# Using Tropos to Model Agent Based Architectures for Adaptive Systems: a Case Study in Ambient Intelligence

Loris Penserini, Paolo Bresciani, Tsvi Kuflik, Paolo Busetta
*ITC-irst, via Sommarive 18, I-38050 Trento-Povo, Italy*
*E-mail:{penserini, bresciani, kuflik , busetta}@itc.it*

## Abstract

*The realization of complex distributed applications required in areas such as e-Business, e-Government, and ambient intelligence calls for new software development paradigms, such as the Service Oriented Computing approach, which accommodates for dynamic and adaptive interaction schemata, carried on at a peer-to-peer level. Multi Agent Systems promise to offer the natural architectural solutions to several requirements imposed by such an adaptive approach. Nevertheless, architectural design patterns commonly adopted to model agent interaction schemata do not totally fit the needs implied by highly adaptable service oriented computing scenarios.*

*In this paper, the agent oriented software engineering methodology Tropos is adopted to discuss common architectural agent patterns, and propose architectural improvements to better deal with high degrees of dynamic system adaptability. The novel architectures are illustrated through a case study in the field of ambient intelligence, namely a real Multi Agent System application implementing a pervasive museum visitors guide.*

## 1. Introduction

Complex distributed applications emerging in areas such as e-Business, e-Government, and the so called *ambient intelligence* (i.e., "intelligent" pervasive computing [7]), need to adopt forms of group communication that are deeply different from classical client-server and Web-based models (see, for instance, [14]). This strongly motivates forms of application-level peer-to-peer interaction, clearly distinct from the request/response style commonly used to access distributed services such as, e.g., Web Services adopting SOAP, XML, and RPC as communication protocol [6], [13]. The so called service oriented computing (SOC) is the paradigm that accommodates for the above mentioned highly dynamic and adaptive interaction schemata.

*Service Oriented Computing* (SOC) [13] provides a general, unifying paradigm for diverse computing environments such as grids, peer-to-peer networks, ubiquitous and pervasive computing. A *service* encapsulates a component made available on a network by a provider. The interaction between a client and a service usually follows a straightforward request/response style, possibly asynchronous; indeed, this is often the case with Web Services, which adopt XML-based message formats and common Web protocols (e.g. SOAP over HTTP) as their basis [6], [13]. Two or more services can be aggregated to offer a single, more complex service, possibly a complete business process; the activity of aggregating and coordinating services is called *service composition*.

Service-oriented computing is applicable to ambient intelligence as a way to access environmental services, e.g., accessing sensors or actuators close to a user. Multi Agent Systems (MAS) naturally accommodate for the SOC paradigm. In fact, each service can be seen as an autonomous agent (or an aggregation of autonomous agents), possibly without global visibility and control over the global system, and characterized by unpredictable/intermitted connections with other agents of the system. However, we argue that some domain specificities – such as the necessity to continuously monitor the environment for understanding the context and adapting to the user needs, and the speed at which clients and service providers come and go within a physical environment populated with mobile devices – impose new challenging system architecture requirements that are not satisfied by traditional agent patterns proposed for request/response interactions. Moreover, in ambient intelligence applications, we often need to effectively deal with service composition based on dynamic agreements among autonomous peers. In fact, group of peers may collaborate at different levels and times during the composition of complex services. This group communication styles should be used as architectural alternatives or extensions to middle agents (e.g., *matchmakers* and *brokers*), simplifying the application logic and moving context-specific decision-making from high-level applications or intermediate agents down to the agents called to achieve a goal. For these reasons, in this paper, we propose novel agent micro-level architectures, which allow for dynamic, collective, and collaborative reconfiguration of service providing schemata.

To illustrate our approach, we use the notion of *service oriented organization* (SOO). We call *Service Oriented Organization* a set of autonomous software agents that, in a given location at a given time, coordinate in order to provide a service; in other words, a SOO is a team of agents whose goal is to deliver a service to its clients. Examples of SOO are not restricted to Web Services and ambient intelligence; for instance, they include *virtual enterprises* or *organizations* [15], [8], in application area of e-Business. This paper also introduces a specific type of SOO that we call *implicit organization* (IO), since it exploits a form of group communication called *channelled multicast* [3] to avoid explicit team formation and dynamically agree on the service composition. As a reference example to illustrate our ideas, we adopt an application scenario from Peach [18], an ongoing project for the development of an interactive museum visitor's guide. Using the Peach system, users can request information about exhibits; these may be provided by a variety of information sources and media types (museum server, online remote servers, video, etc.). As well, we adopt the Tropos software design methodology [5], [2] to illustrate and compare the different agent patterns.

Tropos adopts high-level requirements engineering concepts founded on notions such as actor, agent, role, position, goal, softgoal, task, resource, belief and different kinds of social dependency between actors [5], [2], [12]. Therefore, Tropos allows for a modeling level more abstract than other current methodologies as, e.g., UML and AUML [1]. Such properties well fit with our major interest, which is in modeling environmental constraints that affect and characterize agents roles and their intentional and social relationships, rather than in implementation and/or technological issues.

The paper is organized as follows. Section 2 briefly recalls some background notions both on the Tropos methodology and on two traditional agent patterns, i.e., the matchmaker and broker patterns. Section 3 introduces an excerpt of the Peach project, adopted as a reference case to illustrate our arguments. Section 4 presents two novel micro-level agent architectures — *Service Oriented Organization* and *Implicit Organization*— that we propose to overcome some limitations of traditional patterns. Section 5 provides an example of agent role characterization and describes their dynamic aspects. Some conclusions are given in Section 6.

