Discussing strategies for software architecting and designing from an Agent-oriented point of view*

Anna Perini, Angelo Susi
ITC-irst
Via Sommarive, 18
I-38050, Povo, Trento, Italy
{perini,susi}@irst.itc.it

ABSTRACT

Software design is a complex problem-solving process which requires to evaluate several design options while pursuing the objective of adhering to general principles of good quality. For instance, designing open, robust and secure architectures for novel application areas, calls for evaluating and/or integrating different distributed system technologies, such as peer-to-peer and multi-agent systems.

This paper proposes a discussion on software architecting and designing strategies adopting an agent-oriented approach. Several works face this issue proposing methodologies based mainly on the use of architectural styles. Here we consider general strategies useful in the design process, such as divide-and-conquer, and principles for a good design, such as cohesion and decoupling, and argue that, performing goal analysis, according to *Tropos*, an agent-oriented methodology, provides methods for applying them. A case study in the domain of decision-support systems in agriculture is used to illustrate.

1. INTRODUCTION

Agent-oriented software engineering is being proposed as a general framework for analysis and design, in spite of having adopted Multi-Agent System (MAS) as the target development platform, especially when complex, distributed systems are concerned [4, 22, 9].

The idea is that agent-oriented methodologies, which are founded on notions such as those of agent, goal, plan, are inherently *intentional*, and allow to model explicitly reasons behind the needs of the application domain stakeholder, as well as reasons behind system requirements. On the other side, other approaches, such as object-oriented methodologies, are inherently *not* intentional, since they use ontological primitives which are an abstraction of implementation-level concepts.

Along this line we proposed a *knowledge-level* [17] approach to software development, providing a methodology, called *Tropos* which spans from early requirements to implementation [10, 2]. *Tropos* rests on the uniform use of a small set of intentional notions, such as actor, goal and dependency, which proved to be suitable for modeling the organization where the system-to-be has to be introduced [23, 5], during all phases of software development.

In particular, the methodology covers five software development phases: *early requirements analysis*, *late requirements analysis*, *architectural design*, *detailed design*, and *implementation*. *Early Requirements* analysis focuses on the understanding of a problem domain by studying an *existing organizational setting* where the system-to-be will be introduced. Social actors and software systems that are already present in the domain are modeled as actors with their individual goals and with mutual, intentional dependencies.

Late Requirement analysis focuses on the system-to-be which is introduced as a new actor into the model. The system actor is related to the social actors in terms of dependencies; its goals are analyzed and will eventually lead to revise and add new dependencies involving a subset of the social actors (the users).

Architectural design defines the system's global architecture in terms of subsystems, that are represented as actors. They are assigned subgoals or subplans of the goals and plans assigned to the system.

Detailed design aims at specifying the agent micro-level. At this point, usually, the implementation platform has already been chosen and this can be taken into account in order to perform a detailed design that will map directly to the code.

The *Implementation* activity produces an implementation skeleton according to the detailed design specification. Code is added to the skeleton using the programming language supported by the implementation platform.

The methodology has been illustrated by several case studies ([19], [8], [10], [2]).

*

In this paper we propose a discussion on general strategies for software design and architecting, adopting an agent-oriented approach based on the *Tropos* methodology.

Software design is a well recognized complex problem-solving process aimed at specifying how to implement the system's functional requirements while satisfying the constraints imposed by the non-functional requirements and while pursuing general good quality objectives, such as understandability, maintainability, reusability, etc. [13]. Basic software engineering literature provides general principles and methodologies [6, 20].

Methods which are suitable to apply them in order to deal with design issues which emerge in specific application classes are then to be provided.

For instance, designing complex distributed systems, where different technologies, ranging from e-services to peer-to-peer [16] and MAS, are candidate technologies for building open, robust and secure architectures, we need to decide how to compare and, possibly, how to integrate them in an effective way [1].

In this paper, we focus on general strategies useful in the design process, such as divide-and-conquer, and principles for a good design, such as cohesion and decoupling, and argue that, performing goal analysis, according to the *Tropos* methodology provides methods for applying them.

