UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

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

COORDINATION SPECICATION IN MULTI-AGENT SYSTEMS. FROM REQUIREMENTS TO ARCHITECTURE WITH THE TROPOS METHODOLOGY

Anna Perini, Angelo Susi, and Fausto Giunchiglia

2002

Technical Report # DIT-02-0014

Also in: *Proceedings of* SEKE '02

Coordination specification in Multi-Agent Systems. From requirements to architecture with the Tropos methodology

Anna Perini
ITC-Irst
Via Sommarive, 18
I-38050 Trento-Povo, Italy
perini@irst.itc.it

Angelo Susi
ITC-Irst
Via Sommarive, 18
I-38050 Trento-Povo, Italy
susi@irst.itc.it

Fausto Giunchiglia
Department of Information and Communication Technology
University of Trento
via Sommarive, 14
I-38050 Trento-Povo, Italy
fausto@dit.unitn.it

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.

Building effective Multi-Agent Systems (MASs), 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 [9, 3], (software) agents coordination at the detailed design level [15, 16] (i.e. designing the *micro* level) and agents coordination at the social level [5, 16] (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 a 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 [22, 18] 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* [19] 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 [18].

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 develop a decision support for the technicians of the agricultural advisory service when managing plant diseases. Sections 3.3 describes the initial steps in the architectural design. The related work are considered in Section 4. Finally, conclusions and the future work are presented in Section 5.

2. THE METHODOLOGY

The core process of the *Tropos* methodology consists in performing conceptual modeling using a visual language that provides: intentional and social concepts such as actor, goal and dependency; a graphical notation which allows to visualize a model; a set of diagrams that allow to build views on different properties of the model.

The language syntax is specified by a set of UML [1] class diagrams. We describe here the specification for two basic Tropos notions, namely, actor and goal. Figures 1 and 2 depict the relative UML class diagrams. A more extended description of the language syntax specification is given in [13].

The UML class Actor, in Figure 1, models the concept of actor in Tropos. An actor represents an entity that can have strategic goals and intentionality, as well as an entity with bounded rationality [28]. An actor represents a *physical agent*, a *software agent*,a *role*, i.e. an abstract characterization of the behavior of an actor within some specialized context, as well as a *position*, i.e. a set of roles.

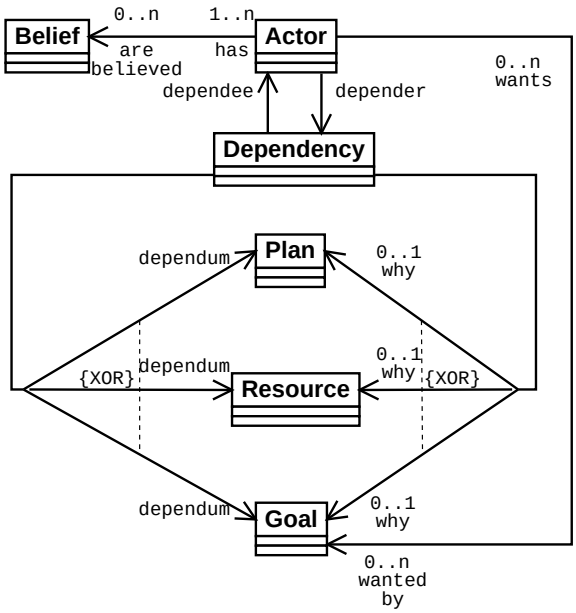

Figure 1: Portion of the UML class diagram specifying the concept of Actor in Tropos.

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. 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 UML class diagram 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 delegated 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.