## 2. Tropos

The Tropos methodology [2], [5] adopts ideas from Multi Agents Systems technologies and concepts from requirements engineering through the ***i**** framework. ***i**** is an organizational modeling framework for early requirements analysis [22], founded on notions such as *actor*, *agent*, *role*, *goal*, *softgoal*, *task*, *resource*, and different kinds of social dependency between actors. In some sense, Tropos extends ***i****, since it spans five phases in both Requirements Engineering and Software Engineering [5], [2]: Early Requirements Analysis, Late Requirements Analysis, Architectural Design, Detailed Design, and Implementation.

One of the key ideas of Tropos is the adoption of the same Agent-Oriented mentalistic notions through all the phases of software engineering, from the Early Requirements Analysis —during which the human organizational setting in which the application will be used, is analyzed from its very social point of view— down to the implementation level —possibly, thought not exclusively [16], adopting an Agent-Oriented-Programming environment. Thus, the notion of *actor* is used to represent any active entity, either human or artificial, and either individual or collective, so to fit, in a uniform and homogeneous way, the representational needs of all the phases of software development life-cycle, starting at requirements, thorough design, to implementation. In particular, an actor may represent a single person or a social group (e.g., the organization running a museum) as well as a whole, complex artificial system, as, e.g., an interactive museum guide, or each of its components (both hardware and software) at different levels of granularity.

*Actors* may be further specialized as *roles* or *agents*. An *agent* represents a physical (human, hardware or software) instance of actor that performs the assigned activities. A *role*, instead, represents a specific function that, in different circumstances, may be executed by different agents —we say that the agent *plays* the role. Actors (agents and roles) are used in Tropos to describe different social dependency and interaction models. In particular, Actor Diagrams (see Figures 1, 4, and 5) describe the network of social dependencies among actors. More in detail, an Actor Diagram is a graph where each node may represent either an actor, a

goal, a softgoal, a task or a resource. Edges between nodes can be used to form paths (called *dependencies*) of the form: *depender ! dependum ! dependee*, where the *depender* and the *dependee* are actors, and the *dependum* is either a goal, a softgoal, a task or a resource. Each path between two actors indicates that the first actor (the depender) depends on the other (the dependee) for something to be attained (the dependum). The type of the dependum describes the nature of the dependency. Goal dependencies are used to represent delegation of responsibility for fulfilling a goal; softgoal dependencies are similar to goal dependencies, but their fulfillment cannot be defined precisely (for instance, the appreciation is subjective, or the fulfillment can occur only to a given extent); task dependencies are used in situations where the dependee is required to perform a given activity; and resource dependencies require the dependee to provide a resource to the depender [22], [5], [2]. As exemplified in Figure 1, actors are represented as circles[1]; dependums —goals, softgoals, tasks and resources— are represented as ovals, clouds, hexagons and rectangles, respectively. Goals and softgoals introduced with Actor Diagrams can be further detailed and analyzed by means of the so called Goals Diagrams [2], in which the rationale of each (soft)goal is described in terms of goal decompositions, means-end-analysis and the like, as, e.g., in Figure 6. Actor and Goal Diagrams are adopted from Early Requirements Analysis to Architectural Design.
In particular, in this paper, we are concerned on using Actor and Goal Diagrams for the Architectural Design, to describe Architectural solution we have devised, and then implemented, in the context of the Peach project.

## 2.1. Architectural Design Patterns in Tropos

In Software Engineering, a common technique to manage the complicated nature of the development of architectural designs of complex systems is the reuse of development experience and know-how, e.g., by adopting architectural styles and design patterns [9], [17] during macro- and micro-level architectural design, respectively [20]. In particular, design patterns describe problems commonly found in software designs, and prescribe adaptable solutions —at the architectural as well as at the detail design and implementation level— for them, so to ease the reuse of these solutions.

[1] In this paper, we do not adopt any graphical distinction between actors, agents, and roles: when needed, we clarify it in the text.

Tropos uses design patterns as a technique to define the micro-level architectural design details, adopting a metaphor inspired by research on cooperative and distributed real-world social structures, namely, social patterns[5],[12].

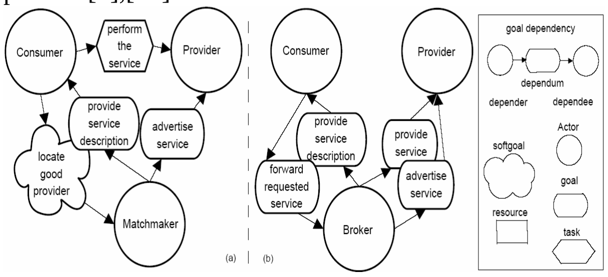

**Figure 1**. a) Matchmaker agent pattern; b) Broker agent pattern, depicted by means of the Tropos Actor Diagrams.

For example, Figure 1 shows two important design patterns, described by means of Tropos actor diagrams. Both the patterns satisfy the need of mediation between a consumer and (one or more) providers. Figure 1.a shows a Matchmaker pattern. Here, the matchmaker searches for the providers that can match the requested service. This is done on the basis of the capabilities advertised by the providers themselves, e.g., by means of a subscription phase that carries out both the service description and the service provider information. Then the consumer itself will directly interact with the provider(s) to obtain the needed service. That is, in terms of Tropos dependencies (as shown in Figure 1.a), the Consumer depends on the Matchmaker to locate good provider, and on the Provider to provide service. The Matchmaker depends on the Provider to advertise service. The advertise service dependency is similarly present also in the Broker pattern (see Figure 1.b), but here the dependency to provide service is on Provider from Broker, since Consumer will depend on the latter to have the requested service delivered (forward requested service), that is, Broker plays an intermediary role between Provider and Consumer.

