UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

DESIGNINIG COORDINATION AMONG HUMAN AND SOFTWARE AGENTS

Anna Perini, Angelo Susi, and Fausto Giunchiglia

February 2002

Technical Report # DIT-02-0060

Designing Coordination among Human and Software Agents

Anna Perini[1] and **Angelo Susi**[1] and **Fausto Giunchiglia**[2]

Abstract. The goal of this paper is to propose a new methodology for designing coordination between human agents and software agents and, ultimately, among software agents. The methodology is based on two key ideas. The first is that coordination should be designed in steps, according to a precise software engineering methodology, and starting from the specification of early requirements. The second is that coordination should be modeled as dependency between actors. Two actors may depend on one another because they want to achieve goals, acquire resources or execute a plan. The methodology used is based on *Tropos*, an agent oriented software engineering methodology presented in earlier papers. The methodology is presented with the help of a case study.

1 Introduction

Agents, either human or software, are social entities that interact, by nature. Coordination among agents is considered a form of interaction devoted to goal attainment and to task completion. Moreover, coordination processes support contrasting agents behaviors, from cooperation to competition. Cooperating agents work together to achieve a common goal, trying to accomplish them as a team. Competitive agents work against each other, trying to optimize their own benefit, because their goals are conflicting. So, building effective Multi-Agent Systems (MAS), requires to carefully design and implement coordination processes. This motivates the large interest on this topic, as emerging from the last ten years MAS literature. Agents coordination has been studied from different perspectives, i.e. considering (software) agents coordination at run-time [8, 2], (software) agents coordination at the detailed design level [11, 12] (i.e. designing the *micro* level) and agents coordination at the group analysis level [3, 12] (i.e. designing the *macro* level). At the macro level, both human and software agents can be considered.

We think that, in order to build effective MAS that operate into human communities, interacting both with software and human agents, we first need to model coordination processes taking place into the social organizational setting where the MAS has to be introduced. Then, we have to analyze how these coordination processes will be affected by introducing a MAS (analogously to what is done during the *macro* level analysis for heterogeneous systems). Only in the following steps we keep designing coordination processes among software agents and detailing interaction and communication mechanisms which support the required coordination processes. This multi-steps process allow us to keep trace of the *why* (i.e. the needs) of the coordination processes modeled at the *micro* level.

Coordination specifications should rest on a deep analysis of the agent intentional dependencies which can be modeled in terms of goal, plan or resource dependencies between pair of agents.

In our approach we adopt the *Tropos* methodology which offers concepts and techniques at support of this idea. The *Tropos* methodology [17, 5] is an agent oriented software development methodology that can be characterized by the following features, namely:

- The notion of agent, goal, plan and various other knowledge level notions are used in all phases of software development, defining a *knowledge level* [15] approach to requirement specification and design activities.
- A crucial role is given to the very early phase of requirements specification when the environment and the system-to-be are analyzed, according to a *requirement driven* approach [14].

The methodology rests on the idea of building a conceptual model that is incrementally refined and extended from an early requirement model, in which the organizational setting where the MAS will be introduced is analyzed, to executable artifacts, along five main development phases, called respectively: *early requirement analysis*, *late requirements analysis*, *architectural design*, *detailed design*, *implementation*.

The paper is structured as follows. Section 2 describes the Tropos methodology stressing how the specification of coordination processes is carried out. Sections 3.1 and 3.2 describe the results of modeling actor coordination during early and late requirements analysis in Tropos, as applied in the context of a real application devoted to provide a decision support to the technicians of the agricultural advisory service when managing plant diseases. The related work are considered in Section 4. Finally, conclusions and the future work are presented in Section 5.

2 Coordination is Dependency among actors

Conceptual models in Tropos are built as instances of intentional and social concepts (such as actor, goal and dependency) and of the relationships among them, which have been defined in the Tropos metamodel [10] specified by a set of UML[3] class diagram. A portion of the metamodel is depicted in Figure 1.

The notion of actor represents an entity that has strategic goals and intentionality. An actor represents a *physical agent* (e.g., a person, an animal, a car), or a *software agent* [16] as well as a *role*, i.e. an abstract characterization of the behavior of an actor within some specialized context [22].