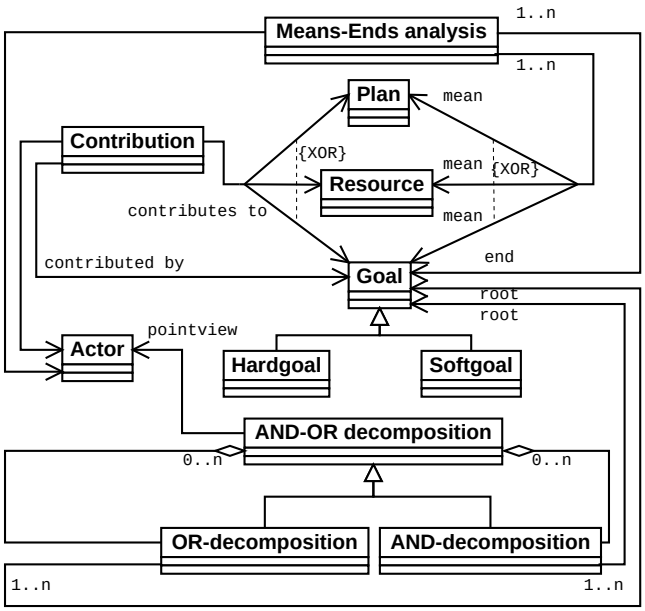

Figure 2: The Goal concept in Tropos.

The UML class Goal, in Figure 2, represents the concept of goal in Tropos. In particular, we distinguish *Hardgoal* from *Softgoal*, specified as two specializations of Goal. The soft goal notion is used to model an actor's objective lacking a clear-cut definition or a criterion for deciding as to whether it is satisfied or not. Soft goals typically represent actor intentions on the way a goal should be achieved, that is on

quality requirements that can lead to the specification of non-functional requirements.

Both soft- and hard- goals can be analyzed, from the point of view of an actor, performing three basic types of analysis, namely, *means-end analysis*, *contribution analysis* and *AND/OR decomposition* (listed in order of strengthness). Let us consider these in turn.

- *Means-end Analysis* is a ternary relationship defined among an *actor*, whose point of view is represented in the analysis, a goal (the end), and a *Plan*, *Resource* or *Goal* (the means). Means-end analysis consists in identifying goals, plans or resources that represent means for reaching a goal.
- *Contribution Analysis* is a ternary relationship between an *actor*, whose point of view is represented, and two goals. Contribution analysis consists in discovering goals, plans or resources that can contribute positively or negatively towards the fulfillment of a goal (see association relationship labelled *contributes to* in Figure 2). A contribution can be annotated with a qualitative metric, as used in [7], denoted by $+, ++, -, --$. In particular, if the goal *g1* contributes positively to the goal *g2*, with metric $++$ then if *g1* is satisfied, so is *g2*.

  Contribution analysis applied to softgoals is often used to evaluate non-functional (quality) requirements.
- *AND/OR Decomposition* is also a ternary relationship which defines an *AND-* or *OR-decomposition* of a root goal into subgoals. AND/OR decomposition allows for a combination of AND and OR decompositions of a root goal into sub-goals, thereby refining a goal structure.

Note that these three techniques are used also when performing plan analysis from the point of view of an actor. The conceptual modeling and the analysis above described can be visualized using two types of diagrams, called respectively *actor* and *goal* diagrams. Examples are given in Section 3. In actor diagrams, actors are depicted as circles, their goals as ovals and their softgoal as cloud shapes. A dependency relationship between two actors is depicted as two arrowed lines connected by a graphical symbol varying according to the dependum: a goal, a plan or a resource. A goal diagram depicts goal and plan analyses performed from the perspective of a specific actor in a balloon that contains graphs whose nodes are goals (ovals) and /or plans (hexagonal shape) and whose arcs represents the different types of relationships that can be identified between its nodes.

Conceptual modeling drives the software development process in Tropos along five main phases, briefly 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 [8, 16], but they emerge 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 requirements 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.

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

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 [23] 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 are also designed at this time.

Implementation activities produce 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.

3. AN EXAMPLE

The example considered in this paper has been extracted from a technology transfer project aimed at developing a Multi-Agent System (MAS) devoted to support decision making by technicians of the agricultural advisory service which operates in our region [21]). The role of the advisors is that of favoring the application of Integrated Production (IP) practices, which are characterized by a low environmental impact, by the local apple producers.

