Developing a Decision Support System for Integrated Production in Agriculture.

Anna Perini [a], Angelo Susi [a,*]

[a] *ITC-irst,*
Via Sommarive 18, I-38050, Povo, Trento, Italy.

Abstract

Recent approaches in building decision support systems (DSS) for agriculture, and more generally for environmental problems, tend to adopt a "systemic" approach. That is to say a problem is analyzed in terms of all the knowledge, the data and the responsibilities it depends on. So, the proposed applications aim to be integrated in larger information systems exploiting the fact that different organizations may manage information sources and resources that are relevant to problem solutions.

The paper focuses on design issues faced during the development of a DSS at use of technicians of the advisory service performing pest management according to an Integrated Production approach.

Designing this type of system requires to analyze basically, two main dimensions of complexity: the organizational dimension dealing with all the dependencies between the domain stakeholders, and the technical dimension concerning the study of natural plant protection techniques.

These considerations motivate the choice of an agent-oriented methodology for software development. The methodology, called Tropos, gives a central role to early requirements analysis and allows to derive system functional and non-functional requirements from a deep understanding of the domain stakeholders goals and of their dependencies.

Two components of the system have been implemented using web technologies and they are currently under evaluation.

Key words: Software Design, Agent Oriented Software Engineering, Agriculture, Integrated Production, Decision Support System, Artificial Intelligence

* Phone: (+39) 0461-314344, Fax: (+39) 0461-302040
Email addresses: perini@irst.itc.it (Anna Perini), susi@irst.itc.it (Angelo Susi).

1 Introduction

Integrated Production (IP) in agriculture consists of a set of practices aimed at favoring the set up of a development model characterized by a reduced environmental impact. So, for instance, the application of IP practices in plant disease management by growers and agronomists requires specialistic skills such as, historical data and information on the disease, as well as on chemicals and on low impact techniques, that can be exploited to minimize or avoid damages on the product and on the plants. These sources of information and knowledge are distributed among different actors in the agriculture production system. So, DSS – including AI applications – for IP can be effective if they are integrated within larger information systems and if they are built taking into account the different roles, in the production systems, that can be covered by its users. For instance, using Machine Learning (ML) techniques (see Mitchell (1997)) in the development of plant disease models that simulate the seasonal evolution of a disease – a relevant activity when assessing the seriousness of an infection – poses critical issues such as providing mechanisms for making the model easily adaptable to different geographical environments or defining a suitable maintenance policy that allows for automatic updates of data (e.g. daily updates of meteo data, pesticides updates twice a year, daily observation on disease manifestations during spring and summer, etc.)[1] . This motivated the adoption of an approach that considers the activities of data acquisition and of data analysis as part of an iterative process aimed at providing accurate predictions on disease evolution (see Avesani et al. (2002)). A process that involves different actors, such as technicians of the meteo center, agronomists, researchers in biology and agronomy.

Analogous considerations resulted from previous experiences aimed at applying AI techniques to environmental problems. In Branting et al. (1997), the problem of predicting the behavior of a biological system, such as grasshoppers, when dealing with pest management activities, has been faced adopting an approach called *model-based adaptation*. This approach integrates case-based reasoning with model-based reasoning in order to overcome problems due to incomplete causal theory and limited empirical data for the biological behaviour of grasshoppers. Avesani et al. (1998) described a solution to the problems related to the intervention planning for fire fighting, where AI techniques for planning and scheduling were integrated with a DBMS and a Geographical Information Systems (GIS). Moreover, features such as the distributedness and the heterogeneity of data and knowledge involved in decision making for environmental problems have been discussed from a Knowledge Management perspective in Cortés et al. (2000).

[1] A discussion of critical issues to be faced when building plant disease model can be found in Susi et al. (2002)

More generally, these issues motivate the adoption of a "systemic" approach in designing software systems for environmental problems. That is to say a problem is analyzed in terms of all the knowledge, the data and the responsibilities it depends on. So, the proposed applications aim to be integrated in larger information systems exploiting the fact that different organizations may manage information sources and resources that are relevant to problem solutions. This basically has a twofold effect: first, an organizational analysis becomes a necessary step when specifying application requirements; second, the resulting applications should be designed in terms of a set of specific, interrelated services, such as information providing or reasoning services, that are provided by specialized software components.

