# Agent-Oriented Software Development: A Case Study

Paolo Giorgini[1] Anna Perini[2] John Mylopoulos[3] Fausto Giunchiglia[4] Paolo Bresciani[2]

[1] Department of Mathematics - University of Trento - Italy - pgiorgini@science.unitn.it
[2] ITC-irst - Povo (Trento) - Italy - {bresciani,perini}@itc.it
[3] Department of Computer Science - University of Toronto - Canada - jm@cs.toronto.edu
[4] Dipartimento di Informatica e Studi Aziendali - University of Trento - Italy - fausto@cs.unitn.it

## Abstract

*We are developing a methodology, called Tropos, for building agent-oriented software systems. The methodology covers five software development phases: early requirements analysis, late requirements analysis, architectural design, detailed design, and implementation. Throughout, the concepts offered by i* are used to model both the stakeholders in the system's environment, and the system itself. These concepts include* actors*, who can be (social) agents (organizational, human or software), positions or roles,* goals*, and* social dependencies *for defining the obligations of actors to other actors (called dependees and dependers respectively.) Dependencies may involve a* goal*, to be fulfilled by the dependee on behalf of the depender, a* task *to be carried out by the dependee, or a* resource *to be delivered. The paper presents a case study to illustrate the features and the strengths of the Tropos methodology.*

## 1. Introduction

The explosive growth of application areas such as electronic commerce, enterprise resource planning and mobile computing has profoundly and irreversibly changed our views on software and Software Engineering. Software must now be based on open architectures that continuously change and evolve to accommodate new components and meet new requirements. Software must also operate on different platforms, without recompilation, and with minimal assumptions about its operating environment and its users. As well, software must be robust and autonomous, capable of serving a naïve user with a minimum of overhead and interference. These new requirements, in turn, call for new concepts, tools and techniques for engineering and managing software.

For these reasons – and more – agent-oriented software development is gaining popularity over traditional software development techniques, including structured and object-oriented ones (see for instance [6]). After all, agent-based architectures *do* provide for an open, evolving architecture which can change at run-time to exploit the services of new agents, or replace under-performing ones. In addition, software agents can, in principle, cope with unforeseen circumstances because they include in their architecture goals, along with a planning capability for meeting them. Finally, agent technologies have matured to the point where protocols for communication and negotiation have been standardized [1].

We are developing a software development methodology for agent-based software systems. The methodology adopts ideas from multi-agent system technologies, mostly to define the implementation phase of our methodology. The implementation platform we use is JACK Intelligent Agents [2], a commercial agent programming platform based on the BDI (Beliefs-Desires-Intentions) agent architecture. We are also adopting ideas from Requirements Engineering, where agents and goals have been heavily used for early requirements analysis [5, 11] . In particular, we adopt Eric Yu's *i** model which offers actors (agents, roles, or positions), goals, and actor dependencies as primitive concepts for modelling an application during early requirements analysis. The key assumption which distinguishes our work from others in Requirements Engineering is that actors and goals are used as fundamental concepts for modelling and analysis during *all phases of software development*, not just early requirements. [1]

Our methodology, named Tropos, is intended to support five phases of software development:

**Early Requirements**, concerned with the understanding of a problem by studying an existing organizational setting; the output of this phase is an organizational model which includes relevant actors and their respective dependencies;

[1] Analogously to the use of concepts such as *object*, *class*, *inheritance* and *method* in object-oriented software development.

**Late requirements**, where the system-to-be is described within its operational environment, along with relevant functions and qualities; this description models the system as a (small) number of actors which have a number of dependencies with actors in their environment; these dependencies define the system's functional and non-functional requirements;

**Architectural design**, where the system's global architecture is defined in terms of subsystems, interconnected through data and control flows; within our framework, subsystems are represented as actors and data/control interconnections are represented as (system) actor dependencies;

**Detailed design**, where each architectural component is defined in further detail in terms of inputs, outputs, control, and other relevant information; our framework adopts elements of AUML [8], to complement the features of *i**;

**Implementation**, where the actual implementation of the system is carried out, consistently with the detailed design.

The motivations behind the Tropos project and an early glimpse of how the methodology would work for particular examples are given in [7, 3]

This paper reports on a case study which applies the Tropos framework to all phases of analysis, design and implementation for fragments of a system developed for the government of Trentino (Provincia Autonoma di Trento, or PAT). The system (which we will call throughout the *eCulture System*) is intended as a web-based broker of cultural information and services for the province of Trento, including information obtained from museums, exhibitions, and other cultural organizations. It is a government's intention that the system be usable by a variety of users, including Trentinos and tourists looking for things to do, or scholars and students looking for material relevant to their studies.