[1] ITC-Irst, Via Sommarive 18, I-38050, Povo, Trento, Italy. email: {perini,susi}@irst.itc.it

[2] Department of Information and Communication Technology, University of Trento, Via Sommarive 14, I-38050, Povo, Trento, Italy. email: fausto@cs.unitn.it

[3] Unified Modeling Language [1]

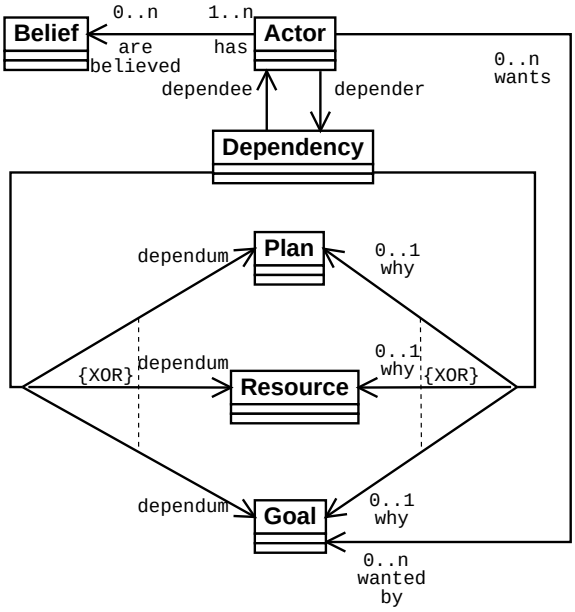

Figure 1. Portion of the Tropos metamodel concerning the concept of Actor, specified in UML class diagrams.

In Figure 1, Actor is represented as a UML class. An actor can have $0 \ldots n$ goals and a goal is wanted by $0 \ldots n$ actors, as specified by the UML association relationship between the class Actor and the class Goal. Moreover, an actor can have $0 \ldots n$ beliefs and, conversely, beliefs are believed by $1 \ldots n$ actors. A dependency between two actors indicates that one actor needs to coordinate itself with another actor in order to attain some goal, execute some plan, exploit a resource. The former actor is called the depender, while the latter is called the dependee (specified in the Tropos metamodel by the two association relationships between the class Dependency and the class Actor). The object around which the dependency centers is called dependum and is an instance of one among the following classes: Goal, Plan, Resource. A few remarks about actor dependency are worth to be mentioned:

- Through goal dependency a goal is delegated by the depender actor to the dependee who will decide autonomously how to satisfy the goal. As a consequence, the depender becomes vulnerable respect to the dependee for the goal satisfaction.
- Plan dependency specifies that the depender requests the dependee to execute a specific plan. The execution control is led to the dependee, the depender is still vulnerable because he/she rests on the plan outcomes for reaching (avoiding) desirable (undesirable) states of affairs.
- Resource dependency model the request of the depender to the dependee of a specific entity (resource). The way this resource should be delivered by the dependee is not specified. (It can eventually be specified by additional softgoal dependencies).

The reason for a given dependency (labelled why in Figure 1) can be specified, in terms of goal, plan or resource. The reason for a given dependency between two actors comes out from goal and plan analyses, performed from the perspective of a specific actor by using three basic reasoning techniques: *means-end analysis*, *contribution analysis*, and *AND/OR decomposition*. For goals, means-end analysis proceeds by refining a goal into subgoals in order to identify plans, resources and softgoals that provide means for achieving the goal (the end). Contribution analysis allows the designer to point out goals that can contribute positively or negatively in reaching the goal being analyzed. In a sense, contribution analysis can be considered as a special case of means-end analysis, where means are always goals. AND/OR decomposition allows for a combination of AND and OR decompositions of a root goal into sub-goals, thereby refining a goal structure.

The conceptual modeling and the analysis above described are used in the five phases of the software development process which have been identified in Tropos, as described in the following.

Early Requirements analysis focuses on the understanding of a problem domain by studying an *existing organizational setting* and *the coordination processes* that characterize the behavior of its elements. Social actors and software systems that are already present in the domain are modeled with their individual goals.

It is worth to be noticed that in Tropos, system goals are not modeled explicitly, as for instance in [7, 12], but emerges from the coordination of actors pursuing individual goals. In particular, a social actor can depend on another actor in order to attain one of its individual goal, for having a resource or resting on the other actor for the execution of a plan.

Late Requirement analysis focuses on the system-to-be which is introduced as a new actor into the model. The system commits itself to taking care of specific goals of the social actors. New goal dependencies between social actors and the system-to-be actor are designed and existing dependencies between the social actors can be modified.