Depending on the required capabilities, each software component can be implemented as a software Agent using specific AI techniques for reasoning or as a generic software component, such as a wrapper to existing DBMS, or GIS.

This paper focuses on the requirement analysis and the design of a software system devoted to support decision making by the technicians of the agricultural advisory service when managing apple plant diseases as described in Perini (2000).

We adopt the *Tropos* methodology, described in Giunchiglia et al. (2002a), an agent-oriented software development methodology which includes intentional analysis techniques. The paper is structured as follows. Section 2 recalls the main concepts and the practical steps of the *Tropos* methodology. Sections 3 and 4 describe the results of early and late requirements analysis, according to Tropos. Section 5 describes the initial phase of the the architectural design of the system. Related work is considered in Section 6. Finally, conclusions and the future work are presented in Section 7.

2 The Methodology

The *Tropos* methodology, described in Giunchiglia et al. (2002a), is an agent-oriented software development methodology based on two key ideas, namely: *(i)* the use of knowledge level concepts, such as actor, goal, plan and dependency between actors, along the whole software development process, and *(ii)* the critical role assigned to the preliminary phase of requirements analysis aimed at understanding the environment in which the system-to-be will operate. Tropos covers five software development phases: *early requirements analysis*, *late requirements analysis*, *architectural design*, *detailed design*, and *implementation*. From a practical point of view, the methodology guides the software engineer along the whole process in building conceptual models, with the help of a visual modeling language which provides an ontology including

knowledge level concepts. 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. The notion of actor in Tropos is a generalization of the classical AI notion of software 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, as described also in Chung et al. (2000). A Tropos model is represented as a set of diagrams: *actor diagrams* describe the network of dependency relationships among actors, *goal diagrams*, illustrates goal and plan analysis from the point of view of a specific actor. Three basic types of analysis are provided: *(i) means-end analysis*, which consists in identifying goals, plans or resources that represent means for reaching a goal (plan); *(ii) contribution analysis* which consists in discovering goals, plans or resources that can contribute positively or negatively towards the fulfillment of a goal (or the execution of a plan); *(iii) AND/OR decomposition* which allows for a combination of AND and OR decompositions of a root goal (plan) into sub-goals (sub-plans), thereby refining a goal (plan) structure.

The purpose of conceptual modeling in each phase of the software development process is briefly recalled below.

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.

Late Requirement analysis focuses on the system-to-be which is introduced as a new actor into the model.

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. Each actor is characterized by: (i) a set of individual capabilities and (ii) a set of social capabilities required by actor coordination. Here, the choice of a specific architectural style for distributed systems (see for instance Garlan and Shaw (1996)), such as MAS, Peer to Peer, Client/Server, can be included. The output of the architectural design is the mapping of the system subactors (with their capabilities) to a set of components (possibly 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.

The *Implementation* activity produces an implementation skeleton according

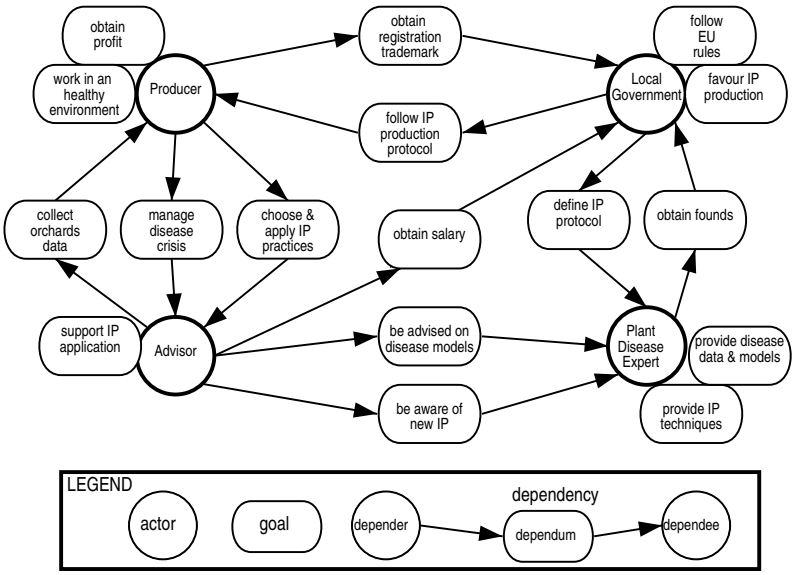

Fig. 1. Early requirements model. A portion of the actor diagram modeling the IP organizational setting.