Both of these (well know, in the agent literature) patterns can be adopted to accommodate architectural needs deriving from a SOC application scenario, since both allows for the discovering of service providers in a dynamically changing environment without requiring, at design time, defined awareness of them. In particular, the matchmaker pattern plays a key role to allow the whole system for searching and matching capabilities, e.g., see [19], as required in a peer to peer scenario.

At the same time, notice that an ever-changing environment, in terms of actors and their information flow, may dreadfully increase the interaction between the system actors and external devices (e.g., the information Consumer). Therefore, an important architectural requirement consists in limiting as much as possible such interactions, e.g., by allowing the system for a more pro-active behavior in order to take decisions that can be taken automatically with reasonable confidence. Such a kind of a requirement may justify a preference for the broker pattern (see, e.g., [11] and next section).

## 3. The Case Study

The Peach project [18] focuses on the development of a mobile museum visiting guide system. The whole system is a MAS, which has been developed following the Tropos methodology. Indeed, agents perform their actions while situated in a particular environment that they can sense and affect. More specifically, in the typical museum visiting guide scenario of the Peach system, a user (the visitor) is provided with several environmental interaction devices. The most evident to her is a Personal Digital Assistant (PDA).The user is walking around the museum and using the PDA for requesting and receiving presentations about the exhibits. Other devices include a set of passive localization hot-spots, based on triangularization of signals coming from the PDA, and (pro) active stationary displays of different sizes and with different audio output quality.

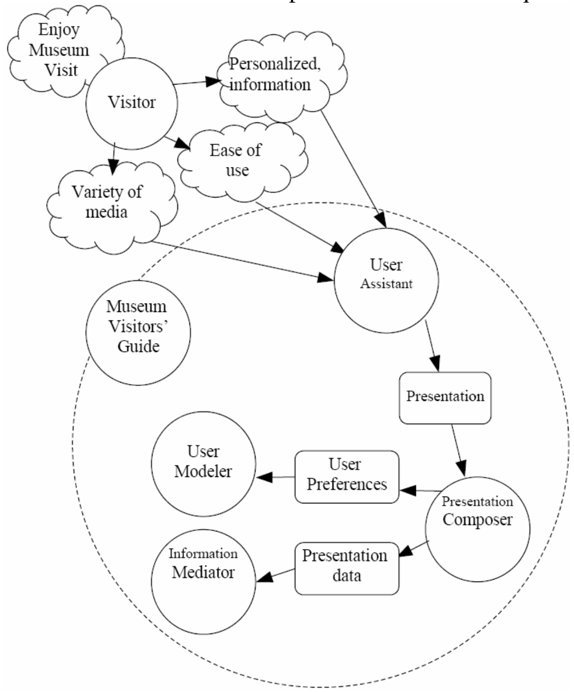

**Figure 2.** A fragment of the system architecture.

A snapshot of the Peach system high level architecture is sketched out in the Tropos Actor Diagram of Figure 2.Here, we have the User Assistant, which provides the primary interface for the user and it acts on her behalf to require presentations (services); the Presentation Composer, capable of generating various types of presentations (audio, video, personal or generic etc.); the Information Mediator, capable of providing the Presentation Composer(s) with relevant information for presentation generation.

Given this environment, we can consider the following user-system interaction scenario:

**Example 1** (*explicit communication*).
A museum visitor requests some information during her tour by using her PDA. To this end, the User Assistant sends a presentation request to the Presentation Composer that, on its turn, is responsible to contact the User Modeler and the Information Mediator, and it properly combines the services they provide. In particular, by adopting a User Modeler, different presentations may be generated accordingly to various user characteristics (e.g., age, mother tongue, etc.).

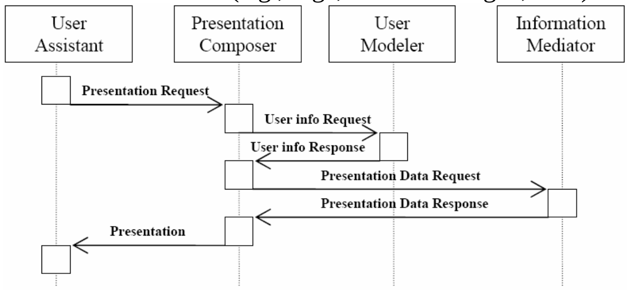

**Figure 3.** Overview of the actor interactions.

Still using Tropos, we can get to Detailed Design and model the communication dimension of the system actors. To this end, Tropos adopts AUML [1] interaction diagrams [2]. The communication diagram of Figure 3 presents the sequence of events from the time a request for presentation is issued until the presentation is presented to the user. It is, here, interesting noticing that a Broker pattern is instantiated among the User Assistant (in the role of Consumer), the Presentation Composer (as Broker), and the User Modeler and Information Mediator (as Service Providers). In other terms, as foreseen, we may here take advantage from the design patterns introduced before, to map the high level Peach architecture on one of them (the broker one), so to ease the next detail design and implementation phases. It is now worth noticing that, though the User Assistant issued by the visitor to interact with the system, this does not necessarily mean that its functionalities (entirely) resides on the PDA. Similarly, also other software components may be distributed (or not) on several HW units. Indeed, here, we are not interested in the concrete instantiations of the actors listed above, but, in the *roles* they play. Thus, the User Assistant, the Presentation Composer, and the User Modeler represent roles—in Tropos terms— which may be played by different agents, i.e., there may be several different information mediators(for video, audio, text, pictures, animation, local and remote information sources and more), there may be several user assistants with different capabilities (PDAs, desk-top stations, wall mounted large plasma screens and more), and there may also be several different user modelers implementing various techniques to get users' profiles. This implies that a team of cooperating agents (possibly "the best" according to some criteria) may be required to be formed, depending on the service request and on contextual parameters, such as theuser's location and profile and the availability of resources. This call for a carefully rethinking of our scenario in terms of dynamic configuration.