Section 2 of the paper introduces the key features of *i** and illustrates its use during early requirements analysis in Tropos. Sections 3, 4 and 5 cover late requirements analysis, architectural design and detailed design, respectively. Section 6 describes the implementation phase, during which the detailed design developed in terms of *i** and AUML diagrams are mapped onto the skeleton of a multi-agent system using the JACK agent programming platform. Conclusions and directions for further research are presented in section 7.

## 2. Early Requirements Analysis

During early requirements analysis, the requirements engineer models and analyzes the intentions of the stakeholders. Intentions are modeled as goals which, through a goal-oriented analysis, eventually lead to the functional and non-functional requirements of the system-to-be [5]. In Tropos, early requirements are assumed to involve social actors who depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished. In analogy with *i**, Tropos includes *actor diagrams* for describing the network of social dependency relationships among actors, as well as *rationale diagrams* for analyzing goals through a means-ends analysis in order to discover ways of fulfilling them.[2] These primitives have been formalized using intentional concepts from AI, such as goal, belief, ability, and commitment. The *i** framework has been presented in detail in [11] and has been related to different application areas, including requirements engineering [10], business process reengineering [13], and software processes [12].

An actor diagram is a graph, where each node represents an *actor*, and each link between two actors indicates that one actor depends on the other for something in order that the former may attain some goal. We call the depending actor the *depender* and the actor who is depended upon the *dependee*. The object around which the dependency centers is called the *dependum*. By depending on another actor for a dependum, an actor is able to achieve goals that it is otherwise unable to achieve on its own, or not as easily, or not as well. At the same time, the depender becomes vulnerable. If the dependee fails to deliver the dependum, the depender would be adversely affected in its ability to achieve its goals. The list of relevant actors for the eCulture project includes, among others, the following stakeholders [3]:

- Provincia Autonoma di Trento (PAT) is the government agency funding the project; their objectives include improving public information services, increasing tourism through new information services, also encouraging internet use within the province (and keeping citizens happy, of course!)
- Museums, who are major cultural information providers for their respective collections; museums want government funds to build/improve their cultural information services, and are willing to interface their systems with the eCulture System.
- Tourist, who will want to access cultural information before or during her visit to Trentino to make her visit interesting and/or pleasant.
- (Trentino) Citizen, who wants easily accessible information, of any sort, and (of course) good government!

Figure 1 shows these actors and their respective goals. In particular, Citizen is associated with a single relevant

[2] In *i**, actor diagrams are called *strategic dependency models*, while rationale diagrams are called *strategic rationale models.*

[3] The list has been scaled down from a longer list which included the organizations responsible for the development of the system as well as the developers.

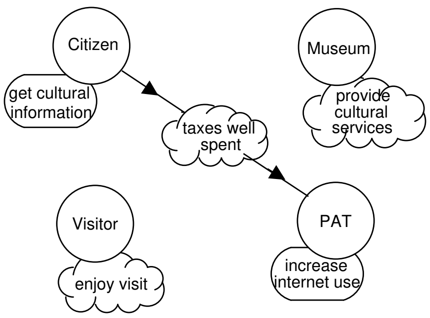


**Figure 1. The stakeholders of the eCultural project**

goal: get cultural information, while Visitor has an associated softgoal enjoy visit. Softgoals are distinguished from goals because they don't have a formal definition, and are amenable to a different (more qualitative) kind of analysis [4]. Along similar lines, PAT wants to increase internet use while Museum wants to provide cultural services. Finally, the diagram includes one softgoal dependency where Citizen depends on PAT to fulfil the taxes well spent softgoal. It should be noted that the starting point of the early requirements analysis is higher level than in most other projects, where initial goals are more concrete. This is due to the fact that the eCulture project is an R&D project involving government and universities, as opposed to a business-to-business project. Once the stakeholders have been identified along with their goals, analysis proceeds by analyzing through a rationale diagram each goal relative to the stakeholder who is responsible for its fulfillment. Figure 2 shows fragments of such an analysis from the perspective of Citizen and Visitor. The rationale diagrams for each actor are shown as balloons within which goals are analyzed and dependencies to other actors are established. For the actor Citizen, the goal get cultural information is decomposed into visit cultural institutions and visit cultural web systems. The latter goal is fulfilled by the task (shown as a hexagonal icon) visit eCulture System, and this task is decomposed into sub-tasks use eCulture System and access internet. At this point, Citizen can rely on other actors, namely PAT to deliver on the eCulture System and make it usable too. The analysis for Visitor is simpler: to enjoy a visit, the visitor must plan for it and for this she needs the eCulture System too. Museum is assumed to rely on funding from PAT to fulfil its objective to offer good cultural services.