The paper is structured as follows. Section 2, introduces some basic concepts of the *Tropos* methodology and illustrate them with examples taken from a case study in the domain of decision-support systems in agriculture, described with more details in [18]. Section 3, focuses on the refinement of the system requirement model into the system design model. Here we present an interpretation of general software engineering principles for a "good" software design, from the point of view of our agent-oriented approach. Related works are discussed in Section 4. Finally, conclusions are presented in Section 5.

2. OUR APPROACH TO AGENT-ORIENTED SOFTWARE DEVELOPMENT

From a practical point of view, the *Tropos* methodology guides the software engineer in building and refining conceptual models, with the help of a visual modeling language which provides an ontology including intentional concepts, such as actor, goal, dependency. An actor models an entity that has strategic goals and intentionality, such as a physical agent, a role or a position. A role is an abstract characterization of the behavior of an actor within some specialized context, while a position represents a set of roles, typically covered by one agent. Goals represent the strategic interests of actors. A dependency between two actors indicates that an actor depends on another in order to achieve a goal, execute a plan, or exploit a resource. *Tropos* distinguishes between hard and soft goals, the latter having no clear-cut definition and/or criteria as to whether they are satisfied. Softgoals are useful for modeling software qualities, such as security, performance and maintainability [1], as described also in [3]. Intentional analysis allow us to focus on why questions: What are the goals of the actors? Who share these goals? What are the divergent goals that lead to different perspectives? Why are particular behavioral or informational structures chosen? What alternatives are considered? What are the reasons for choosing one alternative over the others?

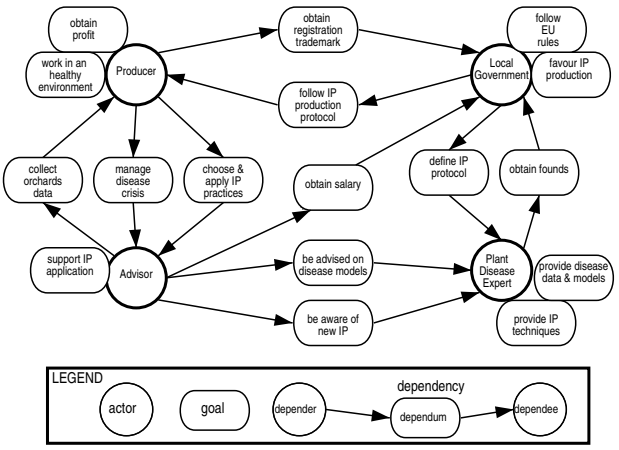

Figure 1: Early requirements model. A portion of the actor diagram modeling the IP organizational setting.

A *Tropos* model is represented as a set of diagrams: *actor diagrams* which describe the network of dependency relationships among actors, *goal diagrams*, which illustrate goal and plan analysis from the point of view of a specific actor.

An example of actor diagram is given in Figure 1, which depicts a fragment of the Early Requirement (ER) model of an application of decision-support systems in agriculture and in particular for Integrated Production (IP) [18] [2].

The ER model in *Tropos* describes the domain stakeholders, their goals, and the mutual dependencies. Stakeholders are modeled as actors.

In Figure 1, the actor Producer represents the apple grower who pursues objectives such as to obtain a profit following acceptable market strategies, and to work in a healthy environment. The actor Advisor models the technician of the advisory service that has been set up by the local government in order to provide a support to producers in choosing and applying the best agricultural practices and techniques (see the goal support IP application). The actor Local Government plays both an institutional and a practical role in promoting IP diffusion in our region (see the goals favor IP production, follow EU rules). The actor Plant Disease Expert represents the researcher in biological phenomena and in agronomical

[1]Softgoals can be further analysed in terms of technological constraints and/or architectural choises [3, 7]

[2]Integrated Production (IP) aims at a sustainable approach to agriculture production. In plant disease control, it promotes the use of low-impact techniques and chemicals, and the exploitation of natural defense mechanisms