to the detailed design specification. Code is added to the skeleton using the programming language supported by the implementation platform.

In the following sections we will describe the application of the methodology to the design of the DSS for the IP Advisor we are developing. We will focus only on the first phases of the development process.

3 Early Requirements

The analysis starts identifying the stakeholders, both social actors and software systems that are already present in the domain, of the agriculture production system of our region. They are modeled as actors, depicted by circles, 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 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 1 shows some of the critical dependencies between the domain stakeholders which, at a macroscopic level, result in a joint effort to disseminate IP.

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 depends: 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 2 shows the analysis of the goal support IP application, from the point of view of the 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

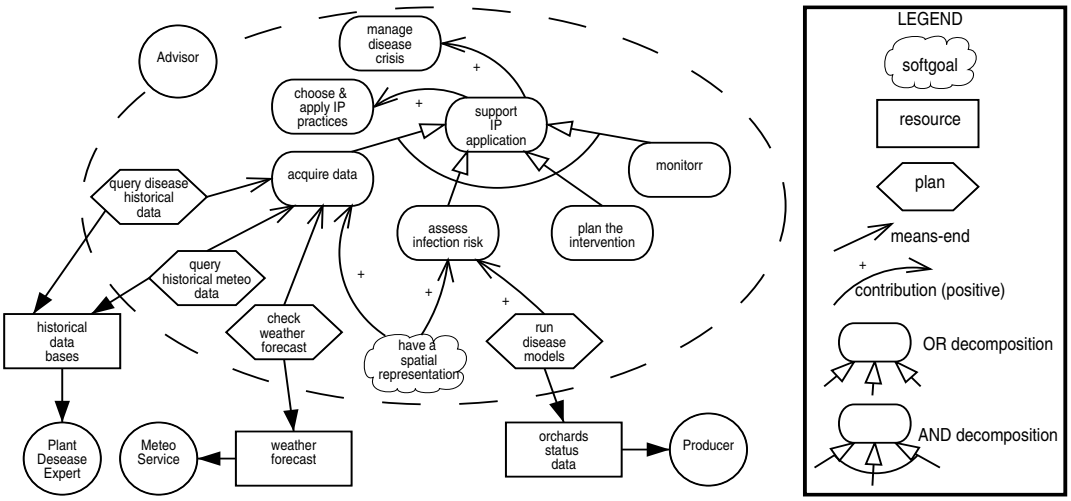

Fig. 2. Early requirements model. The goal diagram of the goal support IP application analyzed from the point of view of the actor Advisor.

Producer depends from 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 2).

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 2 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 interaction 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 current IP practices, the advisors exploit phenology and/or epidemiological models which help them in analyzing the behavior of a plant disease. For instance, they allow

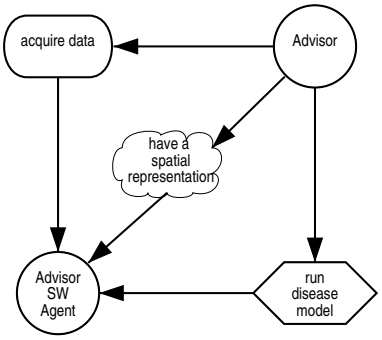

Fig. 3. Late requirements model. Portion of the actor diagram upon the introduction of the system-to-be.