## 3.1. Discussion on Dynamic Configuration

As noticed above, our ambient intelligence scenario naturally calls for some kind of *dynamic configuration*.

More in general, we can deal with dynamic configuration issues with two different —sometimes complementary—approaches. The first is by a *discovery and selection* process driven by the service requester (the User Assistant in our example). This may happen either off-line (when all possible configurations can be anticipated, e.g., in closed environments) or at run-time, in particular in mobile or highly dynamic situations, with the support of technologies such as peer-to-peer networking, as in Jini (see http://www.jini.org/)and UPnP (Universal Plug and Play Device Architecture,http://www.upnp.org), or based on OWL-S ontologies for service descriptions(see http://www.daml.org/services/). This will correspond to a matchmaker pattern.

Alternatively, the configuration issue is left to the service providing agents, rather than to the requester; in other words, a service request has to be posted to a well known location, and who actually takes care of it is transparent to the requester. In this case we have a broker pattern, as in Example 1.

Thus, common agent patterns can be properly adopted to deal with the issue of dynamic configuration (e.g., see [12],[20], [19], [11]). In particular, the traditional patterns introduced in Section 2.1 (matchmaker and broker) are well suited to support the two different approaches described above, respectively; they capture most common solutions found in the literature, possibly by their combination at different levels of abstraction.

However, we argue here that these patterns are not able to capture some vital architectural requirements that may arise from ambient intelligence scenarios —in particular, our scenario. To motivate our assertion, let us consider a variant of Example 1:

**Example 2** (*implicit communication*).
While walking around, the user approaches some presentation devices that are more comfortable and suitable to handle presentations than her PDA, e.g., in terms of pixel resolution and audio quality. We want the User Assistant to autonomously negotiate the most convenient presentation with their locally available peers (presentation devices), on behalf of its human owner. There are several different Presentation Composers capable to generate videos, text, animated explanations, audio, and so on. In turn, each Presentation Composer relies on different Information Mediators to provide the information required for presentation generation. We want Presentation Composers to proactively propose their best services (in terms of available or conveniently producible presentations) to the User Assistant. To complete this picture, we expect agents to be added and removed from the environment at any time, because of local failures (including very common wireless communication problems) or to cope with changing work loads when new types of resources become available. This implies that services can be "dynamically validated" also accordingly to the ever-changing environment and user location.

The scenario presented above calls for architecture flexibility in terms of *dynamic group reconfiguration*. More precisely, we propose here to adopt the metaphor of a *Service Oriented Organization* (SOO). In other terms, we may say that the service providing agents form ad-hoc *Service Oriented Organizations* whose goal is to provide services for a specific role; in our scenario, examples of organizations include the abovementioned Presentation Composer, the Information Mediator, and the User Modeler. Each SOO is characterized by members that collaborate at different levels and times during the service providing process. After a goal is satisfied, an organization may be dissolved and a new one formed —possibly including different agents, as long as they play the correct role—when a new request comes along. This approach is typically implemented with technologies that add a middle layer among requesters and service providers.

In a SOO, indeed, multiple processes should happen concurrently (e.g., negotiation of the best device while composers generate alternative presentations based on the information available at the moment, all this while the user moves and the context changes) and cannot be easily reduced to a set of agent selection mechanisms, the goal of which is to establish the "best" team *before* service execution. In a plausible execution trace, the User Assistant starts an interaction session that triggers the involvement of a group of system actors all playing the role of Presentation Composer, which in turn triggers the involvement of a group of system actors all playing the role of Information Mediator. Each Presentation Composer, also, relies on User Modelers to know the user profile to correctly buildup a user-tailored presentation. While the output and timing of a service provided by a single individual agent may quickly become irrelevant or impossible to complete because of the evolution of the context (e.g., the user changes location, database crashes, a new user enters the same room), relying on a group of agents (an organization) providing the service—considered as a single entity— may, instead, allow for the "right" solution still to be available.

Traditional mediation patterns allow for intentional relationships and request/response communication protocols among individual agents only, and not among groups [10], [11], [12],[20]. Moreover, they adopt a centralized or semi-centralized approach; as shown in Figure 1, some of the dependencies of these patterns, e.g., the advertise service, forces the system actors to rely on the mediator (matchmaker or broker) to keep an up-to-date picture of the situation. This may be hard to manage in real time, introducing single points of failure, and anyhow it does not care for the evolution of the context once requests have been allocated to specific providers. Similar problems arise with direct agent-to-agent communication (e.g., peer-to-peer) and request/response based communication styles, as those presented in [19]. To overcome these issues, we foresee agents exploiting some forms of group-wide "implicit communication", which simplifies the maintenance of a shared view of the context and allows to dynamically decide how to best satisfy a request even while the situation evolves.

During the Peach project, it was clear since the early stages that a SOO approach was needed. To this end, we faced the necessity of as evolving and improving the idea of matchmaker and broker, so to overcome the above mentioned limits, as briefly discussed next. As a result, we come up with the micro level architectures we present in the next section.

## 4. Collaboration Architectures for Group-Based Interaction

In this section, we overview two micro-level architectures aiming at tackling some of the challenges introduced in the previous section.