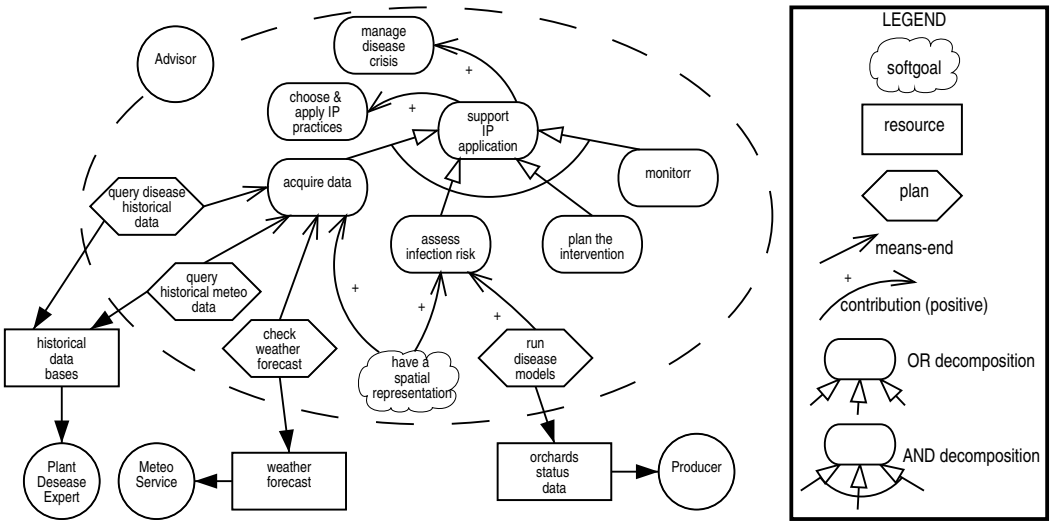

Figure 2: ER model. Goal diagram of the actor Advisor, showing an example of goal analysis.

techniques. Among his/her objectives that of transferring research results directly to the production level, for instance providing infection data and disease simulation models, as well as new effective pest management techniques (see the goals provide disease data & models, provide IP techniques). The actor Producer depends on the actor Local Government for obtaining a product certification (i.e. obtain registration trademark) that states that he/she follows IP practices, as required by specific market sectors. The local government sets up the yearly IP production protocol and issues the desired certification only to the producers that follows it. So, the actor Local Government depends on the actor Producer in order to have its goal follow IP production protocol satisfied. As already noticed, the actor Advisor plays the role of mentor, with respect to the producer, in carrying up apple production according to the IP rule. So the actors Advisor and Producer depend one upon the other: the actor Producer depends on the actor Advisor in order to choose & apply IP practices according to the production protocol and in order to manage disease crisis which may occur in case of unforeseen events and that requires to adopt an appropriate remedy action, still IP compliant. Viceversa, the actor Advisor depends on the actor Producer for satisfying his/her goal to collect orchards data in order to maintain an updated picture of the disease presence and evolution in the area under their control. Moreover, the Advisor depends on the actor Plant Disease Expert in order to use effective disease models (i.e. to attain the goal be advised on disease models and to get information on new IP techniques be aware of new IP).

Three basic types of analysis are exploited in *Tropos* for refining conceptual models: *(i) means-end analysis*, which consists in identifying goals, plans or resources that represent means for reaching an actor's goal (plan); *(ii) contribution analysis* which consists in discovering goals, plans or resources that can contribute positively or negatively towards the fulfillment of an actor's goal (or the execution of a plan); *(iii) AND/OR decomposition* which allows for a combination of AND and OR decompositions of an actor's root goal (plan) into sub-goals (sub-plans), thereby refining a goal (plan) structure.

An example of a refinement of the ER model is depicted in Figure 2 that shows goal analysis of the actor Advisor. Goal analysis allows to discover subgoals and dependencies that can be exploited in the second phase of the *Tropos* methodology, namely Late Requirements (LR) analysis, when the system-to-be is introduced in the domain representation.