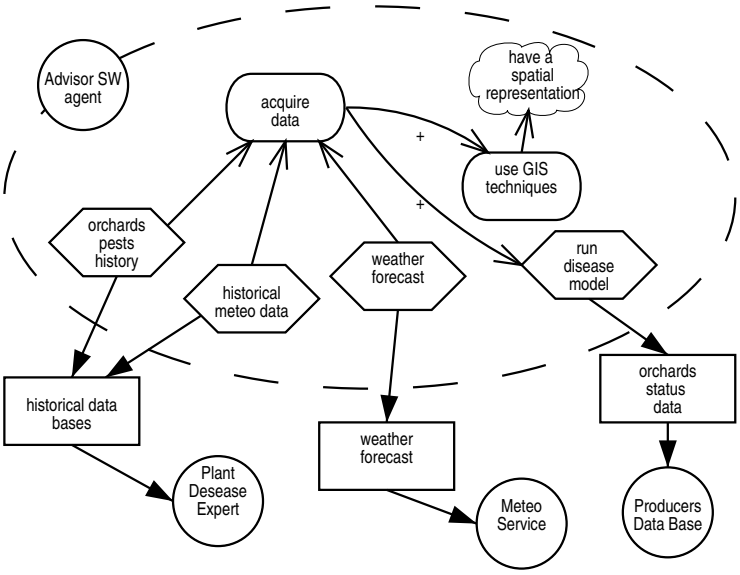

Fig. 4. Late requirements model. The Advisor SW Agent goal diagram.

them to estimate both the disease stage and the infection extent. 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 advisor plans and dependencies with the other actors.

4 Late Requirements

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. Its relationships with social actors, such as Advisor, are modeled in terms of dependencies. Figure 3 depicts a fragment of the late requirements model where the actor Advisor SW

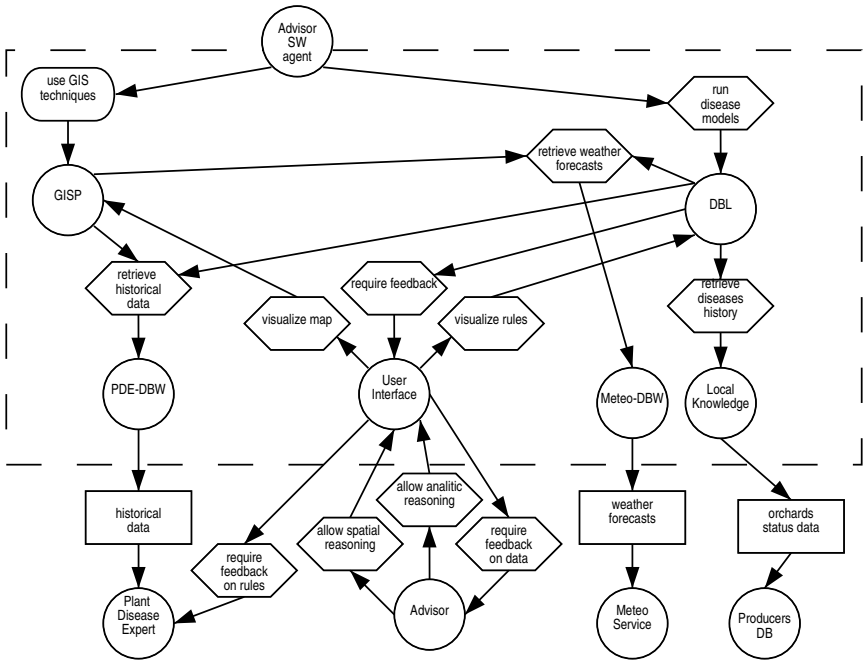

Fig. 5. Architectural Design - step1. The actor diagram refined upon system sub-goals delegation.

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 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 dependencies with actors holding data relevant for disease management have been delegated to the system-to-be actor. Figure 4 shows the resulting goal diagram for the actor Advisor SW Agent. Notice 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.

5 Architectural Design

This phase consists of three steps, namely, refining the system actor diagram taking into account goal and plan decomposition, as well as exploiting useful architectural styles *(i)*, identifying actors capabilities *(ii)* and mapping them to system components (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 5 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 4). 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 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 5);
- 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 relationships between subsystems 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, as in Hayden et al. (1999) and in Busetta et al. (2001), that provide solutions to heterogeneous agents communication, or upon the analysis of non-functional requirements, as described in Giorgini et al. (2001).

