	11

Figure 6.5: An example of threat/service matrix.



	Threat weight		Service1	Service2	Service3	Service4	Service5	Service6	Service7	Service8	Service10	Service11
Thread1	4			L		H	H	H	H	H	M	L
Thread2	3			L		H	H	H	M	H	H	L
Thread3	2		M	H	H	H	M	L	M	M	L	L
Thread4	2		M	M	M	M	H	M	M	M	L	M
Thread5	2			L			L	L			L	L
Thread6			L	L	L		L	M	M	M	M	
Overall Service Risk Exposure			14	27	18	44	46	41	36	42	31	11

Figure 6.6: An example of opportunity/service matrix.

6.3.4.2 Defining the Functional Requirements of Each Required System

During the definition of the non-functional requirements of each required sub-system, its functional requirements will be also defined. For adequate definitions of the functional requirements of each system-level component we use our adapted version of the *Strategic Dependency (SD)* and *Strategic Rationale (SR)* models of the i*

modeling framework [170] as described in Section 6.2.

SD and SR diagrams are constructed iteratively. Figure 6.7 illustrates the building process of a SD diagram. It begins with the identification of an initial set of relevant organizational actors interacting with the system-level component addressed. Then, the dependencies among actors (i.e., *Goal dependency*, *Softgoal dependency*, *Task dependency* and *Resource dependency*) are identified. At this point, a first version of SD diagram is obtained and validated with stakeholders. Next, the actors in the SD diagram will be rationally analyzed as depicted in the Figure 6.8 and, as a result, a corresponding SR diagram is created. If new actors or new dependencies have to be incorporated in the diagram, the process iterates.

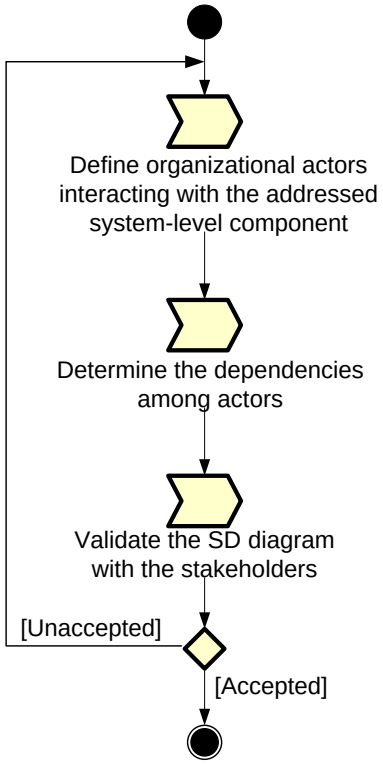


Figure 6.7: A SD diagram building process.

6.3.4.3 Defining the NFRs of Each Required System

This WorkDefinition focuses on the definition of the NFRs of each required system. For adequate analysis of NFRs, we propose to use the NFR framework [53] (see Section 2.2.2). A *Softgoal Interdependency Graph (SIG)* representing the NFRs of each required sub-system will be created.

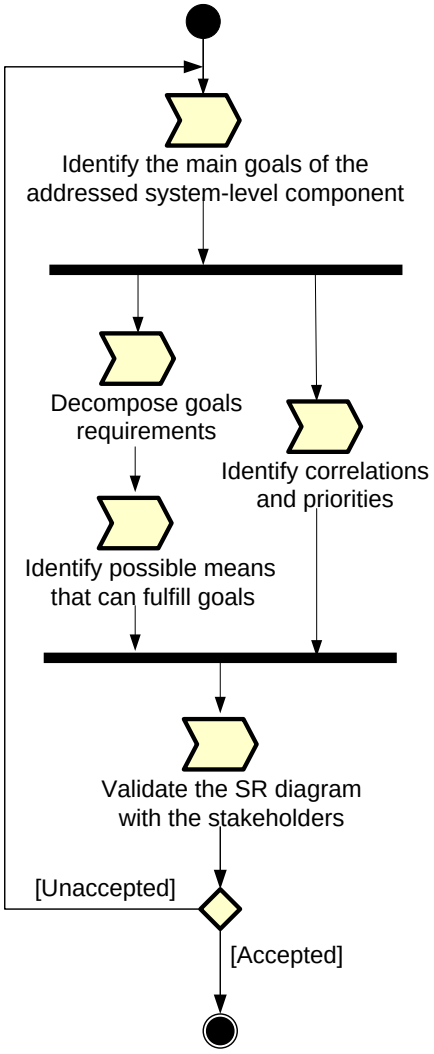


Figure 6.8: A SR diagram building process.

Figure 6.9 shows the process of building a SIG. It begins with the identification of the high-level NFRs that the system should meet (see Section 3.6.2 for quality models). The identified NFRs will be represented as NFR softgoals at the top of the SIG and they will be iteratively refined into more specific ones. At some point, when the NFR softgoals have been sufficiently refined, it will be possible to *operationalize* these NFRs. During the refinement and *operationalization* steps, contribution and possible conflicts should be established, the impact of softgoals onto each other be defined and priorities be identified. The SIG created will be validated with the stakeholders.

The different degrees of desirability described above will be also used in association with the non-functional requirements.

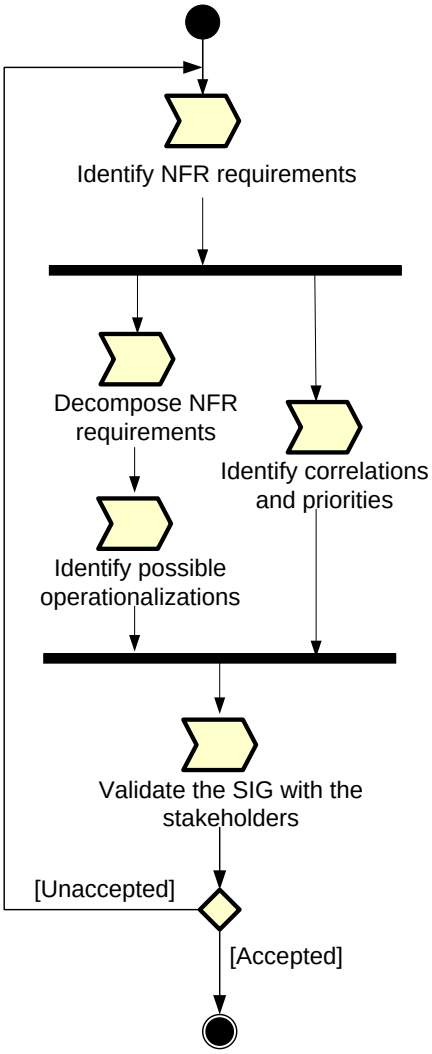


Figure 6.9: A SIG building process.

6.3.5 COTS Product Identification Phase

The main objective of this phase is to identify and find all suitable and potential COTS candidates. The main WorkDefinition of this phase is *Searching for the COTS candidates*. This WorkDefinition is driven by the evaluation criteria which takes as input high level requirements. It can be done by:

- searching the Internet for possible COTS candidates;

- getting recommendation of possible COTS candidates from external sources;
- reading product literature surveys and reviews.

At the first stage, the evaluation criteria do not need to be very detailed or formally defined but it must be unambiguous. Therefore, this WorkDefinition can be initiated as soon as the main features of the required components have been defined. The output of this WorkDefinition is a list of COTS candidates for each required component and a repository storing the *general information* of COTS candidates. The general information includes name, domain, version, vendor, cost, license, technology, etc.

It is quite possible that among the COTS products, some extra functionalities (not initially considered) may be available. These extra functionalities, upon a careful consideration, might indeed be required. This introduces changes to requirements. This feedback mechanism is then used to enhance the development process.

6.3.6 COTS Product Evaluation Phase

This phase focuses on the evaluation of the COTS candidates. The main WorkDefintion of this phase is *Evaluating COTS candidates*. We define two levels of evaluation: *local* and *global*. At local level, it concerns the evaluation of COTS products of each individual sub-system. The evaluated COTS products for each sub-system are classified according to the anticipated fitness of COTS products to the sub-system's requirements if their mismatches are resolved under limited resources. At global level, different combinations of COTS that make up the system are evaluated with respect to the COTS interoperability and interfacing effort and cost. This evaluation is necessary due to the fact that a product that fits in isolation might not be acceptable when combined with other products.

Figure 6.10 depicts the process of COTS evaluation at the local level. The process starts with the definition of a short-list criteria that will be used for filtering the search results. The goal is to reduce the number of candidates for the subsequent detailed evaluation and consequently reduce the time and effort needed for the whole evaluation.

The following is an example of short-list criteria:

- *Vendor size*. Is the vendor too big to pay attention to us? Or is the vendor too small to survive and provide consistent service?
- *Domain knowledge*. What are the target domain and market of the vendor? Do they correspond with the company's needs?
- *Consultant service*. Does the vendor provide consulting services? Does the vendor cooperate with the consultant companies?

- *Vendor's reputation and financial position.* Does the vendor have a good or bad reputation? Does the vendor have a good financial situation or show any sign of potential financial crisis?

- *Price of software.* Is the price of software acceptable?

If there are not any or too few candidates meeting the short-listed criteria, the criteria will be redefined until there is an acceptable set of short-listed candidates.

After eliminating the COTS products according to short-list criteria, the short-listed COTS components will be evaluated against the system requirements. More information on each candidate has to be gathered. This involves preparing a well-organized *Request for Proposal (RFP)* that effectively documents the organization's system requirements to the vendors. In our methodology, the RFP is generated from a goal graph created based on SIG and SR diagrams which respectively represent the non-functional and functional requirements of the system. The RFP will be then sent to the qualified vendors. We propose in Figure 6.11 a template for documenting the RFPs.

Based on the vendor RFP responses, the short-listed candidates are ranked on how well they fulfill the requirements and three to five best fitting candidates are selected to be further evaluated.

In our methodology, we suggest using the Weighted Score Method (WSM) decision-making technique (see Section 3.5.1) to filter the COTS candidates and present results that make sense. On the other hand, we suggest using the gap analysis approach [126] to evaluate the pre-selected candidates (see Section 3.5.3). This involves defining the mismatches between the system requirements and the COTS components as well as their impacts on the project and the possible actions to handle them.

The mismatches analysis is undertaken through component testing due to the fact that only the components testing that can allow us to understand better the pre-selected candidates and also to verify the vendors' claims. It also allows us to see the interactions between features and to detect the mismatches introduced by the interactions. Moreover, there are some evaluation criteria (e.g. security, accuracy, response time, etc.) that can only be verified at runtime. Components testing is time consuming, it is therefore recommended to avoid complex combinations of tests and the idea of "testing everything". Instead, it should focus on testing critical business processes.

On the basis of the RFP responses and components testing, mismatches of each pre-selected candidate are detected. The COTS component evaluation team also needs to decide how to handle them. The decision is made according to the type of mismatch. Based on the literature review, we find several types of mismatches as well as the recommended actions to handle them as listed in Table 6.2. In our methodology, if the recommended action involves adjusting the requirements, this implies some modification on the SIG and the SD and SR diagrams and the mismatch

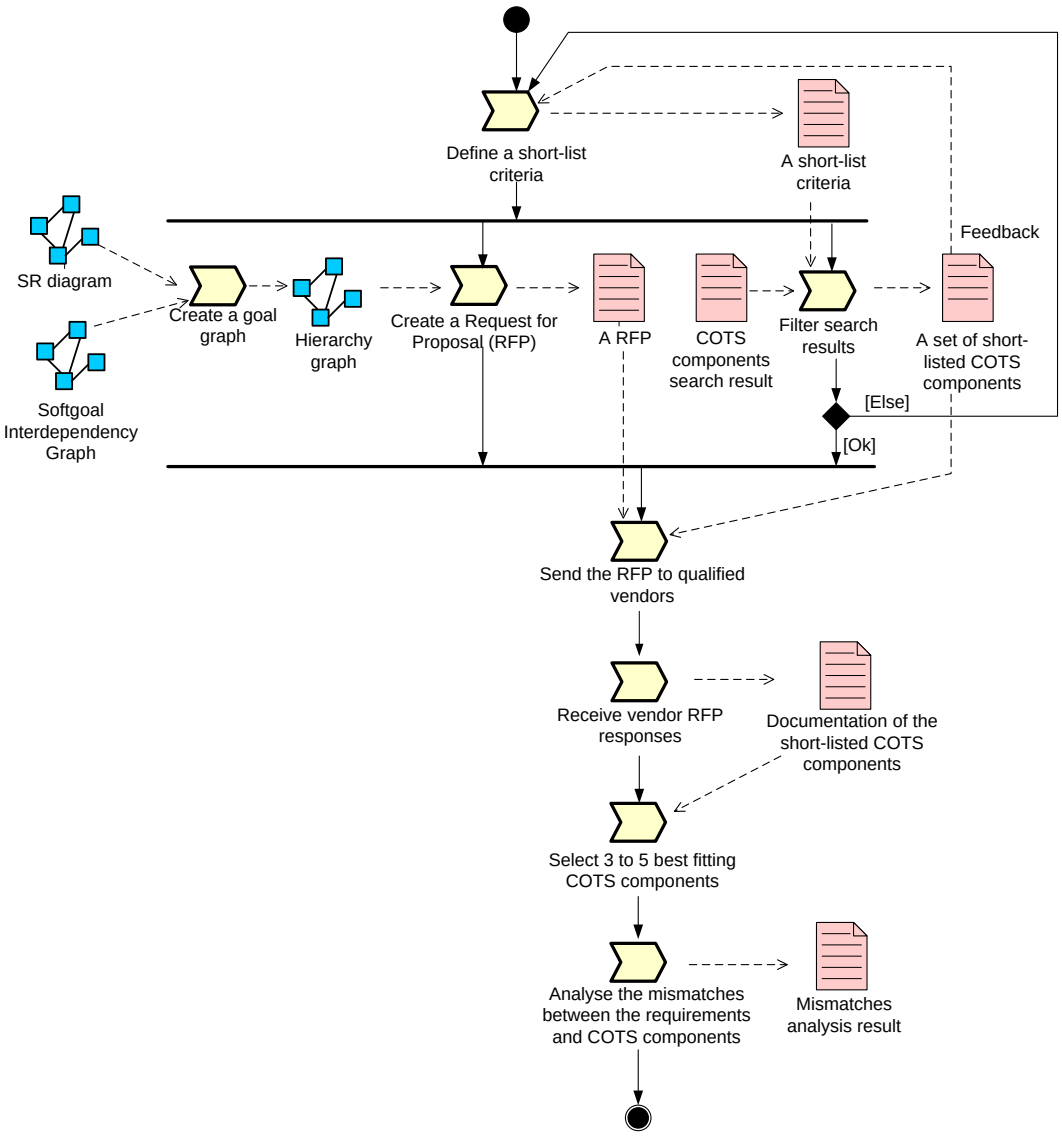


Figure 6.10: Local evaluation process.

is handled during the COTS selection process. Otherwise, if it involves tailoring the product, the cost of handling this needs to be estimated. In this way, features provided by the COTS components might influence the refinement of goals which lead to a very interactive and collaborative process. Moreover, the cost of mismatches handling after the COTS selection will also be taken into consideration for the COTS decision-making.

Hierarchy	Criterion	Priority (2-10)	SUP	MOD	3RD	CST	FUT	NS
1	**Goal 1**	**8**						
1.1	*Sub-goal of goal 1*	*4*						
1.1.1	Criterion 1	2						
1.1.2	Criterion 2	6						
1.1.3	Criterion 3	4						
1.1.4	Criterion 4	10						
1.1.5	Criterion 5	8						
1.1.6	Criterion 6	8						

Priority : 10: Very High 8: High 6: Medium 4: Low 2: Very Low	**SUP**: Supported as delivered "out-of-the-box" **MOD**: Supported via modifications (screen configurations, reports, GUI tailoring, etc) **3RD**: Supported via a third party solution **CST**: Supported via customization (changes to source code) **FUT**: Will be supported in a future release **NS**: Not supported

Figure 6.11: Request for proposal template.

6.3.7 Decision Making Phase

The main WorkDefinition of this phase is *Making decision*. This WorkDefinition involves making the decision on the COTS products selection as well as the action plan to handle the mismatches based the COTS evaluation results from the previous phase. The result of the evaluation is a technical factor for making the decision. However, the *economic factor* influences also this decision. The economic factor concerns *license acquisition cost*, *support and adaptation expenses*, *maintenance prices*, *COTS procurement conditions*, *vendor guaranties*, *availability of training*, *availability of vendors*, and *vendor's reputation and maturity*.

According to [34], using COTS components is the right solution when it lies at the intersection of the three determinants of feasibility as depicted in Figure 6.12. In other words, a COTS component is selected when it fits the project context:

- **Technically**. It has to be able to supply the desired functionalities at the required level of reliability;
- **Economically**. It has to be able to be incorporated and maintained in the target system within the available budget and schedule;
- **Strategically**. It has to meet the needs of the system environment including technical, political and legal consideration.

Table 6.2: Types of mismatches and recommended actions.

Mismatch		Description	Recommended Action
Fulfill conditional		When there is a constraint that has to be accomplished in order to fulfill the initial goal.	Verifying the goal desirability and the affect of constraint to other goals and the cost required to fulfill the constraint before deciding to accept or reject the product.
Fail		When the evaluated COTS does not provide any features to satisfy a goal.	Tailoring the product in order to add the feature that fulfills the goal or adjusting the requirements, it depends on the goal desirability.
Extend	Hurtful	When an extra feature has a negative impact over a particular goal.	Tailoring the product in order to disable the unwanted features.
	Neutral	When an extra feature does not interfere with the achievement of any goals nor the stakeholder want it.	Ignoring this mismatch.
	Helpful	When an extra feature is useful and can be included in the system requirements.	Adjusting the requirement by adding goals to the goal graph.
Differ		When the evaluated COTS partially satisfies a goal.	Tailoring the product or accepting the mismatches depending on their impacts on the project.

6.3.8 COTS Customization Phase

This phase involves customizing the selected COTS products to be used in the project context. This includes :

- parameter initialization, GUI screen and report layout configuration, security protocols set-up, etc.;

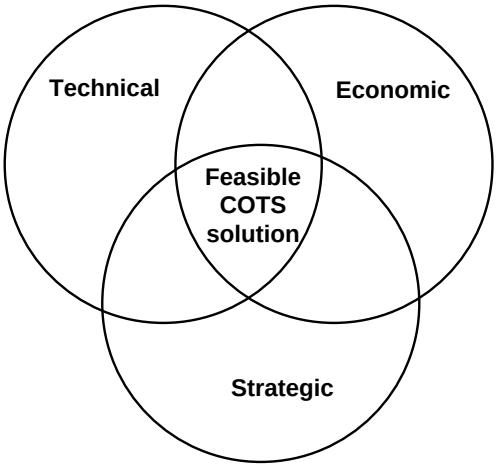


Figure 6.12: The determinants of a feasible COTS solution (from [34]).

- extracting source data from legacy databases, transforming source data into target data and loading target data into the target databases;
- handling mismatches.

Indeed, the selected COTS rarely match perfectly all the requirements and architectural constraints. Many of these mismatches are resolved after the COTS selection. There are different types of mismatches and they require different actions to handle them (see Table 6.2):

- *A fail mismatch.* When the selected COTS does not have any features to satisfy a particular goal. Developers will develop the relevant functionalities fulfilling the desired goal;
- *A hurtful extend mismatch.* When an extra feature has a negative impact over a particular goal. Developers will disable this feature;
- *A differ mismatch.* When the selected COTS partially satisfies a particular goal. Developers need to find the causes of mismatch and their possible solutions. We propose an approach for analyzing such mismatches that will be described in the rest of this section.

Our approach is a systematic, repeatable, agent-oriented process. Figure 6.13 depicts the relevant concepts of our approach and their dependencies using a UML class diagram notation. The model is structured as following: the agent pursues a set of goals. Each goal is realized through a series of capabilities executed by agents in the form of a realization path represented by a Dynamic Diagram proposed in [60]. In our context, we consider a capability as a high-level function that can be

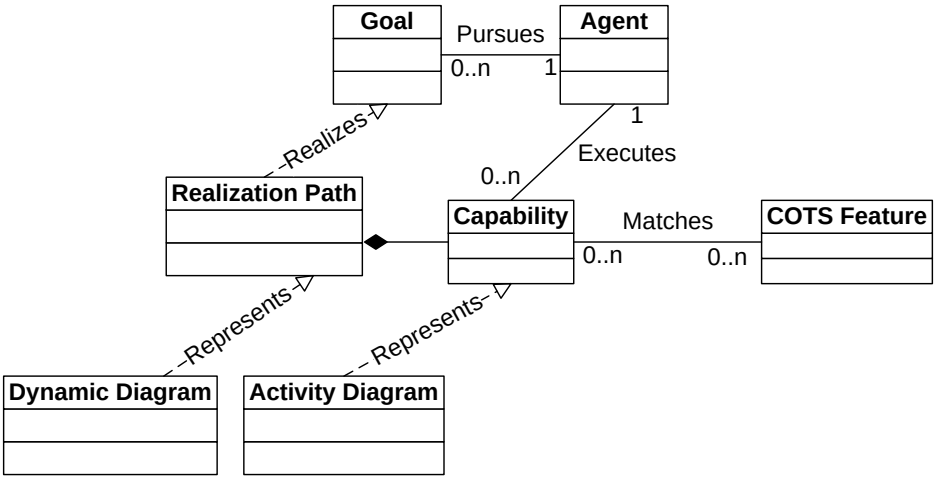


Figure 6.13: Conceptual model of the proposed differ mismatch analysis approach.

matched to COTS features as defined in [10]. At a lower level, each capability could be modeled by a UML activity diagram for further detail if necessary.

The process of the proposed approach for analyzing the differ mismatches of the selected COTS consists of the two WorkDefinition:

- **Defining realization path.** For each differ mismatch that will be handled by the COTS customization, we define the realization path for each partially satisfied goal. A realization path is a series of capabilities executed by agents in order to realize a goal. The corresponding capabilities will be listed in a table with a name, informal definition and the name of the agent the capability belongs to;

- **Defining cause and resolution.** Based on the realization path and the documentation of the selected COTS features provided by the COTS vendor and resulted from COTS feature testing, we define the causes of mismatches and resolution to handle them. Causes of mismatches can come from not only mismatches between COTS features and desired capabilities, but also interactions between COTS features. Indeed, it is possible that desired capabilities for realizing a goal can be matched with the COTS features but undesired interactions between these features make goal partially satisfied. Resolutions are envisaged on a case by case basis.

6.3.9 COTS Integration Phase

This phase focuses on the system integration. It consists of developing glueware which is a type of software that can be used to "glue" or integrate software components to form a seamless integrated system. In our methodology, this involves

implementing our proposed MAS (see Section 5.1 for the architectural description of the proposed MAS).

6.4 Ontology Alignment

The thesis deals with 3 different ontological frames:

(i) The first one is concerned with the problem analysis and represented using the i* models with respect to the specifications we have defined in Section 6.2;

(ii) The second one is concerned with the solution design and represented using the custom syntax and semantics defined in Chapter 4 and Chapter 5;

(iii) The third one is concerned with the description of the software process itself and represented using the syntax and semantics defined by the Software Process Engineering Meta-Model (SPEM) [131].

Although the process description – which is an instantiation of (iii) – explicitly links the analysis models (i) and design ones (ii); higher integration could be brought through a formal alignment study between the elements of these two latter ontologies. More precisely, such a study would allow using the instance of the defined elements of the ontology for problem analysis (i) to instantiate defined elements of the design ontology. This would notably be useful for having the highest possible alignment between concepts issued of the application domain and the software design. In line with this, partial forward engineering abilities could be automated. Such a research nevertheless remains the subject of future research.

On the other hand, the alignment of (i) and (ii) with (iii) is a simple issue since model elements of (i) and (ii) or entire models of the same frameworks are (and can only be) represented in (iii) as *WorkProducts*. Indeed, (iii) is a meta-process description language intended for process engineers that, through an instantiation, is used to describe a particular software development process. Ontologies for describing the problem analysis (i) or the solution design (ii) are thus artifacts of the process input or output to human activities only. The i* models presented in the process have to be created according to the i* ontology we have defined in 6.2. Similarly, the ontology of COTS we have defined in Section 4.2 specifies the information of each COTS component that the project team needs to know during the development process.

6.5 Chapter Summary

Firstly, this chapter has presented the shortcomings we can draw from our literature review on the CBSD and the specifications for our methodology with respects to these shortcomings.

Secondly, it has presented the adaption of agent-oriented techniques for system analysis and modeling for our methodology.

Thirdly, it has described the process development of our methodology. It is an iterative cyclic process wherein each cycle produces a deliverable increment of the software with new COTS added to the system. A project may comprise several development cycles. A cycle of our process development is composed of seven iterative phases including *Business Analysis*, *Requirements Analysis*, *COTS Product Identification*, *COTS Product Evaluation*, *Decision Making*, *COTS Customization* and *COTS integration*. A generic description and WorkDefinitions for each phase have been given. For some WorkDefinitions, guidelines to perform them have been provided.

Finally, it has exposed the ontology alignment in our methodology.

Chapter 7

Methodology Application

This chapter proposes an application of the process presented in the previous chapter for the purpose of validation. It has been done on a case study issued of supply chain management and more particularly outbound logistics (OL). In this case study, we consider COTS as a third party component used to build the global system. We firstly describe the outbound logistics. Next, we present the application of our proposed methodology.

7.1 The TransLogisTIC Project

In august 2005, the *Belgian Walloon Region* introduced the *Marshall Plan*, a vast economic investment to reinforce attractiveness and competitiveness of Walloons companies in order to raise employment rate. This plan notably integrates poles of competitiveness, and, among these poles, *Logistics in Wallonia* created in July 2006, which is concerned with transportation and logistics. Into this latest pole, *TransLogisTIC*, proposed as a driving project, has officially started in March 2007.