The example is intended to illustrate how means-ends analysis is conducted. Throughout, the idea is that goals are decomposed into subgoals and tasks are introduced for their fulfillment. Note that a goal may have several different tasks that can fulfil it. As in other frameworks where these concepts apply, goals describe *what* is desired, tasks describe

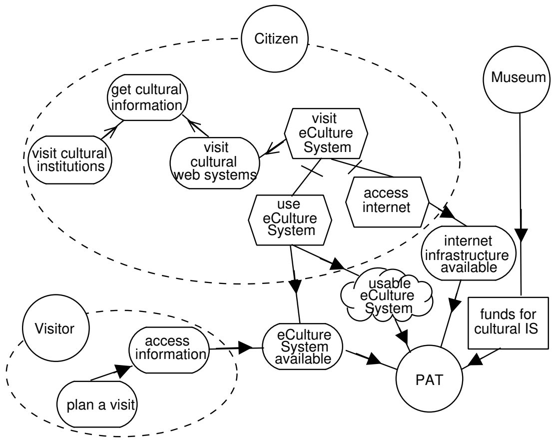


**Figure 2. Means-ends analysis for** Citizen **and** Visitor

*how* goals are to be fulfilled. The result of the means-ends analysis for each goal is a set of dependencies between the actor and other actors through which the goal can be fulfilled. Once a goal dependency is established from, say, Citizen to PAT, the goal is now PAT responsibility and the goal needs to be analyzed from PAT perspective. Portions of the means-ends analysis for PAT are shown in Figure 3. The goals increase internet use and eCulture System available are both well served by the goal build eCulture System. The softgoal taxes well spent gets positive contributions, which can be thought as justifications for the selection of particular dependencies.

The final result of this phase is a set of strategic dependencies among actors, built incrementally by performing means-end analysis on each goal, until all goals have been analyzed. The later it is added, the more specific a goal is. For instance, in the example in Figure 3 PAT goal build eCulture System is introduced last and, therefore, has no subgoals and it is motivated by the higher level goals it fulfills.[4]

## 3. Late Requirements Analysis

During late requirement analysis the system-to-be (the eCulture System in our case) is described within its operating environment, along with relevant functions and qualities. The system is represented as one or more actors which have a number of dependencies with the other actors of the organization. These dependencies define all functional and non-functional requirements for the system-to-be.

Figure 4 includes the eCulture System, introduced as another actor of the actor diagram. PAT depends on the

[4] In rationale diagrams one can also introduce tasks and resources and connect them to the fulfillment of goals.

