


A Goal-Driven Project Management Framework for Multi-Agent Software Development: The Case of I-Tropos

Yves Wautelet

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

Doctor of Philosophy in
Economics and Management Sciences

of the Université catholique de Louvain

Committee in charge:

Prof. Manuel Kolp, Advisor
Prof. Jean Vanderdonckt, Examiner
Prof. Stéphane Faulkner, FUNDP, Examiner
Prof. Philippe Wilmès, Reader
Prof. Haralambos Mouratidis, University of East London, Reader

Summer 2008

A ma grand-mère, Elisabeth Votion, mami.

To my grandmother, Elisabeth Votion, mamy.

ABSTRACT

Today's enterprise information systems have to match with their operational and organizational environment. Unfortunately, software project development methodologies are traditionally inspired by programming concepts rather than by organizational and enterprise ones. In order to reduce as much this distance, agent-orientation is emerging and has been subject to various researches over the last 10 years. Its success comes from the fact that it better meets the increasing complexity and flexibility required to develop software applications built in open-networked environments and deeply embedded into human activities.

Thanks to benefits like efficient software project management, continuous organizational modeling and requirements acquisition, early implementation, continuous testing and modularity, iterative development is more and more used by software engineering professionals especially within object-oriented technologies through the *Unified Process*. Most multi-agent systems development methodologies only use a waterfall development life cycle or advice their users to proceed iteratively without offering a strong project management framework to support that way of proceeding. Consequently they are not suited for the development of huge and complex user-intensive applications. This dissertation presents a research aimed to build an original agent-oriented software development methodology using an iterative life cycle called *I-Tropos*.

The method fills up the traditional Tropos life cycle gaps and offers a goal-oriented project management framework to support project stakeholders when applying the method. The later covers several dimensions including risk, quality, time and process management. The methodology is validated on a real life case study in the steel industry and is supported by *DesCARTES Architect*, a CASE-Tool introduced in the dissertation.

ACKNOWLEDGMENTS

Above all, I would like to express my gratitude to my supervisor, Professor Manuel Kolp, for his guidance, advice, help, and continuing interest in my work. This dissertation has been possible thanks to the chance he gave me to work for the ISYS department. I am, however, not only grateful for what he taught me about research but also for having supported and enlighten me when things went wrong in different areas of life. In a few words I thank him for the friend he has been and will continue to be.

I would like to express my appreciation and thanks to the members of my PhD committee – Professors Jean Vanderdonckt, Stéphane Faulkner, Philippe Wilmès and Haralambos Mouratidis – for accepting to participate in the jury, for carefully reviewing my thesis and for their many useful suggestions and stimulating questions.

Epistemological foundations should never been underestimated by any PhD student! I consequently thank my friend Christophe Schinckus for the perspectives he gave me in that area on both my research topic and other "life stuff".

Also thank to my colleagues of the Louvain School of Management and of the TransLogisTIC project for their friendship, help, support and the nice work environment.

As my old friend Tung Do once said *the path to a PhD has many ups and downs*. When facing downs, family and friends give you the support to strike back. That is why I especially want to thank and express my gratitude to my cousin David Verpeut, my best friend Youssef Achbany, my nephew Corentin as well as Michaël Meyts, Nicolas Neysen and Benoît Wauters who have been there for me at crucial moments.

Finally, my gratitude goes to my father, Pierre – he is gone much too early – and to my mother, Anne-Marie, they both have always supported me, loved me and watched out I would never lack anything. Thank you so much for the education you gave me.

TABLE OF CONTENTS

Chapter 1: Introduction

Chapter 2: From Software Engineering to Project Management

Chapter 3: Epistemological Foundations

Chapter 4: A Framework for Software Process Evaluation

Chapter 5: I-Tropos: Software Process Description

Chapter 6: Case Study

Chapter 7: DesCARTES-Architect: a CASE-Tool for I-Tropos Software Development

1 Introduction

This chapter introduces the motivation for the thesis. In Section 1.1, we describe the advantages of multi-agent systems. Section 1.2 presents the importance of iterative development. We formulate our research proposal in Section 1.3. Section 1.4 presents research validation means while section 1.5 exposes the limitations. Finally, Section 1.6 proposes a reading map for the rest of the dissertation.

1.1 Advantages of Multi-Agent Systems

The meteoric rise of Internet and World-Wide Web technologies has created overnight new application areas for enterprise software, including eBusiness, web services, ubiquitous computing, knowledge management and peer-to-peer networks. These areas demand software that is robust, can operate within a wide range of environments, and can evolve over time to cope with changing requirements. Moreover, such software has to be highly customizable to meet the needs of a wide range of users, and sufficiently secure to protect personal data and other assets on behalf of its stakeholders.