3. TOWARDS SYSTEM ARCHITECTING AND DESIGNING

Approaching system design we face several questions, such as, how can we identify system components which ensure an appropriate level of cohesion? How can we reduce coupling among components? Is there any architectural styles that we can exploit in an effective way? This last issue has been deeply analyzed, in the context of *Tropos*, in [12, 15]. Here we focus on the first questions.

In our IP domain, the system-to-be is represented by the actor Advisor SW Agent which is first introduced in the LR model. The Advisor (i.e. the system's user) rests on it for the fulfillment of some goals and plans discovered during the goal analysis in the ER model depicted in Figure 2, such as goals related to the acquisition of data, to the use of a spatial representation of the territory and of the execution of disease models. Figure 3, shows the resulting LR goal diagram for the Advisor SW Agent.

According to domain knowledge, the goal acquire data can be decomposed in three plans: orchards pests history, historical meteo data, weather forecast, devoted to the acquisition of data from external resources (historical data bases and weather forecast). The acquisition of data positively contributes to the achievement of the plan run disease model and to the goal use GIS techniques (that represents a possible means to satisfy the softgoal have a spatial representation).

This goal analysis provides the basis for system architecting and designing. In fact, Architectural Design (AD) in *Tropos*

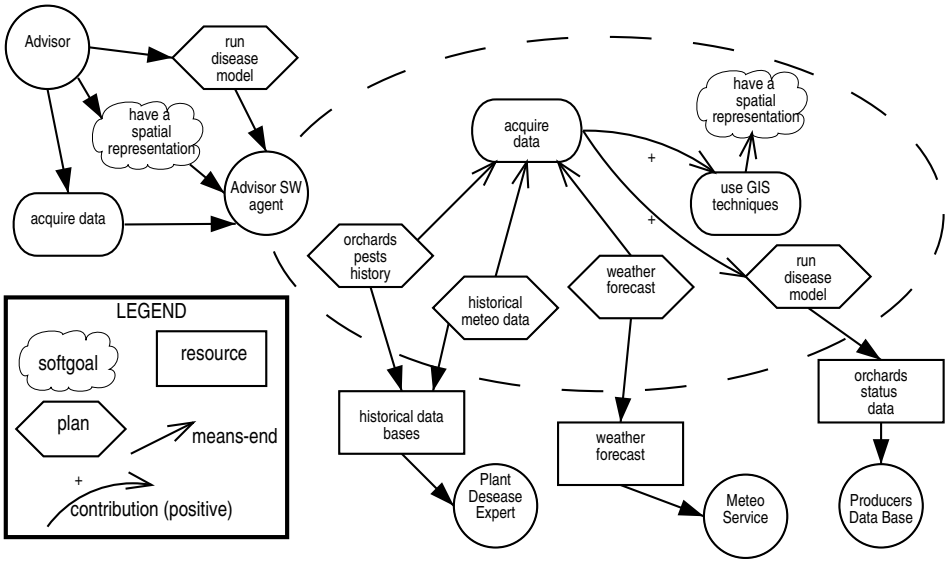

Figure 3: Late requirements model. A part of the Advisor SW Agent goal diagram.

aims at defining a macro-level view of the system architecture, in terms of components (modeled as sub-actors), and interfaces between components (specified in terms of dependencies), which results following to a top-down decomposition strategy.

In a sense, goal analysis applied to the system-to-be actor of the LR model provides a method for implementing a *divide-and-conquer* strategy in software architecting.

The resulting AD model, which is depicted in Figure 4, includes a set of sub-actors which will take care of goals and plans resulting from the goal analysis of the system's goals, that is, each main subgoal and plan identified in the analysis of the actor Advisor SW agent is assigned to a system sub-actor. In particular:

- the actor GISP (Geographic Information Services Provider) to which the Advisor SW agent delegates the goal use GIS techniques;
- the actor DBL (Disease Behavior Learner), which performs the plan run disease models on the basis of information extracted from the seasonal data on the disease;
- three wrapper actors, namely, the PDE-DBW (Plant Diseases Expert DB Wrapper) which takes care of retrieving meteo and orchard historical data; the wrapper of the database of the meteo service, called Meteo-DBW (Meteo Service DataBase Wrapper) which retrieves weather forecast; the Local Knowledge actor, which is the wrapper of the local data base containing data relative to the orchards belonging to the area under the advisor control (represented by the actor Producers DB in Figure 4);
- the actor User Interface which manages the interaction between the user of the application (represented by the actor Advisor in the ER model) and the other specialized system actors.