Currently, the real-time visibility of information flows throughout the whole OL chain fails to ensure competitive integrated logistics. However, decisions at European level request for the development of multi-modal transportation, notably through Eurocorridors as the "C" (Antwerp - Bale - Lyon) which would cross the Belgian Walloon area. For this reason, the TransLogisTIC project has been built up, with, as long-term main objective to develop combined, performing and complete transportations in Walloon Region with transport by rail particularly promoted in accordance with the European policy (see Marco Polo program). The project has initially been planned on a 3 year basis with 14 million euro as overall budget and has involved several complementary actors including 10 private companies and 5 universities and research labs.

The analysis of the socio-economic context in which TransLogisTIC takes place has led to various concrete objectives, among which:

- The development of control systems tailored to the rail freight expectations;
- The development of real time innovative localizing solutions covering the whole OL chain based on a new generation of positioning systems as well as on the optimal and extended use of existing infrastructures;
- The development of an online collaborative logistic platform allowing chargers, carriers, infrastructure managers and final clients (i.e. the major OL actors) to share information for a better optimization of the logistic chain.

Our case study exclusively focuses on the last point especially through the work developed by the Center for Supply Chain Management (CESCM) at the Louvain School of Management (LSM) of the Université catholique de Louvain (UCL).

7.2 RecIProC Application

This section presents an application of RecIProC onto the development of an outbound logistics collaborative platform.

7.2.1 Business Analysis Phase

This phase consists of three main WorkDefinitions including *Studying the business context*, *Defining the scope of the project*, and *Analyzing strategic impacts*. These WorkDefinitions are mainly performed during the first cycle of the project. During the following cycles, they are performed to handle the business changes. The result from the business analysis phase serves as a basis for the following phases.

7.2.1.1 Studying the Business Context

The business context as described in this section is based on the interviews with a set of actors that involved in the OL chain. These interviews have mainly focused on business processes and the main objectives of the project.

OL is the process related to the movement and storage of products from the end of the production line to the end user. In the context of our work, we mostly focus on transportation. The actors of the supply chain play different roles in the outbound logistics flow. The producer will be a logistic client in its relationship with the raw material supplier, which will be consider as the shipper. The carrier will receive transportation orders from the shipper and will deliver goods to the client, while relying on the infrastructure holder and manager. In its relation with the intermediary wholesaler, the producer will then play the role of the shipper and the wholesaler will be the client.

Figure 7.1 summarizes the material flows between the actors of the outbound logistics chain. The responsibilities corresponding to the different roles are:

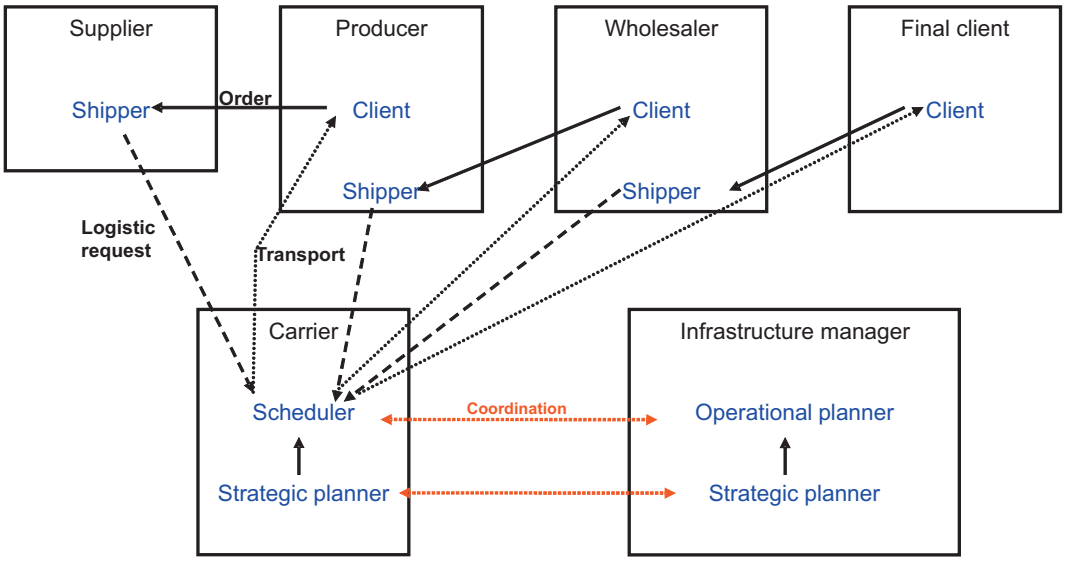


Figure 7.1: Material flows in the outbound logistics chain.

- **The final client**: This role is taken by an actor who has ordered a shipment from a supplier and awaits its delivery. The client is generally not involved in the choice of the carrier, but is impacted by the planned delivery date and the delays that could result in the realization of the transportation.

- **The shipper**: This role is taken by an actor who needs transportation for a shipment. The shipper has received an order from a client and has to deliver this order. The shipment has its own characteristic in term of volume, transportation constraints, pickup and delivery dates, transportation mode, and transportation price. The shipper will turn toward one or several carriers to realize that shipment.

- **The carrier**: This role is taken by a actor owning transportation capacity who receives transportation orders from shippers for the realization of shipments to final clients.

  - **The carrier's strategic planner**: decides on the services that are offered on the long term, on the use of infrastructure, on the logistic resources to hold and on the client's acceptance conditions.
  - **The carrier's scheduler**: orders transports to be realized, according to the strategic network and constraints, coordinates with the infrastructure manager and assign logistic requests to those transports such that the delivery requirements are met.

- **The infrastructure manager**: This role is taken by an actor who owns the logistics infrastructure that is needed by carriers such as hubs, rail tracks, airports and docks. The infrastructure manager coordinates with the carrier's scheduler to offer the network for the planned transports.

The idea behind the system development is to favor these actors' collaboration. Indeed, collaborative decision will tend to avoid local equilibriums (at actor level) and wastes in the global supply chain optimization, giving opportunities to achieve the greatest value that the chain can deliver at lowest cost. The collaborative application package to develop is thus composed of a multitude of aspects including the development of applications and databases to allow the effective collaboration and the use of flexible third party components providing well identified services.

7.2.1.2 Defining the Scope of the Project

The main objective of this WorkDefinition is to build a *Strategic Services Diagram (SSD)* representing the big picture of the project. The SSD of Figure 7.2 overviews the outbound logistics in terms of organizational services. All outbound logistics actors depicted in Section 7.2.1.1 are represented as actors; COTS components we want to integrate in the global applicative package are also represented as an actor. These COTS components are listed as follows:

- **The Fleet Management System (FMS)** is computer software that enables people to accomplish a series of specific tasks in the management of a company's vehicle fleet. It can include vehicle telematics (tracking and diagnostics), driver management, fuel management, vehicle maintenance and so on;

- **The Warehouse Management System (WMS)** is a key part of the systems managing the supply chain. It aids in controlling the movement and storage of materials within a warehouse and processing the associated transactions, such as shipping, receiving, putaway and picking;

- **The Enterprise Resource Planning (ERP)** is an enterprise-wide information system designed to support most of the business system. It maintains in a single database the data needed for a variety of business functions such as Manufacturing, Financial, Projects, and Human Resources;

- **The Transportation Management Systems (TMS)** is computer software designed to manage transportation operations. It aids in determining the most efficient and most cost-effective way to execute the movement of product(s). The TMS will be further described in the case study., it includes various functions such as:

- Planning and optimizing of terrestrial transport rounds;
- Transportation mode and carrier selection ;
- Real time vehicles tracking ;
- Service quality control ;
- Vehicle load and route optimization;
- Transport costs and scheme simulation.

Each service of the SSD depicted in Figure 7.2 represents a complex work-flow summarized below:

- The **Treat Orders** service represents the process to plan the coming material flows by building up *Logistic Requests* on the basis of the customers' orders. *The Shipper's Expedition Representative* is the responsible actor;
- The **Treat Expeditions** service represents the process to build *Transportation Calls* on the available *Logistic Requests*. *The Shipper's Expedition Representative* is the responsible actor;
- The **Plan Logistic Requests** service represents the process to evaluate the possibilities (eventually by relaxing constraints) of accepting the transmitted *Transportation Calls*. *The Carrier's Orders Representative* is the responsible actor;
- The **Manage Transports Services** service represents the process to update the *Carriers' Transportation Services* offer in function of the general environment (demand and capacity on particular origins and destinations, new transportation possibilities, etc). The *Carriers' Planner* is the responsible actor;
- The **Manage Transports** service represents the process to plan the *Carrier's Physical Transports* linked to its *Transportation Services* to answer to the accepted *Transportation Calls* demand. The *Carrier's Scheduler* is the responsible actor;
- The **Manage Resources** service represents the process to ensure an adequate planning of required resources (working teams, docks, cranes, etc.) for the planned *Transports*. The *Infrastructure Manager* is the responsible actor;
- The **Track Transports** service represents the process to real time track transported goods and performs a series of activities such as dynamic replenishments, alternative route calculation, etc. The *Actor's TMS System* is the responsible actor;

- The **Manage Fleet** service represents the process to accomplish a series of specific tasks in the management of a carrier vehicle fleet as for example telematics (tracking and diagnostics), driver management, fuel management, vehicle maintenance and so on. The *Actor's FMS System* is the responsible actor;
- The **Manage Warehouse** service represents the process to manage the movement and storage of materials within the warehouse and processing the associated transactions, such as shipping, receiving, putaway and picking. The *Actor's WMS System* is the responsible actor;
- The **Transfer Orders** service represents the process to manage all the input data flows that must be transferred from the ERP system or to the actor ERP system. The *Actor's ERP System* is responsible actor.

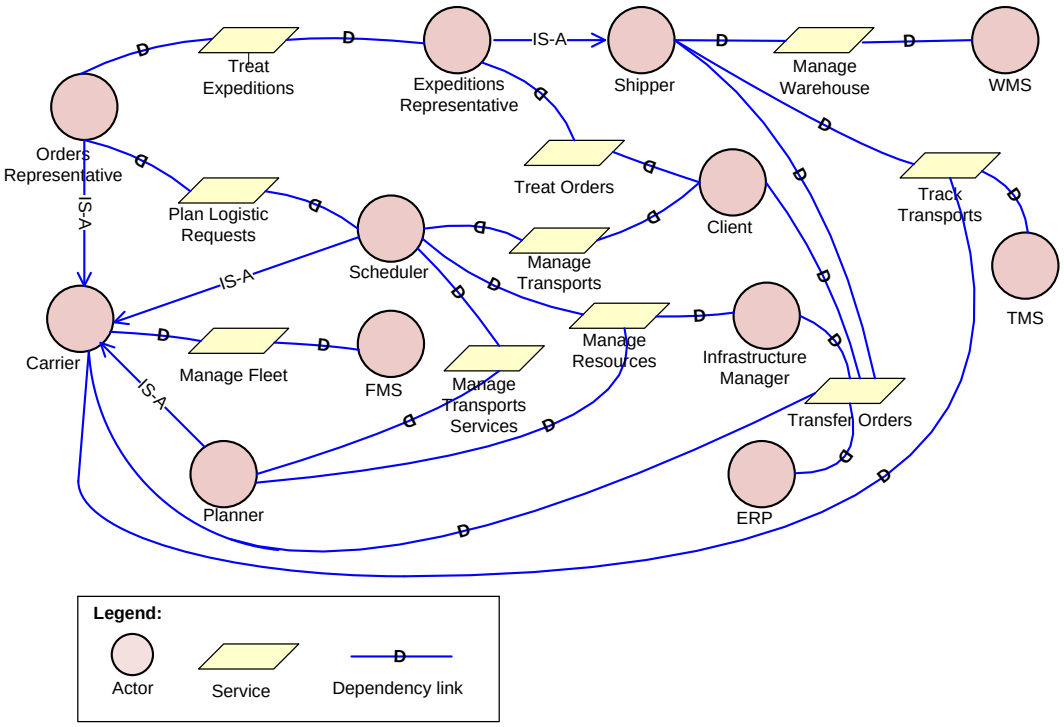


Figure 7.2: Strategic service diagram for outbound logistics.

7.2.1.3 Analyzing Strategic Impacts

The objective of this WorkDefinition is to analyze the strategic impacts in term of threats and opportunities that the project can face. For this purpose, we mainly

study the long-term strategic impact of adopting the collaborative software package through its relevant services for the concerned actors. Indeed, e-collaboration in outbound logistics can potentially deliver series of advantages as information sharing, real-time decision making, online auctions and notably global optimization. Nevertheless, this advantage and others can have drawbacks for some of the concerned actors and they may thus not want to adopt it or, in case of adoption, have disastrous consequences on their businesses.

Global optimization cannot be introduced as an opportunity to the involved actors. However, that target has a particular interest for the environment in the sense that optimized transports leads to less wastes which lowers negative externalities. Indeed, since a variety of stakeholders are involved into the package, each of them must be demonstrated its "local" advantage to adhere. More precisely, we identify here the following opportunities and their associated weight:

- **Real-Time Information Transmission.** Information has become a strategic resource of our modern times so that information availability through adequate transmission represents a strategic tool for each of the actors involved in the supply chain. The applicative package thus represents an opportunity since it enables to lower information transmission time. This opportunity has been given a weight of 5;

- **Business Process Optimization.** Re-engineering business processes notably through standardized information transmission can lead to avoid wasting resources (among which human ones) so that time and money can be spared. This opportunity has been given a weight of 4;

- **Symmetric Information.** The use of the applicative package ensures that when an actor has a specific demand of services it is transmitted to whole of the concerned actors. For example the shippers' logistic requests are transmitted to whole of the carriers so that each of them can propose its transport services until an agreement has been found. Information symmetry tends to lower the eventuality that suppliers are unfairly favored. This opportunity has been given a weight of 3;

- **Unified Data Structures.** The use of unified data structures ease the everyday communications between supply chain actors and constitute a standardized formalization that can serve as a legal basis. This opportunity has been given a weight of 2;

- **Integrated Information Portal.** The use of a single portal for managing all the aspects of an actors' business eases the everyday transactions and lowers maintenance costs so that resources can be spared. This opportunity has been given a weight of 1.

The opportunities/services matrix presented in Figure 7.3 allows determining a hierarchy among the services to determine the one which could benefit of the most added value by adopting the collaborative software package solution. The *Manage Transports* and *Manage Resources* services have an added value of 60 which underlies the solution could potentially bring the most added value for carriers and infrastructure managers. The matrix also shows that the identified opportunities are adding substantial value to all of the collaborative application package's services so that each of the collaborating actors can potentially have an interest by the adoption of the solution which however has only sense if all of them use it or at least feed it with data flows.

	Opportunity weight		Treat Orders	Treat Expeditions	Plan Logistic Requests	Manage Transports Services	Manage Transports	Manage Ressources	Manage Warehouse	Track Transports	Manage Fleet	Transfer Orders
Real-Time Information Transmition	5		H	M	H		H	H	M	H	M	M
Business Process Optimization	4		H	H	H	L	H	H	H	L		
Symetric Information	3		M	M	M	M	H	H				
Unfiied Data Structures	2		L	H	H	M	H	H	M	M	M	H
Integrated Information Portal	1		H	H	H	H	H	H	M	M	M	M
Overall Service Opportunity Added Value			48	44	54	18	60	60	32	30	16	20

Figure 7.3: The opportunities/services matrix.

Global optimization and the adoption of a global applicative package can however have a series of drawbacks. These are studied hereafter in the form of threats. More precisely, we identify here the following threats and their associated weight:

- **Loss of Local Optimality.** Actors are continuously seeking for optimal solutions within their business processes especially when competition is high and concerns rather standardized services as for example for carriers. A global optimization process could lead actors of the supply chain to a worse situation that when they only managed their own business strategy so that they would reject such a tool. This threat has been given a weight of 4;

- **Loss of Local Autonomy.** The autonomy of an actor refers to its capability

to make and influence decisions. The provided information of an actor to the other can lead its autonomy to be adjusted in one sense or the other so that it can be an opportunity or a threat (we nevertheless only consider the "threat side" here). In other words, increased transparency can lead to a loss of autonomy. This threat has been given a weight of 3;

- **Transmission of Strategic Information.** By advertising all of the transportation offer and demand, one could determine information that the company's would prefer to remain confidential. This threat has been given a weight of 3;

- **System Intrusion.** An intrusion takes place when a user of an information system takes an action that he is not legally allowed to take. The intruder may come from outside, or it may be an insider exceeding his limited authority when taking action. The intruder's actions may be detrimental to the health of the system or to the services it offers. This threat has been given a weight of 2;

- **Data loss.** Facing huge data transfers between actors some required data can be lost or never furnished. This threat has been given a weight of 2;

- **System Failure.** The information system may be down and cannot be used for some time. This threat has been given a weight of 2.

The threats/services matrix presented in Figure 7.4 allows determining a hierarchy among the services to determine the one which is the most exposed to risks. The *Manage Transports* service has an overall risk of 46 which underlies risks are potentially highest for carriers. The matrix also shows that the identified threats have an impact on all of the collaborative application package's services so that each of the collaborating actors has to overview the drawbacks and compare them to the potential benefits to run its cost/benefits study.

The strategic analysis of the identified services leads to the following conclusions:

- adopting the collaborative software applicative package with the services defined in the SSD presents opportunities and risks for all of the identified actors;

- carriers have the highest implication since they can take the highest added value of adopting the services but also face highest risks. Adopting the solution can thus be of primary interest for them but they have to set up operational solutions to avoid that threats become executed which has an operational cost. A more sophisticated cost/benefit can then be run by each of the companies playing the role of an actor on the basis of the services subscription fees, opportunities wins and threats'operational costs.

	Threat weight		Treat Orders	Treat Expeditions	Plan Logistic Requests	Manage Transports Services	Manage Transports	Manage Ressources	Manage Warehouse	Track Transports	Manage Fleet	Transfer Orders
Loss of Local Optimality	4			L		H	H	H	H	H	M	L
Loss of Local Autonomy	3			L		H	H	H	M	H	H	L
Transmission of Strategic Information	3		M	H	H	H	M	L	M	M	L	L
System Intrusion	2		M	M	M	M	H	M	M	M	L	M
Data Loss	2			L			L	L			L	L
System Failure	2		L	L	L		L	M	M	M	M	
Overall Service Risk Exposure			14	27	18	44	46	41	36	42	31	16

Figure 7.4: The threats/services matrix.

7.2.2 Requirements Analysis Phase

This phase focuses on defining the system requirements. It consists of three main WorkDefinitions including *Defining the integrated system architecture*, *Defining the functional requirements of each required system*, and *Defining the NFRs of each required system.*

7.2.2.1 Defining the Integrated System Architecture

Different software components involved in the outbound logistics system will be integrated within well-defined system architecture. In this case study context, the outbound logistics system architecture is an instance of our proposed architecture for COTS integration (see Section 5.1).

Figure 7.5 depicts the social dimension of the MAS layer of the outbound logistic system. As we can see in the figure, each outbound logistics system's constituting components is wrapped inside a wrapper agent. There thus four wrapper agents: `ERP`, `FMS`, `WMS` and `TMS`. There is a user agent representing each user; it can be a `Client`, a `Shipper`, a `Carrier` or `Infrastructure manager`, for the three main roles involved in the outbound logistics system.

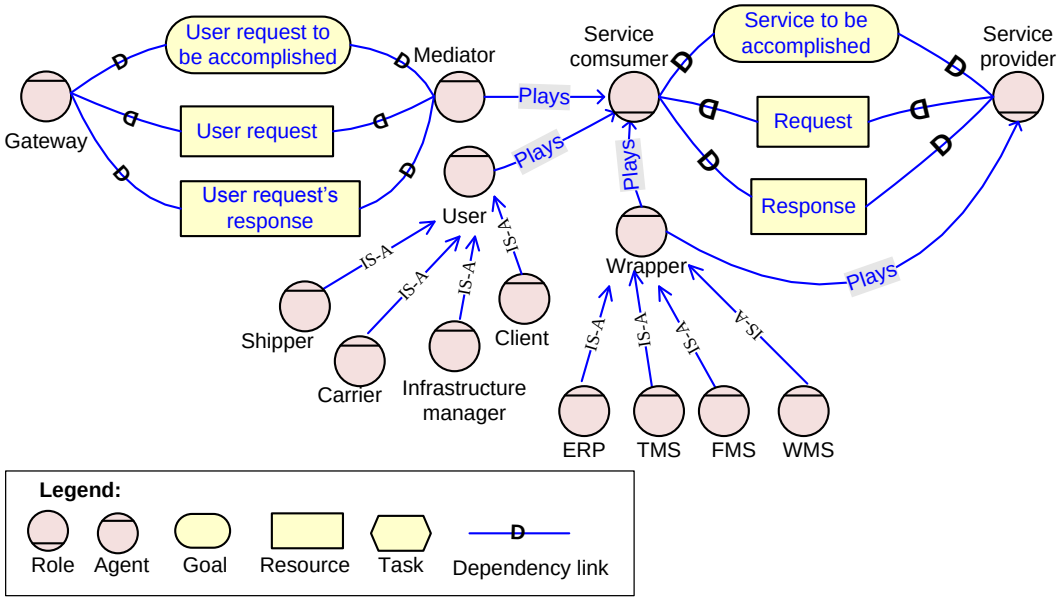


Figure 7.5: Social dimension of the MAS layer for the outbound logistic system.

7.2.2.2 Defining the Functional Requirements of Each Required System

In our framework, the SD and SR diagrams are used to define the functional requirements of each system-level component identified and modeled in the SSD. Figure 7.6 documents the strategic dependency diagram issued of the organizational modeling and requirements engineering for developing the service *Track Transports* offered by the TMS. It depicts the relevant actors and their goals, tasks and resources dependencies involved into the *Track Transports* service.

As illustrated on the SDD of Figure 7.6, TMS components are involved in the realization of the goals:

- *Vehicle load and route to be optimized*, *Events to be handled* and *Transports to be followed up* for the carrier;
- *Carrier to be located* for the shipper;
- *Delivery date to be informed* for the client.

The desirability of each goal is listed in Table 7.1. These goals must be fulfilled otherwise the success of the project will be strongly compromised. These goals will be used to identify and filter the COTS TMS available in the market.

Table 7.1: The desirability of goals to be fulfilled by TMS.

Goal	Desirability
Vehicle load and route to be optimized	Very high
Events to be handled	Very high
Transports to be followed up	High
Carrier to be located	High
Delivery date to be informed	High

The SRD of Figure 7.7 further documents the actors' rationale. Notably, we find the task *Select most adequate transport*, which is aimed to select the best fitting transport offer on the basis of some defined requirements and under defined constraints.

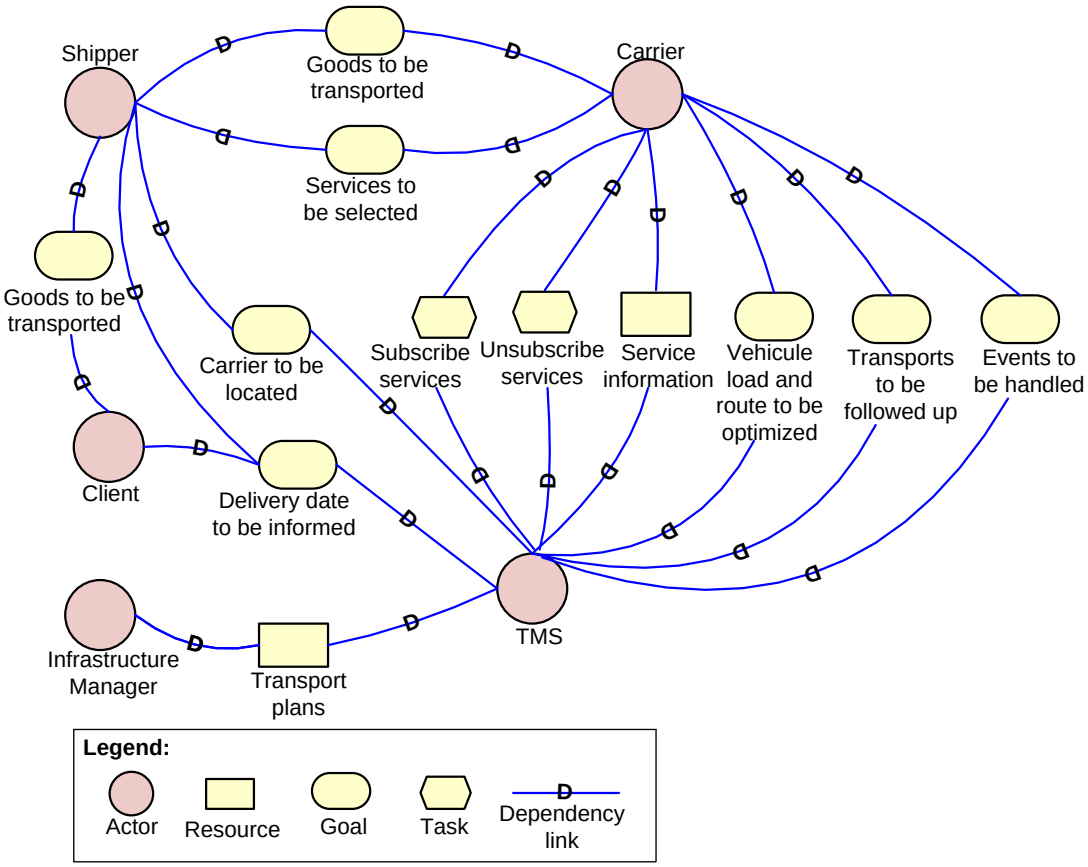


Figure 7.6: Strategic dependency diagram modeling *Track Transport* service.