Not surprisingly, researchers are looking for new software designs that can cope with such requirements. One promising source of ideas for designing such business software is the area of multi-agent systems. Multi-agent system architectures appear to be more flexible, modular and robust than traditional including object-oriented ones. They tend to be open and dynamic in the sense they exist in a changing organizational and operational environment where new components can be added, modified or removed at any time.

Multi-agent systems are based on the concept of an agent which is defined as “a software component situated in some environment that is capable of flexible autonomous action in order to meet its design objective” [AL98]. An agent exhibits the following characteristics:

- Autonomy: an agent has its own internal thread of execution, typically oriented to the achievement of a specific task, and it decides for itself what actions it should perform at what time.

- Situateness: agents perform their actions in the context of being situated in a particular environment. This environment may be a computational one (e.g., a Web site) or a physical one (e.g., a manufacturing pipeline). The agent can sense and affect some portion of that environment.

- Flexibility: in order to accomplish its design objectives in a dynamic and unpredictable environment, the agent may need to act to ensure that its goals

are achieved (by realizing an alternative plan). This property is enabled by the fact that the agent is autonomous in its problem solving.

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 [BMO01], or a goal-driven office delivery mobile device [CKM02]). However, in the overwhelming majority of cases, agents exist in an environment that contains other agents. Such an environment is 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. A multi-agent system is then conceived as a society of autonomous, collaborative, and goal-driven software components (agents), much like a social organization. Each role that an agent has the ability to play has a well defined set of responsibilities (goals) achieved by means of an agent's own abilities, as well as its interaction capabilities.

This *sociality* of MAS is well suited for tackling the complexity of today's organization software systems for a number of reasons:

- It permits a better match between system architectures and its organizational operational environment**,** for example a public organization, a corporation, a non-profit association, a local community.

- The autonomy of an agent (i.e., the ability an agent has to decide what actions it should take and at what time [AL98]) reflects the social and decentralized nature of modern enterprise systems [BMO01] that are operated by different stakeholders [Parunuak97].

- The flexible way in which agents operate to accomplish their goals is suited to the dynamic and unpredictable situations in which business software is now expected to run (see [ZJOW00, ZJW00]).

MAS architectures have become rapidly complicated due to the ever-increasing complexity of these new business domains and their human or organizational actors. As the expectations of the stakeholders change day after day, as the complexity of the systems, communication technologies and organizations continually increases in today's dynamic environments, developers are expected to produce architectures that must handle more difficult and intricate requirements that were not taken into account ten years ago, making thus architectural design a central engineering issue in modern enterprise information system life-cycle [AL98].

1.2 Iterative Development

The waterfall *software development life cycle*[1] (*SDLC*) [Royce70] describes a development process based on a series of phases (often called disciplines) that are fulfilled one after another in a linear/sequential way. This type of SDLC can only be successful when a series of assumptions are met at the same time. Otherwise, lots of risks are introduced into the software process. According to [Boehm00a], these assumptions are:

- The requirements can be known in advance of implementation;
- The requirements have no unresolved high-risk implications, such as risks due to cost, schedule, performance, safety, security, user interfaces, 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, developers, 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 likely be flawed with respect to its key requirements and late discovery of design defects results in costly over-runs and/or project cancellation. Consequently, time and resources will be wasted specifying and implementing requirements that are going to change and implementing a faulty design. The assumptions for an optimal waterfall SDLC are seldom met, root causes are:

- Requirements can rarely be known and defined in advance of implementation, especially for modern user-intensive dynamic systems. Most of them suffer from what is known as the IKIWISI syndrome [Boehm00b]: Users and stakeholders are often, during the early stages of the project, unable to express and describe what their requirements really are. When one asks about them they often reply *I can't tell you, but I'll know it when I see it* (IKIWISI);
- Defining requirements and freezing them early on in the project is a very risky task. Early defined requirements can meet users' expectations but often analysts discover later in the project that their practical achievement is (too)

[1] 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*" [Royce70].

constraining (in terms of cost, development time, processing time, human resources, etc.). These requirements expressed in a less constraining manner (i.e. longer response time from the system, less precision in the data analysis, etc.) keep meeting user's requirements but reduce the identified constraints in a drastic manner. Consequently, it is important not to fix and freeze 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 so that the requirements acquisition process cannot be made only once in the early stages but must be a continuous process. This development-method also avoids spending a huge time on requirements analysis at the beginning of the project specifying requirements that will appear to be out of date or be refined later in the project.