A first architecture we propose, that we name *Service Oriented Organization* (SOO), adopts and evolves the matchmaker pattern, in which the requirement of "service advertising" is removed in favor of a more dynamic and proactive "service proposal" schema. This paves the way for instantiating amore specific case of architectural schema, which is named *Implicit Organization* (IO). This latter does not require of a distinguished agent having the role of mediator among the different "organization forming" service providers. Thus, it allows for a higher level of run-time re-configurability and adaptability, which also leads to a higher system robustness and local failure tolerance. This architecture finally fits the systems needs deriving from the scenario depicted in Example 2.

### 4.1. The Service Oriented Organization

Our notion of *Service Oriented Organization* evolves the idea of team of agents [21] by considering the formation of group of agents characterized by their common role (team) specifically oriented at providing a (possibly complex) service. In this sense we prefer to speak about *Service Oriented Organizations*. Consequently, we define a new micro-level agent architecture —called by us *Service Oriented Organization*(SOO) architecture, and illustrated in Figure 4— which adopts the matchmaker pattern of Figure 1.a. Here, the Matchmaker role corresponds to the joint collaboration of the two roles: Providers Organizer and Initiator, as shown in Figure 4 within the dotted boundary (a). The Initiator acts as a proactive alter goof the Consumer, so that, on the basis of its acquaintances and models of the user, it can autonomously and proactively negotiate services on behalf of the Consumer, and propose them to it without requiring, from the Consumer, complete awareness of the ongoing process. This means that, in the context of the Peach architecture, both the Consumer and the Initiator are part of the User Assistant, where the first represent the very end-user GUI, and the second its "intelligent" part[2].

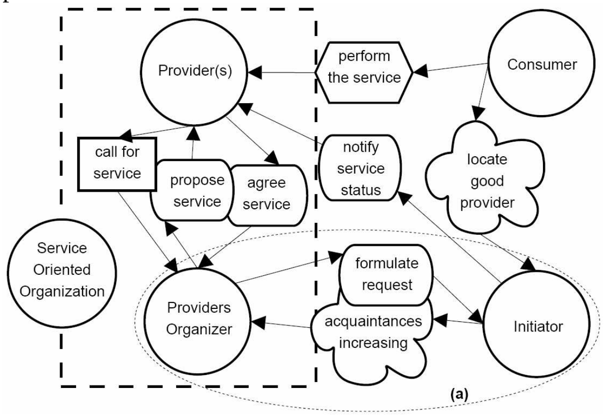

**Figure 4.** Actor Diagram for the *Service Oriented Organization* micro-level architecture.

The dependencies between Consumer and Organization Matchmaker (or, more specifically, Initiator) and between Consumer and Provider(s) are as in the matchmaker pattern. However, in a SOO, there is no longer an advertise service goal dependency between Organization Matchmaker and Provider(s). In fact, our scenario call for dynamic group reconfiguration, which cannot be provided on the basis of a pre-declared and centrally recorded set of service capabilities. The solution we propose is based on agents proactively and dynamically proposing themselves to perform services, in response to a call for service, which is delivered by means of a group communication technological infrastructure. A group communication infrastructure automatically broadcast messages that may be heard by all the agents that are listening on the specific channel associated with the topic of interest; examples include publish/subscribe systems, distributed event systems, and our own LoudVoice (described in Sec. V) based on multicast IP.

Thus, Provider(s) depends on Organization Matchmaker, or, more specifically, on Providers Organizer for a call for service to be broadcasted. That is, the Providers Organizer simply sends its service request to the group and waits that some provider propose service. Finally, Provider(s) depend on the Providers Organizer for the final agreement on service provision (goal agree service).

[2] It is the case to recall that, in general, each high-level architectural agent, as the User Assistant, may be composed by several role-playing components —in this case the Consumer and the Initiator. Moreover, it may also be the case —as it, indeed, happens in this case— that the different components run on different hardware, since the actor concept is a logical, and not physical, abstraction.

It is here worth noticing that the Initiator not only acts as interface towards Consumer —by interpreting Consumer's requests— but also proactively proposes unsolicited services, for instance, on the basis of Consumer's profile and previous interaction history. To this end, the Initiator depends on the Providers Organizer to get new acquaintances about Provider(s) and their services, while Providers Organizer depends on the Initiator to formulate request. In this way, we can drop the dependency provide service description between Matchmaker and Consumer, as instead foreseen in the traditional matchmaker pattern. Finally, Initiator requires that Provider(s) timely notify service status in order to propose only active services.

## 4.2. The Implicit Organization

The architecture discussed above copes with the requirements of a proactive behavior. Nevertheless, it does not fully satisfy yet the specification introduced by [4] for the so called *Implicit Organization* (IO). An IO is a SOO where there is no predefined (or design-time defined) Providers Organizer role. Instead, each of the providers participating at the SOO may, at run-time, assume the role of *organizer*, as a result of an *election* process for the selection of the organizer [4] (here called Winner). The term 'implicit' highlights the fact that there is no group formation phase —since joining an organization is just a matter of tuning on a channel, without explicit awareness about the organization or its members. All the organization members play the same role but they may do it in different ways; just as a particular case, we may have redundancy (as required in fault tolerant and load balanced systems), where agents happen to be perfectly interchangeable. Instead, in a SOO, each time a request (formulate request) is broadcasted, the Providers Organizer plays a central role in order to interpret and reformulate the request (call for service) for the other providers. Then, it coordinates all the offers (goal propose service) selecting the best ones to be forwarded to the Initiator.