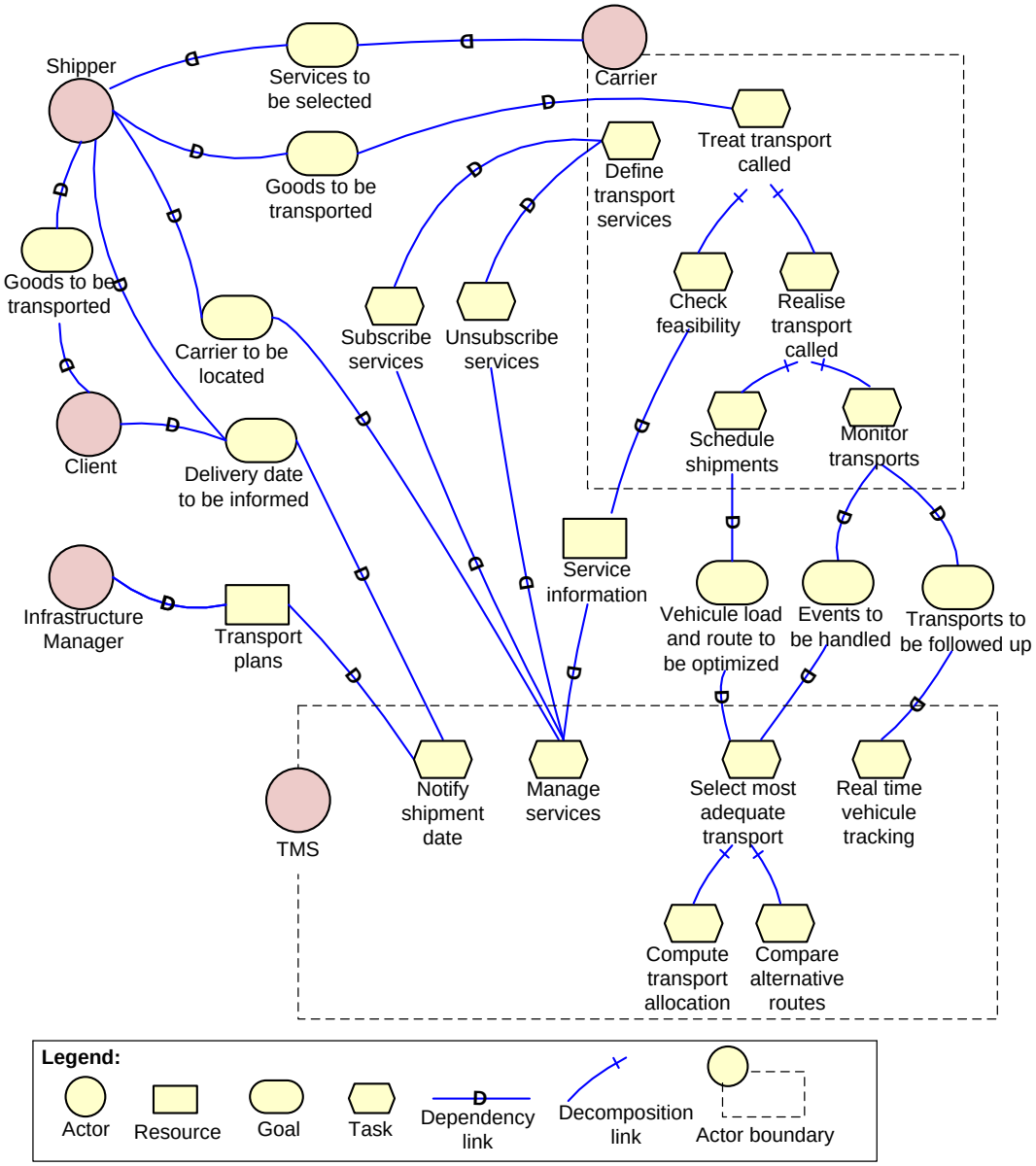


Figure 7.7: Strategic rational diagram modeling *Track Transport* service.

7.2.2.3 Defining the NFRs of Each Required System

The NFR goal graph of Figure 7.8 represents the non-functional aspects that the TMS should fulfill or at least contribute to. Non-functional requirements include *Security*, *Flexibility* and *Adaptability*.

After posing these highest level non-functional requirements as softgoals to satisfy, we tend to refine them into sub-goals through `AND` and `OR` decompositions and the interdependencies among the goals are also studied as shown in Figure 7.8. The NFR *Security* was refined into *Authorization* and *Data encryption*. The NFR *Flexibility* was decomposed into *integrability of data base*, *wrappability of product* and *extensibility of new modules*. The NFR *Flexibility* has a positive influence to the NFR *Adaptability* which has been decomposed into *vendor support*, *developed in a standard technology*, *source code openness*, and *configuration*. The *authority to read source code openness* operationalize goal can partly satisfy the *source code openness* goal which can be fully fulfilled by the *authority to modify code* operationalize goal.The desirability of different goals is listed in Table 7.2. The NFRs will be used to evaluate the COTS candidates.

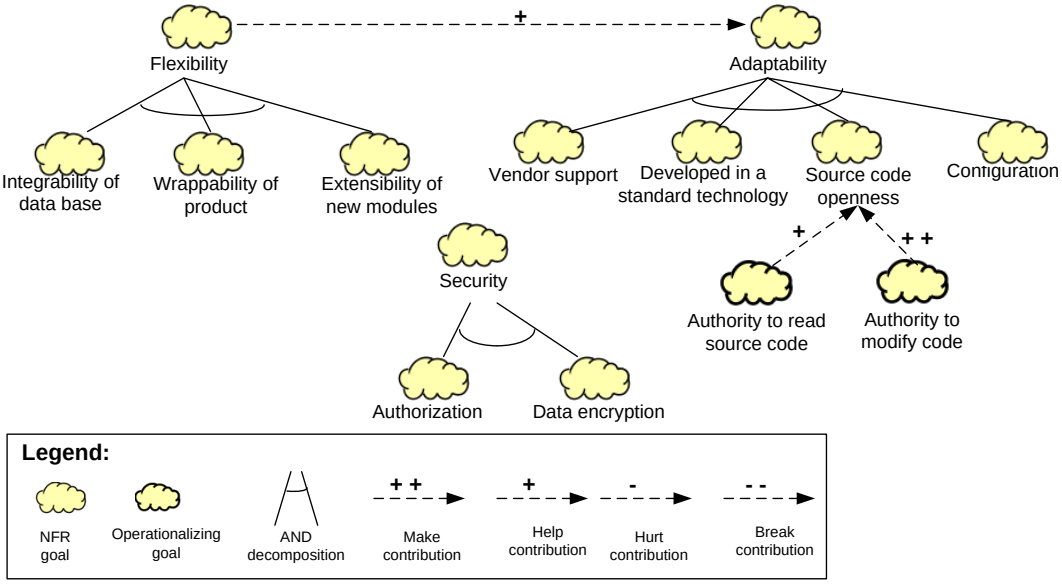


Figure 7.8: NFRs of the TMS.

7.2.3 COTS Product Identification Phase

Several information sources, such as online search engines (e.g. Google, Yahoo), as well as specialized websites that list COTS software packages (e.g. www.vendor-showcase.com) can be used to identify possible COTS candidates. Table 7.3 lists some relevant TMSs found in the market.

Table 7.2: The desirability of global NFRs.

Goal	Desirability
Authorization	High
Data encryption	High
Integrability of database	Very high
Wrappability of product	Very high
Extensibility of new modules	High
Vendor support	High
Developed in standard technology	High
Authority to read source code	Low
Authority to modify code	Low
Configuration	High

Table 7.3: TMS products found in the market.

Product	Vendor	Vendor's website
Oracle Transportation Management	Oracle	www.oracle.com
IBM Sterling Transportation Management System	IBM Sterling Commerce	www.ibm.com
SAP	SAP Transportation Management	www.sap.com
TRIS TMS	INTRIS Group	www.intris-group.com
i2 Transportation Solution	i2 Technologies	www.i2.com
Control Tower	Informore	www.informore.com
Transportation Lifecycle Management System	Manhattan Associates, Inc.	www.manh.com
NaviTrans Transport	Young & Partner	www.youngpartners.com

7.2.4 COTS Product Evaluation and Decision Making Phases

During the evaluation phase, the project team involves filtering search results according to short-list criteria and those that do not provide any possibilities to fulfill the very high desired goals, and evaluating the pre-selected COTS products. The result of the evaluation forms a technical factor for the project team to select a COTS product. However, the economic factor influences also this decision.

For illustration, in this case study we suppose that a COTS TMS is selected. Table 7.4 documents the goals that the selected TMS cannot totally fulfill and the

action to handle the mismatches. Mismatches that will be handled by tailoring the TMS product will be further analyzed in customization step of the COTS-based system development life cycle.

Table 7.4: Mismatches documentation of the selected COTS TMS.

Mismatch goal	**Mismatch type**	**Description**	**Action**
Transport load and route optimization	Differ: Process level	Very strong impact on the success of the project since it concerns the very high desired goal.	Tailoring TMS product
Event handling	Differ: Process level	Very strong impact on the success of the project since it concerns the very high desired goal.	Tailoring TMS product
Transports to be followed up	Differ: Process level	Very strong impact on the success of the project since it concerns the very high desired goal.	Tailoring TMS product
Carrier to be located	Differ: Parameter level	Not strong impact on the success of the project.	Tailoring TMS product
Developed in standard technology	Differ	Some non-standard data types are used but it is acceptable.	Ignoring
Authority to read source code	Fail	Source code cannot be read by the users.	Ignoring
Authority to modify code	Fail	Source code cannot be modified by the users.	Ignoring

7.2.5 COTS Customization Phase

In this case study, we focus on the application of our approach for analyzing the differ mismatch. For illustration, we present the differ mismatch analysis of the *Transport load and route optimization* goal. As illustrated in Figure 7.7, this goal is realized by the *Select Most Adequate Transport* task. We therefore define its realization path as depicted in Figure 7.9. The capabilities involved in this realization path are listed

in Table 7.5. In our

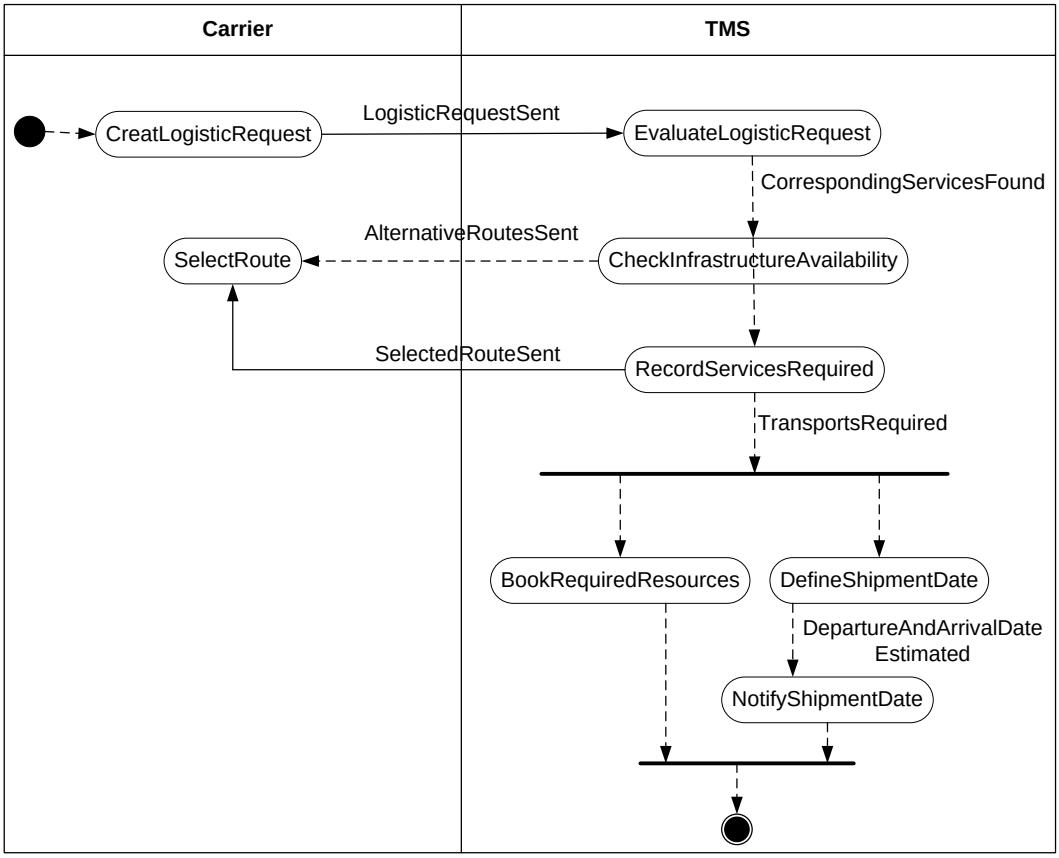


Figure 7.9: Realization path of the *Select most adequate transport* task.

Based on the selected COTS TMS's documentation acquired from the vendor and functional testing, we find that the selected COTS TMS does not provide the features to check the resources availability and to book the required resources at the infrastructure managers *CheckInfrastructureAvailability* and *BookRequiredResources* features.

Figure 7.10 illustrates the selected COTS TMSs scenario for the *Select most adequate transport* task. Consequently, the two evoked features have to be developed for adequate inclusion into the software component so that it can achieve the documented success scenario. The implementation of this inclusion is not in the scope of our research since it is envisaged on a case by case basis. For the other features, the COTS TMS does already provide the same functional behavior and can be used as such or if required overloaded.

Table 7.5: Capabilities relating to the realization path.

Capability name	Informal definition	Agent
`CreateLogisticRequest`	Creating a logistic request.	`Carrier`
`EvaluateLogistic-Request`	Computing the alternative routes to fulfill the logistic request.	`TMS`
`CheckInfrastructure-Availability`	Checking the availability of the resources used by each alternative route before sending possible alternative routes fulfilling a logistic request to the carrier.	`TMS`
`SelectRoute`	Selecting an alternative route	`Carrier`
`RecordServiceRequired`	Recording the required resources on each specific connection of the selected route.	`TMS`
`BookRequiredResources`	Booking the required resources to fulfill the transport at the corresponding infrastructure managers.	`TMS`
`DefineShipmentDate`	Evaluating the departure and arrival date.	`TMS`
`NotifyShipmentDate`	Notifying the shipment date to corresponding infrastructure managers, shipper and final client.	`TMS`

7.2.6 COTS Integration Phase

In our methodology, this phase involves implementing our propose MAS (see Section 5.1. By following the implementation model that we has proposed in section 5.2, the developers primarily need to:

- specify the `Gateway` agent in servlets that will connect with the MAS;
- define the user requests that can be handled by the MAS in an XML file (see code extract 5.16 for the sample);
- define the business rules to be associated with the agents' capabilities;
- define in XML files the services offered by each `Service provider` agent, i.e. `ERP`, `TMS`,`FMS` and `WMS` agents (see code extract 5.24 for the sample);
- define the BeanShell scripts to be executed in the `RealizeRequest` behaviors;
- define the proactive actions for each user agents, i.e. `Shipper`, `Carrier` and `InfrastructureManager` agents.

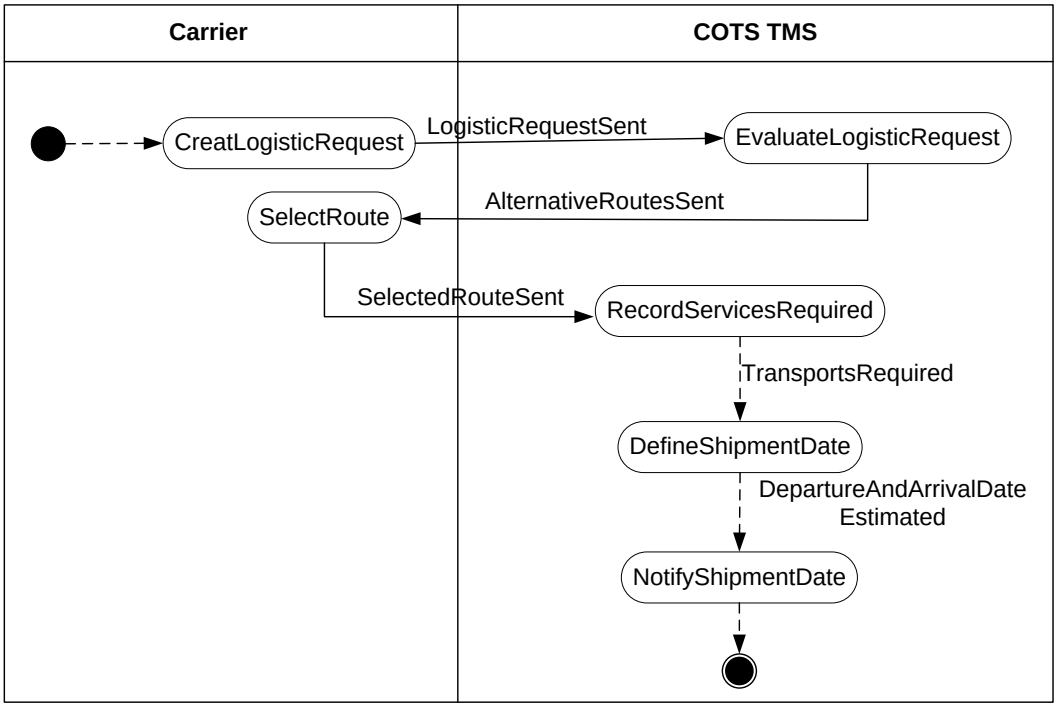


Figure 7.10: The selected COTS TMS's scenario for the *Select most adequate transport* task.

7.3 CASE Tools

Computer-Aided Software Engineering (CASE) tools are a kind of software that provide the assistance to perform the activities involved in various life cycle phases of a project. In this section, we describe the CASE tools that can be used to support some WorkDefinitions in our COTS life cycle process.

7.3.1 DesCARTES

[80] provides an exhaustive list of the available i* tools. According to the comparison of these tools [79], ***Des**ign **C**ASE **T**ool for **A**gent-Oriented **R**epositories*, **T**echniques, **E**nvironments and **S**ystems(DesCARTES) is the tool offering the most exhaustive set of functionalities. It is a plug-in for the Eclipse IDE (Integrated Development Environment). It supports diagram edition of various models including i* models (Strategic Dependency and Strategic Rationale models), NFR models, UML models, and AUML models in the context of Tropos and I-Tropos developments.

At the analysis level, DesCARTES provides:

- an i* editor for editing i* Strategic Dependency (SD) and Strategic Rationale (SR) diagrams;
- an NFR editor for editing Softgoal Interdependency Graph (SIG);
- an use-case editor for editing UML use-cases and business use-cases diagrams;

At the design level, DesCARTES provides:

- a structural diagram editor for editing enhanced UML class diagrams with agent concepts;
- a dynamic diagram editor for editing UML-like activity diagrams and UML-like statecharts;
- a communicational diagram editor for editing UML-like sequence diagrams;
- social patterns templates for software design reuse.

Moreover, DesCARTES provides advanced project management capabilities, particularly in the following aspects:

- *Time Management* through a Gantt charts editor as well as an effort estimation tool implementing Use Case Points, Goal Points and COCOMO II models;
- *Risk Management* through a risk identification and traceability tool;
- *Quality Management* through a quality identification and traceability tool;
- *Configuration and Change Management* through a requirements traceability tool.

In our research context, we have extended DesCARTES to support our extension of the i* model, i.e. the Strategic Service (SS) modeling. Specifically, we have extended the existing i* editor of DesCARTES to support SS diagram editing. In addition, we have added the business process modeling feature, i.e. Business Process Modeling Notation (BPMN) modeling [75], to DesCARTES.

7.3.1.1 Extension of the i* Editor

DesCARTES has included an i* editor for SD and SR modeling. We have extended it in order to support the Strategic Service modeling. The reasons behind the use of the SS model in our methodology has been described in Section 6.2. Figure 7.11 illustrates a SS modeling using DesCARTES.

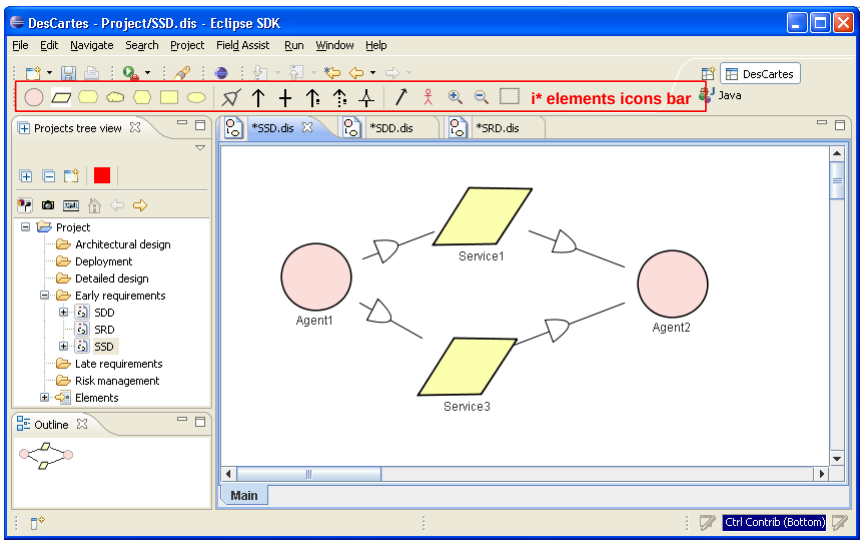


Figure 7.11: The i* editor in DesCARTES.

7.3.1.2 The BPMN Editor

The Figure 7.12 illustrates the business process modeling (BPMN) editor that we have included into DesCARTES. We can find the BPMN elements for their insertion into the drawing zone at its left side.

7.3.2 Hierarchy Graph Modeling

In our methodology, we also need to model goal graph and component feature models which are in the form of hierarchy graph. We have developed a tool for hierarchy graph modeling. This tool allows the user to:

- create a blank new hierarchy graph;
- create a quality model based on the C-QM quality model or the ISO 9216 quality model (see Section 3.6.2 for quality models);
- open the model previously saved.

Figure 7.13 shows the hierarchy graph editor implemented. We find:

- a thumb-index allowing the users to switch from an edited diagram to another (1);
- a buttons bar (2) composed of:
  - a button to add a new node to the tree;

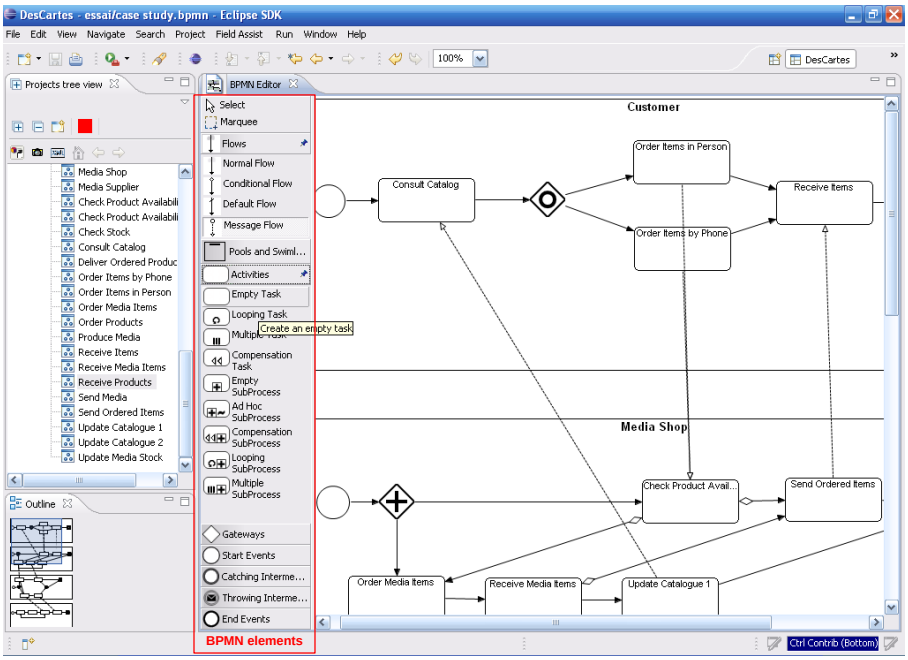


Figure 7.12: The BPMN editor in DesCARTES.

  - a button to edit the selected node;
  - a button to remove the selected node from the tree;
  - a button to remove all the nodes of the tree;
  - a button to save the graph;
  - a button to export the graph into a Excel file. The excel file is generated according to the RFP template that we have defined. Figure 7.14 shows the instructions for the use of our RFP template;
  - a button to save the graph as an image.

- a zone displaying the content of the edited graph in tree form;
- a zone displaying the hierarchy graph with a dynamically allocable scroll region(4).

7.3.3 Analyzing the RFP Response

We have developed a VBA program for analyzing the RFP response. Figure 7.15 depicts the worksheet that contains a button for analyzing the RFP response. When the button is clicked, the user is required to specify the name of worksheet containing the RFP response. The analysis result is displayed in this worksheet as illustrated in

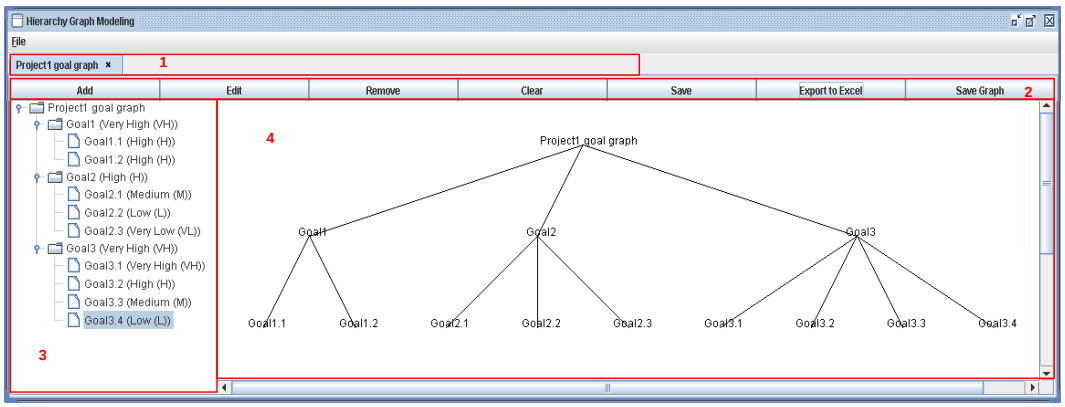


Figure 7.13: Editing a hierarchy graph.

Figure 7.15. The analysis result allows the user to see how well a COTS component meets the selection criteria.