These remarks highlight the need for a SDLC that includes continuous requirements acquisition and modeling. Using an Iterative SDLC implies that risk is considered during the early stages of the project not at the late ones as in a process fully driven by sequential activities.

1.3 Towards a Framework for MAS Iterative Development

To integrate the benefits of MAS into daily industrial software developments, MAS should integrate the latest software engineering evolutions such as iterative development. To this end, **we propose to develop a MAS software engineering methodology called I-Tropos as an evolution of the traditional waterfall Tropos method**.

As a preliminary work, we will envisage the following question: what criteria of a software development methodology must be evaluated to determine its life cycle coverage? Indeed, purely waterfall development methodologies sometimes advise to repeat their stages iteratively during a particular project even when no theoretical framework to support this is furnished (see for example [BS02, BGP02]). We believe this is abusive since a SE methodology using an evolved life cycle has to offer a theoretical framework to support its own development process often called project management (PM). In this perspective we have developed a framework to evaluate a methodology's SDLC in Chapter 3. This framework includes ten evaluation criteria owned by one of the two following families:

- **Engineering concepts**. How does the process organize its life cycle?

- **Project management concepts**. Does the methodology offer a project management perspective to drive software development?

According to us, iterative life cycle software engineering on large projects involving huge resources requires a project management framework. Indeed, modern itera-

tive development methodologies are made of software engineering disciplines allowing to produce requirements models, designing the application, implementing the functionalities, testing and deploying them but also project management disciplines to plan development activities, staff the project, manage risks, quality factors, evaluate costs, etc.

We need to make the distinction between SE and PM clear. That is why we will consider that, in software development, engineering refers to the ontologies[2], models, guidelines, programming languages and other artifacts used when building software allowing to formalize, represent and develop the desired system. On the other hand, the art of balancing human resources onto specific tasks to meet user requirements with respect to available calendar time and financial resources, identified risks, expected quality levels, etc. following a defined life cycle belongs to management. However, risk and quality management do refer to the two disciplines since:

- risk and quality factors are taken into account from a managerial perspective (at process level) to allocate more resources to deal with a risk or meet the desired quality level, prioritize requirements elicitation of the most risky issues etc.

- risk and quality factors are subject to technical choices, solutions, designs and implementations so that they are part of the software engineering field.

As far as we are concerned, we will consider the software engineering field when refeering to the I-Tropos core disciplines (*Organizational Modeling*, *Requirements Engineering*, *Architectural Design*, *Detailed Design*, *Implementation*, *Test* and *Deployment*) because they refer to practical technical choices to build the software product. The I-Tropos support disciplines (*Risk Management*, *Quality Management*, *Time Management* and *Process Management*) will be referred to as project management since they are mostly concerned with the management of resources to specific tasks. As pointed out earlier, risk and quality management could also reefer to software engineering but since, in this dissertation, we mostly focus on the "process" level, we reefer to them as project management.

We can now consider the position of the development methodologies such as Tropos. Tropos as defined in [CKM02] is only a set of phases into which one or several models are built with variable scope. With respect to the above definitions, all the phases described reefer to software engineering, none to project management. One can consider that the process is described in a generic or at least flexible way to be included into a defined software development life cycle (possibly iterative). This statement has, as far as we know, never been introduced by the methodology authors and we argue that:

[2] In both computer science and information science, an ontology is a formal representation of a set of concepts within a domain and the relationships between those concepts. It is used to reason about the properties of that domain, and may be used to define the domain.

- the use of an elaborated development life cycle (as iterative one) requires a specific framework including project management;

- a project management framework can hardly be defined in a completely generic manner particularly if it is model driven. Indeed, the later requires a study of the analysis models and its components to find the adequate scope elements to drive the software process. A couple of principles to define such elements can however be determined.

The project management framework described in this dissertation is applied to Tropos to build I-Tropos methodology. It constitutes a first attempt to include a mature development life cycle into MAS development. We could envisage, with some adaptations, to apply to other development methodologies which could be agent but also object oriented or even other research programmes[3]. The key feature to adapt the framework is the adequate identification and definition of scope elements into analysis models for model driven software project management.

1.4 Validation

The I-Tropos methodology has been applied to case studies. The use of the methodology on the development of a production management system of a coking plant is described in Chapter 6.