Relationships between subsystems are specified in terms of plan dependencies. For instance, the advisor involved in the study of a disease that needs to run the model describing the population dynamic, requires the actor User Interface for the execution of the plan visualize rules; as a consequence a new interaction between the User Interface and DBL is needed, devoted to the running of the disease model in order to obtain the set of rules induced by the meteo and orchards status data; this data can be retrieved by the DBL by means of the plan dependencies retrieve weather forecasts and retrieve disease history among DBL and the actors Meteo-DBW and Local Knowledge respectively.

The resulting architecture satisfies also principles of *cohesion*, in the sense that similar services have been grouped into a single actor. For instance, the actor User Interface collects all the user interface functionalities both for DBL actor and GISP actor. Moreover, coupling among components, here represented in terms of the binary dependencies between actors, is also minimized.

These properties can be highlighted in a component-view of this architecture which can be specified with a UML class diagram. An example is depicted in Figure 5 where only that part of the system, centered on the DBL actor, has been considered. Four main classes are specified. The Disease_Behavior_Learner class, that represent the component corresponding to the DBL actor, and its dependency relationships. This component is in charge of running the learning procedures. The data can be retrieved by means of the connection to the external and internal databases represented, respectively, by the two components Meteo_DBW and Local_Knowledge, which corresponds to the two wrappers actors of the AD model. Finally, the class User_Interface contains methods corresponding to services modeled in term

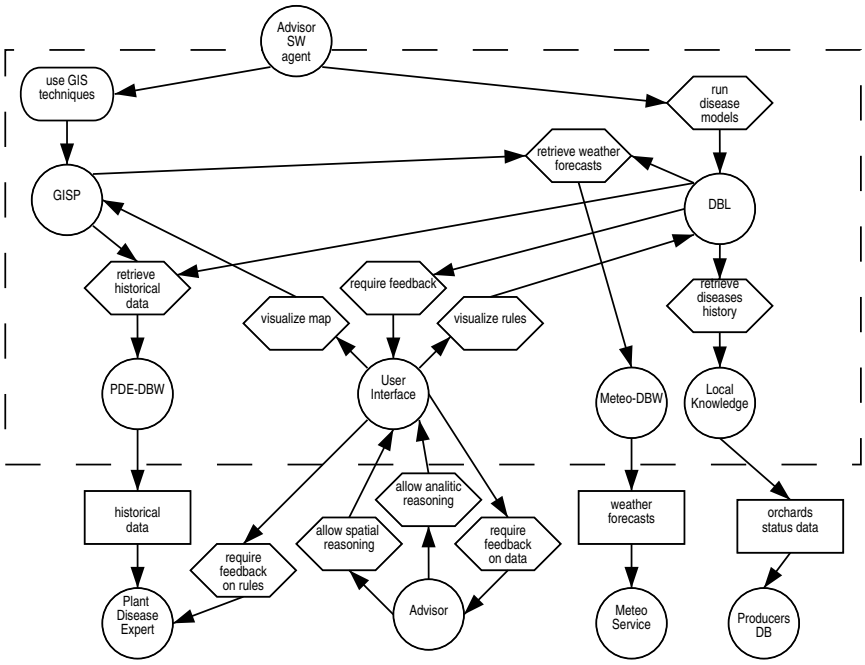

Figure 4: Architectural Design. The actor diagram refined upon system sub-goals delegation.

