Agent Patterns for Ambient Intelligence

Paolo Bresciani, Loris Penserini, Paolo Busetta, and Tsvi Kuflik

ITC-irst
via Sommarive 18
I-38050 Trento-Povo, Italy
{bresciani,penserini,busetta,kuflik}@itc.it

Abstract. The realization of complex distributed applications, required in areas such as e-Business, e-Government, and ambient intelligence, calls for new development paradigms, such as the Service Oriented Computing approach which accommodates for dynamic and adaptive interaction schemata, carried on on a per-to-peer level. Multi Agent Systems offer the natural architectural solutions to several requirements imposed by such an adaptive approach.
This work discusses the limitation of common agent patterns, typically adopted in distributed information systems design, when applied to service oriented computing, and introduces two novel agent patterns, that we call *Service Oriented Organization* and *Implicit Organization Broker* agent pattern, respectivelly. Some design aspects of the Implicit Organization Broker agent pattern are also presented. The limitations and the proposed solutions are demonstrated in the development of a multi agent system which implements a pervasive museum visitors guide. Some of the architecture and design features serve as a reference scenario for the demonstration of both the current methods limitations and the contribution of the newly proposed agent patterns and associated communication framework.

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]), needs to adopt forms of group communication that are deeply different from classical client-server and Web-based models (see, for instance, [13]). 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, 12]. The so called service oriented computing (SOC) is the paradigm that accommodates for the above mentioned more dynamic and adaptive interaction schemata.

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

P. Atzeni et al. (Eds.): ER 2004, LNCS 3288, pp. 682–695, 2004.

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, because group of peers collaborate at different levels and times during the *service providing process life-cycle*, analogously to the product life-cycle process introduced by Virtual Enterprise scenarios [14]. 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 patterns, which allow for dynamic, collective, and collaborative reconfiguration of service providing schemata.

To illustrate our approach, we use the notion of service oriented organization. We call *service oriented organization* (SOO) 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* [14, 8], the name of which reflect the application area in which they have been adopted, i.e., e-Business. As well, this paper focuses on a special type of SOO that we call *implicit organization broker* (IOB), since it exploits a form of group communication called *channelled multicast* [3] to avoid explicit team formation and dynamically agree on the service composition. We will compare SOO and IOB to traditional agent patterns based on brokers or matchmakers. As a reference example to illustrate our ideas, we adopt an application scenario from Peach [15], an ongoing project for the development of an interactive museums 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, 11]. 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.

Section 2 briefly recalls some background notions on Tropos, on service oriented computing, and on agent patterns. Section 3 describes and discusses an excerpt of the Peach project, adopted as a reference case to illustrate our arguments. Section 5.1 tries to overcome some limitations of traditional patterns by proposing two new agent patterns: the *Service Oriented Organization* and the *Implicit Organization Broker*. Section 5.2 aims at justifying group communication as fundamental to effectively deal with the proposed patterns, providing a rationale view and describing dynamic aspects. Some conclusions are given in Section 6.

2 Background

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 [18], founded on notions such as *actor*, *agent*, *role*, *goal*, *softgoal*, *task*, *resource*, and different kinds of social dependency between actors. Actors represents any active entity, either individual or collective, and either human or artificial. Thus, an actor may represent a person or a social group (e.g., an enterprise or a department) or an 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, 3, and 4) describe the network of social dependencies among actors. An Actor Diagram is a graph, where each node may represent either an actor, a goal, a softgoal, a task or a resource. Links between nodes may be used to form paths like: *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 one actor depends on the other for something (represented by the dependum) so that the former may attain some goal/softgoal/task/resource. In other terms, a dependency describes a sort of "agreement" between two actors (the depender and the dependee), in order to attain the dependum. The depender is the depending actor, and the dependee the actor who is depended upon. 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. 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 5.

Tropos spans four phases of Requirements Engineering and Software Engineering activities [5, 2]: Early Requirements Analysis, Late Requirements Analysis, Architectural Design, and Detailed Design. Its key premise is that agents and goals can be used as fundamental concepts for all the phases of the software development life cycle. Actor and Goal Diagrams are adopted from Early Requirements Analysis to architectural design. Here, we use them to describe the agent patterns we are interested in.