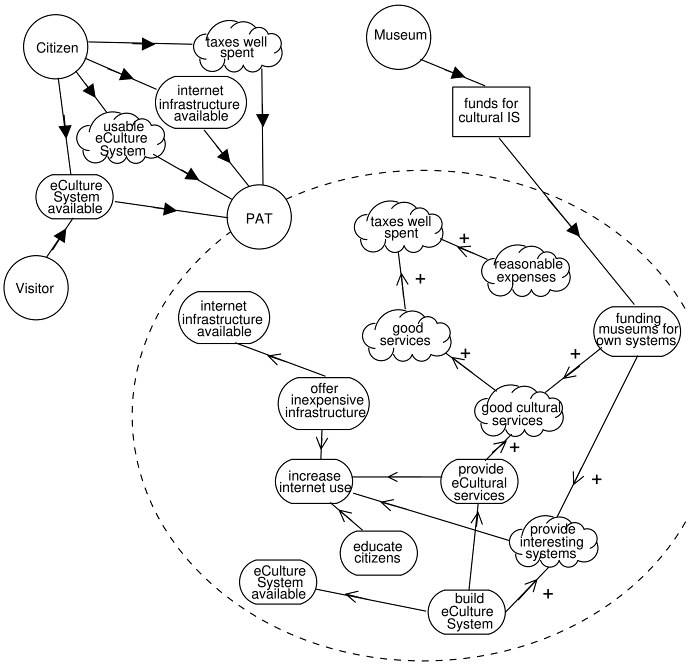


**Figure 3. Means-ends analysis for** PAT

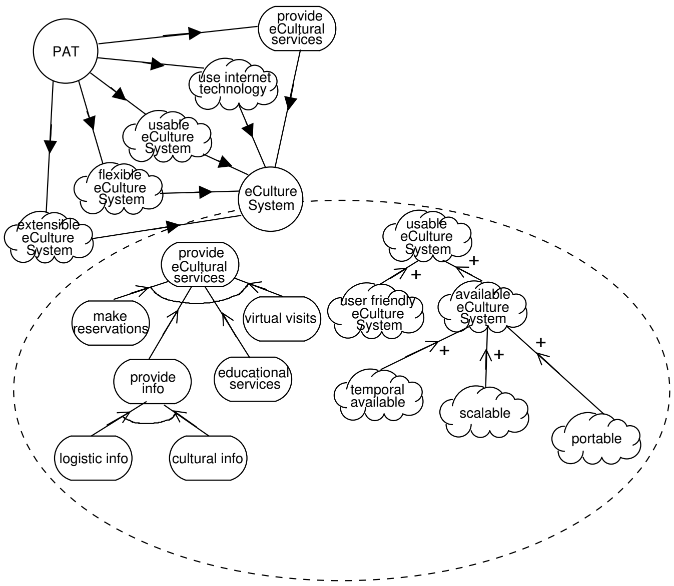


**Figure 4. Rationale diagram for the** eCulture System

eCulture System to provide eCultural services, which is a goal that contributes to the main goal of PAT, namely increase internet use (see Figure 3). The eCulture System is expected to fulfil for PAT softgoals such as extensible eCulture System, flexible eCulture System, usable eCulture System, and use internet technology. In order to exemplify the process, let's concentrate here on an analysis for the goal provide eCultural services and the softgoal usable eCulture System. The goal provide eCultural services is decomposed (AND decomposition) into four subgoals: make reservations, provide info, educational services and virtual visits. As basic eCultural service, the eCulture System must provide information (provide info), which can be logistic info, and cultural info. Logistic info concerns, for instance, timetables and visiting instructions for museums, while cultural info concerns the cultural content of museums and special cultural events. This content may include descriptions and images of historical objects, the description of an exhibition, and the history of a particular region. Virtual visits are services that allow, for instance, Citizen to pay a virtual visit to a city of the past (Rome during Cæsar's time!). Educational services includes presentation of historical and cultural material at different levels (e.g., high school or undergraduate university level) as well as on-line evaluation of the student's grasp of this material. Make reservations allows the Citizen to make reservations for particular cultural events, such as concerts, exhibitions, and guided museum visits. The softgoal usable eCulture System has two positive (+) contributions from softgoals user friendly eCulture System and available eCulture System. The former contributes positively because a system must be user friendly to be usable, whereas the latter contributes positively because it makes the system portable, scalable, and available over time (temporal available).

Once these system goals and softgoals have been defined, new actors (including sub-actors) are introduced. Each new actor takes on the responsibility to fulfil one or more goals of the system. Figure 5 shows the decomposition in sub-actors of the eCulture System and the goal dependences between the eCulture System and its sub-actors. The eCulture System depends on the Info Broker to provide info, on the Educational Broker to provide educational services, on the Reservation Broker to make reservations, on Virtual Visit Broker to provide virtual visits, and on System Manager to provide interface. Additionally, each sub-actor can be itself decomposed in sub-actors responsible for the fulfillment of one or more sub-goals. In Figure 5, System Manager, a position, is decomposed into two roles:[5] System Interface Manager and User Interface Manager, each of which is responsible for an interfacing goal.

Before moving to the architectural design phase, let describe in more detail a portion of the rationale diagram for the eCulture System. Figure 6 shows the analysis for the

[5] An actor can be an agent, a role, or a position (see [11] for more details).