In the rest of the section we shall focus on the early requirement model of the IP domain and on the late requirements model, where the MAS will be introduced as a single

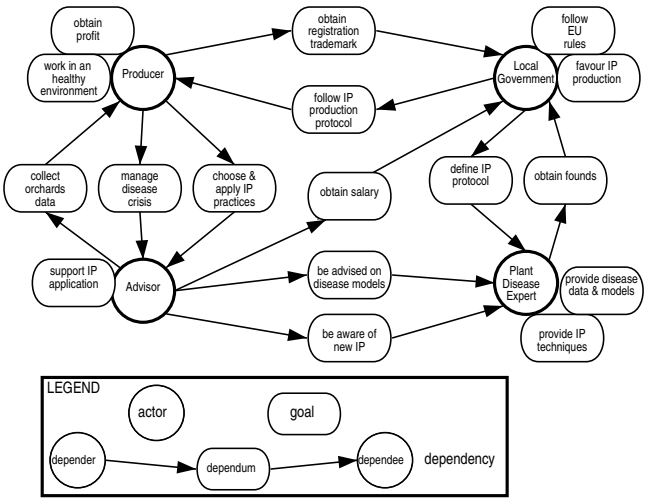

Figure 3: Early requirements model. A portion of the actor diagram modeling the IP organizational setting.

actor. A sketch of the architectural design will be then presented.

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 3:

- 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 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 diagram in Figure 3 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 all its actors and by analyzing their goals. New actors and dependences can be added in the model. The goal diagram depicted in Figure 4 shows the analysis of the goal support IP application, from the point of view of the

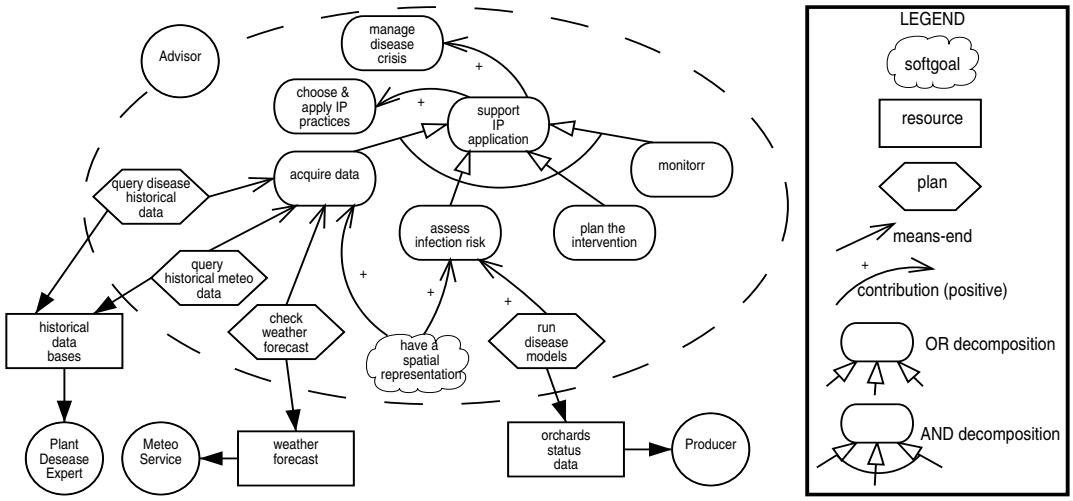

Figure 4: Early requirements model. The goal diagram of the goal support IP application analyzed from the point of view of the actor Advisor.

actor Advisor.

The goal support IP application contributes positively to the fulfillment of both goals choose & apply IP practices and manage disease crisis 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.

Moreover, the softgoal have a spatial representation that is being able to visualize the data on a map of the whole area under control by the advisor shall allow him/her to perform in a more effective way both the data acquisition activity and the assessment of an infection risk (see the two positive contribution links in Figure 4.

In the following we consider the plans that the advisor performs in order to satisfy them according to current practices. Means to satisfy the goal acquire data consists in getting data resulting from observation and measurements activities performed, each season, in the orchards, as well as in getting current meteo data.