These dependencies define *requirements of coordination* involving (some of) the human actors (the users) and the system. They will be further refined and specified in the following design phases.

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.

Dependencies between subactors describe the coordination processes, between the system components. Moreover, the user-system dependencies specified during late requirements analysis can be further refined identifying the system subactors which will play the role of dependee, for each specific dependency.

The result of the architectural design is the mapping of the system subactors to a set of agents. Each agent is characterized by a set of social capabilities providing the coordination mechanisms required by the coordination processes specified as actor dependencies.

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. So, for instance, considering a BDI [19] platform, each agent capability is defined in terms of beliefs, plans and events and each plan in terms of atomic actions. *Interaction* and *communication protocols* required by each coordination process involving the agent is also designed at this time.

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. Now run-time coordination processes can be tested against the design objectives.

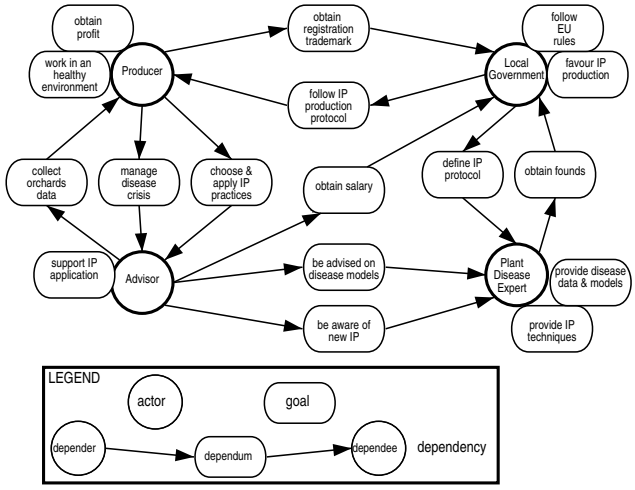

Figure 2. The actor diagram showing a portion of the IP organizational setting model. Early requirements model.

3 An Example

The example considered in this paper has been extracted from a technology transfer project aimed at developing a decision support system for technicians of the agricultural advisory service which operates in our region. The role of the technicians is that of favoring the application of Integrated Production (IP) practices, which are characterized by a reduced environmental impact, by the local apple producers.

In the rest of the paper we focus on the early requirements model of the IP domain and on the late requirements model.

3.1 Early Requirements

Early Requirements focuses on the coordination processes between domain stakeholders; they are modeled as a set of dependencies between pairs of actors.

The analysis starts identifying the stakeholders of the agriculture production system of our region and modeling them as actors, depicted by circles in Figure 2:

- 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 advisor plays a key role in our area since the majority of producers are not professional farmers, they lack specific skills and/or are not confident enough of adopting an IP approach.
- 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). It sets up a list of admissible chemicals and quantity limits, according to the European Union agreements. These rules are yearly updated and coded into a production protocol.
- The actor Plant Disease Expert represents the researcher in biological phenomena and in agronomical techniques. Among his/her objectives that of transferring research results directly to the production level, for instance providing disease data and models and new effective pest management techniques (see the goals provide disease data & models, provide IP techniques).

The actor diagram in Figure 2 shows some of the critical coordination processes between the domain stakeholders which, at a macroscopic level, result in a joint effort to disseminate IP. Coordination is modeled in terms of goal dependencies between the actors.

In particular, 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 closely coordinates: 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). Both actors, the Advisor and the Plant Disease Expert are funded by Local Government). The goal dependency define the IP protocol between the Local Government and the Plant Disease Expert closes the loop. It models the contribution of the expert in providing the technical skills necessary for defining a production protocol that follows the European Union strategic directives.

The Early Requirements model is further refined by considering each actor and by analyzing its goals. New actors and dependences can be added in the model. The goal diagram depicted in Figure 3 shows the analysis of the goal support IP application, from the point of view of the actor Advisor, with reference to the management of a specific disease, a fungus called Venturia inæqualis.