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

Service oriented computing. *Service Oriented Computing* (SOC) [12] 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 normally follows a straightforward request/response style, possibly asynchronous; this is the case with Web Services, which adopt SOAP, XML, and RPC [6, 12] as communication protocol. Two or more services can be aggregated to offer a single, more complex, service or even a complete business process; the process of aggregation is called *service composition*.

As already noticed, MAS naturally accommodate for the SOC paradigm. Since each agent in a MAS may be either an arbiter or an intermediary for the user's requested service, two common agent patterns that appear to be appropriate are the *matchmaker* and the *broker* (see, e.g., [10, 16]).

Agent patterns for SOC. To accommodate the different settings and agents that can be involved, and with the different roles that – from time to time – can be played by each agent, a *pattern based* approach for the description and design of the MAS architectures for SOC systems can be adopted. An agent pattern can be used to describe a problem commonly found in MAS design and to prescribe a flexible solution for that problem, so to ease the reuse of that solution [11, 17, 9]. The literature on Tropos adopts ideas from social patterns [5, 11] to focus on social and intentional aspects that are recurrent in multi-agent or cooperative systems. Here, we adopt Actor and Goal Diagrams to characterize MAS design patterns, focusing on how the goals assigned to each agent[2] are fulfilled [2, 11], rather than on how agents communicate with each other. In the very spirit of Tropos, which naturally carries out the importance of analyzing each problem at a high abstraction level, allowing to reduce and easily manage at 'design time' the system components complexity, we aim at enhancing the reuse of design experience and knowledge by means of the adoption of agent patterns.

In our context, such patterns have to cope with the important issue of locating information/service providers, which is an architectural requirement. Indeed, as also investigated in [13], such a requirements strongly affect coordination issues in decentralized (pure) peer-to-peer scenarios. Thus, to support the peer-to-peer scenario, the matchmaker agent pattern (see Figure 1a) play a key/centric role in order to allow the whole system for the searching and matching capabilities, e.g., see [16].

At the same time, the focus on the *service providing process life-cycle* puts the consumer in the center, and when the consumer demands novel services the system architecture should provide them without overwhelming her with additional interactions. Moreover, in a decentralized scenario, it may have several local failures may happen, when trying to locate new services; hence, a huge number of interactions, before reaching the related provider, are possible. Of course, the reduction of the interaction complexity decreases the customer overload. Such a requirement calls for a broker pattern too, as detailed in Figure 1b (e.g., see [10]).

[2] Indeed, accordingly with the Tropos terminology, we should speak about roles, but we drop, here, this distinction, to ease the reading of the diagrams.

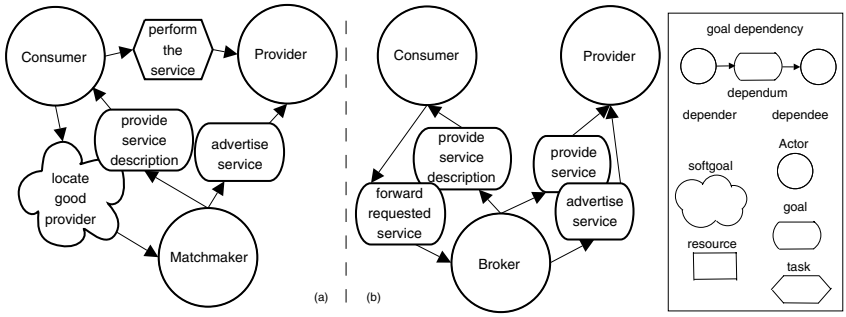

Fig. 1. a) Matchmaker agent pattern; b) Broker agent pattern, depicted by means of the Tropos Actor Diagrams.

The Tropos diagram, of Figure 1.a, shows that each time a user's information/service request arrives[3], Consumer depends on Matchmaker to locate good provider. On the contrary, Figure 1.b shows that Consumer depends on Broker in order to forward requested service, that is, Broker plays an intermediary role between Provider and Consumer. In essence, both Broker and Matchmaker depend on Provider to advertise service(s). Namely, the two patterns skills consist of mediating, among both consumers and providers, for some synergic collaborations to satisfy global goals. In particular, Matchmaker lets Consumer directly interact with Provider, while Broker handles all the interactions between Consumer and Provider.