By contrast, as illustrated by Figure 5, inside the IO no control role is required, since each member receives (overhears) the initial request and immediately decides by itself whether to participate at achieving the organizational goal established by the request. Thus, an IO is composed only by all the Providers which autonomously and proactively agree to participate at the IO itself (see Figure 5). Moreover, in the IO architecture there is no call for service dependency. Instead, each IO member directly depends on the Initiator for the goal formulate request. To highlight the collective nature of this dependency from each member of the to-be-formed implicit organization, in Figure 5the dependency formulate request is among the whole Implicit Organization and the Initiator.

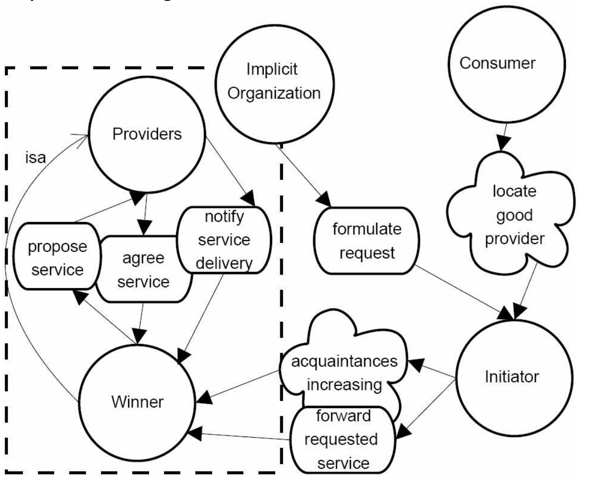

**Figure 5.** Actor Diagram for the *Implicit Organization* (IO) pattern.

Among all the Providers of the Implicit Organization there is a (see the "isa" link) run-time selected —accordingly with a defined selection policy [4]– Winner, which plays the role of coordinator, and which may change from time to time at every IO formation. For example, a very simple coordination policy can be established by assuming the winner as the member that first answers to the service request. At this level of design, we do not care about the details concerning the winner election process; anyway, we notice that several negotiating policies can be adopted, like, for example, the *Plain Competition* policy, the *Simple Collaboration* policy, the *Multicast Contract Net* policy, or the *Master-Slave* policy, as described in [4].

## 5. Supporting Implicit Organization

The two agent-oriented architectures, *Service Oriented Organization* and *Implicit Organization* presented so forth, have been experimented within the Peach project in order to build an interactive, pervasive, museum guide system. As mentioned, our patterns require a group communication infrastructure. To this end, we adopt the *LoudVoice* [4] experimental communication infrastructure based on channeled multicast and developed at our institute. Specifically, LoudVoice uses the fast but inherently unreliable IP multicast —which is not major limitation in our domain, since the communication media in use are unreliable by their own nature. However, we had to deal with message losses and temporary network partitions by carefully crafting protocols and using time-based mechanisms to ensure consistency of mutual beliefs within organizations.

### 5.1. Agent role characterization

Analyzing agent roles means figuring out and characterizing the main capabilities (e.g., internal and external services) required to achieve the intentional dependencies already identified during the agent-based architectural design phase. Note that, a capability (or skill) is not necessarily justified by external requests (like a service), but it can be an internal agent characteristic, required to enhance its autonomous and proactive behavior. To deal with the rationale aspects of an agent at 'design time', that is, in order to look inside and to understand how an agent exploits its capabilities, we adopt the *Goal Modeling Activity* of Tropos [2]. In Figure 6, we adopt the *means-end* and *AND/OR decomposition* reasoning techniques of the Goal Modeling Activity [2], [5]. Means-end analysis aims at identifying goals, tasks, and resources that provide means for achieving a given goal. AND/OR decomposition analysis combines AND and OR decompositions of a root goal into sub goals, modeling a finer goal structure. Notice that, we have modeled every agent capability as a goal to be achieved.

For the sake of briefness, here, we only briefly consider our IO architecture. According to Figures 6 and 5, each time Initiator formulates a request, the Winner among the Providers tries to achieve its main goal cope with the request relying on its three principal skills: provide service, deal with fipa-acl performatives, and support organizational communication. This root goal success depends on the satisfaction of all the three subordinate goals (AND-decomposition). For the sake of simplicity, Figure 6 does not consider Initiator and its intentional relationships. An adequate organizational communication infrastructure is used to enhance the system actor autonomous and proactive behavior by means of group communication based on channeled multicast [3] (see the goal provide channeled multicast) that allows messages to be exchanged over open channels identified by topics of conversation. Thus, a proper structuring of conversations among agents allows every listener to capture its partner intentions without any explicit request, thanks to its agent introspection skill (see goal allow agents introspection). Indeed, each actor is able to overhear on specific channels every ongoing interaction; hence, it can choose the best role to play in order to satisfy a goal, provide a resource, perform a task, without any external decision control, but only according to its internal beliefs and desires.

Exploiting the provide channeled multicast ability, each actor can decide by itself what channels to listen to, represented by the tasks discover channels and maintain a channel list. These communication mechanisms well support group communication for service oriented and implicit organizations composed by members with the same interests or skills.