The goal support IP application contributes positively to the fulfillment of both goals choose & apply IP practices for which the actor Producer needs to coordinate with the actor Advisor. The goal can be AND decomposed into a set of more specific subgoals, i.e. acquire data, assess infection risk, plan the intervention and monitor the situation after the intervention. In the following we consider the plans that the advisor performs in order to satisfy them in a specific case, that is when dealing with the disease caused by Venturia inæqualis. Considering the goal acquire data the following plans are means to satisfy it getting data that are relevant in the case of Venturia inæqualis, i.e. historical data on the presence of the disease in the area, historical meteo data and weather forecast (see the plans, depicted as hexagonal shapes in Figure 3, query disease historical data, query historical data and check weather forecast). The analysis points out a set of coordination processes related to the execution of these plans, they are modeled in terms of resource dependencies. For instance, the dependency between the actor Advisor and the actor Plant Disease Expert for the resource disease historical

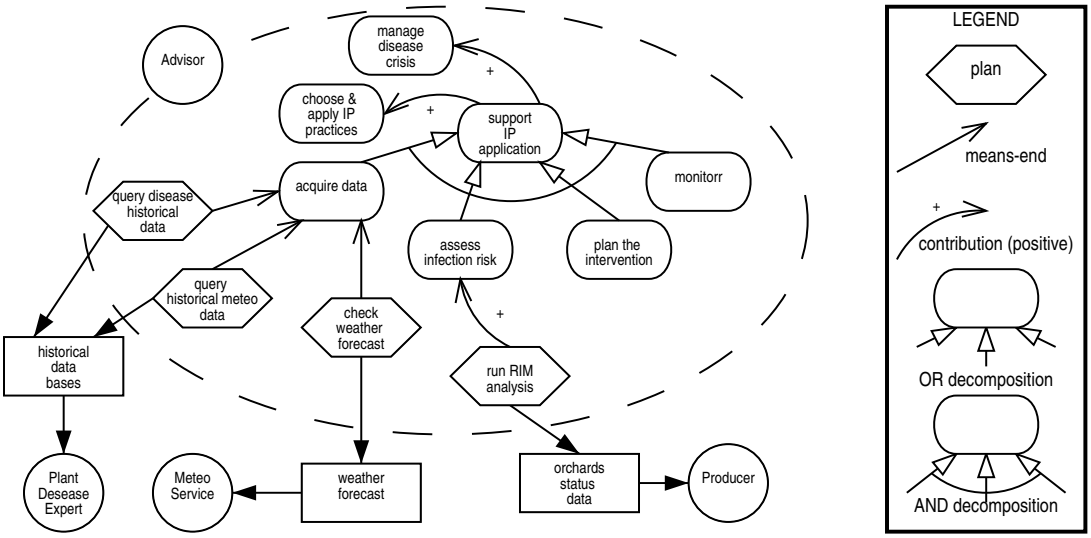

Figure 3. The goal diagram of the goal support IP application analyzed from the point of view of the actor Advisor.

data models the fact that the advisors usually perform searches into the data bases on disease data held by the experts. Analogously, historical meteo data and weather forecast are data to be obtained from the actors that institutionally held them. The plan run RIM model is a means to attain the goal assess infection risk. The RIM model is a mathematical model for estimating both the disease stage and the infection extent. It requires specific data from the orchard in order to produce updated estimates. Analogously, the remaining subgoals can be analyzed with the aim of identifying advisors plans and coordination processes with the other actors that allow for the execution of these plans.

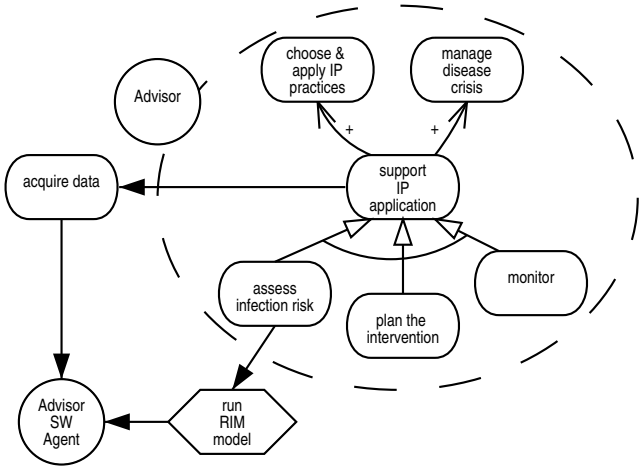

Figure 4. The advisor goal diagram for the venturia inæqualis management. Late requirements model.

3.2 Late Requirements

Late Requirements analysis focuses on the system-to-be actor and on the coordination processes between it and the human actors.