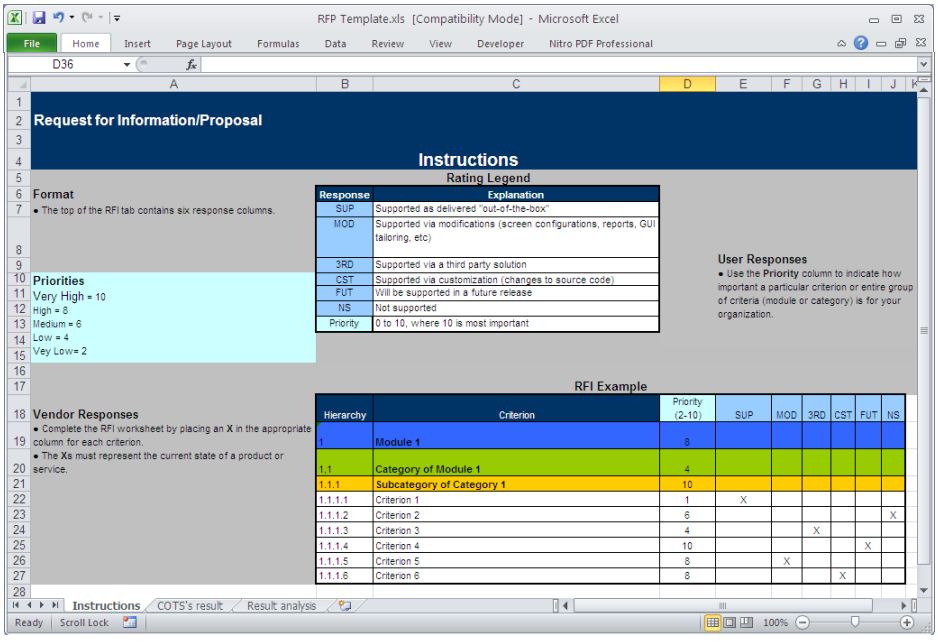


Figure 7.14: Instructions for using our RFP template.

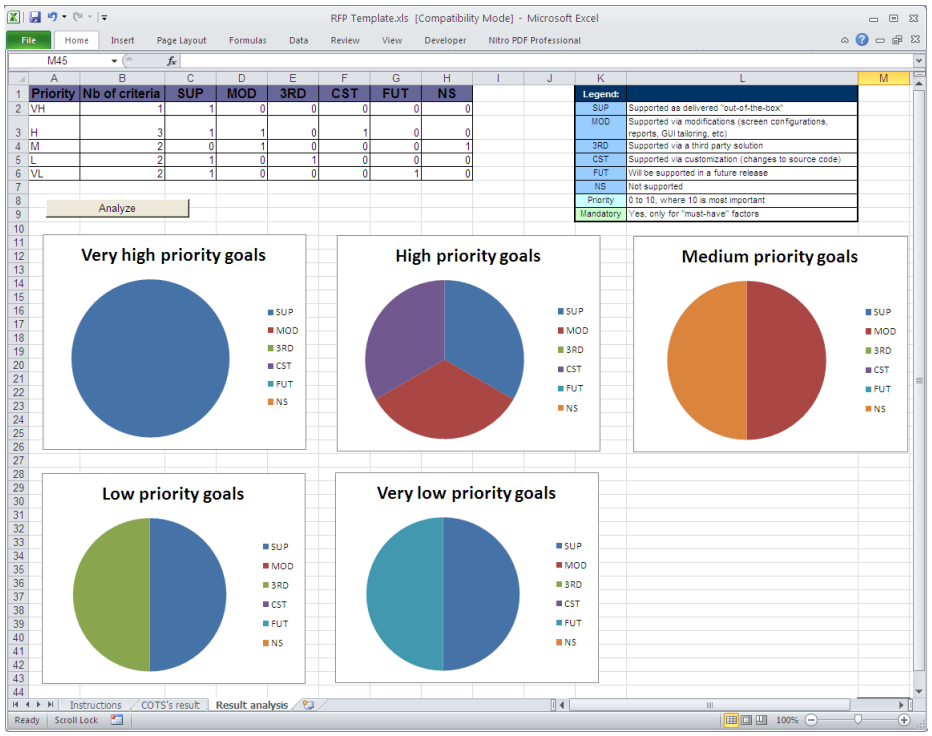


Figure 7.15: A simple example of RFP response analysis result.

7.3.4 Components Management System

In the context of our methodology, the project team needs to have a *Components Management System (ComMS)* so that they can easily see the information of the COTS components. We have developed such system using Ms Access. Our CMS is developed based on our component meta-model that we have presented in Section 4.2. This system allows user to:

- to edit and view the information of a component including name, cost, version, programming language, vendor name, domain name, component type, list of supported platforms, file name storing information of its quality, file name of the feature model representing its features and file name containing API documentation (see Figure C.1);
- to search for components that correspond to the search criteria composed of domain, type, platform and programming language (see Figure C.2). Figure C.3 presents the search result. The result can also be printed as illustrated in Figure C.4;
- to view the list of components per vendor (see Figure C.5);

- to print the list of components grouped by vendor (see Figure C.6);
- to view the list of components per domain (see Figure C.7);
- to print the list of components grouped by domain (see Figure C.8);
- to view the list of components per type (see Figure C.9);
- to print the list of components grouped by type (see Figure C.10);
- to view the list of components per platform (see Figure C.11);
- to print the list of components grouped by platform (see Figure C.12);
- to view the list of components per programming language (see Figure C.13);
- to print the list of components grouped by programming language (see Figure C.14);
- to edit and view information of a vendor including (see Figure C.15):
  - vendor name;
  - contact phone number;
  - vendor website;
  - contact email address;
  - vendor size of the vendor: *Too big to pay attention to us*, *Too small to survive and provide consistent service*, *Fit to us* or *Unknown*;
  - vendor's reputation: *Good*, *Bad* or *Unknown*;
  - vendor's financial position: *Good*, *Bad* or *Unknown*;
  - consultant service: vendor offers consult service or not;
  - list of target domains and market of the vendor;
  - vendor address: number, street name, city and country.
  -
- to print the list of vendors (see Figure C.15);
- to edit and view information of domain, type, platform and programming language that will be associated with components and vendors. Figure C.17 shows the form for editing platform. It consists of editing its name and description. It is the same for domain, type and programming editing forms.

7.4 Chapter Summary

This chapter has presented the application of our proposed methodology onto the development of an outbound logistics collaborative platform. Specifically, we have presented:

- the use of (SSM) for modeling the big picture of the project and for strategic reasoning about the project;
- the definition of system requirements using the i* and NFR frameworks;
- the use of our approach for analyzing differ mismatches during the COTS customization phase;
- definition and implementation of the integrated system architecture based on our proposal.

It has also presented the CASE tools that have been developed to support some WorkDefinitions in our development process. Indeed, we have presented:

- i*, NFR, Use-Case, and BPMN editors in DesCARTES;
- hierarchy graph editor;
- RFP template;
- RFP generator from a hierarchy graph;
- a VBA program in Ms Excel for analyzing the RFP answers from vendors;
- a Component Management System for managing components' information.

Part V

Conclusion

Chapter 8

Conclusion

This chapter concludes this research work by presenting our main contributions and points to future work.

8.1 Summary of Contributions

This thesis aimed to contribute to the improvement of CBSD. We have conducted our research works for this purpose. The results of our works constitute contributions to the field of CBSD research.

For a common conceptual basis in CBSD, we have defined a meta-model illustrating the relevant elements of a COTS component. It indicates the informations that a component user needs to know in order to select appropriate components and to properly integrate the selected ones into the system. A component management system can be built based on this meta-model. With such a system, components can be found and reused effectively.

From the COTS integration aspect, we have defined a system architecture for COTS dynamic integration. Indeed, existing OOP frameworks for COTS integration such as OMG CORBA, or Microsoft DCOM fail to support COTS dynamic integration. In these frameworks, the integrated components are statically bound, and collaboration mode among them is fixed so that it cannot be adjusted and modified especially when the system is running. Consequently, they do not help the developers to handle changing requirements and component substitution. For this reason, we have designed a system architecture that can be customized with respect to the project-specific business logic and adapted to business rules and component substitution.

In order to benefit from the advantages of agent-oriented programming, our proposed system architecture is centered on the agent concept. It is designed to be able to support COTS dynamic integration. It is a wrapper-based multi-agent architecture. Logically, it is composed of vertical architectural layers: GUI, MAS,

and Component layers. The MAS is in charge of realizing user requests from the GUI layer with respect to available components at the component layer. We have defined a meta-model illustrating the main parts constituting our MAS and characteristics of our agents. Agents that we have identified in our MAS are further specified in multiple complementary dimensions illustrating the different aspects of the proposed MAS. In order to ground our proposed system architecture, we have implemented it using the JADE framework. This constitutes an implementation model to developers.

From the development process aspect, we have established a rationale incremental and iterative process for CBSD (RecIProC). We have not only provided a high level description of the process but also some practical models and guidelines for accomplishing works defined in the process. Concerning the practical models, we firstly propose to use the SSM for modeling the most aggregate static view of the project for adequate identification of different individual systems. For each system, the SDM and the SRM are used to model its functional requirements; the SIG is used to model its non-functional requirements; and the hierarchy goal graph is used to model its selection criteria. We have also defined some guidelines for performing some works in the process, i.e. a guideline for analyzing strategic impacts of adopting the software project, a RFP template, a guideline for COTS evaluation, a mismatch analysis approach, and guidelines for constructing the SSD, the SDD, the SRD, and the SIG.

RecIProC is a business-driven development process. It starts with the business analysis phase. It provides the ability to develop IT solutions that meet the business needs and strategies and can be easily adapted to the business changes.

RecIProC addresses diverse issues related to CBSD including business analysis, requirement analysis, COTS evaluation and selection, and COTS mismatches handling. For business and requirements analysis, RecIProC provides practical models (i.e. the SSM, the SDM, the SRP and the SIG) as well as guidelines to perform them as evoked earlier. For COTS evaluation and selection, it provides an approach that deals with different dimensions (i.e. functional requirements, NFRs and non-technical aspect). It is suitable for both single and multiple COTS selections. It is originally proposed for multiple COTS selection, but adaptable for a single one by conducting only the COTS evaluation at local level in case that the selected COTS will be used as an isolated system. Otherwise, the COTS evaluation at global level helps for defining the COTS that best fits with other existing or in-house developed systems. For COTS mismatches handling, RecIProC provides the ability to address mismatches between the COTS products and system's requirements and architectural constraints. We model them in terms of goals, and these goals are then matched with the COTS features during the evaluation phase. Some mismatches detected are handled directly during the evaluation and some others will be handled after the selection.

RecIProC is an iterative cyclic approach. Each cycle produces a deliverable increment of the software with new COTS added to the system. A RecIProC cycle consists of seven iterative phases including business analysis, requirements analysis, COTS product identification, COTS product evaluation, decision making, COTS customization, and COTS integration. The incremental and iterative aspects of RecIProC allows it to support a "simultaneous definition and tradeoffs" among requirements, architecture, and component marketplace and management aspects of the project including cost and time estimation and risk management.

We have also developed CASE tools to support our methodology. Based on the meta-model of COTS components, we have implemented a component management system for managing components' information so that they can be found and reused effectively. Concerning the modeling tool, DesCARTES is a tool developed by our research unit. It supports diagram edition of various models including i* models (Strategic Dependency and Strategic Rationale models), NFR models, UML models, and AUML models. We have extended DesCARTES to add the SSM modeling into its i* editor and a business process modeling editor which is based on BPMN. We have also developed a hierarchy graph editor to model goals to be fulfill by each COTS component. This editor also allows users to generate a RFP, which follows our RFP template, from a hierarchy graph. Finally, we have implemented a VBA program in Ms Excel for analyzing the RFP answers from vendors.

The methodology proposed in this thesis focuses on system-level COTS components. Such COTS components are far more expensive and have bigger organizational impact than fine grained components, i.e. distributed component. Consequently, the selection of system-level COTS components has to be properly conducted at the analysis and design time while fine grained components can be selected at runtime. In [162], we have proposed an approach for selecting COTS components at runtime but this approach is intended to be used for fine grained COTS components selection.

From the managerial and economical perspective, this thesis contributes to the improvement of enterprise information system development. Specifically, it focuses on CBSD seen by research and industry as an efficient, manageable and cost effective development approach. This thesis proposes a business-driven methodology for the development of IT solutions that satisfy business requirements. In this sense, the proposed development process starts with business analysis phase that consists of studying the business context in order to define the business problems to be solved or the business opportunities to be addressed by the information system; defining the scope of the project in order to identify the different systems that will be integrated together to build the intended system; and analyzing strategic impacts in order to study the long-term strategic impacts of adopting the software project. The result of the business analysis phase serves as the basis for the following phases of the development process.

The development process has also drawn attention to the economic factor concerning the use of COTS components. It states that using COTS components the right solution when it fits the project context:

- **Technically**. It is able to offer the desired functionally at the required level of reliability;
- **Economically**. It is able to be incorporated and maintained in the target system within the available budget and schedule;
- **Strategically**. It meets the needs of the system environment including technical, political and legal consideration.

In addition, our methodology proposes a flexible integration system architecture allowing IT systems to be easily adapted to the business changes.

8.2 Advantages of the Proposed Architectural Design

The advantages of the proposed architecture is extremely difficult to quantify. A same case study should be performed using the proposed methodology, once with the architectural pattern, once without. The result could then be evaluated on the basis of development time, cost, the application performance on general and particular aspects, etc. Such an experiment would nevertheless be polluted by external factors.

In this thesis, we aim to illustrate the advantages of the proposed architecture in term of its qualitative aspects. Indeed, the proposed architecture was designed with respect to a set of quality requirements that we have defined. We used the NFR framework to conduct the quality analysis.

As depicted in Figure 8.1, *Usability* is the highest-level quality requirement that the proposed architecture needs to fulfill. It is the ability of the proposed architecture to be used by the application developers. It is refined into *Compatibility*, *Learnability* and *Modifiability*.

The NFR *Compatibility* requires the proposed architecture to be able to work with other systems. The operationalizing goal *Use of standard technology* has a sufficient positive (++) contribution to fulfill this quality requirement. With respect to this operationalizing goal, technologies used for the implementation of the proposed architecture are all standard, e.g. Java, XML, Servlet, etc. The MAS is also designed in accordance with the FIPA-specification.

The NFR *Learnability* requires the proposed architecture to be easy for developers to understand and to be able to use it. The operationalizing goals *Use of standard technology*, *Use of an implementation model*, *Description over multiple complementation dimensions* and *Social-based architecture* contribute positively to fulfill this requirement. These operationalizing goals are implemented in our proposal.

The NFR *Modifiability* requires the proposed architecture to be easy for developers to modify it. It is refined into *Maintainability* and *Flexibility*. *Maintainability* is the ease with which a software system can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment. *Flexibility* is the ease with which a system can be modified to adapt to the environment changes. The operationalizing goals *Use of wrapping architecture* and *Ease of customization to business logic* contributes to the fulfillment of these requirements.

With respect to the operationalizing goal *Use of wrapping architecture*, our proposed architecture is a wrapper-based architecture. It is designed with respect to the integration architecture constraints for easing component integration defined in [157] that:

- all components are wrapped;
- components do not talk directly to each other;
- the MAS is independent of underlying components.

The operationalizing goal *Ease of customization to business logic* encompasses *Use of XML*, *Use of business rules*, and *Use of agent technology* operationalizing goals. These goals are all implemented in our proposal. More precisely, our integration architecture is an agent-oriented architecture. The XML files are used to store the belief base and the service base of each agent. Logical rules, which implement the business rules, are separated from the application code.

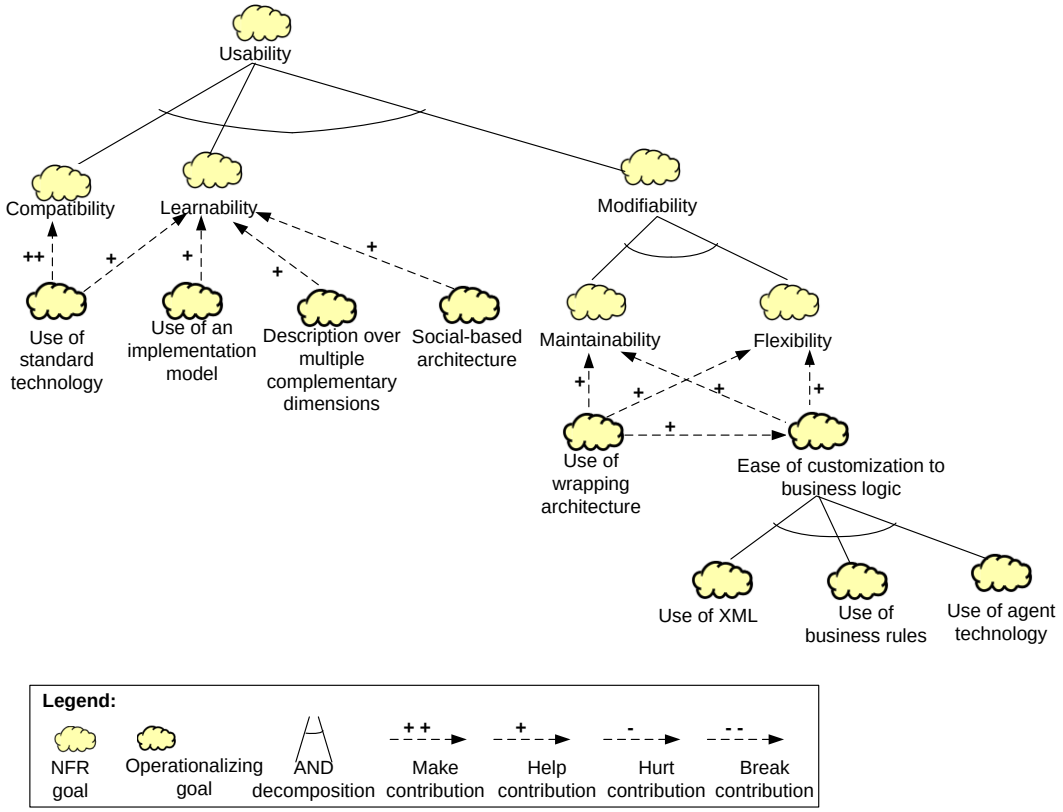


Figure 8.1: Quality analysis of the proposed architectural design.

8.3 Future Work

This section introduces the future work, that can be done to improve the work done in this thesis, as follows:

- *Including the project management dimension into our methodology.* We have reviewed the state of the art on project management for CBSD (i.e. effort estimation models, quality management, risk management, and organizational change management). The relevant approaches found in the literature can be integrated into our methodology in order to deal with the project management issues;

- *More case studies.* The methodology needs to gain experience with its use. It should be tested on more case studies, and eventually compared to other methodologies. This would contribute to the refinement of the methodology;

- *Dealing with system evolution.* The methodology can be extended to cover the system evolution phase of CBSD life cycle.

8.4 List of Publications

International Journals

1. S. Kiv, Y. Wautelet, Manuel Kolp: "Agent-Driven Integration Architecture for Component-Based Development", Transactions on Computational Collective Intelligence (8), 2012.

2. Y. Wautelet, S. Kiv, Manuel Kolp: "An Iterative Process for Component-Based Software Development Centered on Agents", Transactions on Computational Collective Intelligence (4), 2011.

International Conferences

1. S. Kiv, Y. Wautelet, and M. Kolp, "A Multi-Agent Architectural Pattern for Wrapping Off-the-Shelf Components", in Proceeding of the 5th KES-AMSTA International Conference, 2011.

2. S. Kiv, Y. Wautelet and M. Kolp, "A Process For COTS-Selection And Mismatches Handling: A Goal-Driven Approach", in Proceedings of the 2nd International Conference on Agent and Artificial Intelligence (ICAART), 2010.

3. Y. Wautelet, S. Kiv, V. Tran and M. Kolp, "Round Tripping in Component Based Software Development", in Proceeding of the 2010 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2010.

4. Y. Wautelet, S. Kiv, V. Tran and M. Kolp, "Strategic Reasoning in Software Development", in Proceeding of the 12th International Conference on Enterprise Information System, 2010.

5. Y. Wautelet, Y. Achbany, S. Kiv and M. Kolp, "A Service-Oriented Framework for Component-Based Software Development, An i* Driven Approach", in Proceedings of the 11th International Conference on Enterprise Information Systems (ICEIS09), Lecture Notes in Business Information Processing, Springer, 24, pp 551-563, 2009.

6. Y. Wautelet, S. Kiv and M. Kolp, "A Methodology for COTS-based Software Customization: an Agent-Oriented Approach", accepted for poster presentation at the 4th International Conference on Software and Data Technologies (ICSOFT09), 2009.

Book Chapters

1. M. Kolp, Y. Wautelet, S. Kiv and V. Tran, "Engineering Software Systems with Social-Driven Templates" in Methodological Advancements in Intelligent

Information Technologies: Evolutionary Trends, Advances in Intelligent Information Technologies Series, Information Science Publishing, 2009.

Technical Report

1. M. Kolp, Y. Wautelet, S. Kiv and Y. Achbany, "A Unified Data Model for European Schoolnet Databases", Final Report, May 2009.

Bibliography

[1] Microsoft office. *http://www.office.microsoft.com/*.

[2] Sap erp. *http://www.sap.com/solutions/bp/enterprise-resource-planning/index.epx*.

[3] C. Abts, B.W. Boehm, and E.B. Clark. COCOTS: A COTS software integration lifecycle cost model-model overview and preliminary data collection findings. In *ESCOM-SCOPE Conference*. Citeseer, 2000.

[4] C. Albert, L. Brownsword, C.D. Bentley, T. Bono, E. Morris, et al. Evolutionary process for integrating COTS-based systems (EPIC): An overview. *SEI CMU, Pittsburgh, PA, Technical Report CMU/SEI-2002-TR-009*, 2002.

[5] A.J. Albrecht. Measuring application development productivity. In *Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium*, pages 83–92, 1979.

[6] E. Altendorf, M. Hohman, and R. Zabicki. Using j2ee on a large, web-based project. *IEEE Software,*, pages 81–89, 2002.

[7] C. Alves. Cots-based requirements engineering. *Component-Based Software Quality*, pages 21–39, 2003.

[8] C. Alves and J. Castro. CRE: A systematic method for COTS components selection. In *XV Brazilian Symposium on Software Engineering (SBES)*. Rio de Janeiro, Brazil, 2001.

[9] C. Alves and A. Finkelstein. Negotiating requirements for cots-based systems. In *proceedings of 8th Int. Workshop on Requirements Engineering: Foundation for Software Quality, in conjunction with RE*, volume 2. Citeseer.

[10] C. Alves and A. Finkelstein. Investigating conflicts in cots decision-making. *International Journal of Software Engineering and Knowledge Engineering*, 13(5):473–493, 2003.

[11] B. Anda, H. Dreiem, D. Sjøberg, and M. Jørgensen. Estimating software development effort based on use casesexperiences from industry. *UML 2001The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, pages 487–502, 2001.

[12] A.I. Anton. Goal identification and refinement in the specification of software-based information systems. *PhD thesis*, 1997.

[13] B. Arinze and M. Anandarajan. A framework for using oo mapping methods to rapidly configure erp systems. *Communications of the ACM*, 46(2):61–65, 2003.

[14] J. Arlow and I. Neustadt. Uml and the unified process. *The Object Technology Series*, 2002.

[15] C. Ayala. Systematic construction of goal-oriented cots taxonomies. *PhD thesis*, 2008.

[16] C Ayala, Carlos Cares, Juan P Carvallo, Gemma Grau, Mariela Haya, Guadalupe Salazar, Xavier Franch, Enric Mayol, and Carme Quer. A comparative analysis of i*-based agent-oriented modeling languages. In *Proc. of the Conf. on Software Engineering and Knowledge Engineering (SEKE'05), Taipei, Taiwan, Republic of China*, pages 43–50, 2005.

[17] B. Balachandran. Developing intelligent agent applications with jade and jess. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 236–244. Springer, 2008.

[18] V.R. Basili and B. Boehm. COTS-based systems top 10 list. *Computer*, 34(5):91–95, 2001.

[19] F. Bellifemine, A. Poggi, and G. Rimassa. Jade–a fipa-compliant agent framework. In *Proceedings of PAAM*, volume 99, pages 97–108. Citeseer, 1999.

[20] F.L. Bellifemine, G. Caire, and D. Greenwood. *Developing multi-agent systems with JADE*, volume 5. Wiley, 2007.

[21] A. Beneventi, A. Poggi, M. Tomaiuolo, and P. Turci. Integrating rule and agent-based programming to realize complex systems. *WSEAS Trans. on Information Science and Applications*, 1(1):422–427, 2004.

[22] C. Bernon, V. Camps, M.P. Gleizes, and G. Picard. Tools for self-organizing applications engineering. *Engineering Self-Organising Systems*, pages 283–298, 2004.

[23] M. Bertoa and A. Vallecillo. Quality attributes for cots components. *ID Computación*, 1(2):128–144, 2002.

[24] M.F. Bertoa, J.M. Troya, and A. Vallecillo. Measuring the usability of software components. *Journal of Systems and Software*, 79(3):427–439, 2006.

[25] B. Boehm. Requirements that handle ikiwisi, cots, and rapid change. *Computer*, 33(7):99–102, 2000.

[26] B. Boehm. Spiral development: Experience, principles and refinements, report special report cmu. Technical report, SEI-2000-SR-008, Carnegie Mellon Software Engineering Institute, 2000.

[27] B. Boehm, D. Port, M. Abi-Antoun, and A. Egyed. Guidelines for the life cycle objectives (lco) and the life cycle architecture (lca) deliverables for model-based architecting and software engineering (mbase). *USC, Los Angeles, USC Technical Report USCCSE-98-519*, 1999.

[28] B. Boehm, D. Port, and Y. Yang. Winwin spiral approach to developing cots-based applications. In *EDSER-5 5 th International Workshop on Economic-Driven Software Engineering Research*, 2003.