3 A Reference Scenario

The Peach project [15] 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 Peach museum visiting guide scenario, a user (the visitor) is provided with several environmental interaction devices. The most evident to her is a personal hand-held mobile I/O device, namely a PDA. Other devices include: i) passive localization hot-spots, based on triangularization of signals coming from the PDA; ii) (pro)active stationary displays of different sizes and with different audio output quality. Depending on the dimensions, the displays may be used to deliver visual/audio information (images and/or motion pictures possibly with audio comments; text) to a single user at a time, or to a group of users.

Given this environment, let us start from the following possible user–system interaction scenario:

Example 1 (explicit communication). A museum visitor requests some information during her tour by using her mobile device. To deliver on such a goal, the PDA con-

[3] In the context of our simplified Peach example (see below), the Consumer is the role plaid by the software agent acting as the interface for the human user, that is the *User Assistant*.

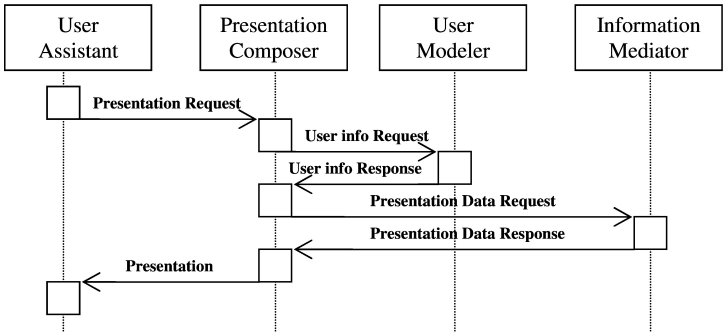

Fig. 2. Overview of the actor interactions.

tains an agent (the *User Assistant*) which, on behalf of the user, sends a presentation request to the museum central system. Here, three system actors take the responsibility of generating a presentation: the *Presentation Composer*, the *User Modeler* and the *Information Mediator*.

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. The communication diagram of Figure 2 presents the sequence of events from the time a request for presentation is issued until the presentation is presented to the user. The *User Assistant*, the *Presentation Composer*, and the *User Modeler* are generic roles that may be played by different software agents, e.g., 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 (hand-held devices, 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.

In any case, here, we are not interested in the specific agents implementing such functionalities (i.e., playing the assigned role), but, instead, in the roles themselves. In fact, they – i.e., the *User Assistants*, the *Presentation Composer*, and the *Information Mediator* – form ad-hoc service-oriented organizations, in order to achieve the service goal. Each SOO is characterized by members that collaborate at different levels and times during the *service providing process life-cycle*. After the goal is satisfied, the organization is dissolved and a new one will be formed – possibly including different agents, provided they play the listed roles – to serve a new request.

3.1 Discussion

The previous section motivates the need of some agent patterns to effectively deal with distributed computing issues (e.g., see [11, 17, 16, 10]).

Nevertheless, if we proceed by adopting traditional agent patterns, as, e.g., the matchmaker and broker introduced in Section 2, probably we could not be able to capture few but interesting and vital architectural requirements that arise from our ambient intelligence scenario, specially if we want to fully exploit the flexibility – in terms of self organizing presentation delivery channels – that can be provided.

In particular, to motivate our assertion, let us consider the following new scenario:

Example 2 (implicit communication). Let us assume that, while walking around, the user is approaching some presentation devices that are more comfortable and suitable to handle the presentation than the mobile device (*User Assistant*), e.g., in terms of pixel resolution and audio quality. So, we may assume the *User Assistant* is autonomously capable to exploit its intelligent behavior by negotiating the most convenient presentation, on behalf of its human owner. Let us also assume that there are several different *Presentation Composers* for each single device (capable to generate video, text, animated explanation, audio, etc.) and that each *Presentation Composer* relies on different *Information Mediators* to provide the information required for presentation generation. Moreover, we may also assume that each *Presentation Composer* is able to proactively propose its best services (in terms of available or conveniently producible presentations) to the *User Assistant*, possibly through some mediation interface. As well, we expect that all the services (negotiated or proposed) are "dynamically validated", that is, due to the fact that the environment and the user location are quickly changing, only the appropriate services are considered.