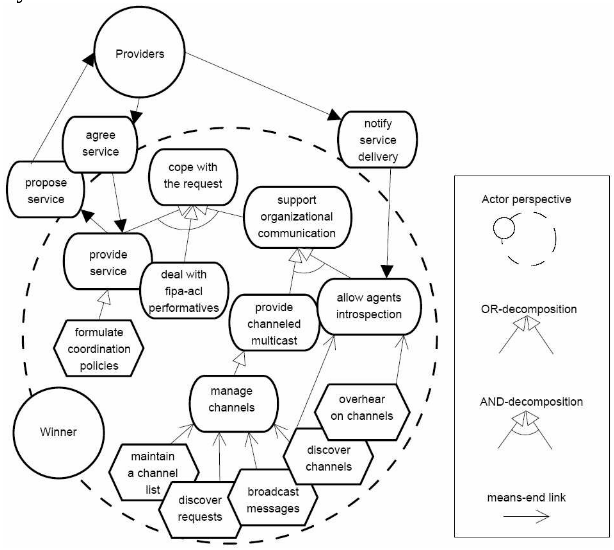

**Figure 6**. Goal Diagram for the Winner role (service provider) characterization by means of its capabilities.

## 5.2. Group Communication: dynamics

As described earlier, the museum visitor guide system is composed of several different types of agents. Rather than by individual agents, most components are formed by a group of coordinated agents. Modeling communication dynamics in this case requires the representation of implicit organizations, which cannot be done by a regular AUML communication diagram, as presented, e.g., in [2], [5]. Therefore, in Figure 7, we propose a new type of Interaction Diagram that deals with the group communication features required by the scenario introduced with Example 2. Here, the shaded rectangles represent the implicit organizations, and contain their internal communications. Requests sent to an organization are presented as arrows terminating in a dot at the border of the organization; the organization reply is presented by an arrow starting from a dot on the organization border. Obviously, we consider an asynchronous message-based communication model.

In the example diagram of Figure 7, the request for presentation is initiated by a certain User Assistant on behalf of a specific user. The request is addressed to the Implicit Organization of Presentation Composers. Presentation Composers have different capabilities and require different resources. Hence, every Presentation Composer may require user information to the User Modeler (only a single agent in this example) and presentation data (availability, constraints, etc.) from the Implicit Organization of Information Mediators. In turn, the Implicit Organization of Information Mediators holds an internal conversation. Each member suggests the service it can provide. The “best” service is selected and returned, as a group decision, to the requesting presentation composer. At this stage, the presentation composers request additional information, regarding the availability of assistants capable to show the presentation being planned, to the Implicit Organization of User Assistants. When all the information has been received, the implicit organization of presentation composers can reason and decide on the best presentation to prepare. This will be sent from the composers as a group response to the selected User Assistant.

Of course, Figure 7 —which, at present, represents just a very preliminary tentative to properly represent IO internal communication— only shows a possible —prototypical—instance of the run-time interaction evolution, and can not be seen as a prescriptive, design-time fixed communication schema. In fact, the very nature of Implicit Organizations, which are characterized by very flexible and unpredictable run-time configurations, does not allow to *freeze* the communication behavior inside each IO, and, indeed, not even the IO composition itself.

## 6. Conclusions

Ambient intelligence scenarios characterized by service oriented organizations, such as our active museum guide, generates new architectural requirements. Such scenarios, where group of agents collaborate at different levels and times during the service providing process, cannot be modeled, as easily or as well, by using traditional agent patterns. In fact,

traditional request/response communication protocols are not appropriate to cope with service negotiation and aggregation that must be 'dynamically validated’, since the environment conditions and the user location are quickly changing.

For such reasons, we propose, in this paper, two specific micro-level architectures (Service Oriented Organization and Implicit Organization) partially derived from traditional architectural patterns for agent pairing [11], [12]. Specifically, we adopt the agent oriented software development methodology Tropos [2], [5], to effectively figure out the new requirements.

Thus, we have been able to capture important aspects of ambient intelligence requirements and to build up novel agent micro-level architectures, more adequate to them. In particular, as future work, we are trying to apply our new micro level architectures to other complex systems in order to prove their extensive applicability, so to properly rank them as reusable agent patterns. To this end, we also need to further investigate on the syntax an semantics of the extension of AUML Interaction Diagrams, we preliminary proposed in Section 5.2.

## 7. References

[1] B. Bauer, J. P. Muller, and J. Odell. Agent uml: A formalism for specifying multiagent software systems. International Journal of Software Engineering and Knowledge Engineering, 11(3):1–24, 2001.

[2] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An agent-oriented software development methodology. Journal of Autonomous agents and Multi-agent Systems (JAAMAS), 8(3):203 –236, 2004.

[3] P. Busetta, A. Don´a, and M. Nori. Channeled multicast for group communications. In Proceedings of the first international joint conference on Autonomous agents and multiagent systems, pages 1280–1287. ACM Press, 2002.

[4] P. Busetta, M. Merzi, S. Rossi, and F. Legras. Intra-role coordination using group communication: A preliminary report. In F. Dignum, editor, Advances in Agent Communication, LNAI 2922. Springer Verlag, 2003.

[5] J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven information systems engineering: The tropos project. Information Systems (27), pages 365–389, Elsevier, Amsterdam, The Netherlands, 2002.

[6] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana. The next step in web services. Commun. ACM, 46(10):29–34, 2003.

[7] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J.-C. Burgelman. Scenarios for ambient intelligence in 2010. Technical report, Information Society Technologies Programme of the European Union Commission (IST), Feb. 2001. http://www.cordis.lu/ist/.

[8] U. J. Franke. Managing Virtual Web Organizations in the 21th century: Issues and Challenges. Idea Group Publishing, Pennsylvania, 2001.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns:Elements of reusable Object-Oriented Software. Addison-Wesley, 1995.

[10] S. Hayden, C. Carrick, and Q. Yang. Architectural design patterns for multiagent coordination. In Proc. of the 3rd Int. Conf. on Agent Systems (Agents'99), 1999.