[29] B. Boehm, D. Port, Y. Yang, and J. Bhuta. Not all CBS are created equally: COTS-intensive project types. *COTS-Based Software Systems*, pages 36–50, 2003.

[30] B.W. Boehm. *Software engineering economics*. Prentice-Hall, 1981.

[31] B.W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, 1988.

[32] B.W. Boehm, J.R. Brown, H. Kaspar, M. Lipow, G.J. MacLeod, and M.J. Merrit. Characteristics of software quality. 1978.

[33] B.W. Boehm, J.R. Brown, and M. Lipow. Quantitative evaluation of software quality. In *Proceedings of the 2nd international conference on Software engineering*, pages 592–605. IEEE Computer Society Press, 1976.

[34] B.W. Boehm, R. Madachy, B. Steece, et al. *Software Cost Estimation with Cocomo II*. Prentice Hall PTR, 2000.

[35] G. Booch. *Object-oriented analysis and design with applications*. The Addison-Wesley object technology series. Benjamin/Cummings Pub. Co., 1994.

[36] A.W. Brown and K.C. Wallnau. The current state of CBSE. *Software, IEEE*, 15(5):37–46, 1998.

[37] L. Brownsword, T. Oberndorf, and C.A. Sledge. Developing new processes for COTS-based systems. *Software, IEEE*, 17(4):48–55, 2000.

[38] T. Budd. *An introduction to object-oriented programming*. Addison-Wesley Reading, Massachusetts, 1991.

[39] X. Burgués, C. Estay, X. Franch, J.A. Pastor, and C. Quer. Combined selection of COTS components. *COTS-Based Software Systems*, pages 54–64, 2002.

[40] P. Burrafato and M. Cossentino. Designing a multi-agent solution for a bookstore with the passi methodology. In *Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002)*, pages 27–28, 2002.

[41] J. Cadle and D. Yeates. *Project management for information systems*. Prentice Hall, 2004.

[42] G. Caire, W. Coulier, F. Garijo, J. Gomez, J. Pavón, F. Leal, P. Chainho, P. Kearney, J. Stark, R. Evans, et al. Agent oriented analysis using message/uml. *Agent-oriented software engineering II*, pages 119–135, 2002.

[43] Esther Cameron and Mike Green. *Making Sense of Change Management: A Complete Guide to the Models Tools and Techniques of Organizational Change*. Kogan Page, 2012.

[44] L.F. Capretz. Y: a new component-based software life cycle model. *Journal of Computer Science*, 1(1):76–82, 2005.

[45] S.K. Card, T.P. Moran, and A. Newell. *The psychology of human-computer interaction*. CRC, 1983.

[46] D. Carney. Assembling large systems from COTS components: opportunities, cautions, and complexities. *SEI Monographs on Use of Commercial Software in Government Systems*, 1997.

[47] D. Carney and F. Leng. What do you mean by COTS? Finally, a useful answer. *Software, IEEE*, 17(2):83–86, 2000.

[48] J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven information systems engineering: the tropos project. *Information systems*, 27(6):365–389, 2002.

[49] L. Chung and K. Cooper. A cots-aware requirements engineering (care) process: Defining system level agents, goals, and requirements. *Department of Computer Science, The University of Texas, Dallas, TR UTDCS-23-01*, 2001.

[50] L. Chung and K. Cooper. A knowledge-based cots-aware requirements engineering approach. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 175–182. ACM, 2002.

[51] L. Chung and K. Cooper. Defining goals in a cots-aware requirements engineering approach. *Systems engineering*, 7(1):61–83, 2004.

[52] L. Chung and K. Cooper. Matching, ranking, and selecting components: A cots-aware requirements engineering and software architecting approach. In *Proceedings 1st MPEC Workshop*, 2004.

[53] L. Chung, BA Nixon, E. Yu, and J. Mylopoulos. Non-functional requirements in software engineering. 2000, 2000.

[54] S. Comella-Dorda, J. Dean, E. Morris, P. Oberndorf, et al. A process for cots software product evaluation. In *Proceedings of the 1st International Conference on COTS-Based Software System*, 2002.

[55] A. Corradi, N. Dulay, R. Montanari, and C. Stefanelli. Policy-driven management of agent systems. *Policies for Distributed Systems and Networks*, pages 214–229, 2001.

[56] I. Crnkovic and M.P.H. Larsson. *Building reliable component-based software systems*. Artech House Publishers, 2002.

[57] A. Dardenne, A. Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of computer programming*, 20(1-2):3–50, 1993.

[58] A. Davis. A comparative study of dcom and soap. In *Multimedia Software Engineering, 2002. Proceedings. Fourth International Symposium on*, pages 48–55. IEEE, 2002.

[59] S.A. DeLoach, M.F. Wood, and C.H. Sparkman. Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258, 2001.

[60] T. T. Do. A framework for multi-agent systems detail design. *PhD thesis, Universit catholique de Louvain, Institut d'Administration et de Gestion (IAG), Belgium*, 2005.

[61] T.B. Downing. *Java RMI: remote method invocation*. IDG Books Worldwide, Inc., 1998.

[62] R.G. Dromey. A model for software product quality. *Software Engineering, IEEE Transactions on*, 21(2):146–162, 1995.

[63] C.A. Ellis and J. Wainer. Goal-based models of collaboration. *Collaborative Computing*, 1(1):61–86, 1994.

[64] H. Estrada, A. Rebollar, O. Pastor, and J. Mylopoulos. An empirical evaluation of the i* framework in a model-based software generation environment. In *Advanced Information Systems Engineering*, pages 513–527. Springer, 2006.

[65] Y. Fan, W. Shi, and C. Wu. Enterprise wide application integration platform for cims implementation. *Journal of intelligent Manufacturing*, 10(6):587–601, 1999.

[66] M. Fayad and D.C. Schmidt. Object-oriented application frameworks. *Communications of the ACM*, 40(10):32–38, 1997.

[67] M.E. Fayad, D.S. Hamu, and D. Brugali. Enterprise frameworks characteristics, criteria, and challenges. *Communications of the ACM*, 43(10):39–46, 2000.

[68] P. Felber and P. Narasimhan. Experiences, strategies, and challenges in building fault-tolerant corba systems. *IEEE Transactions on Computers*, pages 467–511, 2004.

[69] FIPA. Agent uml. *http://www.auml.org/*, 2010.

[70] K. Forsberg and H. Mooz. System engineering overview. *Software Requirements Engineering*, pages 44–72, 1997.

[71] E. Friedman-Hill. *Jess in action: rule-based systems in java*. Manning publications, 2003.

[72] R.B. Grady. *Practical software metrics for project management and process improvement*. Prentice-Hall, Inc., 1992.

[73] G. Grau, J.P. Carvallo, X. Franch, and C. Quer. Descots: a software system for selecting cots components. In *Euromicro Conference, 2004. Proceedings. 30th*, pages 118–126. IEEE, 2004.

[74] S. Gregor, J. Hutson, and C. Oresky. Storyboard process to assist in requirements verification and adaptation to capabilities inherent in cots. *COTS-Based Software Systems*, pages 132–141, 2002.

[75] Object Management Group. Business process modeling notation bpmn v2.0. *Available at http://www.omg.org/spec/BPMN/2.0/*.

[76] H.T.T. Hang. Quality-aware agent-oriented information-system development. *PhD thesis*, 2010.

[77] B. Henderson-Sellers. From object-oriented to agent-oriented software engineering methodologies. *Software Engineering for Multi-Agent Systems III*, pages 1–18, 2005.

[78] P. Herzum and O. Sims. *Business Components Factory: A Comprehensive Overview of Component-Based Development for the Enterprise*. John Wiley & Sons, Inc. New York, NY, USA, 2000.

[79] i* Wiki. Available i* tools. *Available at http://istar.rwth-aachen.de/tiki-index.php?page=Comparing+the+i*+Tools*.

[80] i* Wiki. Comparing the i* tools. *Available at http://istar.rwth-aachen.de/tiki-index.php?page=i*+Tools&structure=i*+Wiki+Home*.

[81] ISO. *ISO/IEC IS 9126: Software Product Evaluation - Quality Characteristics and Guidelines for their Use*. 1991.

[82] ISO. Iso/iec 9126: Software engineering-product quality-part 1: Quality model. *International Organization for Standardization, Geneva, Switzerland*, 2001.

[83] ISO/IEC. Software engineering – software product quality requirements and evaluation (square)– guide to square. *International Organization for Standardization*, 2005.

[84] I. Jacobson, G. Booch, and J. Rumbaugh. *The unified software development process*. Addison-Wesley Longman, 1999.

[85] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard. *Object-oriented software engineering: a use case driven approach*. Addison-Wesley, 1992.

[86] N. Jennings. Agent-oriented software engineering. *Multi-Agent System Engineering*, pages 1–7, 1999.

[87] N.R. Jennings and M.J. Wooldridge. *Agent technology: foundations, applications, and markets*. Springer Verlag, 1998.

[88] S. Joosten and S. Purao. A rigorous approach for mapping workflows to object-oriented is models. *Journal of Database Management (JDM)*, 13(4):1–19, 2002.

[89] S Kalaimagal and R Srinivasan. Q'facto 10-a commercial off-the-shelf component quality model proposal'. *J. Software Eng*, 4:1–15, 2010.

[90] Sivamuni Kalaimagal and Rengaramanujam Srinivasan. Q'facto 12: an improved quality model for cots components. *SIGSOFT Softw. Eng. Notes*, 35(2):1–4, 2010.

[91] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, and A.S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, DTIC Document, 1990.

[92] E. Kavakli and P. Loucopoulos. Goal driven requirements engineering: Evaluation of current methods. In *Proceedings of the 8th CAiSE/IFIP8*, 2003.

[93] V. Kavakli and P. Loucopoulos. Goal-driven business process analysis application in electricity deregulation. In *Advanced Information Systems Engineering*, page 305. Springer, 1998.

[94] SD Kim and JD Park. C-qm: A practical quality model for evaluting cots components. In *Applied Informatics*, pages 991–996. IASTED/ACTA Press, 2003.

[95] R. Kishore, H. Zhang, and R. Ramesh. Enterprise integration using the agent paradigm: foundations of multi-agent-based integrative business information systems. *Decision Support Systems*, 42(1):48–78, 2006.

[96] M. Kolp, S. Faulkner, and Y. Wautelet. Social structure based design patterns for agent-oriented software engineering. *IJIIT*, 4(2):1–23, 2008.

[97] M. Kolp, P. Giorgini, and J. Mylopoulos. Multi-agent architectures as organizational structures. *Autonomous Agents and Multi-Agent Systems*, 13(1):3–25, 2006.

[98] J. Kontio. OTSO: a systematic process for reusable software component selection. 1995.

[99] J. Kontio. A case study in applying a systematic method for COTS selection. In *Proceedings of the 18th international conference on Software engineering*, pages 201–209. IEEE Computer Society, 1996.

[100] J. Kontio, G. Caldiera, and V.R. Basili. Defining factors, goals and criteria for reusable component evaluation. In *Proceedings of the 1996 conference of the Centre for Advanced Studies on Collaborative research*, page 21. IBM Press, 1996.

[101] G. Kotonya and A. Rashid. A strategy for managing risk in component-based software development. In *Euromicro Conference, 2001. Proceedings. 27th*, pages 12–21. IEEE, 2001.

[102] P. Kruchten. The rational unified process : An introduction. *Longman (Wokingham), Addison-Wesley, December*, 2003.

[103] P. Kruchten. *The rational unified process: an introduction.* Addison-Wesley Professional, 2004.

[104] T.S. Kuhn. *The structure of scientific revolutions*, volume 2. University of Chicago press, 1996.

[105] D. Kunda and L. Brooks. Applying social-technical approach for cots selection. In *Proceedings of the 4th UKAIS Conference*, pages 552–565. Citeseer, 1999.

[106] D. Kunda and L. Brooks. Identifying and classifying processes (traditional and soft factors) that support cots component selection: a case study. *European Journal of Information Systems*, 9(4):226–234, 2000.

[107] Y. Labrou, T. Finin, and Y. Peng. Agent communication languages: The current landscape. *Intelligent Systems and their Applications, IEEE*, 14(2):45–52, 1999.

[108] I. Lakatos. *The Methodology of Scientific Research Programmes: Philosophical Papers: Vol.: 1.* Cambridge University Press, 1978.

[109] R.W. Lichota, R.L. Vesprini, and B. Swanson. Prism product examination process for component based development. In *Assessment of Software Tools and Technologies, 1997., Proceedings Fifth International Symposium on*, pages 61–69. IEEE, 1997.

[110] N.A. Maiden and C. Ncube. Acquiring cots software selection requirements. *Software, IEEE*, 15(2):46–56, 1998.

[111] A. Marco and J. Buxton. *The craft of software engineering.* Addison-Wesley Longman Publishing Co., Inc., 1986.

[112] J.A. McCall, P.K. Richards, G.F. Walters, Rome Air Development Center, and United States. Air Force. Systems Command. Electronic Systems Division. *Factors in software quality.* Rome Air Development Center, Air Force Systems Command, 1977.

[113] J. McDermid and P. Rook. Software development process models. *Software engineer's reference book*, pages 1–35, 1991.

[114] J. McManus and T. Wood-Harper. *Information systems project management: Methods, tools and techniques.* Pearson Education, 2003.

[115] B.C. Meyers and P. Oberndorf. *Managing software acquisition: open systems and COTS products.* 2001.

[116] A. Mohamed, G. Ruhe, and A. Eberlein. COTS selection: past, present, and future. 2007.

[117] A. Mohamed, G. Ruhe, and A. Eberlein. Decision support for handling mismatches between COTS products and system requirements. 2007.

[118] A.S.A.S. Mohamed. Decision support for selecting COTS software products based on comprehensive mismatch handling. *PhD thesis*, 2007.

[119] E.F. Monk and B.J. Wagner. *Concepts in enterprise resource planning*. Course Technology, 2006.

[120] M. Morisio, C.B. Seaman, A.T. Parra, V.R. Basili, S.E. Kraft, and S.E. Condon. Investigating and improving a COTS-based software development process. 2000.

[121] M. Morisio and M. Torchiano. Definition and classification of COTS: a proposal. *COTS-Based Software Systems*, pages 165–175, 2002.

[122] M. Morisio and A. Tsoukias. Iusware: A methodology for the evaluation and selection of software products. In *Software Engineering. IEE Proceedings*, volume 144, pages 162–174. IET, 1997.

[123] L.F. Motiwalla and J. Thompson. *Enterprise systems for management*. Pearson Prentice Hall, 2009.

[124] H. Mouratidis. *A security oriented approach in the development of multiagent systems: applied to the management of the health and social care needs of older people in England*. PhD thesis, University of Sheffield, 2004.

[125] P. Naur and B. Randell. Software engineering: Report of a conference sponsored by the nato science committee. NATO, 1968.

[126] C. Ncube and J. Dean. The limitations of current decision-making techniques in the procurement of COTS software components. *COTS-Based Software Systems*, pages 176–187, 2002.

[127] C. Ncube and N.A.M. Maiden. PORE: Procurement-oriented requirements engineering method for the component-based systems engineering development paradigm. In *International Workshop on Component-Based Software Engineering*, page 1. Citeseer, 1999.

[128] M. Northover, D. G. Kourie, A. Boake, S. Gruner, and A. Northover. Towards a philosophy of software development: 40 years after the birth of software engineering. In *Zeitschrift fr allgemeineWissenschaftstheorie, 39(1)*.

[129] P.A. Oberndorf. Workshop on cots-based systems. Technical report, Carnegie Mellon University, 1997.

[130] M. Ochs, D. Pfahl, G. Chrobok-Diening, and B. Nothhelfer-Kolb. A cots acquisition process: Definition and application experience. *ISERN Report*, 2000.

[131] OMG. The software process engineering metamodel specification. version 1.1. 2005.

[132] J. Pavón and J. Gómez-Sanz. Agent oriented software engineering with ingenias. *Multi-Agent Systems and Applications III*, pages 1069–1069, 2003.

[133] S.L. Pfleeger. *Software engineering: theory and practice*. Prentice-Hall, 2001.

[134] S.L. Pfleeger and J.M. Atlee. *Software engineering: theory and practice*. Prentice Hall, 1998.

[135] PMI. *A Guide to the Project Management Body of Knowledge (PMBOK guide)*. Project Management Institute, 2004.

[136] A. Poggi, G. Rimassa, and P. Turci. What agent middleware can (and should) do for you. *Applied Artificial Intelligence, 16*, 9(10):677–698, 2002.

[137] L.H. Putnam. A general empirical solution to the macro software sizing and estimating problem. *Software Engineering, IEEE Transactions on*, (4):345–361, 1978.

[138] A. Rashid and G. Kotonya. Risk management in component-based development: A separation of concerns perspective. In *ECOOP Workshop on Advanced Separation of Concerns (ECOOP Workshop Reader)*. Citeseer, 2001.

[139] A. Rawashdeh and B. Matalkah. A new software quality model for evaluating cots components. *Journal of Computer Science*, 2(4):373–381, 2006.

[140] P. Rogers. Software engineering: a practitioner's approach, 1991.

[141] L. Rose. Risk management of COTS based systems development. *Component-Based Software Quality*, pages 352–373, 2003.

[142] W. Royce. *Software project management: a unified framework*. The Addison-Wesley object technology series. Addison-Wesley, 1998.

[143] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-oriented modeling and design*, volume 38. Prentice hall, 1991.

[144] T.L. Saaty. Analytic hierarchy process. 1980.

[145] R. Simao and A. Belchior. Quality characteristics for software components: Hierarchy and quality guides. *Component-Based Software Quality*, pages 184–206, 2003.

[146] BeanShell Web Site. http://www.beanshell.org/.

[147] DesCARTES Web Site. http://www.isys.ucl.ac.be/descartes/.

[148] Drools Web Site. http://www.jboss.org/drools.

[149] FIPA Web Site. http://fipa.org.

[150] JADE Web Site. http://jade.tilab.com/.

[151] UML Web Site. http://www.uml.org/.

[152] I. Sommerville. *Software Engineering*. Addison-Wesley USA, 2005.

[153] M. Torchiano and M. Morisio. Overlooked aspects of COTS-based development. *Software, IEEE*, 21(2):88–93, 2004.

[154] V. Tran and D.B. Liu. A procurement-centric model for engineering component-based software systems. In *Assessment of Software Tools and Technologies, 1997., Proceedings Fifth International Symposium on*, pages 70–79. IEEE, 1997.

[155] A. Van Lamsweerde. Goal-oriented requirements engineering: A guided tour. *re*, page 0249, 2001.

[156] R. Vieira, A. Moreira, M. Wooldridge, and R.H. Bordini. On the formal semantics of speech-act based communication in an agent-oriented programming language. *Journal of Artificial Intelligence Research*, 29(1):221–267, 2007.

[157] M.R. Vigder and J. Dean. An architectural approach to building systems from COTS software components. In *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, page 22. IBM Press, 1997.

[158] G. Wagner. Agent-object-relationship modeling. In *Proceedings of the 2nd International Symposium: From Agent Theory to Agent Implementation*, 2000.

[159] Y. Wautelet. A goal-driven project management framework for multi-agent software development: The case of i-tropos. *PhD thesis, Universit catholique de Louvain, Louvain School of Management (LSM), Louvain-La-Neuve, Belgium, August*, 2008.

[160] Y. Wautelet. Representing, modeling and engineering a collaborative supply chain management platform. *IJISSCM*, 5(3):1–23, 2012.

[161] Y. Wautelet, Y. Achbany, and M. Kolp. A service-oriented framework for mas modeling. In *Proceedings of the 10th International Conference on Entreprise Information Systems (ICEIS), Bacelona*, 2008.

[162] Y. Wautelet, S. Kiv, and M. Kolp. An iterative process for component-based software development centered on agents. *T. Computational Collective Intelligence*, 5:41–65, 2011.

[163] Y. Wautelet and M. Kolp. Goal driven iterative software project management. In *ICSOFT (2)*, pages 44–53, 2011.

[164] Y. Wautelet, M. Kolp, and S. Poelmans. Requirements-driven iterative project planning. In María José Escalona Cuaresma, José Cordeiro, and Boris Shishkov, editors, *Software and Data Technologies*, volume 303 of *Communications in Computer and Information Science*, pages 121–135. Springer, 2013.

[165] Y. Wautelet, C. Schinckus, and M. Kolp. A modern epistemological reading of agent orientation. *IJIIT*, 4(3):46–57, 2008.

[166] Y. Wautelet, C. Schinckus, and M. Kolp. Towards knowledge evolution in software engineering: An epistemological approach. *IJITSA*, 3(1):21–40, 2010.

[167] M. Wooldridge. Agent-based software engineering. In *Software Engineering. IEE Proceedings*, volume 144, pages 26–37. IET, 1997.

[168] M. Wooldridge, N.R. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[169] D. Xu, J. Yin, Y. Deng, and J. Ding. A formal architectural model for logical agent mobility. *IEEE Transactions on Software Engineering*, pages 31–45, 2003.

[170] E. Yu. Modeling strategic relationships for process reengineering. *PhD thesis, University of Toronto, Department of Computer Science, Canada*, 1995.

[171] E. Yu. *Social Modeling for Requirements Engineering*. MIT Press, 2011.

[172] E. Yu and J. Mylopoulos. Why goal-oriented requirements engineering. In *Proceedings of the 4th International Workshop on Requirements Engineering: Foundations of Software Quality*, pages 15–22, 1998.

[173] Robert W Zmud. Management of large software development efforts. *MIS Quarterly*, pages 45–55, 1980.

Appendix A

Epistemological Foundation

This appendix presents an epistemological analysis of the emergence of CBSD. More precisely, we argue that Lakatos' "research programme" is more suitable than Kuhn's "paradigm shift" to explain this emergence.

A.1 Changes in Software Development Methodology

In the early years of computing, there was no clear notion of software development methodology. Software was developed in a rather ad-hoc fashion, based on the available computer technology. As software projects grew in size project managers with classical engineering backgrounds adopted their traditional style of management in order to control challenging extent of system complexity wherein a project is viewed as a sequence of discrete phases, each of which needed to be articulated, documented, executed and approved before proceeding the next phase. Therefore, once the requirements had been elicited, documented and approved by the client, an analysis phase began. The analysis phase involved defining various modules of software that could be written by different developers. This phase was followed by further software project phases including design, implementation, testing, deployment, and maintenance. This phased approach was called the waterfall methodology and it symbolized a change from informally to formally and rigorously managed software development project. Such a change clearly involved an important conceptual shift, and in this sense, the adoption of the waterfall methodology could be seen as "revolutionary". However, since there were relatively few large projects in the emerging software industry at that time, the uptake in applying the waterfall methodology was gradual [128].

Nevertheless, this type of methodology can only be successful when a series of assumptions are met at the same time. Otherwise, risks are introduced in the development process. Following [26], these assumptions are:

- The requirements can be known in advance of implementation;
- The requirements have no unresolved high-risk implication, such as risks due to cots, schedule, performance, safety, security, user interface, and organizational impacts;
- The nature of the requirements will not change very much either during development or evolution;
- The requirements are compatible with all the key system stakeholders' expectations, including users, customers, developer, maintainers, investors;
- The right architecture for implementing the requirements is well understood;
- There is enough calendar time to proceed sequentially.

If these assumptions are not met, the initial design will possibly be flawed with respect to its key requirements and late discovery of design defects results in costly overruns and/or project abandonment. Consequently, time and money will be wasted on specifying and implementing requirements that are going to change and implementing a faulty design. By the late of 1980s, the inappropriate nature of this type of methodology had become increasingly apparent because of the large number of unsuccessful software project because the assumptions needed for an optimal waterfall methodology are rarely met due to:

- Requirement can rarely be known and defined in advance of implementation, especially for new user-interactive systems. Users and stakeholders are often, during the early stages of the project, unable to express and describe the system's requirements. Most of them suffer from what Boehm calls *IKIWISI* syndrome [25]. It is when they are asked about the system requirements they reply *"I can't tell you, but I'll know it when I see it (IKIWISI)"*;
- Defining requirements and freezing them early in the project is very risky. Early defined requirements can meet users' needs but sometimes analysts discover later in the project that their practical achievement is too constraining (in terms of cost, development time, processing time, human resources, etc.). Thus, it is important not to fix requirements too early in the project but to keep the ability to refine them later if their definition appears to be too constraining;
- Requirements often evolve during the development process. Stakeholders express new ideas, needs or perspectives and consequently the requirements elicitation process cannot be done only once in the early stage but must be a

continuous process. In this way, we can avoid spending a huge time on requirements analysis at the beginning of the project specifying requirements that will tend to be out of date or be refined later in the project.

These remarks highlight the need for a new type of methodology that includes continuous requirements acquisition and modeling. As a result, there was a strong shift to *"Iterative Incremental Development"* which later evolved into a spiral model of software development [31].

By contrast with the waterfall methodology, the iterative incremental development assumes that it is improbable to produce a correct system without the repetition of cycles and the close interaction of users and developers contributing to the creation of an evolving system. However, some people found even modernized concepts of software development processes supporting the development from scratch to be unable to cope with the challenges of the current enterprise information systems which have become more and more large scale and complex. The development from scratch of such systems has become extremely expensive and time-consuming.

In order to gain competitive advantages software organizations are forced to develop systems quickly and cost-efficiently. Many organizations shift from traditional system development in which user requirements are specified and custom solutions are developed, to a market-generated, product-based approach in which users themselves select and arrange meaningful-to-them components as solutions to their requirements. This new approach assumes that a knowledgeable project team comprehends its own requirements, is able to obtain the required components for building systems, and is competent enough to assemble these components into useful and desired systems. This COTS-based system development approach seems to be a promising solution to reduce development cost and effort, while maintaining overall software products quality.)

COTS-based system development has several characteristics that distinguish it from the traditional system development in which software systems are built from scratch. On the epistemological perspective, the CBSD constitutes a knowledge evolution in software engineering. In the thesis, we argue that *Lakatos' "research programme"* is more suitable than *Kuhn's "paradigm shift"* to explain this evolution. Before developing our argumentation, we present in this section the concepts of these two epistemological approaches.