This is modeled in Figure 4 with a set of plans, depicted as hexagonal shapes, which are related to the goal acquire data trough specific means-end relationships, i.e. query disease historical data, which refers to historical data on the presence of the disease in the area, query historical meteo data which refers to historical climate and check weather forecast (the current meteo data).

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 historical data bases models the fact that the advisors usually perform searches into the data bases on disease data held by the experts, as well as on climate data relative to the area under their control.

The plan run disease models is a means to attain the goal assess infection risk. In IP practices currently used, the advisors exploit phenology and/or epidemiological models which help them in analyzing the behavior of a plant disease. For instance, they allow them to estimate both the disease stage and the infection extent[1].

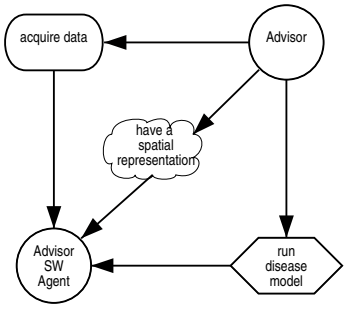

Figure 5: Late requirements model. Portion of the actor diagram upon the introduction of the system-to-be.

These model require 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.

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 5 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

[1]The models are developed by experts (entomologists and/or biologists) using mathematical and/or artificial intelligence techniques [25]. Here, we refer to those plant diseases for which effective models exist.

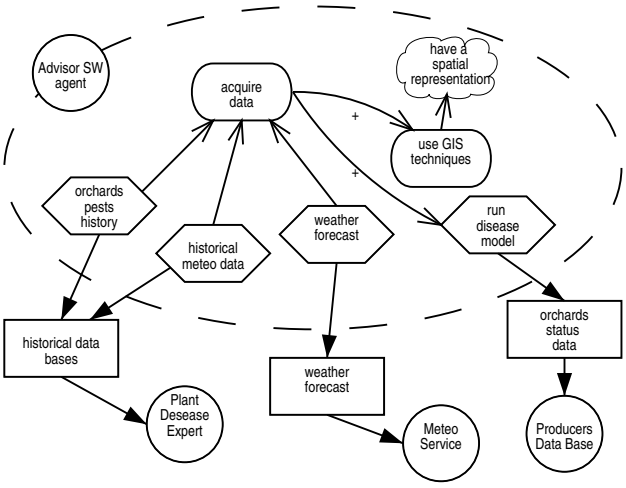

Figure 6: Late requirements model. The Advisor SW Agent goal diagram.

goal acquire data and of the softgoal have a spatial representation, and for the execution of the plan run disease models. 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.

Figure 6 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,i.e., appropriate interaction procedures have to be defined between the system actor and the social actors who held the specific data. In the goal diagram a new goal use GIS techniques has been introduced as a mean to satisfy the softgoal have a spatial representation.

3.3 Architectural Design

> *Architectural design defines the system's global architecture in terms of subsystems represented as actors. Dependencies between subactors describe the coordination processes, between the system components.*

This phase consists of three steps, namely, refining the system actor diagram taking into account goal and plan decomposition, as well as exploiting useful design patterns *(i)*, identifying actors capabilities *(ii)* and assigning them to agents *(iii)*. We will focus on the first two steps.

The system actor diagram is refined by including new actors which will take care of subgoals which have been discovered upon goal analysis of the system's goals in the late requirement phase.

Figure 7 depicts the refined actor diagram. Six sub-system actors have been introduced. The actor Advisor SW agent delegates them the sub-goals and the plans that were found during the goal analysis performed from the point of view of the system actor (see Figure 6). They are:

- 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 Experts 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 7);
- the actor User Interface which manages the interaction between the user of the application (the actor Advisor) and the other specialized sub-actors of the Advisor SW agent.

The actor diagram shows some of the subsystems coordination processes specified in terms of plan dependencies. For instance, the user that wants to do spatial reasoning, such as analyzing the distribution on a geographical area of a given pest in a certain period of time, requires the actor User Interface for the execution of the plan allow spatial reasoning (see the correspondent plan dependency between the two actors). As a consequence, a new interaction between the User Interface and the actor GISP is needed, devoted to the definition of an appropriate electronic map to be visualized by the user interface (see the plan dependency visualize map between the actor User Interface and the actor GISP).

