OPEN SOCIAL AGENT ARCHITECTURE FOR DISTRIBUTED MULTIMEDIA

Busetta P., Zancanaro M.

December 2002

Technical Report # 0212–20

Open Social Agent Architecture for Distributed Multimedia

Paolo Busetta,
Massimo Zancanaro
ITC-irst
Povo, Trento - Italy
busetta@itc.it
zancana@itc.it

ABSTRACT

We propose an architecture for large scale, multi-user, distributed multimedia based on cooperating agents communicating by means of an observable communication channel. In addition to the traditional protocols based on point-to-point communication, coordination and cooperation should be supported via social awareness and overhearing. Overhearing also allows the collection of contextual information without interfering with already deployed systems. Our domain of application is interactive museums, which are typical examples of so-called "active environments" or "ambient intelligence". Purpose of this paper is to present the core concepts, and outline the future lines of research.

1. INTRODUCTION

Supporting a visit to an museum may consist of many different things: for instance, driving a visitor through the exhibits; explaining them; illustrating their history and the biography of their authors; answering to the visitor's questions; highlighting important architectural points in the building. In traditional settings, these supportive actions are performed as a mixture of active human intervention (e.g., by a guide) and dissemination of static information in the environment (e.g., maps of the building, explanatory notes on the walls). Their digital equivalent are interactive systems able somehow to adapt to an individual visitor's needs.

Although, many research projects are exploring the new possibilities offered by Personal Digital Assistants as multimedia guides (among others, [28, 9, 21, 26]), there has been very view attempts to investigate the architectural issues emerging from the embedding of the mobile guide into an overall "intelligent museum". The "intelligent museums" scenario belongs to the area of ubiquitous computing dealing with "ambient intelligence" [15] or "active environments" (AE) [27].

An intelligent museum clearly calls for sophisticated multi-media, multi-modal interfaces and applications that are physically and logically distributed and autonomous from each other, but able to cooperate in order to provide a consistent user experience. Indeed, a museum must cater for many simultaneous visitors, but each individual must be recognized and consistently followed during her visit. To make a simple example, all presentations must be in the visitor's language; more interestingly, each presentation of an art exhibit must take into account what has been already told during the visit; ideally, an electronic personal guide should be allocated and virtually accompany and drive the visitor during the entire visit, interacting with her by means of a PDA – if she carries one – or whatever device is close to her.

Significant challenges have to be faced for integrating the components of such a complex scenario, and having them adapting to continuous changes in the environment. These may vary from simple faults (e.g. power and network failures), to the continuously changing availability of different types of devices (including the PDAs used as multimedia guides), and possibly hardware and software upgrades being rolled out while the system is running.

This paper proposes a novel architecture for large scale, distributed, multi-modal multi-media systems, whose objective is to tackle some of these challenges. In short, a multi-agent architecture is adopted, where coordination among different types of components is obtained via role-based, observable communication. This style of communication allows, from the one hand, to cater for continuous changes in the number of components, for the collection of contextual information and for unforeseeable interactions among different types of components; on the other hand, it requires proper coordination among components able to play the same role. A multicasting technique is used, which eases the problem of finding interfaces and computing elements based on their capabilities and physical distribution. The architecture has been adopted and is under test within the PEACH project[1] [29], whose goal is to experiment new technologies for cultural heritage appreciation.

The rest of this paper is organized as follows. Next section gives some basic references to multi-agent systems that are at the basis of our approach. Section 3 highlights what are, in our perception, the main differences between active environments and traditional systems. Section 4 outlines the approach we have taken. Section 5 presents a practical application scenario. In Section 6, we discuss how we want to gather contextual information. The communication infrastructure we have chosen is introduced in Section 7. Section 8 presents the general architecture, while Section 9 discusses in more detail the technique for component coordination. Finally, in Section 10 we show how the architecture is applied to the application scenario introduced earlier.

---

[1] http://peach.itc.it

2. AGENTS AS COOPERATIVE COMPONENTS

A multi-agent architecture is composed of distributed, autonomous, communicating components called agents [35].

Much research on engineering multi-agent systems has focused on the issues of communication. Within this, two main areas can be identified. The first revolves around the issues of syntax and semantics of protocols, and has led to the definition of some agent communication languages (ACLs) such as FIPA [18] and KQML [17] and their supporting infrastructures. A typical ACL provides a set of predefined, high-level message types called *performatives* (usually inspired by speech-acts [30]) with a clear – even if somewhat simplistic – semantics in terms of pre- and post-conditions on the state of the sender and of the receiver, a generic syntax for the message's payload, and a way to specify a communication ontology, i.e. a domain-specific, semantically precise content language.

The second main research area in agent communication focuses on coordination protocols, such as the well-known Contract Net [31] for task distribution and its numerous variants and derivations. This work has lead to the identification of some typical patterns, which have been reduced to five main ones in [13]: "blackboard" (subsuming all sort of pattern-based indirect interaction by means of a facilitator or a tuple-space), "meeting" (typically supported by infrastructures for mobile software agents), "market maker" (subsuming Contract Net and its derivatives). "master-slave" (subsuming all sort of RPC-like interactions), and "negotiating agents" (typical of supply-chains or other situations where an agreement has to be reached before performing something, e.g. on the quality of the service to be provided).