A.2 Kuhn's "paradigm shift"

Kuhn is an epistemologist and historian of science who is most known for his book The Structure of Scientific Revolutions [104] in which he presented the idea that science does not progress via a linear accumulation of new knowledge, but undergoes periodic revolutions, also called paradigm shift. According to Kuhn, a scientific rev-

olution or paradigm shift occurs when scientists encounter anomalies which cannot be explained by the current principal paradigm. For Kuhn, a paradigm is "*A constellation of concepts, values, perceptions and practices shared by a community and which forms a particular vision of reality that is the basis of the way a community organizes itself*".

Kuhn's analysis of the history of science suggests to him that the practice of science comes in three phases. The first phase is the pre-scientific phase, in which there is no consensus on any theory of explanation because multiple paradigms are put forward by different schools of thought. This phase is generally characterized by several incompatible and incomplete theories and scientists may disagree with one another as they propose and support their individual theories. Over time, as the ideas compete, scientists cluster around a small set of paradigms, each trying to support their own ideas and destroy the opposing paradigms. Finally, one paradigm wins through and becomes the dominant principle. This comes to the normal science phase in which scholars accept the dominant paradigm of the moment, performing experiments that test and prove its efficacy in a range of situations. New explanations may extends the paradigm but do not change its fundamental nature so that the paradigm may grow with many extensions to explain the various exception cases that are not easily covered by the original paradigm. During normal science, scientists do not attempt to refute the paradigm. They resist anomalies until so many have accumulated that they can no longer be ignored and the scientific community is thrown into a state of crisis. During this crisis, new ideas are proposed and proven. Eventually a new paradigm is formed and replaces the current dominant paradigm. This is what Kuhn calls a paradigm shift.

For Kuhn, paradigm shift is revolutionary since the rules of the game have changed. It is a complete change of world-view and is holistic rather than piecemeal. In this sense, the new paradigm is incommensurable with the old one. Kuhn states that all scientific fields go through these paradigm shifts multiple times, since new theories displace the old, however, no scientific community will abandon the dominant paradigm unless a new one becomes available.

Applying Kuhnian perspective to software engineering ingenuously, we could consider COTS-based system development to be a new paradigm of software engineering, whereas *"waterfall"* or other software engineering methodologies for the traditional system development in which software systems are built from scratch would be considered part of the old paradigm. The emergence of COTS-based system development could be seen as a paradigm shift.

In the following, we will demonstrate that Kuhn's concept of paradigm shift does not adequately account for the change in software development methodology toward COTS-based system development. After presenting the research program concept developed by Lakatos, we will explain why Lakatosian perspective is more appropriate to describe the evolution to COTS-based system development.

A.3 Lakatos' "research programme"

Lakatos replaced the Kuhnian paradigm with an entity called a *"research programme"* which involves a succession of theories [108]. A Lakatosian research programme is a kind of scientific construction, a theoretical framework, which guides future research in a specific field in a positive or negative way. Each research programme is constituted of a hard core, a protective belt of auxiliary hypotheses, and a positive and a negative heuristic.

The theories are linked by a common *"hard core"* of shared commitments. Each theory in the sequence constitutes a new and more detailed articulation of these commitments. The hard core is surrounded with a protective belt composed of auxiliary hypotheses which shelters the hard core from immediate empirical refutation. These auxiliary hypotheses will be thoroughly studied again, widened and completed by successive theories in the program, but the core assumptions remain intact. This widening of these hypotheses contributes to the evolution of the research programme without modifying the core assumptions. According to Lakatos, the evolution of knowledge can be characterized by a series of problems shifts which allow the scientific theories to evolve without rejecting the basic axioms.

The third important of characteristic of a research programme is its ability to stimulate the development of more complex and adequate theories. This capacity for development, which Lakatos called the *"heuristic"*, is taken as an objective feature of the program. Lakatos distinguished between positive and negative heuristic. The positive heuristic represents the agreement among the theoreticians over the scientific evolution of the research programme. It is a kind of "problem solving machinery" composed by proposals and indications on the way to widen and enrich the research programme. The negative heuristic is the opposite of the positive. Within each research programme, it is important to maintain the core assumptions intact. It means that all the questions or methodologies that are not in accordance with the core assumptions must be rejected. All doubts appearing about the shared commitments of the main theoretical framework become a kind of negative heuristic of the research programme. When the negative heuristic considerable progresses, a research programme can become degenerative (i.e. it has more and more empirical anomalies).This means that theoreticians have to reconsider the core assumptions of the research programme, which can lead to the creation of another research programme. Kuhnian scientific revolutions are characterized by Lakatos as the defeat of one research programme by another.

At first glance, the concept of Lakatosian research programme seems to be close to the concept of Kuhnian paradigm. A research programme roughly corresponds to a paradigm, and a program change approximately resembles to a paradigm shift. The retention of the hard core and the positive heuristic in pursing a program reproduce the continuity of normal science.

Indeed, these two concepts are different. For Kuhn, the evolution of science could be represented by a broken line where discontinuity would mark the passage from one paradigm to another. In this sense, a particular science could contain only one paradigm at a given time and the new paradigm is incommensurable with the old one. In contract to the Kuhnian vision, Lakatos assumed that the simultaneous existence of several research programmes is the norm. Moreover, rival programs may contribute elements to each other, and degenerating programs are sometimes revived. In this vision, there is no discontinuity between the different research programmes and they are comparable to each other.

More precisely, Lakatos decompose the evolution of science into successive methodological and epistemological steps. These steps form a kind of vertical structure built with a multitude of *"layer of knowledge"* and where each layer represents a particular research programme. The emergence of a new research programme is induced by an empirical degeneration of a previously dominating research programme. The new research programme will constitute a superior layer of knowledge.

In the following section, we present the argumentations for the use of the Lakatosian research programme to explain the emergence of COTS-based system development.

A.4 CBSD: Paradigm Shift vs Research Programme

[165, 166] have presented a modern epistemological reading of software engineering as an evolving science. These articles have advocated the use of the Lakatosian research programme concept as an epistemological basis for the knowledge evolution in software engineering. Following [165, 166], we will argue thoroughly that it is not appropriate to see the emergence of COTS-based system development as a Kuhnian paradigm shift.

First, by adoption this vision, we would consider that the CBSD will completely replace the methodologies of the old paradigm. In reality, in most software projects, the custom and COTS-based system developments are used jointly because it is rarely to find the software components available in the market that meet all the system requirements. The required functionalities that are specific to each project will need to be developed in-house. In a Lakatosian vision, this cohabitation represents a progressive evolution of knowledge in software engineering. Indeed, the Lakatosian epistemology implies that the transition between research programmes is not clear and depends on the specific aspects of the experiment conducted (the software project is the experiment in our case). In this sense, the Kuhnian discontinuity between paradigms is not appropriate to explain the emergence of CBSD and custom system development.

Another drawback of the adoption of the Kuhnian paradigm shift is the incommensurability between paradigms in the sense that scientists within different paradigms are unable to understand each other. As evoked earlier, some activities

involved in building COTS-based systems are similar to those of custom system development such as requirements engineering, system architecture design, and testing. Based on numerous articles in the software literature, the methodologies employed to undertake these activities in the custom system development have been adapted to suit the changes brought by the use of COTS components. For instant, the goal-driven requirements engineering have been used in both custom and COTS-based system developments but the requirements defined in CBSD are more flexible and abstract than the requirements defined in custom system development. In other words, the methodologies used in CBSD are evolved from existing methodologies. Therefore, the methodologies of custom and COTS-based system developments are comparable. On the other hand, if we adopt the Lakatosian research programme wherein each research programme constitutes a higher level of knowledge and raises the abstraction level, CBSD can be seen as an evolution of the system development from scratch and as a new research programme because it raises that level higher. Moreover, the old and new research programmes are comparable which means that the custom and COTS-based system developments are commensurable.

Based on these arguments, we could see that the emergence of CBSD can not be described as a revolution or paradigm shift, in the Kuhnian sense but rather as a Lakatosian new research programme. CBSD is based on the basic knowledge that existed before its emergence. We could say that it has a hard core composed of the common activities that need to be done when acquiring a system including requirements engineering, system architecture design, and testing. Its protective belt could be characterized by the use of existing software components that bring new activities and changes to the new activities specific to the CBSD including selecting COTS components and developing glueware.

A.5 Conclusion

The use of COTS components in software developments bring a lot of changes to the traditional software development approach wherein every software components are built from scratch. We have adopted Lakatos' "research programme" rather than the Kuhn's "paradigm shift" for the emergence of CBSD. In other words, CBSD should be envisaged as a natural evolution rather than as a complete revolution. This adoption implies that our methodology is defined on the basis of existing methods and techniques for custom system development.

Appendix B

MAS Implementation with JADE: Source codes

B.1 MAS

B.1.1 `Gateway` Agent

```
1  package MAS;
2
3  import java.io.IOException;
4  import java.util.Hashtable;
5
6  import jade.core.AID;
7  import jade.core.Agent;
8  import jade.core.behaviours.Behaviour;
9  import jade.core.behaviours.FSMBehaviour;
10 import jade.core.behaviours.OneShotBehaviour;
11 import jade.lang.acl.ACLMessage;
12 import jade.lang.acl.MessageTemplate;
13 import jade.lang.acl.UnreadableException;
14 import jade.wrapper.gateway.GatewayAgent;
15
16 public class Gateway extends GatewayAgent {
17
18   private static final String STATE_A = "A";
19   private static final String STATE_B = "B";
20   private static final String STATE_C = "C";
21   private static final String STATE_D = "D";
22
23   private Request req;
24   private ACLMessage msg;
25   private MessageTemplate mt;
26
27   @Override
28   protected void processCommand(Object obj) {
29
30     // Creating a FMS behavior for handling the user request
31     FSMBehaviour fsm = new FSMBehaviour(this);
32
```

```
33    // Registering state A (first state)
34    fsm.registerFirstState(new GetUserRequest(this, obj), STATE_A);
35
36    // Registering state B
37    fsm.registerState(new SendUserRequest(this), STATE_B);
38
39    // Registering state C
40    fsm.registerState(new GetUserRequestResult(this), STATE_C);
41
42    // Registering state C (last state)
43    fsm.registerLastState(new SendUserRequestResult(this, obj), STATE_D);
44
45    // Registering the transitions
46    fsm.registerTransition(STATE_A, STATE_B, 1);
47    //Getting to the final state if the request is unknown
48    fsm.registerTransition(STATE_A, STATE_D, 0);
49    fsm.registerDefaultTransition(STATE_B, STATE_C);
50    fsm.registerDefaultTransition(STATE_C, STATE_D);
51
52    // Adding the FSM to the Gateway agent
53    addBehaviour(fsm);
54   }
55
56   private class GetUserRequest extends OneShotBehaviour {
57
58    Object obj;
59    int exitValue;
60
61    public GetUserRequest(Agent agent, Object obj) {
62     super(agent);
63     this.obj = obj;
64
65    }
66
67    public void action() {
68     if (obj instanceof Request) {
69
70      req = (Request) obj;
71      exitValue = 1;
72
73     } else {
74      exitValue = 0;
75
76     }
77
78    }
79
80    @Override
81    public int onEnd() {
82
83     return exitValue;
84    }
85
86   }// End of inner class
87
88   private class SendUserRequest extends OneShotBehaviour {
89
90    public SendUserRequest(Agent agent) {
91     super(agent);
92    }
```

```
93  |
94  |   public void action() {
95  |    msg = new ACLMessage(ACLMessage.REQUEST);
96  |    msg.addReceiver(new AID("Mediator", AID.ISLOCALNAME));
97  |    try {
98  |     msg.setContentObject(req);
99  |    } catch (IOException e) {
100 |
101 |     e.printStackTrace();
102 |    }
103 |    msg.setReplyWith("request" + System.currentTimeMillis());
104 |    send(msg);
105 |   }
106 |
107 |  }// End of inner class
108 |
109 |  private class GetUserRequestResult extends Behaviour {
110 |   boolean done = false;
111 |
112 |   public GetUserRequestResult(Agent agent) {
113 |    super(agent);
114 |   }
115 |
116 |   public void action() {
117 |    mt = MessageTemplate.and(
118 |      MessageTemplate.MatchSender(new AID("Mediator", AID.ISLOCALNAME)),
119 |      MessageTemplate.MatchInReplyTo(msg.getReplyWith()));
120 |    ACLMessage reply = receive(mt);
121 |    if (reply != null) {
122 |     try {
123 |      req.setResult((Hashtable) reply.getContentObject());
124 |      System.out.println(req.getResult());
125 |     } catch (UnreadableException e) {
126 |
127 |      e.printStackTrace();
128 |     }
129 |     done = true;
130 |
131 |    } else {
132 |     block();
133 |    }
134 |
135 |   }
136 |
137 |   public boolean done() {
138 |
139 |    return done;
140 |   }
141 |  }// End of inner class
142 |
143 |  private class SendUserRequestResult extends OneShotBehaviour {
144 |
145 |   Object obj;
146 |
147 |   public SendUserRequestResult(Agent agent, Object obj) {
148 |    super(agent);
149 |    this.obj = obj;
150 |   }
151 |
152 |   @Override
```

```
153     public void action() {
154
155      releaseCommand(obj);
156
157     }
158
159   }
160
161 }
```

Code extract B.1: Extract code of the `Gateway` agent.

B.1.2 `Mediator` Agent

```
1  package MAS;
2
3  import jade.core.Agent;
4  import java.io.FileNotFoundException;
5  import java.io.IOException;
6  import java.util.Hashtable;
7
8  import jade.core.behaviours.Behaviour;
9  import jade.core.behaviours.FSMBehaviour;
10 import jade.core.behaviours.OneShotBehaviour;
11 import jade.domain.DFService;
12 import jade.domain.FIPAException;
13 import jade.domain.FIPAAgentManagement.DFAgentDescription;
14 import jade.domain.FIPAAgentManagement.ServiceDescription;
15 import jade.lang.acl.ACLMessage;
16 import jade.lang.acl.MessageTemplate;
17 import jade.lang.acl.UnreadableException;
18 import MAS.Request;
19 import bsh.EvalError;
20 import bsh.Interpreter;
21
22 public class Mediator extends Agent {
23   private static final String STATE_A = "A";
24   private static final String STATE_B = "B";
25   private static final String STATE_C = "C";
26   private static final String STATE_D = "D";
27
28   private Request req;
29   private ACLMessage msgRequest, reply;
30   private MessageTemplate mt; // The template to receive replies
31   DFAgentDescription[] serviceProviders;
32   private String beanShellFileName;
33   private String userRequestFileName;
34
35   protected void setup() {
36
37     Object[] args = getArguments();
38     if (args != null && args.length > 1) {
39       beanShellFileName = (String) args[0];
40       userRequestFileName = (String) args[1];
41     }
42
43     FSMBehaviour fsm = new FSMBehaviour(this);
44
45     // Register state A (first state)
```

```
46    fsm.registerFirstState(new GetRequest(this), STATE_A);
47
48    // Register state B
49    fsm.registerState(new AnalyzeRequest(this), STATE_B);
50
51    // Register state C
52    fsm.registerState(new RealizeRequest(this), STATE_C);
53
54    // Register state D
55    fsm.registerState(new SendResult(this), STATE_D);
56
57    // Register the transitions
58    fsm.registerDefaultTransition(STATE_A, STATE_B);
59    fsm.registerTransition(STATE_B, STATE_C, 1);
60
61    //Getting back to the first state if the request is unknown
62    fsm.registerTransition(STATE_B, STATE_A, 0);
63
64    fsm.registerDefaultTransition(STATE_C, STATE_D);
65    fsm.registerDefaultTransition(STATE_D, STATE_A);
66
67    addBehaviour(fsm);
68
69  }
70
71  private class GetRequest extends Behaviour {
72    Boolean done = false;
73    private MessageTemplate mt;
74
75    public GetRequest(Agent agent) {
76      super(agent);
77
78    }
79
80    public void action() {
81      mt = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
82      msgRequest = receive(mt);
83      if (msgRequest != null) {
84        reply = msgRequest.createReply();
85        try {
86          Object contentObject = msgRequest.getContentObject();
87          if (contentObject instanceof Request) {
88            req = (Request) contentObject;
89            done = true;
90          }
91        } catch (UnreadableException e) {
92          e.printStackTrace();
93        }
94
95      } else {
96        block();
97      }
98    }
99
100   public boolean done() {
101
102     return (done);
103   }
104 }// End of inner class
105
```

```
106 |  private class AnalyzeRequest extends OneShotBehaviour {
107 |   int exitValue = 0;
108 |
109 |   public AnalyzeRequest(Agent agent) {
110 |    super(agent);
111 |
112 |   }
113 |
114 |   public void action() {
115 |
116 |    if (req != null) {
117 |     UserRequestDB userRequestDB = new UserRequestDB(userRequestFileName);
118 |     if (userRequestDB.contains(req))
119 |      exitValue = 1;
120 |    }
121 |   }
122 |
123 |   @Override
124 |   public int onEnd() {
125 |    return exitValue;
126 |   }
127 |
128 |  }// End of inner class
129 |
130 |  public class RealizeRequest extends OneShotBehaviour {
131 |
132 |   public RealizeRequest(Agent agent) {
133 |    super(agent);
134 |
135 |   }
136 |
137 |   public void action() {
138 |
139 |    Interpreter i = new Interpreter();
140 |    try {
141 |     i.set("myAgent", myAgent);
142 |     i.set("behaviour", this);
143 |     i.set("req", req);
144 |     i.source(beanShellFileName);
145 |    } catch (FileNotFoundException e) {
146 |     e.printStackTrace();
147 |    } catch (IOException e) {
148 |     e.printStackTrace();
149 |
150 |    } catch (EvalError e) {
151 |     e.printStackTrace();
152 |    }
153 |
154 |   }
155 |
156 |   public void executeSubRequest(Request subReq, String serviceType,
157 |     String serviceName) {
158 |    searchServiceProvider(serviceType, serviceName);
159 |    sendRequest(subReq);
160 |    getSubResult(subReq);
161 |   }
162 |
163 |   public void searchServiceProvider(String type, String name) {
164 |    try {
165 |     // Build the description used as template for the search
```

```
166     DFAgentDescription template = new DFAgentDescription();
167     ServiceDescription templateSd = new ServiceDescription();
168     System.out.println("Service provider search: " + serviceType + ":"
169       + serviceName);
170     templateSd.setType(type);
171     templateSd.setName(name);
172     template.addServices(templateSd);
173
174     DFAgentDescription[] results = DFService.search(myAgent, template);
175     System.out.println(results.length);
176     serviceProviders = results;
177
178    } catch (FIPAException fe) {
179     fe.printStackTrace();
180     serviceProviders = null;
181    }
182
183   }
184
185   public void sendRequest(Request request) {
186
187    if (serviceProviders.length > 0) {
188
189     // Send the cfp to all sellers
190     ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
191     for (int i = 0; i < serviceProviders.length; ++i) {
192      msg.addReceiver(serviceProviders[i].getName());
193     }
194     try {
195      msg.setContentObject(request);
196     } catch (IOException e1) {
197
198      e1.printStackTrace();
199     }
200     msg.setReplyWith("request" + System.currentTimeMillis());
201     myAgent.send(msg);
202     System.out.println("Message sent to service provider.");
203     // Prepare the template to get proposals
204     mt = MessageTemplate.and(
205       MessageTemplate.MatchConversationId("conversationId"),
206       MessageTemplate.MatchInReplyTo(msg.getReplyWith()));
207
208    } else {
209     System.out.println("Service provider not found.");
210
211    }
212   }
213
214   public void getSubResult(Request request) {
215    if (serviceProviders.length > 0) {
216     ACLMessage reply = myAgent.receive(mt);
217     while (reply == null)
218      reply = myAgent.receive(mt);
219     try {
220      request.setResult((Hashtable) reply.getContentObject());
221
222     } catch (UnreadableException e) {
223      // TODO Auto-generated catch block
224      e.printStackTrace();
```

```
225       }
226     }
227    }
228
229   }// End of inner class
230
231   private class SendResult extends OneShotBehaviour {
232
233    public SendResult(Agent agent) {
234     super(agent);
235
236    }
237
238    public void action() {
239
240     try {
241      reply.setPerformative(ACLMessage.INFORM);
242      reply.setContentObject(req.getResult());
243      myAgent.send(reply);
244     } catch (IOException e) {
245      e.printStackTrace();
246     }
247
248    }
249
250   }// End of inner class
251
252 }
```

Code extract B.2: Extract code of the `Mediator` agent.

B.1.3 `Service provider` Agent

```
1  package MAS;
2
3  import java.io.FileNotFoundException;
4  import java.io.IOException;
5  import java.util.Hashtable;
6  import java.util.Iterator;
7  import bsh.EvalError;
8  import bsh.Interpreter;
9  import jade.core.Agent;
10 import jade.core.behaviours.Behaviour;
11 import jade.core.behaviours.CyclicBehaviour;
12 import jade.core.behaviours.FSMBehaviour;
13 import jade.core.behaviours.OneShotBehaviour;
14 import jade.core.behaviours.ParallelBehaviour;
15 import jade.domain.DFService;
16 import jade.domain.FIPAException;
17 import jade.domain.FIPAAgentManagement.DFAgentDescription;
18 import jade.domain.FIPAAgentManagement.ServiceDescription;
19 import jade.lang.acl.ACLMessage;
20 import jade.lang.acl.MessageTemplate;
21 import jade.lang.acl.UnreadableException;
22
23 public class ServiceProvider extends Agent {
24  private static final String STATE_A = "A";
25  private static final String STATE_B = "B";
```

```
26  private static final String STATE_C = "C";
27  private static final String STATE_D = "D";
28
29  DFAgentDescription dfd;
30
31  private Request req;
32  private ACLMessage msgRequest, reply;
33  private String servicesFileName;
34  private String beanShellFileName;
35  private ServiceDB serviceDB;
36
37  @Override
38  protected void setup() {
39
40   dfd = new DFAgentDescription();
41   dfd.setName(getAID());
42   Object[] args = getArguments();
43   if (args != null && args.length > 0) {
44    servicesFileName = (String) args[0];
45    beanShellFileName = (String) args[1];
46    servicesRegister();
47
48   }
49
50   FSMBehaviour fsm = new FSMBehaviour(this);
51   // Register state A (first state)
52   fsm.registerFirstState(new GetRequest(this), STATE_A);
53
54   // Register state B
55   fsm.registerState(new AnalyzeRequest(this), STATE_B);
56
57   // Register state C
58   fsm.registerState(new RealizeRequest(this), STATE_C);
59
60   // Register state D
61   fsm.registerState(new SendResult(this), STATE_D);
62
63   // Register the transitions
64   fsm.registerDefaultTransition(STATE_A, STATE_B);
65   fsm.registerTransition(STATE_B, STATE_C, 1);
66
67   //Getting back to the first state if the request is unknown
68   fsm.registerTransition(STATE_B, STATE_A, 0);
69
70   fsm.registerDefaultTransition(STATE_C, STATE_D);
71   fsm.registerDefaultTransition(STATE_D, STATE_A);
72
73   //The fsm behavior for handling the request
74   //from other agent and a behavior for proactive
75   //actions are executed in parallel
76   ParallelBehaviour b = new ParallelBehaviour();
77   b.addSubBehaviour(fsm);
78   b.addSubBehaviour(new ProactiveAction(this));
79   addBehaviour(b);
80
81  }
82
83  protected void takeDown() {
84   // Deregister from the yellow pages
85   servicesDeregister();
```

```
86  |                                                                    |
87  |  }                                                                 |
88  |                                                                    |
89  |  public void servicesRegister() {                                  |
90  |   serviceDB = new ServiceDB(servicesFileName);                     |
91  |   Iterator it = serviceDB.getServiceDB().iterator();               |
92  |   Service service;                                                 |
93  |   while (it.hasNext()) {                                           |
94  |    service = (Service) it.next();                                  |
95  |    ServiceDescription sd = new ServiceDescription();               |
96  |    sd.setType(service.getType());                                  |
97  |    sd.setName(service.getName());                                  |
98  |    System.out.println(service.getType() + "␣" + service.getName());|
99  |    dfd.addServices(sd);                                            |
100 |   }                                                                |
101 |   try {                                                            |
102 |    DFService.register(this, dfd);                                  |
103 |   } catch (FIPAException fe) {                                     |
104 |    System.out.println("Service␣registrering␣exception.");         |
105 |    fe.printStackTrace();                                           |
106 |   }                                                                |
107 |                                                                    |
108 |  }                                                                 |
109 |                                                                    |
110 |  public DFAgentDescription[] searchServiceProvider(String type,    |
111 |    String name) {                                                  |
112 |   try {                                                            |
113 |    // Build the description used as template for the search       |
114 |    DFAgentDescription template = new DFAgentDescription();         |
115 |    ServiceDescription templateSd = new ServiceDescription();       |
116 |    System.out.println("Service␣provider␣search:␣" + serviceType + ":"|
117 |      + serviceName);                                               |
118 |    templateSd.setType(type);                                       |
119 |    templateSd.setName(name);                                       |
120 |    template.addServices(templateSd);                               |
121 |                                                                    |
122 |    DFAgentDescription[] results = DFService.search(this, template);|
123 |    return results;                                                 |
124 |                                                                    |
125 |   } catch (FIPAException fe) {                                     |
126 |    fe.printStackTrace();                                           |
127 |    return null;                                                   |
128 |   }                                                                |
129 |                                                                    |
130 |  }                                                                 |
131 |                                                                    |
132 |  public void servicesDeregister() {                                |
133 |   try {                                                            |
134 |    DFService.deregister(this);                                     |
135 |   } catch (FIPAException fe) {                                     |
136 |    fe.printStackTrace();                                           |
137 |   }                                                                |
138 |                                                                    |
139 |  }                                                                 |
140 |                                                                    |
141 |  public void servicesUpdate() {                                    |
142 |   servicesDeregister();                                            |
143 |   dfd.clearAllServices();                                          |
144 |   servicesRegister();                                              |
```

```
145 |   System.gc();
146 |  }
147 |
148 |  public void sendRequest(Request request, MessageTemplate mt,
149 |    DFAgentDescription[] serviceProviders) {
150 |
151 |   if (serviceProviders.length > 0) {
152 |    System.out.println("Service provider found: " + serviceProviders.length);
153 |    // Send the cfp to all sellers
154 |    ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
155 |    for (int i = 0; i < serviceProviders.length; ++i) {
156 |     msg.addReceiver(serviceProviders[i].getName());
157 |    }
158 |    try {
159 |
160 |     msg.setContentObject(request);
161 |
162 |    } catch (IOException e1) {
163 |      e1.printStackTrace();
164 |    }
165 |
166 |    msg.setReplyWith("request" + System.currentTimeMillis());
167 |    send(msg);
168 |
169 |    // Prepare the template to get proposals
170 |    mt = MessageTemplate.MatchInReplyTo(msg.getReplyWith());
171 |   } else {
172 |    req.setResult(null);
173 |   }
174 |  }
175 |
176 |  public void getSubResult(Request request, MessageTemplate mt) {
177 |   ACLMessage reply = receive(mt);
178 |   while (reply == null)
179 |    reply = receive(mt);
180 |   try {
181 |    Request result = (Request) reply.getContentObject();
182 |    request.setResult(result.getResult());
183 |
184 |   } catch (UnreadableException e) {
185 |    e.printStackTrace();
186 |   }
187 |  }
188 |
189 |  private class GetRequest extends Behaviour {
190 |   Boolean done = false;
191 |   private MessageTemplate mt;
192 |
193 |   public GetRequest(Agent agent) {
194 |    super(agent);
195 |
196 |   }
197 |
198 |   public void action() {
199 |    mt = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
200 |    msgRequest = receive(mt);
201 |    if (msgRequest != null) {
202 |     reply = msgRequest.createReply();
203 |     try {
```

```
204 |       Object contentObject = msgRequest.getContentObject();
205 |       if (contentObject instanceof Request) {
206 |        req = (Request) contentObject;
207 |        done = true;
208 |       }
209 |
210 |      } catch (UnreadableException e) {
211 |       e.printStackTrace();
212 |      }
213 |
214 |    } else {
215 |     block();
216 |    }
217 |   }
218 |
219 |   public boolean done() {
220 |
221 |    return (done);
222 |   }
223 |
224 |  }// End of inner class
225 |
226 |  private class AnalyzeRequest extends OneShotBehaviour {
227 |   int exitValue = 0;
228 |
229 |   public AnalyzeRequest(Agent agent) {
230 |    super(agent);
231 |
232 |   }
233 |
234 |   public void action() {
235 |
236 |    if (req != null) {
237 |     if (serviceDB.containServiceName(req.getName()))
238 |      exitValue = 1;
239 |    }
240 |   }
241 |
242 |   @Override
243 |   public int onEnd() {
244 |    return exitValue;
245 |   }
246 |
247 |  }// End of inner class
248 |
249 |  private class RealizeRequest extends OneShotBehaviour {
250 |
251 |   public RealizeRequest(Agent agent) {
252 |    super(agent);
253 |   }
254 |
255 |   public void action() {
256 |
257 |    Interpreter i = new Interpreter();
258 |    try {
259 |     i.set("myAgent", myAgent);
260 |     i.set("behaviour", this);
261 |     i.set("req", req);
262 |     i.source(beanShellFileName);
263 |    } catch (FileNotFoundException e) {
```

```
264       e.printStackTrace();
265     } catch (IOException e) {
266       e.printStackTrace();
267
268     } catch (EvalError e) {
269       e.printStackTrace();
270     }
271
272   }
273
274  } // End of inner class
275
276  private class SendResult extends OneShotBehaviour {
277
278   public SendResult(Agent agent) {
279    super(agent);
280
281   }
282
283   public void action() {
284
285    try {
286     reply.setPerformative(ACLMessage.INFORM);
287     reply.setContentObject(req.getResult());
288     myAgent.send(reply);
289
290    } catch (IOException e) {
291     e.printStackTrace();
292    }
293
294   }
295
296  }// End of inner class
297
298  private class ProactiveAction extends CyclicBehaviour {
299   public ProactiveAction(Agent agent) {
300    super(agent);
301   }
302
303   @Override
304   public void action() {
305    // TODO Auto-generated method stub
306
307   }
308
309  } // End of inner class
310
311 }
```