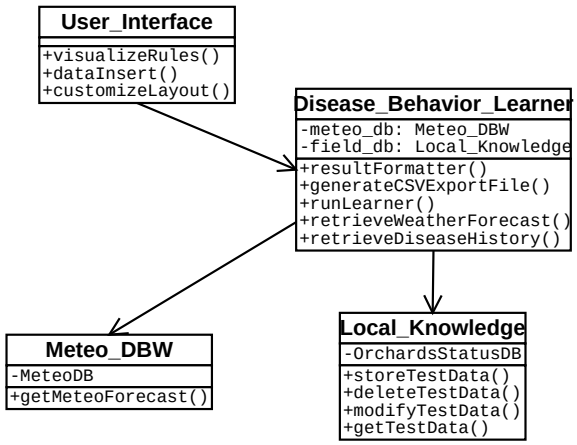

Figure 5: Architectural Design - UML class diagram of a portion of the system architecture.

of plan dependencies: the requests of data and feedback from DBL where processed by the methods visualizeRules, dataInsert, respectively.

At the detailed design level, this system architecture has been refined into a client-server architecture which has been implemented as a set of JavaScript and HTML pages (client side), while the server components have been implemented in Java and exploit a Tomcat servlet container and a PostGreSQL data base.

4. RELATED WORK

Several research lines are interesting for the work presented in the paper. Here we will mention two of them: first, approaches aiming at narrowing the semantic gap between a requirement specification and the software architecture to be produced from it; second, research pursuing the integration of agent-oriented and object-oriented methodologies, at different stages in the software development process.

Among the works pertaining to the first research line, the proposal of a design approach based on a family of pattern (architectural style), in the context of the *Tropos* effort [12, 15]. This work is rooted in organizational theory which provides business organization models. The authors propose to use these models as software architectural styles (specified in i* notation [23]) for MAS and evaluate them respect to software quality attributes.

An approach to component-based design that resembles several analogies with our work, is that presented in [14]. Here, design issues related to component identification and specification are put in relation with a parallel analysis of both business goals and processes, which provides with the identification of enterprise components. Component specification

is given in UML notation (class diagram).

In a sense, *Responsibility-driven* design [21] can be considered an approach which bears analogies with our, but adopting an object-oriented point of view. Instead of considering actors' goals, here, class responsibilities are analyzed, together with collaborations among classes, which are devoted to fulfill responsibilities.

Among the approaches aiming at integrating agent-oriented and object-oriented paradigms we shall mention the work by [7] where an application domain is analysed according to intentional analysis techniques. The resulting intentional model provides the basis from which a use-case model of system requirements is derived. A general framework for integrating scenario-based techniques for requirements engineering into object-oriented, model-based, processes, is given in the CREWS project [11]. Finally, [2] proposes an extension of UML for agent-oriented software development. Actor diagrams, analogous to those described in Section 3, are specified by UML class diagrams including stereotypes for actors and classes modeling resources, specified through resource dependencies in the original actor diagram.

5. CONCLUSION

This paper focused on software architecting and designing for distributed applications (component-based or MAS) adopting an agent-oriented approach.

In particular, we considered general strategies useful in the design process, such as divide-and-conquer, and principles for a good design, such as cohesion and decoupling. We discussed how to apply them upon guidance of goal analysis, performed on system-actor goals, according to the *Tropos* methodology. We used a case study in the domain of decision-support systems in agriculture. This DSS has been implemented on a web-based architecture and it is currently under evaluation by a group of users (a group of technicians of the local agriculture advisory service).

As recalled in Section 4 we share with other ongoing researches basic motivations of providing methodologies for enhancing the coherence when moving from requirement analysis to software design.

6. ACKNOWLEDGMENTS

Research partially funded by MIUR, FIRB Project ASTRO.

7. REFERENCES

[1] D. Bertolini, P. Busetta, A. Molani, M. Nori, and A. Perini. Designing Peer-to-Peer Applications: an Agent-Oriented Approach. In *Proceedings of International Workshop on Agent Technology and Software Engineering (AgeS) - Net Object Days 2002 (NODe02) Erfurt, Germany*, October 2002. To appear in R. Kowalszyk, J. Müller, H. Tianfield, R. Unland (editors) Agent Technologies, Infrastructures, Tools, and Applications for e-Services (LNAI 2592).