[11] M. Klusch and K. Sycara. Brokering and matchmaking for coordination of agent societies: A survey. In A. Omicini, F. Zambonelli, et al., editors, Coordination of Internet Agents: Models, Technologies, and Applications, pages 197–224. Springer-Verlag, Mar. 2001.

[12] M. Kolp, P. Giorgini, and J. Mylopoulos. A goal-based organizational perspective on multi-agents architectures. In Proceedings of the Eighth International Workshop on Agent Theories, architectures, and languages(ATAL-2001), 2001.

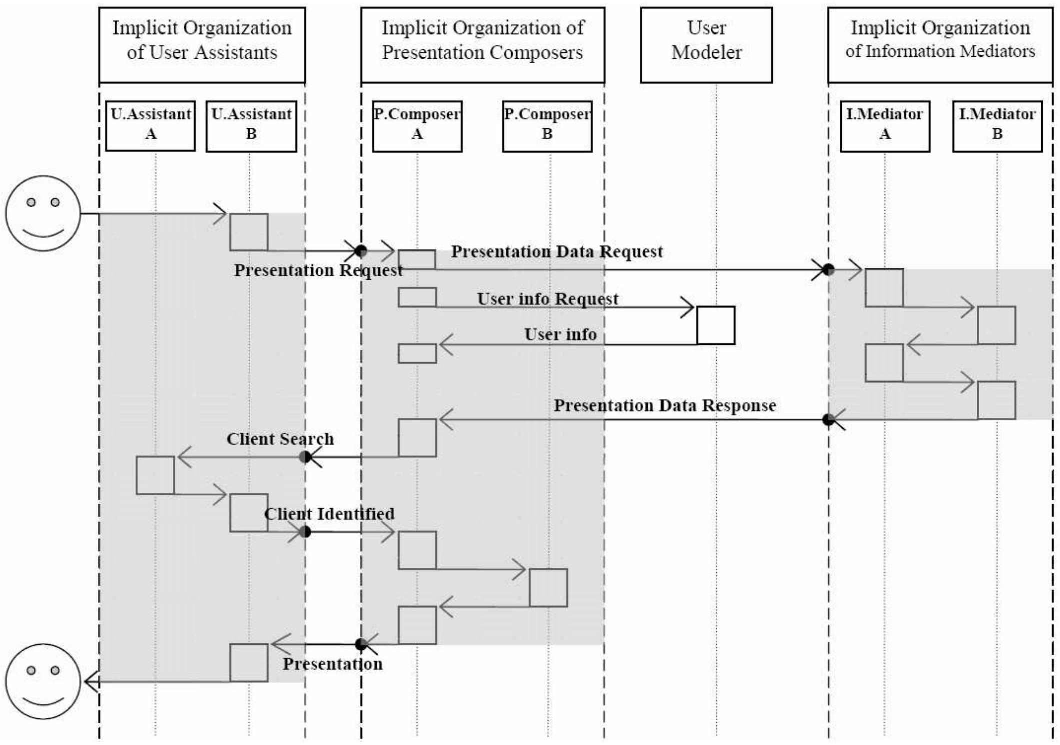
Implicit Organization of User Assistants
Implicit Organization of Presentation Composers
User Modeler
Implicit Organization of Information Mediators
U.Assistant A
U.Assistant B
P.Composer A
P.Composer B
I.Mediator A
I.Mediator B
Presentation Request
Presentation Data Request
User info Request
User info
Presentation Data Response
Client Search
Client Identified
Presentation

**Figure 7.** Interaction of organizations.

[13] M. P. Papazoglou and D. Georgakopoulos. Introduction to the special section on Service Oriented Computing. Commun. ACM, 46(10):24–28, 2003.

[14] L. Penserini, L. Liu, J. Mylopoulos, M. Panti, and L. Spalazzi. Cooperation strategies for agent-based p2p systems. WIAS: Web Intelligence and Agent Systems: An International Journal, 1(1):3–21, 2003.

[15] L. Penserini, L. Spalazzi, and M. Panti. A p2p-based infrastructure for virtual-enterprise's supply-chain management. In Proc. of the Sixth Int Conference on Enterprise Information Systems (ICEIS-04). vol.4, pp 316-321, 2004.

[16] A. Perini, P. Bresciani, P. Giorgini, F. Giunchiglia, and J. Mylopoulos.Towards an Agent Oriented approach to Software Enginee ring. In A. Omicini and M. Viroli, editors, WOA 2001 – Dagli oggetti agli agenti: tendenze evolutive dei sistemi software, Modena, Italy, Sept. 2001. Pitagora Editrice Bologna. Published on CD media.

[17] M. Shaw and D. Garlan. Software Architecture: Perspectives on an Emerging Discipline. Prentice-Hall, 1996.

[18] O. Stock and M. Zancanaro. Intelligent Interactive Information Presentation for Cultural Tourism. In Proc. of the International CLASS Workshop on Natural Intelligent and Effective Interaction in Multimodal Dialogue Systems, Copenhagen, Denmark, 28-29 June 2002.

[19] K. Sycara, S. Widoff, M. Klusch, and J. Lu. Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. AutonomousAgents and Multi-Agent Systems, 5(2):173–203, 2002.

[20] Y. Tahara, A. Ohsuga, and S. Honiden. Agent system development method based on agent patterns. In Proc. of the 21st Int. Conf. on Software Engineering (ICSE'99). IEEE Computer Society Press, 1999.

[21] G. Tidhar, A. S. Rao, and E. A. Sonenberg. Guided team selection. In Proceedings of the Second International Conference on Multi-Agent Systems, Kyoto, Japan, December 1996. AAAI Press.

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