Empirical evaluation of the benefits of AOSE iterative development is hard to measure. This could be done by achieving a similar case study with I-Tropos on the one side and with Tropos or other waterfall AOSE methodologies on the other using development teams with similar experience. Such an experience is practically unachievable. However, if it was, we could measure both PM and SE metrics to evaluate the impact of iterative development on each aspect.

First of all, the PM metrics that could be interesting to evaluate are the following:

- The general users' satisfaction towards the developed application;

- Risk management: for each of the I-Tropos identified risks, we could measure quantifiably users' satisfaction on the functions and properties concerned with the particular threat; measure the threat impact in case of occurence;

- Quality management: for each of the I-Tropos identified quality factors, we could measure quantifiably users' satisfaction on the functions and properties concerned with the particular quality factor;

- Time management: on a global basis we could compare:

[3] In the Lakatosian sense.

- Development time;
- Development costs.

In a more general manner, we could test the I-Tropos process performance by comparing every estimated factor and comparing it to the effectively measured so that a general process performance level could be computed. This would allow to calibrate and to refine the process notably in the process management discipline.

Moreover, a series of SE metrics for the evaluation of the software produced by the I-Tropos process is given into Chapter 3.

1.5 Limitations

The I-Tropos methodology constitutes more than a simple evolution of Tropos since the ambition is to get a fully operational development process. Inherited from Tropos, we use i* [Yu95] models for analysis disciplines and extended UML diagrams (mostly issued from the Skwyrl process [DFK03]) and do not consider the equivalent models defined into other agent-oriented methodologies such as GAIA [WJK00], MASE [WDS01] and MESSAGE [Cairo02].

The process description proposed in this dissertation constitutes a contribution to the definition of agent-oriented iterative development. This dissertation focuses on the process description by conceptualizing a framework to explore and facilitate the building of MAS. It models and introspects the process along different complementary stages. We, nevertheless, point out some important limitations of our research:

- We only consider the design of cooperative MASs. Indeed, MAS may be either cooperative or competitive. In a cooperative MAS, the agents cooperate together in order to achieve common goals. Inversely, a competitive MAS is composed of agents that pursue personal goals and defend their own interests. The design of competitive MAS is left for future developments;

- The process needs to gain experience with its use. In this dissertation, we have applied it on a case study. By doing so, we have tailored the process to a particular project and shown how our framework can help the design of MAS. However, it should be tested on more case studies and knowledge base should evolve from project to project. This is particularly true for effort evaluation, risk and quality assessments, etc. where only an accumulation of empirical data can lead to more accurate predictions. Indeed, each organization shall calibrate models with its own collected data betraying its own specificities;

- Process description includes the use of software patterns, the reuse of the *Commercial Off The Shelf* (*COTS*) and open source components, an explicit

and systematic approach for selecting and applying those features could be included into the process rather than using an ad hoc manner;

- The methodology is mostly designed for large**,** user intensive, organizational projects. It is especially suited for companies IS development rather than for stand alone developments.

1.6 Reading Map

In addition to the introduction and the conclusion, this dissertation is organized in six chapters.

Chapter 2 overviews software engineering (SE) and project management (PM) states of the art. The chapter defines SE, overviews the main SE phases before turning on to some of the main well known SDLCs. The Rational Unified Process and agile modeling techniques are also evoked. The chapter then turns on to PM, it overviews risk, time and quality management techniques.

Chapter 3 introduces the epistemological foundations of agent-oriented iterative development. In this chapter we propose a modern epistemological reading of SE as an evolving science as well as software project knowledge evolution; project actors' reasoning context is also studied. Popperian theories have been used in literature to describe the evolution of knowledge within a software project (at micro level). Due to the nature of SE belonging as well to social sciences we will show that this vision is abusive at both micro level (knowledge within the project) and macro level (knowledge on development frameworks). We also propose a modern epistemological validation of the emergence of Object-Orientation (OO) and Agent-Orientation (AO).

Chapter 4 is entirely dedicated to the context of the development of the I-Tropos process that is why it is called "*Towards an Iterative MAS Development Methodology*". The chapter firstly introduces Agent-Oriented Software Engineering (AOSE) then positions SE against PM. Software development methodologies can use plenty of different life cycles, a multiple-criteria framework for evaluating a method life cycle is developed and applied on four main AOSE processes. The building process of I-Tropos is finally overviewed, it encompasses the theoretical foundations of the process, the study of a mean to formalize (document) the process, the way the process has been designed, a comparison with the traditional Tropos development method and finally points to further works and developments.