Such a scenario calls for architecture flexibility in terms of dynamic group reconfiguration to support SOOs involvement. Traditional approaches allow for intentional relationships and request/response communication protocols among single agents only, and not among group of agents [9–11, 17]. More specifically, we may assume that the *User Assistant* starts an interaction session that triggers the involvement of a group of system actors all with the ability of *Presentation Composer*, which in turn trigger the involvement of a group of system actors all with the ability of *Information Mediator*. Each *Presentation Composer*, instead, relays on the *User Modeler* to know the user profile to correctly build up a user-tailored presentation.

Therefore, such an architecture has to adopt group communication in order to support an 'intelligent' pervasive computing model among users' assistant devices and the system actor information/service providers. To cope with these new challenges, we can imagine that the system agents exploit a form of 'implicit communication', where they can autonomously build up SOOs in order to satisfy a request at the best they can do at that time. This is not possible by means of traditional approaches that adopt simple request/response based communication styles (e.g., [16]). In fact, as shown in Figure 1, using classical matchmaker and broker approaches, we assume that there is an advertise service dependency (e.g., based on a preliminary registration phase) forcing the system actors to rely on a centralized computing model.

4 Agent Patterns-Based Detailed Design

The discussion above highlights the limits of traditional patterns when applied to our ambient intelligence pervasive computing scenario; hence, the necessity of characterizing our system architecture by means of new agent patterns.

4.1 The Service Oriented Organization

In distributed computing and especially in *'intelligent' pervasive* computing based scenarios, each time an information consumer explicitly or implicitly causes a specific

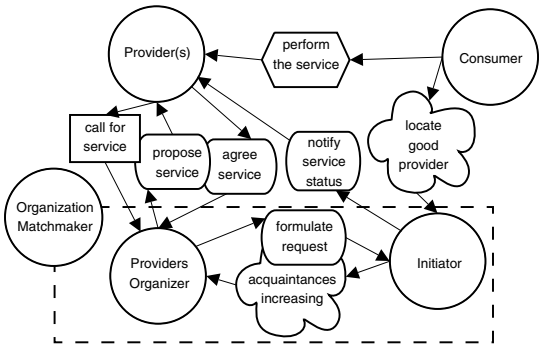

Fig. 3. Actor Diagram for the *Service Oriented Organization* pattern.

service request, it inherently needs a searching capability in order to locate the service provider. In particular, in our scenario, when the *User Assistants* is looking for a *Presentation Composer* in order to ask for a personalized presentation, a matchmaker (e.g., the one presented in Section 2) or a facilitator architecture is required [10, 11, 13, 16].

As previously discussed, the matchmaker pattern illustrated in Section 2 does not completely fit the requirements of our pervasive computing scenario (Example 2). Here, we define a new agent pattern – the *Service Oriented Organization* pattern – illustrated in Figure 3, which extends and adapts the matchmaker pattern of Figure 1.a. Here, the actor Matchmaker is replaced by Organization Matchmaker, which is further decomposed in two component system actors: Service oriented Organization and Initiator. The dependencies between Consumer and Organization Matchmaker (or, more specifically, Initiator) and between Consumer and Provider(s) are as before. The main difference, instead, is that now there is no 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, as foreseen in the classical matchmaker approach. The solution we propose, instead, is based on a proactive and, specially, dynamic capability of service proposal, on the basis of the actual, current requests or needs of services. In particular, our system low level communication infrastructure is based on a group communication, which has been designed to support channelled multicast [3]. That is, a form of group communication that allows messages addressed to a single agent or a group of agents (Provider(s)) to be received by everybody tuned on the channel, i.e., the agent "introspection" capability described in Section 5. Thus, Provider(s) depends now on Organization Matchmaker, or, more specifically, on Providers Organizer to have a call for service. That is, because of each SOO member adopts an IP channelled multicast approach that allows to overhear on channels (see Section 5 for details), the organizer simply sends its service request message on a specific channel and it waits for some providers offers[4]. On the basis of such calls, Provider(s) may notify their current services availability. Thus, the Providers Organizer depends on Provider(s) for propose service and,