During late requirements analysis the system-to-be, that is the decision support system at use of the advisors when dealing plant disease management, is introduced as a new actor into the conceptual model. Figure 4 [4] depicts a fragment of the late requirements model where the actor Advisor SW Agent models the system-to-be. In particular, the actor Advisor delegates the system-to-be for the fulfillment of the goal acquire data and the execution of the plan run RIM model. This implies that also the dependencies to the other social actors related with these model elements have to be appropriately revised. For instance all the coordination processes with actors holding data relevant for disease management have been delegated to the system-to-be actor.

The goal diagrams of the actor Advisor is modified accordingly (Figure 4) and it should be compared with the analogous diagram of the early requirement model, depicted in Figure 3. Figure 5 shows the resulting goal diagram for the actor Advisor SW Agent. Note that the plans that the actor executes in order to fulfill the goal acquire data have to be redefined from the point of view of the system actor.

4 Related work

As already mentioned, agents coordination has been largely recognized as a topical issue in both MAS and Distributed Artificial Intelligence [21]. As a consequence, several interesting approaches for studying this topic, from different perspectives, can be found in the literature on MAS. We first consider work that focused mainly on studying software agents coordination at run time, then work on designing agents coordination at the *micro* and *macro* level (according to the definition recalled in [21]).

[4] Note that for these dependencies also the why argument, in the 4-argument dependency relationship, has been explicitly modeled. So, for instance, the depender is the actor Advisor, the dependee is the actor Advisor SW Agent, the dependum is the goal acquire data and the reason for this dependency is the goal support IP management.

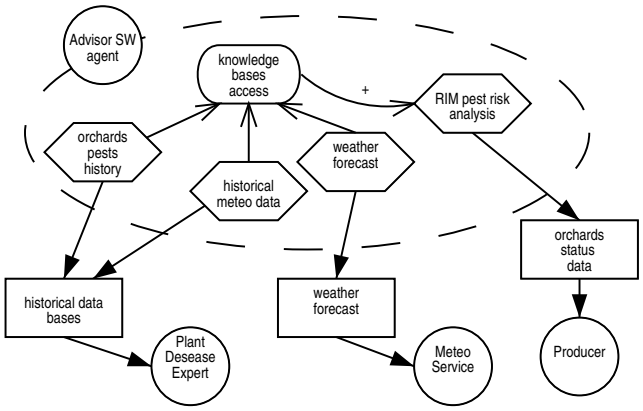

Figure 5. The Advisor SW Agent goal diagram for the venturia inæqualis management. Late requirements model.

Dealing with agents coordination at run-time consists, basically, in setting up interaction and communication protocols that effectively support coordination processes. For instance, one of the most widely used interaction protocols for cooperative problem solving in practical MAS application is *contract net* [20], as discussed in [11]. At a more theoretical level, the coordination problem at run-time has been faced using dynamic programming strategies, see for instance [8] or using multi-agent Markov Decision Processes, see for instance [2, 18].

A relevant approach to the design of agents coordination processes is [12], where commitments and conventions mechanisms are used to specify coordination processes. Goal analysis techniques similar to those used in *Tropos* are also proposed, the main difference resting on the fact that in [12] global MAS goals are considered, while for us, coordination processes emerges from the actors intentions to pursue their own goals. Analyses of agents coordination based on an explicit model of the dependencies among agents actions have been proposed in [3, 4].

Finally, work in Computer Supported Cooperative Work [9], provides useful ideas for dealing with the problem of designing MAS coordination, especially when heterogeneous agents, human and software, are considered. In particular, the work of Malone [13] is worth to be mentioned. Malone considers coordination as a phenomenon that occurs in different kinds of systems (e.g. human, computational, biological) and this allows to set up a framework for studying coordination which exploits analysis techniques provided by different disciplines, such as economics, computer science organizational theory. A definition of coordination, as the process of *managing dependencies between activities* has been defined and a research agenda on this topic is proposed.

5 Conclusion and Future Work

This paper describes a new methodology for designing coordination between human agents and software agents based on *Tropos*, an agent oriented software engineering methodology. The approach rests on the basic idea that coordination can be modeled as dependencies between actors.

Coordination modeling has been described in details, with reference to a real application for the agriculture domain which is currently being developed in our group. In particular, we have presented the early requirements model that concerns the understanding of the organizational setting (the environment), and late requirements model which focuses on the system-to-be and its relationships with the environment. The architectural design and the detailed design are not presented, due to lack of space.