[2] J. Castro, M. Kolp, and J. Mylopoulos. A Requirements-Driven Development Methodology. In *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01), Interlaken, Switzerland*, June 2001.

[3] L. K. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Publishing, 2000.

[4] P. Ciancarini and M. Wooldridge, editors. *Agent-Oriented Software Engineering*, volume 1957 of *Lecture Notes in AI*. Springer-Verlag, March 2001.

[5] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1–2):3–50, 1993.

[6] T. DeMarco. The Paradox of Software Architecture and Design. In *Stevens Prize Lecture*. August 1999.

[7] P. Donzelli and M. R. Moulding. Application Domain Modelling for the Verification and Validation of Synthetic Environments: from Requirements Engineering to Conceptual Modelling. In *Proceedings of the 2000 Spring Simulation Interoperability Workshop*, Orlando, Florida, USA, 2000.

[8] P. Giorgini, A. Perini, J. Mylopoulos, F. Giunchiglia, and P. Bresciani. Agent-oriented software development: A case study. In S. Sen J.P. Müller, E. Andre and C. Frassen, editors, *Proceedings of the Thirteenth International Conference on Software Engineering - Knowledge Engineering (SEKE'01)*, Buenos Aires - Argentina, June 13 - 15 2001.

[9] F. Giunchiglia, J. Odell, and G. Weiß, editors. *Agent-Oriented Software Engineering III*. LNCS. Springer-Verlag, Bologna, Italy, Third International Workshop, AOSE2002 edition, July 2002.

[10] F. Giunchiglia, A. Perini, and F. Sannicolò. Knowledge level software engineering. In Springer Verlag, editor, *In Proceedings of ATAL 2001, Seattle*, December 2001. Also IRST Technical Report 0112-22, Istituto Trentino di Cultura, Trento,Italy.

[11] Matthias Jarke. CREWS: Towords Systematic Usage of Scenarios, Use Cases and Scenes. Technical report, Lehrstuhl Informatik V RWTH Aachen, 1999.

[12] Manuel Kolp and John Mylopoulos. Software Architecture as Organizational Structures. In *Proceedings of ASERC workshop*. Edmonton, Canada, August 2001.

[13] T. C. Lethbridge and R. Laganière. *Object-Oriented Software Engineering*. McGrow-Hill, 2001.

[14] Keith Levi and Ali Arsanjani. A goal driven approach to enterprise component identification and specification. *Communications of the ACM*, 45(10), October 2002.

[15] John Mylopoulos, Manuel Kolp, and Paolo Giorgini. Agent-Oriented Software Engineering. In *Proceedings of the 2nd Hellenic Conference on Artificial Intelligence (SETN-02)*. 2002.

[16] Andy Oram, editor. *Peer-to-Peer Harnessing the Power of Disruptive Technologies*. O'Reilly Associates, 2001.

[17] A. Perini, P. Bresciani, F. Giunchiglia, P. Giorgini, and J. Mylopoulos. A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. In *Proceedings of Agents 2001*, Montreal CA, May 2001. ACM.

[18] A. Perini and A. Susi. Developing a Decision Support System for Integrated Production in Agriculture. *Environmental Modelling and Software Journal*, 2003. to appear.

[19] A. Perini, A. Susi, and F. Giunchiglia. Coordination specification in multi-agent systems. from requirements to architecture with the Tropos methodology. In *Proceedings of 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02)*, Ischia, Italy, July 2002. ACM Press.

[20] G. Tremblay. Guide to the Software Engineering: Body of Knowledge. http://www.swebok.org/.

[21] Rebecca Wirfs-Brock and Alan McKean. *Object Design: Roles, Responsibilities, and Collaborations*. Pearson Education, 2002.

[22] M.J. Wooldridge, G. Weiß, and P. Ciancarini, editors. *Agent-Oriented Software Engineering II*. LNCS 2222. Springer-Verlag, Montreal, Canada, Second International Workshop, AOSE2001 edition, May 2001.

[23] E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, Department of Computer Science, University of Toronto, 1995.