[4] In fact, channels are classified by topics and each provider is free to overhear on the preferred channels according to its main interest and capabilities.

vice-versa, Provider(s) depend on the Providers Organizer for the final agreement on service provision (goal agree service). Moreover, in an 'intelligent' pervasive computing based scenario, the system awareness allows to proactively propose services to the consumer without any explicit service request. Thus, Initiator acts as interface towards Consumer. It is able to interpret Consumer's requests and, specially, proactively propose not explicitly requested services, on the basis of Consumer's profile and previous interaction history[5]. To this end, Initiator depends on 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, which is instead present in the traditional matchmaker pattern. Finally, Initiator requires that Provider(s) timely notify service status in order to only propose active services.

4.2 The Implicit Organization Broker

As observed in Section 1, ambient intelligence environments are often characterized by intermitted communication channels. This problem is even more relevant when proactive broadcasting is adopted, as in the scenario suggested by Example 2. In this case communications to/from the *User Assistant* need to be reduced at a minimum. To this end, we propose here to exploit the *implicit communication* paradigm towards the adoption of an *implicit organizations broker* (IOB) agent pattern that is inspired to the implicit organization introduced by [4]. That is, we define the IOB as a SOO formed by all the agents tuned on the same channel to play the same *role* (i.e., having same communication API) and willing to coordinate their actions. 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 – and no name for it – since the role and the channel uniquely identify the organization. Its members play the same role but they may do it in different ways; redundancy (as in fault tolerant and load balanced systems) is just a particular case where agents happen to be perfectly interchangeable. In particular, we can consider to have *implicit organizations* playing a kind of broker role. In other terms, each time the system perceives the visitor's information needs, the system actors set up a SOO (as described in Section 4.1), which, in addition to the already presented matchmaking capabilities, can also manage the whole service I/O process; that is, the SOO is able to autonomously and proactively cope with the whole *service providing process life cycle*. Such a system ability enhances the ambient intelligence awareness, a system requirement that cannot be captured by adopting traditional agent patterns [10, 11].

Figure 4 introduces a IOB pattern as a refinement/adaptation of the SOO pattern introduced in Section 4.1. Provider(s) are now part of the organization itself, which plays the role of an Organization Broker. Thus, the latter include both Providers Organizer and Provider(s) (see the inside of the dashed-line rectangle). It is worth noticing that the IOB members are characterized by the same (required) skill (see ahead Section 5).

The differences between the two traditional agent patterns of Figure 1 are naturally reflected also between the two patterns illustrated in Figures 3 and 4. In particular, Fig-

[5] For example, every system actor, through environmental sensors, can perceive and profile users during their visits across museum media services, as in the scenario of Example 2.

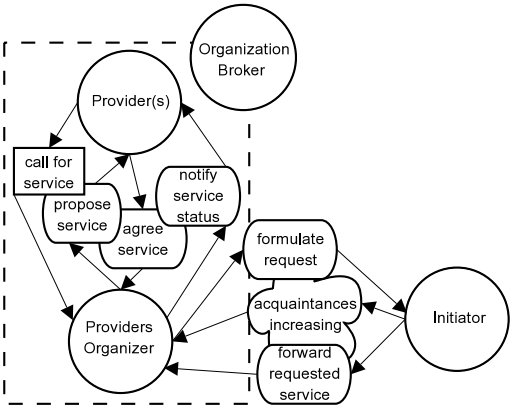

Fig. 4. Actor Diagram for the *Implicit Organization Broker* (IOB) pattern.

ure 3 tries to capture intentional aspects for more general group communication scenarios, i.e., general SOO. On the contrary, Figure 4 gives a level of pattern based detailed design focusing more on special kind of SOO, tailor-made for ambient intelligence scenarios. In other words, Figure 3 does not consider a strictly 'intelligent' pervasive computing scenario that, on the contrary, characterizes our IOB of Figure 4.