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 5, 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 associated to the actor User Interface

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.

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 software components (agents). Each component is characterized by a set of the capabilities identified in the actor diagram.

The next phase in the Tropos development process is detailed design, where the components micro-level is specified in terms of component capabilities and plans. Here a set of diagrams proposed in Agent UML by Odell et al. (2000) can be used. The detailed design specifies the interaction between software components, that will allow to realize the dependencies designed at the architectural design level.

6 Related work

Different lines of research are relevant to the work presented here. We already mentioned in the introduction works which deal with the problem of applying AI techniques to complex environmental problems (Avesani et al. (1998), Branting et al. (1997)). Here we focus on two research lines which aim at defining the design methodologies of complex distributed systems. The first concerns how the Agent paradigm has been exploited in software systems devoted to environmental management. The second focuses on Agent Oriented Software Engineering methodologies.

A natural use of the Agents paradigm can be found at the logical and technological level in environmental simulation models. Here an agent may represent a counterpart of a physical entity in the domain. Agents interacts among them, according to elementary rules, resulting in the macroscopic phenomena or behaviors at interest. Agent based simulation requires to define (assume hypotheses on the) micro-level interactions while the traditional simulation approach assumes a set of rules (laws) governing the interaction among entities, at the macroscopic level. In Bruse (2002), the agent based systems may allow to discover and simulate the dependencies and relationships between environmental variables although they are not included in the model a priori. Agent based simulation has been also exploited in the analysis of the behavior of social networks, as in Pahl-Wostl (2002). Here the simulation involves the relationship between agents that represents people involved in the management of environmental resources and "Integrated Assessment"; the objective is that of building of a domain model at support of decision making processes. Another interesting application of the Agent paradigm is the control and management of complex plants, such as in Borrel et al. (2002), where an application to Wastewater Treatment Plants is described. In this case, the software system interacts with the domain experts and gives them support in supervising the treatment process collecting plant sensors' data, supporting the planning and the execution of suitable control actions, according to macro-level strategies previously chosen by the experts.

Agent Oriented Software Engineering aims at providing methodologies and tools at support of requirements analysis and design of complex systems, such as Multi Agent Systems. Several interesting approaches have been proposed (see the AOSE workshops acts in Ciancarini and Wooldridge (2001), Wooldridge et al. (2001), Giunchiglia et al. (2002b)). Most of them focus on architectural design and detailed design issues, see for instance GAIA in Wooldridge et al. (2000) and AUML in Bauer et al. (2001). The Tropos methodology aims at covering the whole software development process, from early requirements to the implementation of the system using the same notions of agent, goal, dependencies. Tropos exploits techniques for goal analysis originally proposed for Requirement Engineering, like the Eric Yu's i^* (introduced in Yu (1995)) and can be combined with other agent and non-agent software development paradigms like UML or AUML for the system design phases.

7 Conclusion and Future Work

This paper described the requirement analysis and the design of a decision support system, for the agriculture Advisory Service of our region which have been performed using *Tropos*, an agent oriented software engineering method-