An in-depth discussion of these patterns goes well beyond the scope of this paper; however, some observations are necessary. The first three patterns (namely, blackboard, meeting and market maker) require the presence of a mediator of some sort. Participants to multiparty protocols thus need to contact this mediator beforehand, as well as know message formats (or patterns or tuple spaces). This leaves no mechanism for on-the-fly additions or removal of agents to a conversation due to its content, or to the appearance or disappearance of mobile devices, or to the adaptation to – or learning of – the protocols used in a specific environment. Also notably missing from the patterns presented above is the publish / discovery mechanism supported by state-of-the-art peer-to-peer systems, such as JXTA [1]; this again shows that research on coordination has focused on mediators, rather than supporting awareness of other agents and of the execution environment.

Various researchers have looked at groups (or *societies*) of agents as first-order entities with goals and other attributes of their own, rather than being just an aggregation of the knowledge and behavior of their members. In this approach, the coordination protocols mentioned above are used for maintaining a group's attributes synchronized with those of its individual members. Some research has focused on *teamwork*, initiated by the classic work on joint attitudes (goals, intentions, and mutual beliefs) [10]; a very recent review of its potential application to ubiquitous computing is in [8]. A number of implemented systems supports team programming; for instance, STEAM [32] uses SOAR rules to define teams and their behavior; SimpleAgent [23] extends a commercial BDI product, JACK Intelligent Agents, with programming facilities aimed at describing roles, team formation, and team coordination. More recently, Tidhar [33] introduced the concept of *organization-oriented programming*, where organizations are made up of teams that coordinate by means of *command, control, and communication (C3)* relationships. Other approaches, rather than using social structuring for coordination, focus on group planning (e.g., Partial Global Planning [16], Shared Plans [22]).

3. ACTIVE ENVIRONMENTS: INTEGRATION ISSUES

In this section, we contrast some important assumptions on application requirements and run-time environments taken in traditional computing and HCIs (including agent-based approaches) against those in ubiquitous computing for "active environments" (AE). A very recent and up-to-date discussion of most of the points highlighted below can be found in [12].

In traditional computing and HCIs, including multi-media and multimodal systems, the programmer (sometimes the system configurator) knows beforehand which hardware and software components are available. The maximum run-time flexibility left to a component that needs a service or a resource is the on-the-fly selection of one (or other well-defined number) service/resource provider within a set of equivalent ones, e.g. by means of an auction or a match-making mechanism or any other of the coordination patterns mentioned earlier. Shared planning is rarely performed in real-time, real-world applications.

Composition of components is typically hierarchical, and by level of abstractions. In most cases, communication is directed, component-to-component – even multi-party protocols are commonly implemented as a set of point-to-point message exchanges. Moreover, no unanticipated messages can "sneak" into the communication logic. Redundancies of components (e.g., duplications) or on-the-fly additions are just not part of this picture, other than to provide some carefully engineered fault-tolerance mechanism. A single HCI concentrates both input and output, often following a model-view-controller pattern where the three main application layers are tightly coupled and the controller has full ownership of one or two input mechanisms (typically, keyboard and mouse).

In active environments, it is expected that many different HCI devices operate simultaneously in the same place. A single device may able to perform more than one operation (e.g., a touch screen is at the same time an output and an input device); more than one device can perform the same operations (e.g., multiple screens may be available in a room); one or more may come and go at any time (e.g., wearable computers); and, communication may easily fail, in particular when using wireless links, because of noise or many other reasons. Of course, "device" means something more than pure hardware: indeed, a typical device is expected to embed software with more or less sophisticated capabilities of self-configuration and adaptation to specific situations. Consider, for instance, a speech recognition system including microphones and software. The most complex example is probably a mobile device such as a PDA, which can include multiple applications running simultaneously.

Input from users in AE is often partially implicit, i.e. determined by context – for instance, where the user is, what objects she is closed to, what she has been doing recently. More than one user may be present in a single place at the same time; normally, a device operates for a single user, but there may be instances where group HCI is required (e.g., spoken or written information to a group of visitors; or, a wall-size screen showing messages for many different

users); conversely, there may be instances where multiple devices of the same type work for a single user (e.g., multiple screens, each showing a part of a picture).

Some application components may be permanently running for a user, on her mobile device if she has one but possibly on an invisible host in a computer room, maybe remotely on a grid. Other applications may be invoked on demand, again locally or remotely. Adaptation to local conditions, in terms of devices and context, is absolutely required. Since no application can foresee all possible contexts, it is necessary to have a set of intermediary components acting as context interpreters, information aggregators and contextual adaptors on behalf of the applications; in turn, these intermediaries may be permanent or created on demand.