As well, it is worth noticing that the IOB pattern incorporate in the Initiator role both the roles of Consumer and Initiator of the SOO pattern. As already said, this is a consequence of the fact that, in ambient intelligence, some system actors concurrently play the consumer and initiator roles, which allows the system to enhance autonomy and proactivity skills. Moreover, and similarly to what happen in Figure 1.b between the Consumer and the Broker, in Figure 4, the Initiator depends on the Organization Broker – or, more specifically, on the Providers Organizer – to forward requested service, in order to avoid *User Assistant* message/interaction overloading. Nevertheless, the IOB pattern allows for acquaintance increasing (for Initiator), so to consent a more precise service requests during future interactions, as already foreseen for the generic *Service Oriented Organization* pattern.

5 Supporting Implicit Organization Brokers

The two agent patterns *Service Oriented Organization* and the *Implicit Organization Broker* presented so forth have been experimented within the Peach project to build an interactive, pervasive, museum guide. As mentioned, our patterns require a group communication infrastructure. To this end, we adopt the *LoudVoice* [4] experimental communication infrastructure based on channelled multicast and developed at our institute. Specifically, LoudVoice uses the fast but inherently unreliable IP multicast – which is not a 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.

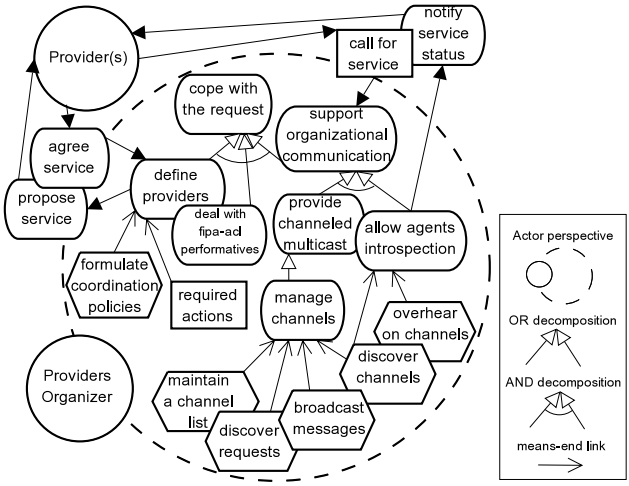

Fig. 5. Goal Diagram for an agent's role characterization by means of its capabilities.

5.1 Agent Roles Characterization

Analyzing agent roles means figuring out and characterizing its main capabilities (e.g., internal and external services) required to achieve its intentional dependencies already identified by the agent patterns analysis of Section 4. 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 5, we adopt the *means-end* and *AND/OR decomposition* reasoning techniques of the Goal Modeling Activity [2, 5, 18]. 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 subgoals, 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 consider only the IOB pattern. According to Figures 5 and 4, each time Initiator formulates a request, Providers Organizer achieves its main goal cope with the request (i.e., the goal that Providers Organizer internally adopts to satisfy Initiator's request) relying on its three principal skills: define providers, deal with fipa-acl performatives, and support organizational communication. The principal goal success depends on the satisfaction of all the three goals (i.e., AND decomposition). For the sake of simplicity, Figure 5 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 channelled multicast [3] (see goal provide channelled multicast) that allows messages to be exchanged over open channels identified by topic of conversation. Thus, a proper structuring of conversations among agents allows every listener to