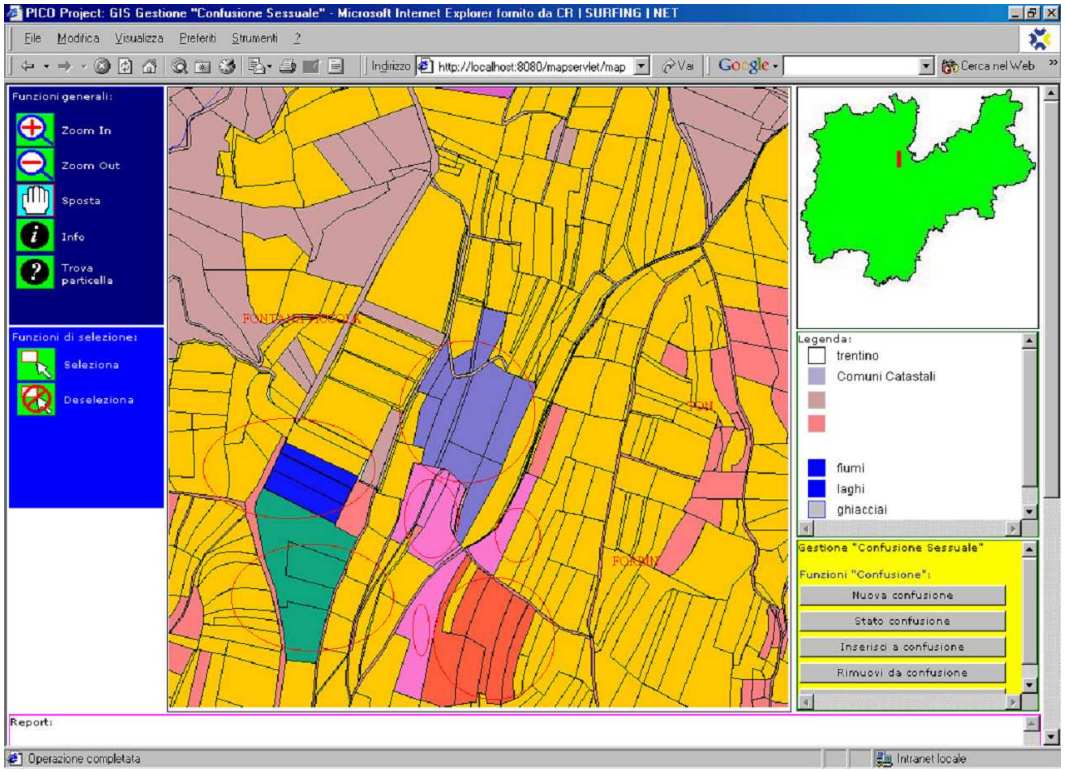

Fig. 6. The GISP component graphic user interface.

ology that allows to model explicitly the domain stakeholders goal and mutual dependencies.

We discussed the early requirement and late requirement analysis specifying the reasons for dependencies between social and system actors. In particular, goal and plan delegations from social actors (the users) to system actors were pointed out.

A sketch of the architectural design, according to the *Tropos* methodology has been also given. The architecture includes a set of software components (agents) wrapping existing information systems and interacting with agents providing estimates on the evolution of a specific plant disease.

We are currently developing some of the agents of the Advisor SW Agent system: the GISP component and the DBL component with reference to a critical pest for apple, called the Codling Moth (*Cydia Pomonella*). A first prototype of the system has been developed and evaluated by the technicians and researchers of the Advisory Service providing useful suggestions for its improvement. The GISP component is based on a set of Geographic Information System (GIS) functionalities that allow to visualize territorial data and to perform spatial queries relatively to the apple orchards in the Trentino area.

In particular, we have developed a set of functions supporting the design of

a pheromones sex trapping plant (a technique for reducing the effects of the *Cydia Pomonella* infections in an orchard), on a multi-orchard basis. Figure 6 depicts the browser based Graphical User Interface of this component. The visualization area is subdivided in three major areas: in the center is depicted a map of the area of interest showing the organizational setting of the orchards; on the left, a set of functions allow the user to interact with the map; on the right, the user can find a set of functions related to the management of a pheromones trapping plant.

The DBL component is based on the Weka software library (see Witten and Frank (1999)) which provides Machine Learning and Data Mining Algorithm, applied to the analysis of the data collected by researchers from the field and from lab tests, during the last ten years.

8 Acknowledgments

The work presented in the paper has been realized inside the *PICO* project (see http://sra.itc.it/project.epl?name=DSS-PICO) which is partially funded by the Italian Ministry of Scientific and Technological Research.

References

Avesani, A., Perini, A., Ricci, F., Jul. 1998. The Twofold Integration of CBR in Decision Support Systems. In: AAAI98 Workshop on Case-Based Reasoning Integrations. Madison.

Avesani, P., Olivetti, E., Susi, A., July 2002. Feeding Data Mining. Tech. Rep. 0207-01, Istituto Trentino di Cultura - IRST.