Code extract B.3: Extract code of the `Service provider` agent.

B.2 Supporting Classes

B.2.1 A Simple Example of Servlet Connecting to `Gateway` agent

```
1 package MAS;
2
```

```
3  import jade.core.Profile;
4  import jade.util.leap.Properties;
5  import jade.wrapper.gateway.JadeGateway;
6  import java.io.IOException;
7  import java.util.Hashtable;
8
9  import javax.servlet.ServletException;
10 import javax.servlet.http.HttpServlet;
11 import javax.servlet.http.HttpServletRequest;
12 import javax.servlet.http.HttpServletResponse;
13
14 public class MyServlet extends HttpServlet {
15  private final static String HOST = "localhost";
16  private final static String PORT = "8888";
17
18  @Override
19  public void init() throws ServletException {
20   super.init();
21
22   // Setting which class will be the GateWayAgent
23   Properties pp = new Properties();
24   pp.setProperty(Profile.MAIN_HOST, HOST);
25   pp.setProperty(Profile.MAIN_PORT, PORT);
26   JadeGateway.init("MAS.Gateway", pp);
27  }
28
29  protected void doGet(HttpServletRequest request,
30          HttpServletResponse response)
31    throws ServletException, IOException {
32
33   // Reading the user input
34   String bookTitle = request.getParameter("bookTitle");
35
36   // Createing a request to send to the Gateway agent
37   Hashtable parameters = new Hashtable();
38   parameters.put("title", bookTitle);
39   Request req = new Request("BookPrice");
40   req.setInput(parameters);
41   try {
42    // Sending request to Gateway agent
43    JadeGateway.execute(req);
44   } catch (Exception e) {
45    e.printStackTrace();
46   }
47   // Getting reply from the Gateway agent
48   if (req.getResult() == null)
49    request.setAttribute("Reply", "Unknown request.");
50   else {
51    Hashtable result = req.getResult();
52    int price = (Integer) result.get("price");
53    request.setAttribute("Reply", Integer.toString(price));
54   }
55
56   // Forwarding result to a JSP
57   this.getServletContext().getRequestDispatcher("myjsp.jsp")
58     .forward(request, response);
59  }
60 }
```

Code extract B.4: A simple example of a Servlet connecting with `Gateway` agent.

B.2.2 XMLReader

```
1
2  package MAS;
3
4  import java.io.IOException;
5  import java.util.ArrayList;
6  import java.util.Iterator;
7  import java.util.Scanner;
8
9  import javax.xml.parsers.DocumentBuilder;
10 import javax.xml.parsers.DocumentBuilderFactory;
11 import javax.xml.parsers.ParserConfigurationException;
12
13 import org.w3c.dom.Document;
14 import org.w3c.dom.Element;
15 import org.w3c.dom.NodeList;
16 import org.xml.sax.SAXException;
17
18 public abstract class XMLReader {
19  protected Document dom;
20  protected DocumentBuilder db;
21  protected DocumentBuilderFactory dbf;
22  protected String fileName;
23
24  public abstract void update();
25
26  protected abstract void parseDocument();
27
28  public abstract void printData();
29
30  public XMLReader(String fileName) {
31   dbf = DocumentBuilderFactory.newInstance();
32   // Using factory get an instance of document builder
33   try {
34    db = dbf.newDocumentBuilder();
35   } catch (ParserConfigurationException e) {
36    e.printStackTrace();
37   }
38   this.fileName = fileName;
39  }
40
41  protected void parseXmlFile() {
42
43   try {
44
45    // parse using builder to get DOM representation of the XML file
46    dom = db.parse(fileName);
47   } catch (SAXException se) {
48    se.printStackTrace();
49   } catch (IOException ioe) {
50    ioe.printStackTrace();
51   }
52  }
53
54  protected String[] getTextValues(Element e, String[] tags) {
55
56   String[] textValues = new String[tags.length];
57   for (int i = 0; i < tags.length; i++)
58    textValues[i] = getTextValue(e, tags[i]);
```

```
59
60    return textValues;
61   }
62
63   protected String getTextValue(Element ele, String tagName) {
64    String textVal = null;
65    NodeList nl = ele.getElementsByTagName(tagName);
66    if (nl != null && nl.getLength() > 0) {
67     Element el = (Element) nl.item(0);
68     textVal = el.getFirstChild().getNodeValue();
69    }
70
71    return textVal;
72   }
73
74   protected int getIntValue(Element ele, String tagName) {
75    // in production application you would catch the exception
76    return Integer.parseInt(getTextValue(ele, tagName));
77   }
78
79  }
```

Code extract B.5: Code extract of `XMLReader`.

B.2.3 ServiceDB

```
1
2   package MAS;
3
4   import java.util.ArrayList;
5   import java.util.Iterator;
6
7   import org.w3c.dom.Element;
8   import org.w3c.dom.NodeList;
9
10  public class ServiceDB extends XMLReader {
11
12   private ArrayList serviceDB;
13
14   public ServiceDB(String fileName) {
15    super(fileName);
16    serviceDB = new ArrayList();
17    parseXmlFile();
18    parseDocument();
19
20   }
21
22   public void update() {
23    serviceDB.clear();
24    parseXmlFile();
25    parseDocument();
26
27   }
28
29   public boolean containServiceName(String serviceName) {
30    Iterator it = serviceDB.iterator();
31    while (it.hasNext()) {
32     Service service = (Service) it.next();
```

```
33      if (service.getName().equalsIgnoreCase(serviceName))
34       return true;
35
36     }
37     return false;
38
39   }
40
41   protected void parseDocument() {
42    // get the root element
43    Element docEle = dom.getDocumentElement();
44
45    // get a nodelist of elements
46    NodeList nl = docEle.getElementsByTagName("Service");
47    if (nl != null && nl.getLength() > 0) {
48     for (int i = 0; i < nl.getLength(); i++) {
49      // get the employee element
50      Element el = (Element) nl.item(i);
51      // get the Employee object
52      String[] tags = { "Type", "Name" };
53      String[] textValues = getTextValues(el, tags);
54      Service service = new Service(textValues[0], textValues[1]);
55      // add it to list
56      serviceDB.add(service);
57
58     }
59    }
60   }
61
62   public ArrayList getServiceDB() {
63    return serviceDB;
64   }
65
66   public void printData() {
67
68    Iterator it = serviceDB.iterator();
69    while (it.hasNext()) {
70     System.out.println(it.next().toString());
71    }
72   }
73
74 }
```

Code extract B.6: Code extract of `ServiceDB`.

B.2.4 Service

```
1
2  package MAS;
3
4  public class Service {
5   String type;
6   String name;
7
8   public Service(String type, String name) {
9    super();
10   this.type = type;
11   this.name = name;
```

```
12  }
13
14  public String getType() {
15   return type;
16  }
17
18  public void setType(String type) {
19   this.type = type;
20  }
21
22  public String getName() {
23   return name;
24  }
25
26  public void setName(String name) {
27   this.name = name;
28  }
29
30  @Override
31  public String toString() {
32   return "Service [type=" + type + ", name=" + name + "]";
33  }
34
35 }
```

Code extract B.7: Code extract of `Service`.

B.2.5 UserRequestDB

```
1  package MAS;
2
3  import java.io.IOException;
4  import java.util.ArrayList;
5  import java.util.Iterator;
6
7  import javax.xml.parsers.DocumentBuilder;
8  import javax.xml.parsers.DocumentBuilderFactory;
9  import javax.xml.parsers.ParserConfigurationException;
10
11 import org.w3c.dom.Document;
12 import org.w3c.dom.Element;
13 import org.w3c.dom.NodeList;
14 import org.xml.sax.SAXException;
15
16 public class UserRequestDB extends XMLReader {
17
18  ArrayList userRequestDB;
19
20  public UserRequestDB(String fileName) {
21   super(fileName);
22   userRequestDB = new ArrayList();
23   parseXmlFile();
24   parseDocument();
25
26  }
27
28  public void update() {
29   userRequestDB.clear();
```

```
30    parseXmlFile();
31    parseDocument();
32
33   }
34
35   protected void parseDocument() {
36
37    // get the root element
38    Element docEle = dom.getDocumentElement();
39
40    // get a nodelist of elements
41    NodeList nl = docEle.getElementsByTagName("Request");
42    if (nl != null && nl.getLength() > 0) {
43     for (int i = 0; i < nl.getLength(); i++) {
44      // get the employee element
45      Element el = (Element) nl.item(i);
46      // get the Employee object
47      String[] tags = { "Name" };
48      String[] textValues = getTextValues(el, tags);
49      Request request = new Request(textValues[0]);
50
51      // add it to list
52      userRequestDB.add(request);
53     }
54    }
55   }
56
57   public void printData() {
58
59    Iterator it = userRequestDB.iterator();
60    while (it.hasNext()) {
61     System.out.println(it.next().toString());
62    }
63   }
64
65   public boolean contains(Request req) {
66    Iterator it = userRequestDB.iterator();
67    Request r;
68    while (it.hasNext()) {
69     r = (Request) it.next();
70     if (r.getName().equalsIgnoreCase(req.getName()))
71      return true;
72    }
73    return false;
74   }
75
76  }
```

Code extract B.8: Code extract of `UserRequestDB`.

B.2.6 Request

```
1  package MAS;
2  import java.io.Serializable;
3  import java.util.Hashtable;
4
5  public class Request implements Serializable {
6  private String name;
```

```
7  private Hashtable input;
8  private Hashtable result;
9
10 public Request(){
11         super();
12 }
13
14 public Request(String name) {
15         this.name = name;
16
17 }
18
19 public Request(String name, Hashtable data) {
20         this.name = name;
21         this.input = data;
22 }
23
24 public String getName() {
25         return name;
26 }
27 public void setName(String name) {
28         this.name = name;
29 }
30 public Hashtable getInput() {
31         return input;
32 }
33 public void setInput(Hashtable input) {
34         this.input = input;
35 }
36
37 public Hashtable getResult() {
38         return result;
39 }
40
41 public void setResult(Hashtable result) {
42         this.result = result;
43 }
44
45 @Override
46 public String toString() {
47         return "UserRequest [name=" + name + "]";
48 }
49
50 }
```

Code extract B.9: Code extract of `Request`.

B.2.7 BookDB

```
1  package MAS;
2
3  import java.util.ArrayList;
4  import java.util.Iterator;
5
6  import org.w3c.dom.Element;
7  import org.w3c.dom.NodeList;
8
9  public class BookDB extends XMLReader {
```

```
10 |
11 |  private ArrayList bookDB;
12 |
13 |  public BookDB(String fileName) {
14 |   super(fileName);
15 |   bookDB = new ArrayList();
16 |   parseXmlFile();
17 |   parseDocument();
18 |
19 |  }
20 |
21 |  public void update() {
22 |   bookDB.clear();
23 |   parseXmlFile();
24 |   parseDocument();
25 |
26 |  }
27 |
28 |  public boolean containBookTitle(String title) {
29 |   Iterator it = bookDB.iterator();
30 |   while (it.hasNext()) {
31 |    Book book = (Book) it.next();
32 |    if (book.getTitle().equalsIgnoreCase(title))
33 |     return true;
34 |
35 |   }
36 |   return false;
37 |
38 |  }
39 |
40 |  protected void parseDocument() {
41 |
42 |   // get the root element
43 |   Element docEle = dom.getDocumentElement();
44 |
45 |   // get a nodelist of elements
46 |   NodeList nl = docEle.getElementsByTagName("Book");
47 |   if (nl != null && nl.getLength() > 0) {
48 |    for (int i = 0; i < nl.getLength(); i++) {
49 |     // get the employee element
50 |     Element el = (Element) nl.item(i);
51 |     // get the Employee object
52 |     String[] tags = { "Title", "Price" };
53 |     String[] textValues = getTextValues(el, tags);
54 |     String title=textValues[0];
55 |     int price=Integer.parseInt(textValues[1])
56 |     Book book = new Book(title, price);
57 |     // add it to list
58 |     bookDB.add(book);
59 |    }
60 |   }
61 |  }
62 |
63 |  public ArrayList getBookDB() {
64 |   return bookDB;
65 |  }
66 |
67 |  public Book getBook(String title) {
68 |   Iterator it = bookDB.iterator();
```

```
69    while (it.hasNext()) {
70     Book book = (Book) it.next();
71     if (book.getTitle().equalsIgnoreCase(title))
72      return book;
73
74    }
75    return null;
76
77   }
78
79   public void printData() {
80
81    Iterator it = bookDB.iterator();
82    while (it.hasNext()) {
83     System.out.println(it.next().toString());
84    }
85   }
86
87  }
```

Code extract B.10: Code extract of `BookDB`.

B.2.8 Book

```
1  package MAS;
2  import java.io.Serializable;
3
4  public class Book implements Serializable {
5  private String title;
6  private int price;
7  public Book(String title,int price){
8   this.title=title;
9   this.price=price;
10 }
11 @Override
12 public String toString() {
13  return "Book [title=" + title + ", price=" + price + "]";
14 }
15 public String getTitle() {
16  return title;
17 }
18 public void setTitle(String title) {
19  this.title = title;
20 }
21 public int getPrice() {
22  return price;
23 }
24 public void setPrice(int price) {
25  this.price = price;
26 }
27
28
29
30 }
```

Code extract B.11: Code extract of `Book`.

B.3 XML Files

B.3.1 UserRequests

```
<?xml version="1.0" encoding="UTF-8"?>
<UserRequest>
  <Request>
        <Name>BookPrice</Name>

  </Request>
  <Request>
        <Name>BookCatalogue</Name>
  </Request>

</UserRequest>
```

Code extract B.12: Code extract of the `UserRequest` XML file.

B.3.2 Services

```
<?xml version="1.0" encoding="UTF-8"?>
<Services>
 <Service>
                <Type>Book</Type>
        <Name>BookPrice</Name>
  </Service>
  <Service>
                <Type>Book</Type>
        <Name>BookCatalogue</Name>
  </Service>

</Services>
```

Code extract B.13: Code extract of `Services` XML file.

B.3.3 Books

```
<?xml version="1.0" encoding="UTF-8"?>
<BookCatalogue>
  <Book>
        <Title>JADE</Title>
        <Price>10</Price>
  </Book>
    <Book>
        <Title>Agent</Title>
        <Price>20</Price>
  </Book>
  <Book>
        <Title>Java</Title>
        <Price>30</Price>
  </Book>
</BookCatalogue>
```

Code extract B.14: Code extract of `Books` XML file.

B.4 Bean Shell Files

B.4.1 A Simple Example of Bean Shell File for the `Mediator` Agent

```
1  import java.util.Hashtable;
2  import jade.core.Agent;
3  import jade.core.behaviours.OneShotBehaviour;
4  import MAS.Request;
5  import MAS.Mediator;
6
7
8  public void executeSubRequest(Request subReq, String serviceType,
9      String serviceName) {
10     behaviour.searchServiceProvider(serviceType, serviceName);
11     behaviour.sendRequest(subReq);
12     behaviour.getSubResult(subReq);
13   }
14
15 if (req.getName().equals("BookPrice")) {
16
17         executeSubRequest(req,"Book","BookPrice");
18
19 } else if (req.getName().equals("BookCatalogue")){
20         executeSubRequest(req,"Book","BookCatalogue");
21 }
```

Code extract B.15: Code extract of Bean shell script for the `Mediator`.

B.4.2 A Simple Example of Bean Shell File for a `Service provider` Agent

```
1  import java.util.Hashtable;
2  import jade.core.Agent;
3  import jade.core.behaviours.OneShotBehaviour;
4  import MAS.Request;
5  import MAS.BookDB;
6  import MAS.Mediator;
7  import MAS.Book;
8
9    if (req.getName().equals("BookPrice")) {
10    BookDB bookDB = new BookDB("BookCatalogue.xml");
11    Hashtable parameters = new Hashtable();
12    Hashtable inputs = req.getInput();
13    String title = inputs.get("title");
14    Book book = bookDB.getBook(title);
15    if (book != null) {
16     Hashtable result = new Hashtable();
17     result.put("price", book.getPrice());
18     req.setResult(result);
19    }
20   } else if (req.getName().equals("BookCatalgue")) {
21    BookDB bookDB = new BookDB("BookCatalogue.xml");
22    req.setResult(bookDB.getBookDB());
23   }
```

Code extract B.16: Code extract of Bean shell script for a `Service provider`.

Appendix C

Component Management System: Screenshots

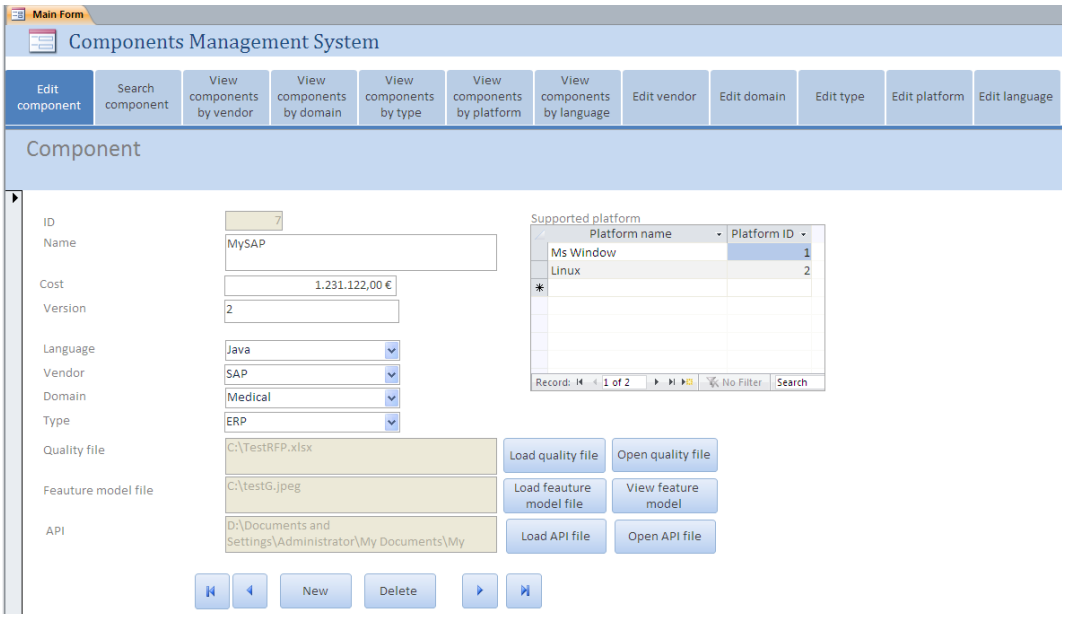


Figure C.1: Edit component in ComMS.

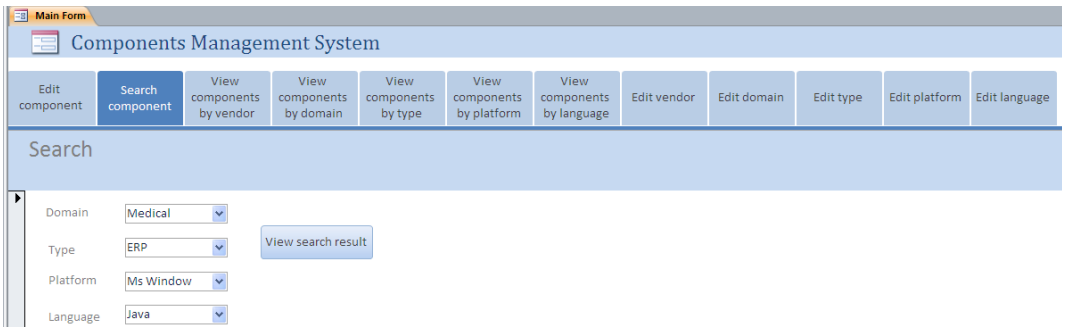


Figure C.2: Search component in ComMS.

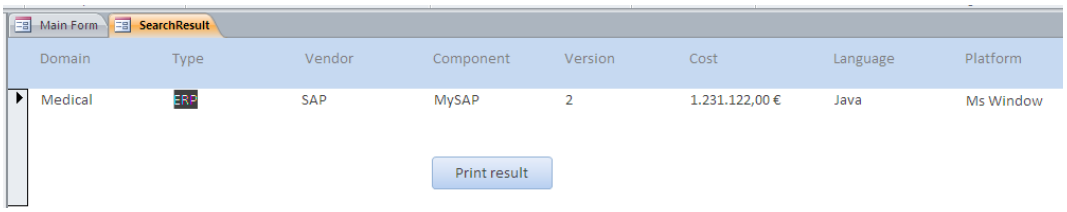


Figure C.3: Components search result in ComMS.

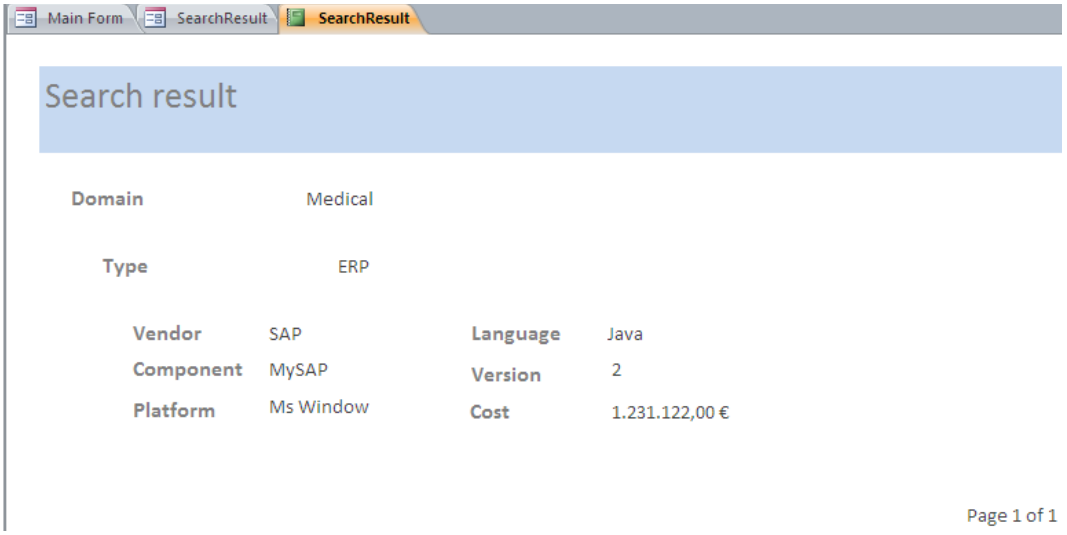


Figure C.4: Components search result report in ComMS.