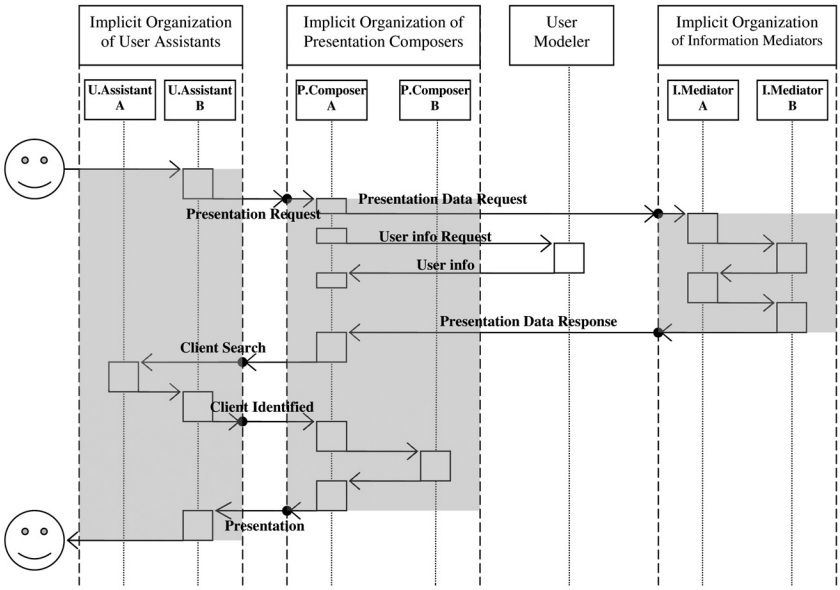

Fig. 6. Interaction of organizations.

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 channelled multicast ability, each actor can decide by itself what channels to listen to, by means of a subscription phase (represented by the tasks discover channels and maintain a channel list). This communication mechanisms well support group communication for service oriented and implicit organizations composed by members with the same interests or skills. Such organizations assist the *User Assistant* avoiding it to know how directly interact with the museum multi-media services.

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, as presented by Example 2 of Section 3. Modeling this example 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 6, we propose a new type of diagram that deals with the group communication features required by the scenario introduced with Example 2. Here, the shaded rectangles and the dashed lines below them represent the implicit organizations, and the gray rectangles represent the communication internal to implicit organizations. 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 6, 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 requests user information on the user model 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 to the *Implicit Organization of User Assistants*, regarding the availability of assistants capable to show the presentation being planned. 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.

6 Conclusions

Ambient intelligence scenarios characterized by service oriented organizations, where group of agents collaborate at different levels and times during the service providing life-cycle, generates new software architectural requirements that traditional agent patterns cannot satisfy. For example, 'intelligent' pervasive computing and peer-to-peer computing models naturally support group communication for ambient intelligence, but they also call for architecture flexibility in terms of dynamic group reconfiguration. Traditional request/response communication protocols are not appropriate to cope with service negotiation and aggregation that must be 'dynamically validate', since the environment conditions and the user location are quickly changing.

For such reasons, we propose two new agent patterns (Service Oriented Organization and Implicit Organization Broker) and compare them with traditional patterns [10, 11]. Specifically, we adopt the agent oriented software development methodology Tropos [2, 5], to effectively figure out the new requirements. For example, using Tropos, we can keep the agent conversation and social levels independent from complex coordination activities, thanks to an inherently pure *peer-to-peer* computing model [13]. Such a way of modeling has been thought for enriching the Tropos methodology detailed design phase with new capabilities, more oriented towards sophisticated software agents, which requires advanced modeling mechanisms to better fit group communication, goals, and negotiations. Thus, we have been able to capture important aspects of ambient intelligence requirements and to build up new agent patterns, more flexible and reusable than the traditional ones.

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. *Autonomous agents and Multi-agent Systems (JAAMAS)*, 8(3):203–236, May 2004.
3. P. Busetta, A. Donà, 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. Springer Verlag (to Appear), 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. 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.
10. M. Klusch and K. Sycara. Brokering and matchmaking for coordination of agent societies: A survey. In A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, editors, *Coordination of Internet Agents: Models, Technologies, and Applications*, pages 197–224. Springer-Verlag, Mar. 2001.
11. 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.
12. M. P. Papazoglou and D. Georgakopoulos. Introduction to the special section on Service Oriented Computing. *Commun. ACM*, 46(10):24–28, 2003.
13. 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*, IOS Press, 1(1):3–21, 2003.
14. 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)*. INSTICC-Institute for Systems and Technologies of Information, Control and Communication, vol.4, pp 316-321, 2004.
15. 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.
16. K. Sycara, S. Widoff, M. Klusch, and J. Lu. Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Autonomous Agents and Multi-Agent Systems*, 5(2):173–203, 2002.
17. 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.
18. E. Yu. *Modeling Strategic Relationships for Process Reengineering*. PhD thesis, Department of Computer Science, University of Toronto, Toronto, Canada, 1995.