Bauer, B., Müller, J. P., Odell, J., 2001. Agent UML: A formalism for specifying multiagent software systems. Int. Journal of Software Engineering and Knowledge Engineering 11 (3), 207–230.

Borrel, F., Riaño, D., Sànchez-Marrè, M., Rodríguez-Roda, I., 2002. Agent Based Simulation in Integrated Assessment and Resources Management. In: Rizzoli, A. E., Jakeman, A. J. (Eds.), International Environmental Modelling and Software Society (iEMSs 2002). No. 3. Lugano, Switzerland, pp. 402–407.

Branting, K., Hastings, J.D., and Lockwood, J.A., 1997. Integrating cases and models for prediction in biological systems, AI Applications, Vol. 11, No. 1, pp. 29–48.

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A., 2003. Tropos: An Agent-Oriented Software Development Methodology. To appear in Journal of Autonomous Agent and Multi-Agent Systems.

Bruse, M., 2002. Multi-Agent Simulations as a Tool for the Assessment of Urban Microclimate and its Effect on Pedestrian Behaviour. In: Rizzoli, A. E., Jakeman, A. J. (Eds.), International Environmental Modelling and Software Society (iEMSs 2002). No. 2. Lugano, Switzerland, pp. 196–201.

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

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

Ciancarini, P., Wooldridge, M. (Eds.), March 2001. Agent-Oriented Software Engineering. Vol. 1957 of Lecture Notes in AI. Springer-Verlag.

Cortés, U., Sànchez-Marrè, M., Comas, J., Rodríguez-Roda, I., Poch, M., 2000. Knowledge management in environmental decision support systems. In: ECAI2000 WS AI in Agriculture and Nat.Res.

Garlan, D., Shaw, M., 1996. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall Publishing, 1996.

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

Giunchiglia, F., Mylopoulos, J., Perini, A., 2002a. The Tropos Software Development Methodology: Processes, Models and Diagrams. In: Giunchiglia et al. (2002b).

Giunchiglia, F., Odell, J., Weiß, G. (Eds.), 2002b. Agent-Oriented Software Engineering III, Third International Workshop, AOSE2002 Edition. LNCS. Springer-Verlag, Bologna, Italy.

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

Mitchell, T., 1997. Machine Learning. McGraw Hill.

Odell, J., Parunak, H., Bauer, B., 2000. Extending UML for agents. In: Wagner, G., Lesperance, Y., Yu, E. (Eds.), Proc. of the AOIS 2000 workshop at the 17th National conference on Artificial Intelligence. Austin, TX, pp. 3–17.

Pahl-Wostl, C., 2002. Agent Based Simulation in Integrated Assessment and Resources Management. In: Rizzoli, A. E., Jakeman, A. J. (Eds.), International Environmental Modelling and Software Society (iEMSs 2002). No. 2. Lugano, Switzerland, pp. 239–244.

Perini, A., 2000. A web advisor for integrated protection. In: ECAI2000 WS AI in Agriculture and Nat.Res.

Rao, A., Georgeff, M., 1991. Modelling rational agents within a BDI-architecture. In: Proceedings of Knowledge Representation and Reasoning (KRR-91) Conference. San Mateo CA.

Susi, A., Perini, A., Olivetti, E., 2002. Plant disease models. Critical issues

in development and use. In: Rizzoli, A. E., Jakeman, A. J. (Eds.), International Environmental Modelling and Software Society (iEMSs 2002). Lugano, Switzerland, pp. 426–431.

Witten, I. H., Frank, E., October 1999. Data Mining. Morgan Kaufmann.

Wooldridge, M., Jennings, N. R., Kinny, D., 2000. The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems 3 (3).

Wooldridge, M., Weiß, G., Ciancarini, P. (Eds.), May 2001. Agent-Oriented Software Engineering II, Second International Workshop, AOSE2001 Edition. LNCS 2222. Springer-Verlag, Montreal, Canada.

Yu, E., 1995. Modelling strategic relationships for process reengineering. Ph.D. thesis, University of Toronto, Department of Computer Science, University of Toronto.