The system architecture model can be further enriched with other system actors resulting from the inclusion of design patterns [14, 4] that provide solutions to heterogeneous agents communication. We omit here further details due to lack of space. How to perform in Tropos this type of architectural model refinement , as well as analogous refinements that can result from the analysis of non-functional requirements, is described in [12].

Further steps are required in *Tropos* to complete the architectural design, such as that aimed at identifying actor capabilities from the analysis of the dependencies going-in and -out from the actor and from the goals and plans that the system actors will carry on.

For instance, focusing on the system actor User Interface in Figure 7, and in particular on its ongoing and outgoing dependencies we can identify the following capabilities: ask for map visualization, provide rule feedback, ask for rules visualization, ask for rule feedback, support analytic reasoning, support spatial reasoning. Table 1 lists the capabilities asso-

Actor	Capability
UI	ask for map visualization
	provide rule feedback
	ask for rules visualization
	ask for rule feedback
	support analytic reasoning
	support spatial reasoning

Table 1: Architectural design - step 2. Actors capabilities.

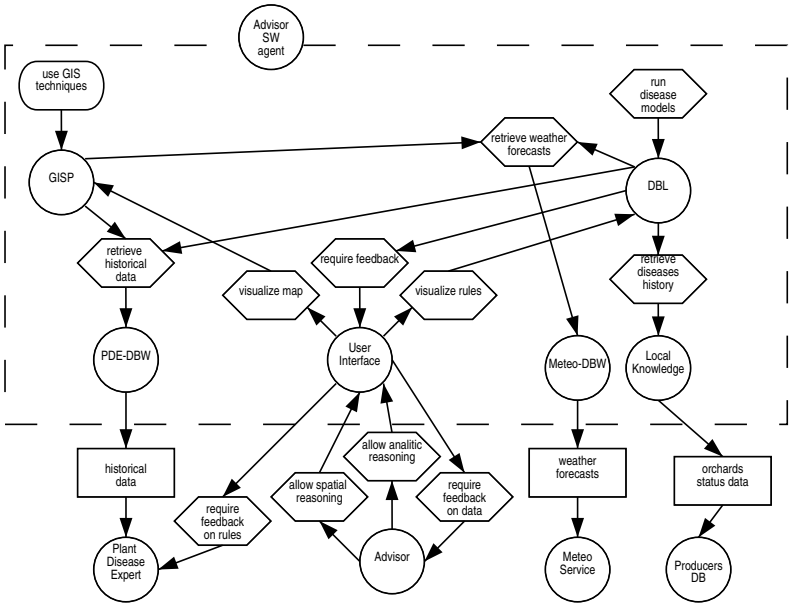

Figure 7: Architectural Design - step1. The actor diagram refined upon system sub-goals delegation.

ciated to the actor User Interface and should be completed considering all the system actors included in the architectural design. A given capability may be needed by different actors.

The architectural design ends with a mapping of the system subactors to a set of agents. Each agent is characterized by a set of the capabilities identifyied in the actor diagram.

The next phase in the Tropos development process is detailed design, where the agent micro-level is specified in terms of agent capabilities and plans. Here a set of diagrams proposed in Agent UML [20] can be used. The detailed design specifies the interaction between agents, that will allow to realize the coordination processes designed at the architectural design level. At this stage we exploit specification of agent communication protocols available in the MAS literature (e.g. [11]).

4. RELATED WORK

As already mentioned, agents coordination has been largely recognized as a critical issue in both MAS and Distributed Artificial Intelligence [26]. 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 [26]).

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* [24], as discussed in [15]. At a more theoretical level, the coordination problem at run-time has been faced using dynamic programming strategies, see for instance [9] or using multi-agent Markov Decision Processes, see for instance [3, 2].

A relevant approach to the design of agents coordination processes is [16], 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 [16] 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 [5, 6].