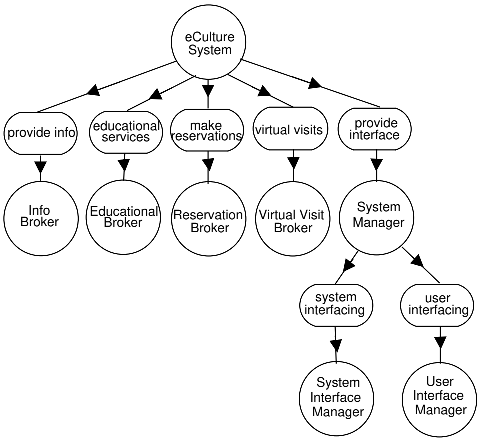


**Figure 5. Sub-actors decomposition for the** eCulture System

fulfillment of the goal get cultural information that the Citizen depends on the eCulture System for. The analysis starts from a root goal search information which can be fulfilled by four different tasks: search by area (thematic area), search by geographical area, search by keyword, and search by time period. The decomposition into sub-tasks is almost the same for all four kind of search. For example, to search information about a particular thematic area, the Citizen provides information using an area specification form. Such information will be used to classify the area, get info on area, and synthesize results. The sub-task get info on area is decomposed in find info sources, that finds which information sources are more appropriate to provide information concerning the specified area, and query sources, that queries the information sources. The sub-task find info sources depends on the museums for the description of the information that the museums can provide (info about source), and synthesize results depends on museums for query result.

# 4. Architectural Design

The architectural design phase consists of four steps: addition of new actors, actor decomposition, capabilities identification and agents assignment.

In the first step new actors are added to the overall actor diagram in order both to make system interact with the external actors and to contribute positively to the fulfillment of some non-functional requirements. The final result of this step is the extended actor diagram, in which the new actors and their dependencies with the other actors are presented. Figure 7 shows the extended actor diagram with respect to the Info Broker. The User Interface Manager

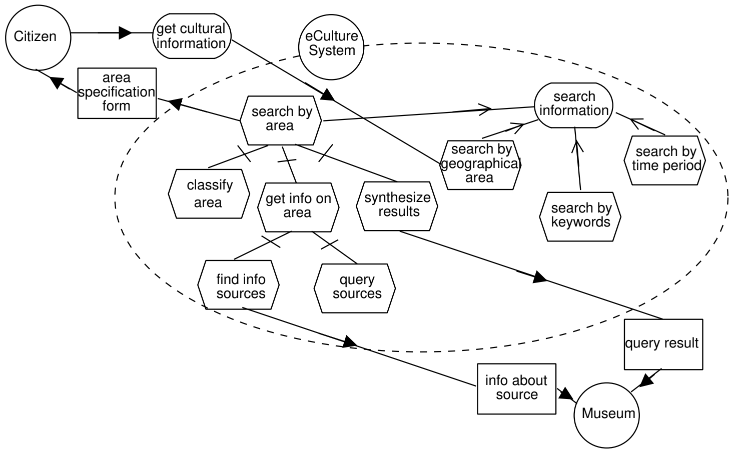


**Figure 6. Rationale diagram for the goal** get cultural information

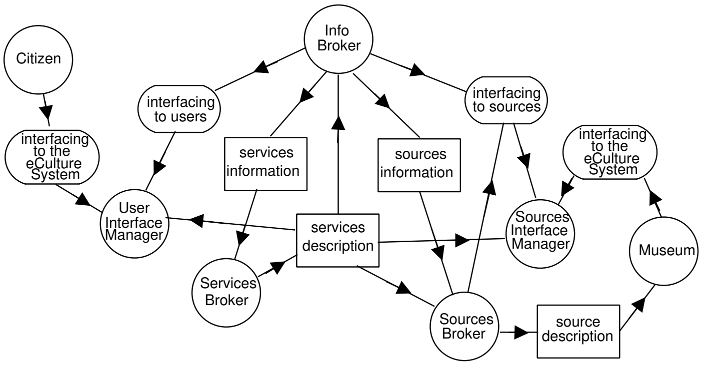


**Figure 7. Extended actor diagram w.r.t. the** Info Broker **(step 1)**

and the Sources Interface Manager are responsible for interfacing the system to the external actors Citizen and Museum. To facilitate actor interactions inside the system, we have introduced two more actors: the Services Broker and Sources Broker[6]. Services Broker manages a repository of descriptions for services offered by actors within the eCulture System. Analogously, Sources Broker manages a repository of descriptions for information sources available outside the system. The introduction of the Services Broker and Sources Broker contributes positively to soft-goal extensible eCulture System that the PAT has delegated to the eCulture System (see Figure 4). In fact, the Services Broker supports extension of the system through new services (possibly provided by new actors), whereas the Sources Broker allows the system to use new information sources. The soft-goal analysis from the system perspective can be helpful for the architecture refine-