Chapter 5 presents the I-Tropos software process itself. It studies i* elements and refines some of their definitions in order to find the most appropriate scope element to drive the software process. After that, the process characteristics are exposed. Finally, the process' core and support disciplines are described in a generic way using the SPEM.

In Chapter 6, the I-Tropos software process is applied on a case study, a production management system development for a coking plant. Production of coke is one of the steps of steel making; the reasons for the choice of such a case study are exposed. After that, the chapter explains the steps in steel production as well as the main project characteristics. Developments made during the four project phases are depicted in details.

Chapter 7 presents DesCARTES-Architect, the CASE-Tool developed to support I-Tropos developments. This tool addresses to analysts, designers, software architects, developers and even project managers since it allows to represent the models in a graphical way, to reuse the catalogue of architectural styles and other design patterns to construct the MAS, to generate code on the basis of design models and to manage the software project.

Finally, Chapter 8 concludes this dissertation. It summarizes the work done, exposes the main contribution of the thesis and points to future work.

References

[AL98] Y. Aridor and D.B. Lange. “Agent Design Patterns: Elements of Agent Application Design”. In *Proceedings of the 2nd Int. Conf. on Autonomous Agents (Agents’98*), Minneapolis, USA , pp.108-115, 1998.

[BMO01] B. Bauer, J. P. Muller and J. Odell. “Agent UML: A Formalism for Specifying Multiagent Interaction”. In *Proceedings of the 1st Int. Workshop on Agent-Oriented Software Engineering (AOSE’00)*, Limerick, Ireland, pp.91-103, 2001.

[BGP02] C. Bernon, M-P. Gleizes, G. Picard and P. Glize. “The Adelfe Methodology for an Intranet System Design”. In *Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002) at CAiSE'02*, Toronto, Ontario, Canada, 2002.

[Boeh00a] B. Boehm. “Spiral Development: Experience, Principles, and Refinements”. In *Proceedings of the Spiral Development Workshop*, 2000.

[Boeh00b] B. Boehm. “Requirements that Handle IKIWISI, COTS, and Rapid Change”. *Computer,* IEEE, 34(5), pp. 99-102, 2000.

[BS02] P. Bresciani and F. Sannicolo. “Applying Tropos Requirements Analysis for defining a Tropos tool”. In P. Giorgini, Y. Lespérance, G.Wagner, and E. Yu, editors, *Agent-Oriented Information System. Proceedings of AOIS-2002: Fourth International Bi-Conference Workshop*, Toronto, Canada, pages 135-138, 2002.

[Cairo02] J. Caire & al. “Agent-oriented analysis using MESSAGE/UML”. In *Proceedings of the 2nd Int. Workshop on Agent-Oriented Software Engineering*, LNCS 2222, 119-135, 2002.

[CKM02] J. Castro, M. Kolp and J. Mylopoulos. “Towards A Requirements-Driven Development Methodology : The Tropos Project”. In *Information Systems*, Elsevier, 2002.

[DFK03] T.T. Do, S. Faulkner and M. Kolp. “Organizational Multi-Agent Architectures for Information Systems”. In *Proceedings of the 5th International Conference on Enterprise Information Systems* (*ICEIS 2003*), 89-96, 2003.

[Parunak97] V. Parunak. “Go to the ant: Engineering principles from natural agent systems”. In *Annals of Operations Research*, 75, 69-101, 1997.

[Royc70] W. Royce. “Managing the Development of Large Software Systems”, In *Proceedings of the IEEE WESCON*, pp. 1-9, August 1970.

[WDS01] M. Wood, S.A. DeLoach and C. Sparkman, “Multi-Agent System Engineering”. In *International Journal of Software Engineering and Knowledge Engineering*, 11(3), 231-258, 2001.

[WJK00] M. Woodridge, N.R. Jennings and D. Kinny. “The Gaia Methodology for Agent-Oriented Analysis and Design”. In *Autonomous Agents and Multi-Agent Systems*, 3(3), 285-312, 2000.

[Yu95] E. Yu. “Modeling Strategic Relationships for Process Reengineering”. *PhD thesis*, *University of Toronto*, *Department of Computer Science*, Canada, 1995.

[ZJOW00] F. Zambonelli, N.R. Jennings, A. Omicini and M. Wooldridge. “Agent-Oriented Software Engineering for Internet Applications”. In *Coordination of Internet Agents: Models, Technologies and Applications*, Springer Verlag, 326-346, 2000.

[ZJW00] F. Zambonelli, N.R. Jennings and M. Wooldridge. “Organizational abstractions for the analysis and design of multi-agent systems”. In *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering*, pp. 243-252, 2000.