Our long term objective is to provide a complete and detailed account of the methodology. We are also considering how to combine our methodology, which covers early and late requirements analysis, with others, for instance those discussed in Section 4, suitable for detailed design.

ACKNOWLEDGEMENTS

The work presented in the paper is partially funded by the Italian Ministry of Scientific and Technological Research.

REFERENCES

[1] G. Booch, J. Rambaugh, and J. Jacobson, *The Unified Modeling Language User Guide*, The Addison-Wesley Object Technology Series, Addison-Wesley, 1999.
[2] C. Boutilier, 'Sequential optimality and coordination in multiagent systems', in *IJCAI*, pp. 478–485, (1999).
[3] C. Castelfranchi, 'Modeling Social Action for AI Agents', in *IJCAI*, pp. 1567–1576, (1997).
[4] C. Castelfranchi, M. Miceli, and A. Cesta, 'Dependence relations among autonomous agents', in *Decentralized AI 3 – Proceedings of the Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-91)*, eds., E. Werner and Y. Demazeau, pp. 215–231. Elsevier Science B.V.: Amsterdam, Netherland, (1992).
[5] J. Castro, M. Kolp, and J. Mylopoulos, 'A requirements-driven development methodology', in *Proceedings Thirteen International Conference on Advanced Information Systems Engineering CAiSE 01*, Stafford UK, (June 2001).
[6] *Agent-Oriented Software Engineering*, eds., P. Ciancarini and M. Wooldridge, volume 1957 of *Lecture Notes in AI*, Springer-Verlag, March 2001.
[7] A. Dardenne, A. van Lamsweerde, and S. Fickas, 'Goal-directed requirements acquisition', *Science of Computer Programming*, **20**(1–2), 3–50, (1993).
[8] E. H. Durfee, 'Practically Coordinating', *AI Magazine*, **20**(1), (1999).
[9] C. Ellis and J. Wainer, *Groupware and Computer Supported Cooperative Work*, chapter 10. In Weiss [21], 1999.
[10] F. Giunchiglia, J. Mylopoulos, and A. Perini, 'The Tropos Software Development Methodology: Processes, Models and Diagrams', Technical Report 0111-20, ITC-irst, (2001).
[11] M. N. Huhns and L. M. Stephens, *Multiagent systems and Societies of agents*, chapter 2. In Weiss [21], 1999.
[12] N. R. Jennings, 'Commitments and conventions: The foundation of coordination in multi-agent systems', *The Knowledge Engineering Review*, **8**(3), 223–250, (1993).
[13] Thomas W. Malone and Kevin Crowston, 'The Interdisciplinary Study of Coordination', *ACM Computing Surveys*, **26**(1), 87–119, (1994).
[14] J. Mylopoulos and J. Castro, 'Tropos: A framework for requirements-driven software development', in *Information System Engineering: State of the Art and Research Themes*, eds., J. Brinkkemper and A. Solvberg, Lecture Notes in Computer Science, Springer-Verlag, (2000).
[15] A. Newell, 'The knowledge level', *Artificial Intelligence*, **18**, 87–127, (1982).
[16] H. Nwana, 'Software agents: An overview', *Knowledge Engineering Review Journal*, **11**(3), (November 1996).
[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 the Fifth International Conference on Autonomous Agents*, Montreal CA, (May 2001). ACM.
[18] N. R. Jennings R. A. Bourne, C. Excelente-Toledo, 'Run-time selection of coordination mechanisms in multi-agent systems', in *Proc. of the 14th European Conf. on Artificial Intelligence (ECAI'2000)*, pp. 348–352. IOS Press, (2000).

[19] A.S. Rao and M.P. Georgeff, 'Modelling rational agents within a BDI-architecture', in *Proceedings of Knowledge Representation and Reasoning (KRR-91) Conference*, San Mateo CA, (1991).
[20] R. G. Smith, 'The contract net protocol: High-level communication and control in a distributed problem solver', in *IEEE Transactions on Computers*, volume C-29, pp. 1104–1113, (1980).
[21] *Multiagent System: a modern approach to Distributed AI*, ed., G. Weiss, MIT Press, 1999.
[22] E. Yu, 'Agent-oriented modeling: Software versus the world', In Ciancarini and Wooldridge [6].