[6] These actually correspond to the Directory Facilitator and the Agent Resource Broker in the FIPA recommendations [1].

ment; in particular, for characterizing new actors in terms of functionalities to be inserted into the architecture. In our eCulture System, for example, we can think of adding a new actor called User Profiler which determines and maintains user preferences (profiles) and makes suggestions to the Info Broker accordingly. This actor could contribute positively to the soft-goal user friendly eCulture System.

The second step consists in the decomposition of actors in sub-actors. The aim of the decomposition is to expand in details each actor with respect to its goals and tasks. Figure 8 shows the Info Broker decomposition with respect to the goal of searching information, and in particular, the task search by area reported in Figure 6. The Info Broker is decomposed in three sub-actors: the Area Classifier, the Results Synthesizer, and the Info Searcher. Area Classifier is responsible for the classification of the information provided by the user. It depends on the User Interface Manager for interfacing to the users, and on the Service Broker to have information about the services provided by other actors. The Info Searcher depends on Area Classifier to have information about the thematic area that the user is interested in, on the Source Broker for the description of the information sources available outside the system, and on the Sources Interface Manager for interfacing to the sources. The Results Synthesizer depends on the Info Searcher for the information concerning the query that the Info Searcher asked, and on the Museum to have the query results.

The third step of the architectural design consists in the identification of the capabilities needed by the actors to fulfill their goals and tasks. Capabilities can be easily identified by analyzing the extended actor diagram. In particular each dependency relationship can give place to one or more capabilities triggered by external events. Table 1 lists the capabilities associated to the extended actor diagram of Figure 8. They are listed with respect to the system-to-be actors, and then numbered in order to eliminate possible copies whereas.

The last step of the architectural design is the agents assignment, in which a set of agent types is defined assigning to each agent one or more different capabilities (agent assignment). Table 2 reports the agents assignment with respect to the capabilities listed in Table 1. The capabilities concern exclusively the task search by area assigned to the Info Broker. Of course, many other capabilities and agent types are needed in case we consider all the goals and tasks associated to the complete extended actor diagram.

In general, the agents assignment is not unique and depends on the designer. The number of agents and the capabilities assigned to each of them are choices driven by the analysis of the extend actor diagram and by the way in which the designer think the system in term of agents. Some of the activities done in architectural design can be compared to what Wooldridge et al. propose to do within the Gaia methodology [9]. For instance what we do in actor diagram refinement can be compared to "role modeling" in Gaia. We instead consider also non-functional requirements. Similarly, capability analysis can be compared to "protocols modeling", even if in Gaia only external events are considered.

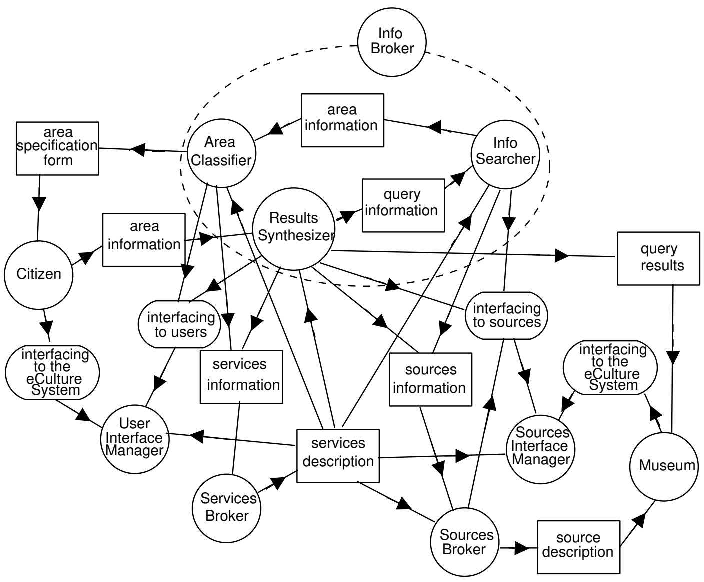


**Figure 8. Extended actor diagram w.r.t. the** Info Broker **(step 2)**

## 5. Detailed design

The detailed design phase aims at specifying agent capabilities and interactions. The specification of capabilities amounts to modeling external and internal events that trigger plans and the beliefs involved in agent reasoning. Practical approaches to this step are often used.[7] In the paper we adapt a subset of the AUML diagrams proposed in [8]. In particular:

1. *Capability diagrams*. The AUML activity diagram allows to model a capability (or a set of correlated capabilities), from the point of view of a specific actor. External events set up the starting state of a capability diagram, activity nodes model plans, transition arcs model events, beliefs are modeled as objects. For instance Figure 9 depicts the capability diagram of the query results capability of the User Interface Agent.
2. *Plan diagrams*. Each plan node of a capability diagram can be further specified by AUML action diagrams.

[7]For instance the *Data-Event-Plan diagram* used by JACK developer. Ralph Rönnquist, personal communication.

| Actor Name | N | Capability |
|---|---|---|
| Area Classifier | 1 | get area specification form |
| | 2 | classify area |
| | 3 | provide area information |
| | 4 | provide service description |
| Info Searcher | 5 | get area information |
| | 6 | find information source |
| | 7 | compose query |
| | 8 | query source |
| | 9 | provide query information |
| | | provide service description |
| Results Synthesizer | 10 | get query information |
| | 11 | get query results |
| | 12 | provide query results |
| | 13 | synthesize area query results |
| | | provide service description |
| Sources Interface Manager | 14 | wrap information source |
| | | provide service description |
| Sources Broker | 15 | get source description |
| | 16 | classify source |
| | 17 | store source description |
| | 18 | delete source description |
| | 19 | provide sources information |
| | | provide service description |
| Services Broker | 20 | get service description |
| | 21 | classify service |
| | 22 | store service description |
| | 23 | delete service description |
| | 24 | provide services information |
| User Interface Manager | 25 | get user specification |
| | 26 | provide user specification |
| | 27 | get query results |
| | 28 | present query results to the user |
| | | provide service description |

**Table 1. Actors' capabilities**

| Agent | Capabilities |
|---|---|
| Query Handler | 1, 3, 4, 5, 7, 8, 9, 10, 11, 12 |
| Classifier | 2, 4 |
| Searcher | 6, 4 |
| Synthesizer | 13, 4 |
| Wrapper | 14, 4 |
| Agent Resource Broker | 15, 16, 17, 18, 19, 4 |
| Directory Facilitator | 20, 21, 22, 23, 24, 4 |
| User Interface Agent | 25, 26, 27, 28, 4 |

**Table 2. Agent types and their capabilities**

3. *Agent interaction diagrams.* Here AUML sequence diagrams can be exploited. In AUML sequence diagrams, agents corresponds to objects, whose life-line is independent from the specific interaction to be modeled (in UML an object can be created or destroyed during the interaction); communication acts between agents correspond to asynchronous message arcs. It can be shown that sequence diagrams modeling Agent Interaction Protocols, proposed by [8], can be straightforwardly applied to our example.

## 6. Implementation Using JACK

The BDI platform chosen for the implementation is JACK Intelligent Agents, an agent-oriented development environment built on top and fully integrated with Java. Agents in JACK are autonomous software components that have explicit goals (desires) to achieve or events to handle. Agents are programmed with a set of plans in order to make them capable of achieving goals.

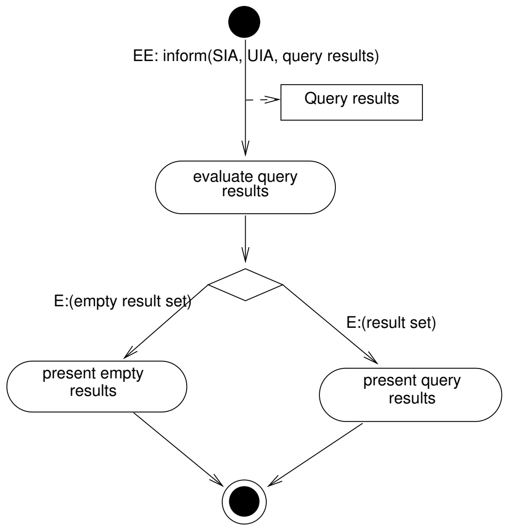


**Figure 9. Capability diagram using AUML activity diagram.**

The implementation activity follows step by step, in a natural way, the detailed design specification described in section 5. In fact, the notions introduced in that section have a direct correspondence with the following JACK's constructs, as explained below:

- *Agent*. A JACK's agent construct is used to define the behavior of an intelligent software agent. This includes the capabilities an agent has, the types of messages and events it responds to and the plans it uses to achieve its goals.
- *Capability*. A JACK's capability construct can include plans, events, beliefs and other capabilities. An agent can be assigned a number of capabilities. Furthermore, a given capability can be assigned to different agents. JACK's capability provides a way of applying reuse concepts.
- *Belief*. Currently, in Tropos, this concept is used only in the implementation phase, but we are considering to move it up to earlier phases. The JACK's database construct provides a generic relational database. A database describes a set of beliefs that the agent can have.
- *Event*. Internal and external events specified in the detailed design map to the JACK's event construct. In JACK an event describes a triggering condition for agents actions.
- *Plan*. The plans contained into the capability specification resulting from the detailed design level map to the

JACK's plan construct. In JACK a plan is a sequence of instructions the agent follows to try to achieve goals and handle designed events.

As an example, the definition for the User Interface Agent, in JACK code, is as follows:

```
public agent UserInterface extends Agent {
  #has capability GetQueryResults;
  #has capability ProvideUserSpecification;
  #has capability GetUserSpecification;
  #has capability PresentQueryResults;
  #handles event InformQueryResults;
  #handles event ResultsSet; }
```

The capability present query results, analyzed in Figure 9 is defined as follows:

```
public capability PresentQueryResults
                       extends Capability {
  #handles external event InformQueryResults;
  #posts event ResultsSet ;
  #posts event EmptyResultsSet ;
  #private database QueryResults ();
  #private database ResultsModel ();
  #uses plan EvaluateQueryResults;
  #uses plan PresentEmptyResults;
  #uses plan PresentResults; }}
```

## 7. Conclusions

In this paper we have reported on a case study which applies the Tropos framework to all phases on analysis, design and implementation for fragments of a system developed for the government of Trentino. Tropos is a new software development methodology for agent-based software systems, which allows us to exploit the advantages and the extra flexibility (if compared with other programming paradigms, for instance object oriented programming) coming from using Agent Oriented Programming. The basic assumption which distinguishes our work from others in Requirements Engineering is that actors and goals are used as fundamental concepts for modelling and analysis during all the phases of software development, not just early requirements.

Of course, much remains to be done to further refine the proposed methodology. We are currently working on several open points, such as the development of formal analysis techniques for Tropos, and also the development of tools which support different phases of the methodology.

## References


[1] FIPA. Foundation for Intelligent Physical Agents. http://www.fipa.org.

[2] Paolo Busetta, Ralph Rönnquist, Andrew Hodgson, and Andrew Lucas. Jack intelligent agents - components for intelligent agents in java. AOS TR9901, January 1999. http://www.jackagents.com/pdf/tr9901.pdf.

[3] J. Castro, M. Kolp, and J. Mylopoulos. Developing agent-oriented information systems for the enterprise. In *Proceedings Third International Conference on Enterprise Information Systems*, Stafford UK, July 2000.

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

[5] A. Dardenne, A. van Lamsweerde, and S. Fickas. "goal" directed requirements acquisition. *Science of Computer Programming*, (20), 1993.

[6] N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2), 2000.

[7] J. Mylopoulos and J. Castro. *Tropos: A Framework for Requirements-Driven Software Development*. Lecture Notes in Computer Science. Springer-Verlag, 2000.

[8] J. Odell and C. Bock. Suggested UML extensions for agents. Technical report, OMG, December 1999. Submitted to the OMG's Analysis and Design Task Force in response to the Request for Information entitled "UML2.0 RFI".

[9] M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3), 2000.

[10] E. Yu. Modeling organizations for information systems requirements engineering. In *Proceedings First IEEE International Symposium on Requirements Engineering*, pages 34–41, San Jose, January 1993. IEEE.

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

[12] E. Yu and J. Mylopoulos. Understanding 'why' in software process modeling, analysis and design. In *Proceedings Sixteenth International Conference on Software Engineering*, Sorrento, Italy, May 1994.

[13] E. Yu and J. Mylopoulos. Using goals, rules, and methods to support reasoning in business process reengineering. *International Journal of Intelligent Systems in Accounting, Finance and Management*, 1(5), January 1996.