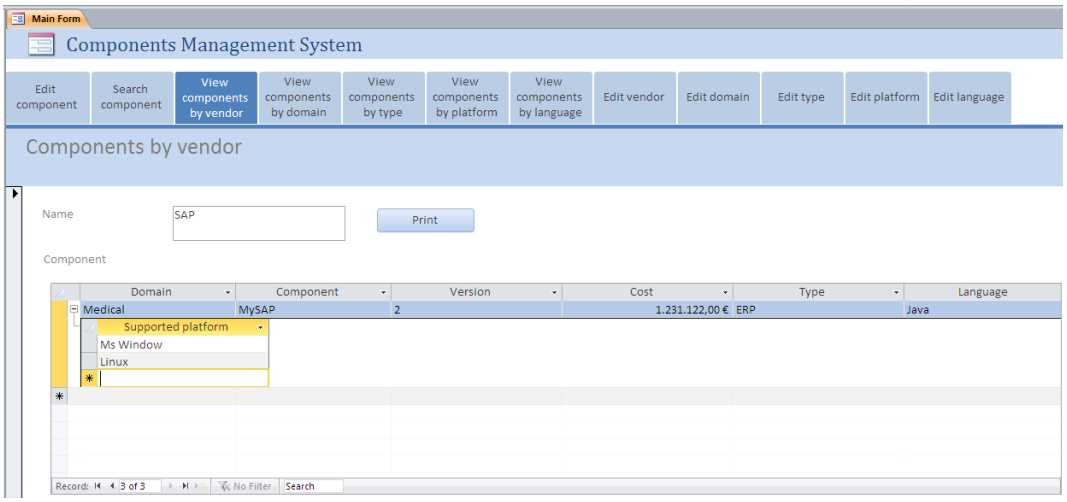


Figure C.5: View list of components per vendor in ComMS.

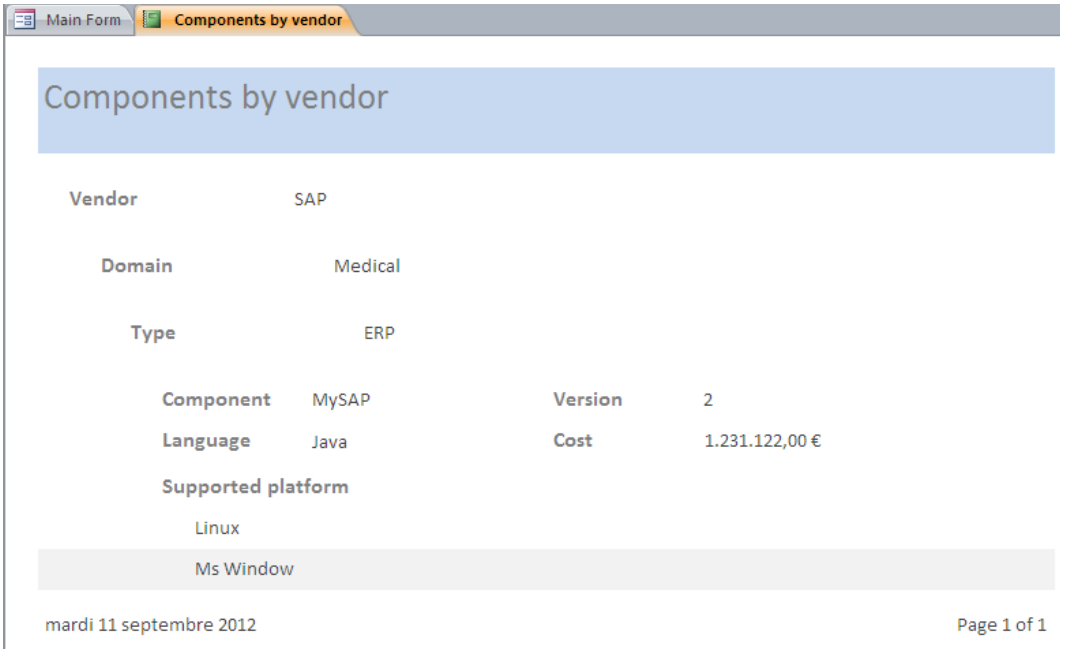


Figure C.6: List of components grouped by its vendor report in ComMS.

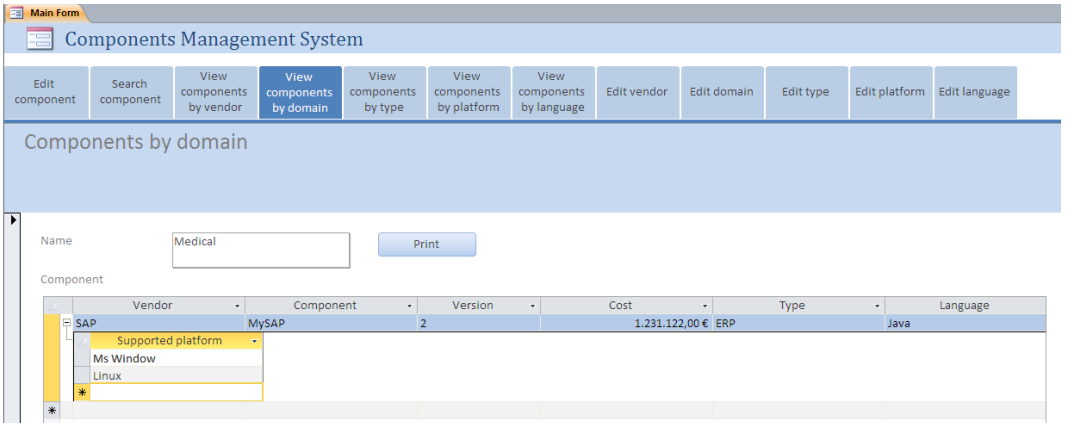


Figure C.7: View list of components per domain in ComMS.

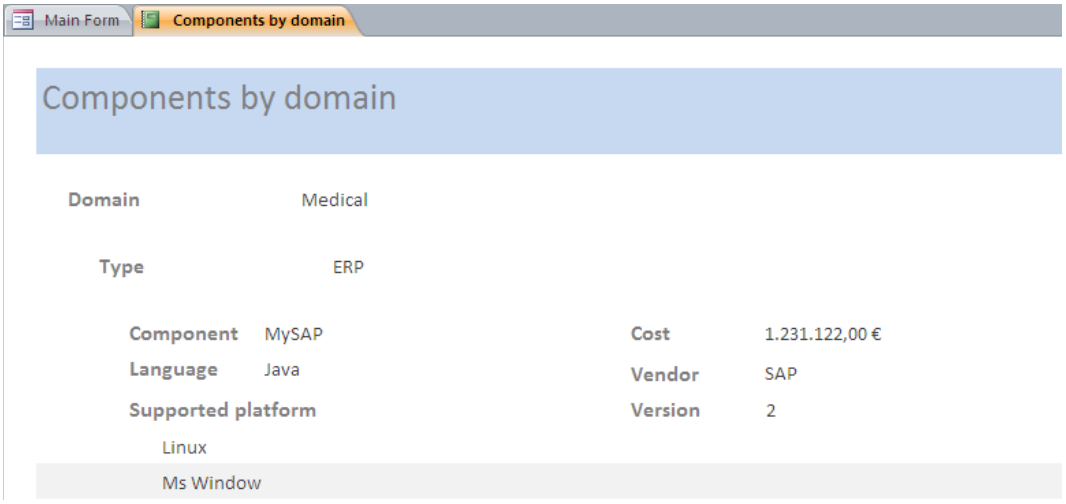


Figure C.8: List of components grouped by its domain report in ComMS.

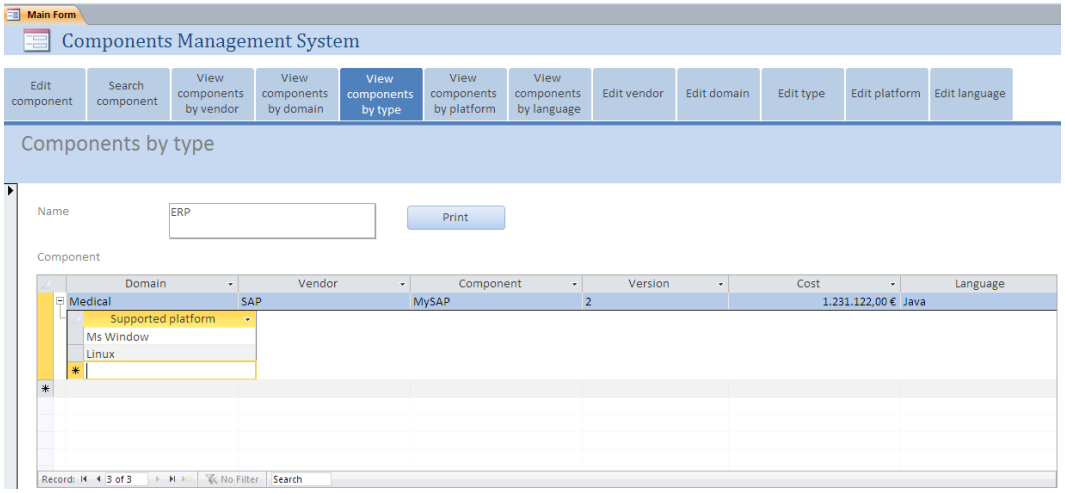


Figure C.9: View list of components per vendor in ComMS.

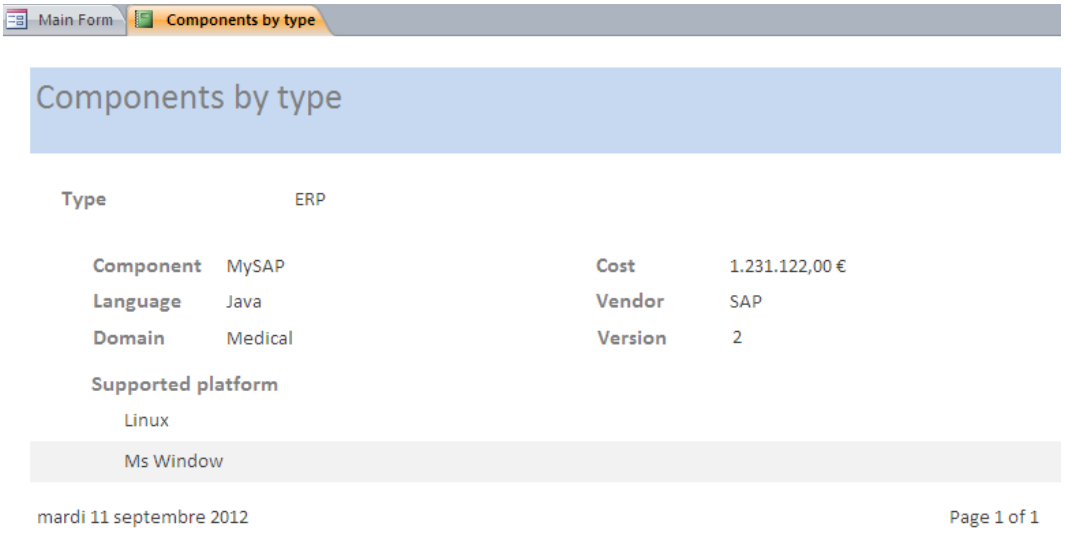


Figure C.10: List of components grouped by its type report in ComMS.

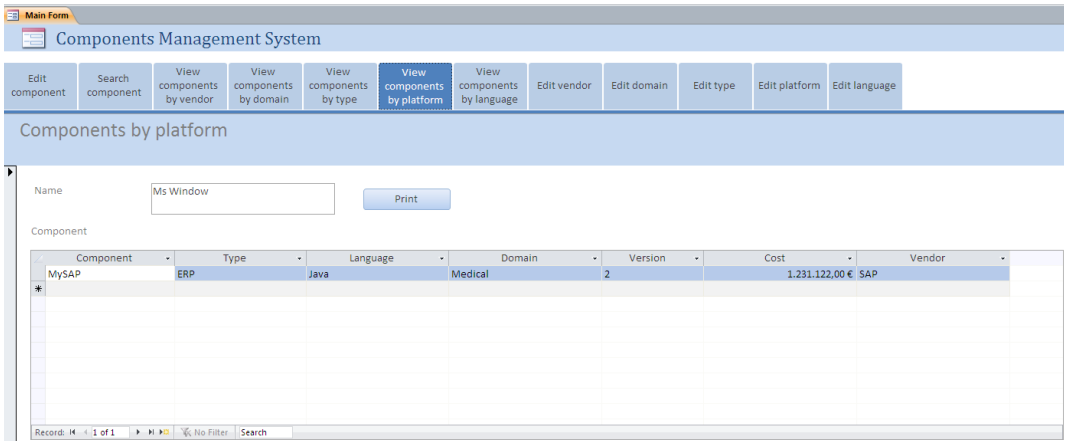


Figure C.11: View list of components per platform in ComMS.

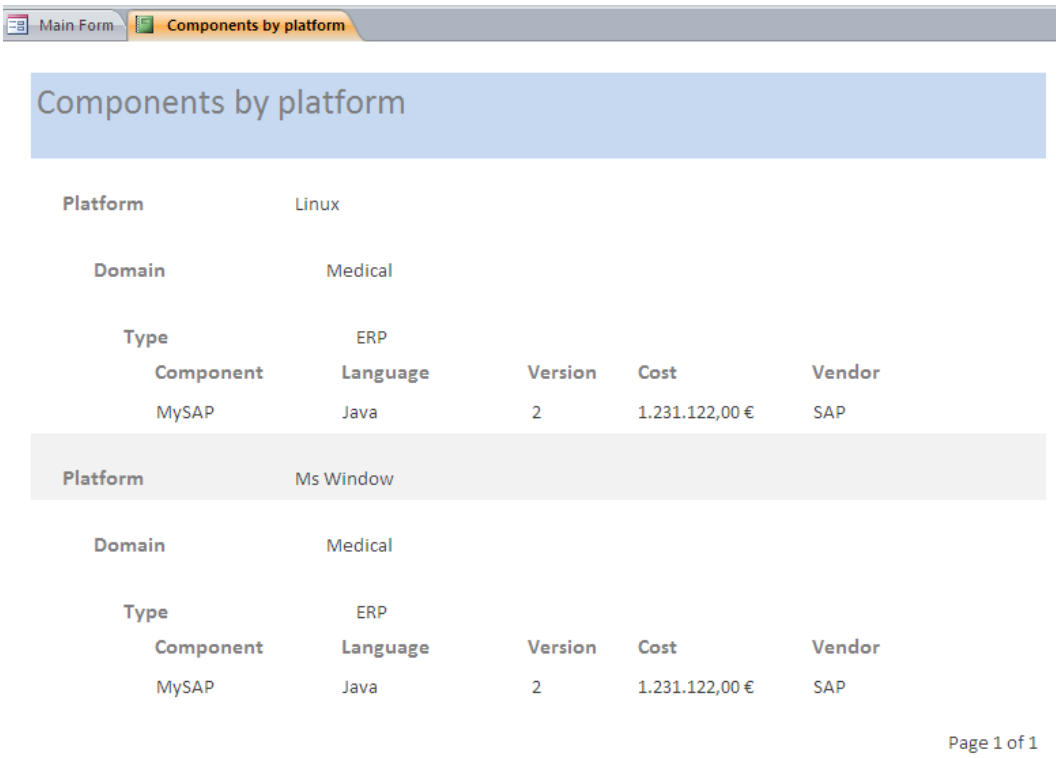


Figure C.12: List of components grouped by its platform report in ComMS.

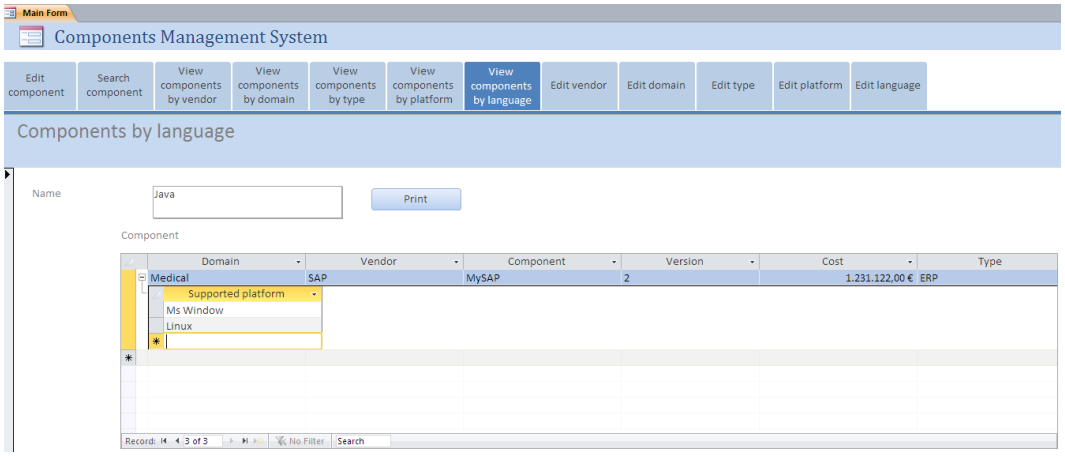


Figure C.13: View list of components per programming language in ComMS.

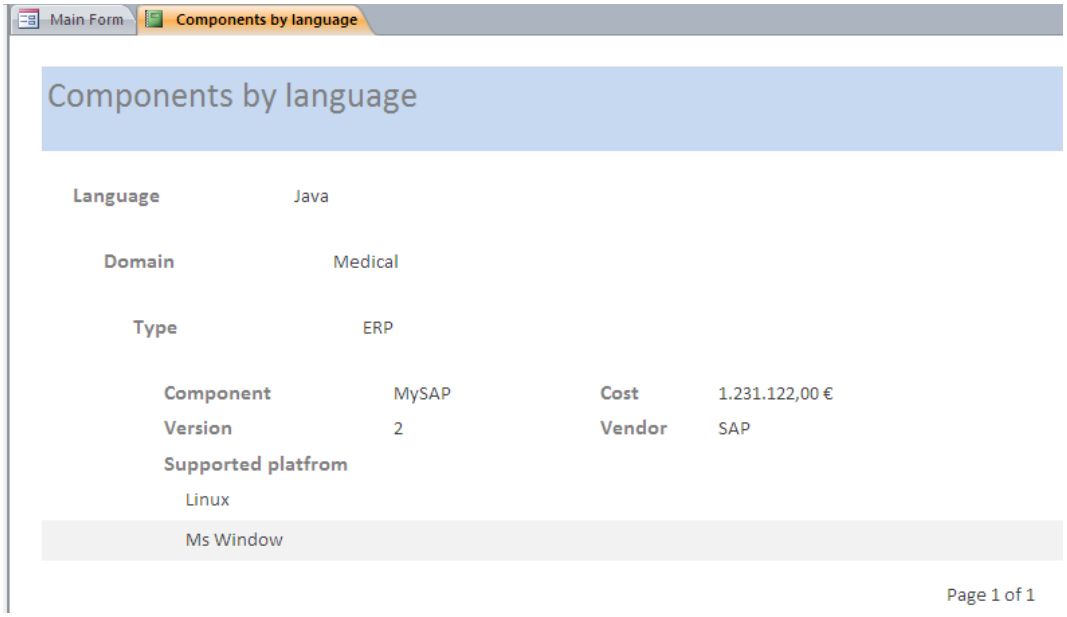


Figure C.14: List of components grouped by its programming language report in ComMS.

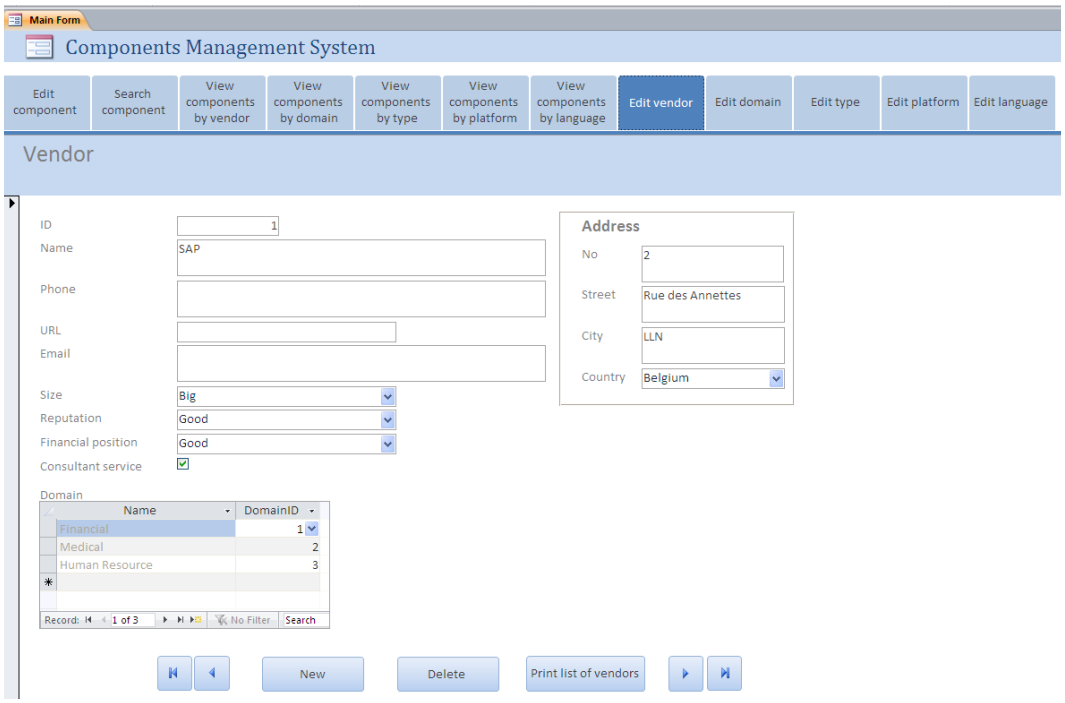


Figure C.15: Edit vendor information in ComMS.

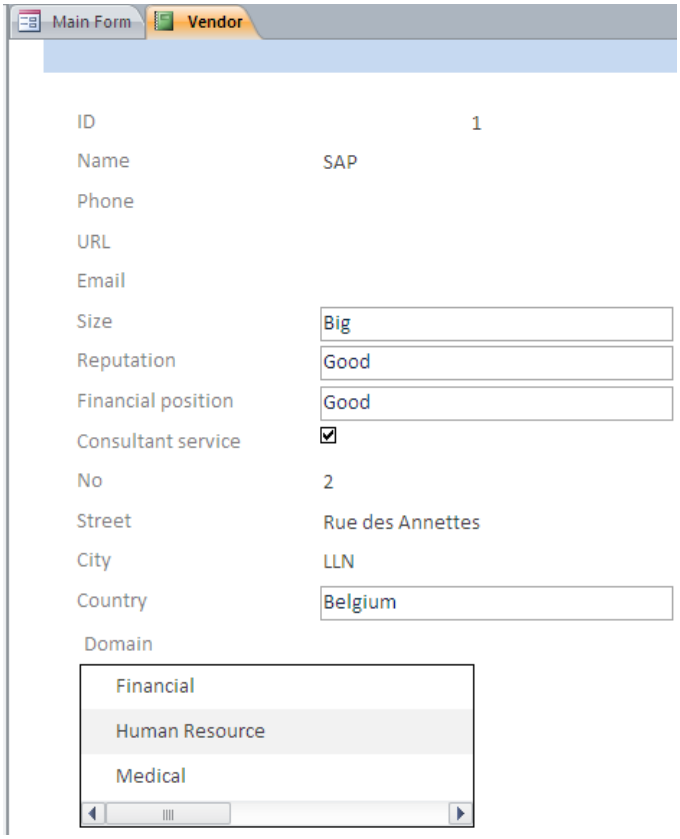


Figure C.16: List of vendors report in ComMS.

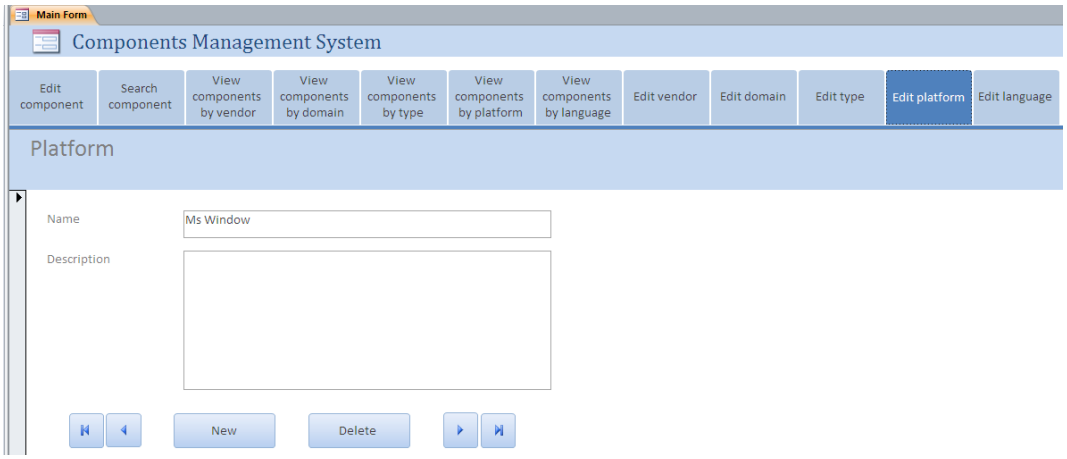


Figure C.17: Edit platform information in ComMS