Finally, work in Computer Supported Cooperative Work [10], 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 [17] 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. Our definition of coordination is rather similar to the one given by Malone, the main difference resting on the fact that Malone doesn't take into account the concept of actor, and, as a consequence, the intentionality behind a coordination requirement remains implicit.

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 detail, 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 is briefly sketched.

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, as well as architectural design, with others, for instance those discussed in Section 4, suitable for detailed design.

6. 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] R. A. Bourne, C. Excelente-Toledo, and N. R. Jennings. Run-time selection of coordination mechanisms in multi-agent systems. In *Proc. of the 14th European Conf. on Artificial Intelligence (ECAI'2000)*, pages 348–352. IOS Press, 2000.

[3] C. Boutilier. Sequential optimality and coordination in multiagent systems. In *IJCAI*, pages 478–485, 1999.

[4] P. Busetta, L. Serafini, D. Singh, and F. Zini. Extending multi-agent cooperation by overhearing. In *9th International Conference on Cooperative Information Systems (CoopIS 2001)*, volume 2172 of *Lecture Notes in Computer Science*, Trento, Italy, September 2001. Springer Verlag.

[5] C. Castelfranchi. Modeling Social Action for AI Agents. In *IJCAI*, pages 1567–1576, 1997.

[6] C. Castelfranchi, M. Miceli, and A. Cesta. Dependence relations among autonomous agents. In E. Werner and Y. Demazeau, editors, *Decentralized AI 3 – Proceedings of the Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-91)*, pages 215–231. Elsevier Science B.V.: Amsterdam, Netherland, 1992.

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

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

[9] E. H. Durfee. Practically Coordinating. *AI Magazine*, 20(1), 1999.

[10] C. Ellis and J. Wainer. *Groupware and Computer Supported Cooperative Work*, chapter 10. In Weiss [26], 1999.

[11] Foundation for Intelligent Physical Agents. *FIPA Agent Communication Specifications*. http://www.fipa.org/repository/aclspecs.html.

[12] 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 (SEKE01)*, Buenos Aires - Argentina, June 13 - 15 2001.

[13] F. Giunchiglia, J. Mylopoulos, and A. Perini. The Tropos Software Development Methodology: Processes, Models and Diagrams. Technical Report 0111-20, ITC-irst, 2001.

[14] S. Hayden, C. Carrick, and Q. Yang. Architectural design patterns for multi-agent coordination, 1999.

[15] M. N. Huhns and L. M. Stephens. *Multiagent systems and Societies of agents*, chapter 2. In Weiss [26], 1999.

[16] N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.

[17] Thomas W. Malone and Kevin Crowston. The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, 26(1):87–119, 1994.

[18] J. Mylopoulos and J. Castro. Tropos: A framework for requirements-driven software development. In J. Brinkkemper and A. Solvberg, editors, *Information System Engineering: State of the Art and Research Themes*, Lecture Notes in Computer Science. Springer-Verlag, 2000.

[19] A. Newell. The knowledge level. *Artificial Intelligence*, 18:87–127, 1982.

[20] J. Odell, H. Parunak, and B. Bauer. Extending UML for agents. In G. Wagner, Y. Lesperance, and E. Yu, editors, *Proc. of the Agent-Oriented Information Systems workshop at the 17th National conference on Artificial Intelligence*, pages 3–17, Austin, TX, 2000.

[21] A. Perini. A web advisor for integrated protection. In *ECAI2000 Workshop AI in Agriculture and Natural resources*, 2000.

[22] 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.

[23] 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.

[24] 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, pages 1104–1113, 1980.

[25] A. Susi, A. Perini, and E. Olivetti. Plant disease models. Critical issues in development and use. Technical report, ITC-IRST, 2002.

[26] G. Weiss, editor. *Multiagent System: a modern approach to Distributed AI*. MIT Press, 1999.

[27] M. J. Wooldridge, G. Weiß, and P. Ciancarini, editors. *Agent-Oriented Software Engineering II*, volume 2222 of *Lecture Notes in Computer Science*. Springer-Verlag, May 2001.

[28] E. Yu. Agent-Oriented Modeling: Software Versus the World. In Wooldridge et al. [27], pages 206 – 225.