In summary, these are some of the major characteristics unique to AE that make it different from traditional computing and HCIs:

- Multiple users may be in a single place, interacting with different applications simultaneously. (The very meaning of "application" is probably different from traditional). Context is shared among multiple applications being active at the same time for the same user (e.g., one in foreground interacting with the user, the others in background overhearing and interacting only when appropriate e.g. to provide additional information or when invoked either explicitly or implicitly).

- The set of users changes dynamically over time. For example, people may enter or exit a smart room as they please.

- Users are unaware that the system is formed by a number of components and and they are also not interested in the internal mechanisms that the system uses to provide services. Therefore, they interact with the system as if it were a single entity.

- However, services are provided by a variable set of components that can join and leave the system (plug-and-play as well as mobile computing, both physical and logical), and can be running anywhere (from a local computer to any machine in the world connect by Internet).

- Services provided by components can (partially) overlap; therefore these components need to coordinate in order to decide, for instance, who provides a specific service, and how.

As a consequence, coordination policies among components cannot be fully predetermined, but have to depend at least partially on the context. For example, the way a service is provided to a user may vary according to who the user is, her preferences, and the availability of devices (sensors as well as actuators, possibly mobile on the user's PDAs or wearable) in the place where she currently is.

4. GENERAL ARCHITECTURAL APPROACH

From the assumptions described above, we derive a set of architectural requirements.

The type of systems we want to support are composed by many physically and logically distributed components. Most of them are able to operate either autonomously (i.e., off-line), or in a limited environment where only a few other system components are present. Conversely, the addition at run-time of a new component does not disrupt the operations of those already deployed; in some cases, their behavior may change so to take advantage of the added component.

Henceforth, those individual components will be called **agents**. However, we do not make any assumption about their sophistication or their inferential capabilities. The only requirement for an agent is to be able to communicate with others following a common, basic syntax (our choice is described later).

As discussed above, agents then include systems embedded into hardware – sensors, or contextual interpreters of sensors data ("context widgets" and "context aggregators" using the terminology of [14]), actuators, output devices – as well as middleware components close to sensors and actuators, e.g. context interpreters and context aggregators up to application-level components (presentation generators, user profilers, and so on).

One of our goals is to investigate group communication techniques suited to an active environment, extending if necessary the set of known agent coordination patterns. As an architectural choice, unless there are serious limits in performance or message size, we want communication to be "observable", i.e. to be realized by means of some form of broadcasting; to this end, we chose the communication infrastructure described later. Communication observability allows group communication (one-to-many), which is key to addressing dynamic additions or removal of agents, as well as additional capabilities such as, for instance, monitoring, intention recognition, and consequently collaboration triggered by overhearing [7].

We will refer to a *role* to mean an abstract agent interface, i.e. a set of protocols used to obtain a certain type of services. Differently from traditional approaches, our goal is to distribute functionality on a group of agents without necessarily know which agents are or will be delivering them, and without involving mediators, so to achieve a higher flexibility and adaptivity to continuously changing conditions. Our approach to this is by favoring role-to-role rather than agent-to-agent interaction, as described in a later section. Of course, this has a significant impact on the way agents behave in their interactions, also because we want to allow all possible combinations of roles and agents. For instance, we can have roles provided in full by a single agent; roles only partially fulfilled by a group of agents; and so on. In particular, we are interested in three main broad classes of relationship among roles and agents that support them:

redundancy: multiple agents support a certain role in an equivalent way. That is, a request for a service from that role can be serviced by anyone of the agents;

partitioning: the services in the role are partitioned among the agents; that is, a service request can be fulfilled by at most one of the agents. Note that partitioning is often based not on the type of service being requested, but on its parameters; e.g., two agents may maintain user profiles with the same schema but for different users;

coalition: the services of the role can be provided by the group but there is no single agent that can do it by its own; thus, collaboration is necessary.

The group of agents entirely or partially fulfilling a role may change over time, with the consequent impact on the ability of the system

of delivering a specific service. Consequently, it is necessary for an agent to be aware of who else is able to play its same roles in the current context, and coordinate accordingly.

5. AN EXAMPLE SCENARIO

In this section, we introduce a scenario of a distributed architecture for delivering multimedia presentations on a PDA to support a visitor of a museum. The architecture is inspired by the result of the HIPS project [28].

Figure 1: Reacting to user movements (sequence diagram)

The visitor is carrying a PDA, connected to a main computer with a wireless link. The PDA is a rich client yet all the system components are on the main computer. In particular, the PDA is able to display a multimedia presentation composed by synchronized audio and video, and lets the user operate the standard control of an hypermedia system (that is, pause, stop, reply and selection of links). Moreover, the PDA is able to capture infrared codes emitted by beamers scattered in the environment.

The communication among the components of the system takes place through message passing. In the following, we will briefly present the flows of messages that arise from the reception of an infrared code by the PDA (see diagram in Fig. 1, where the PDA is represented by the Visitor actor). This is meant to illustrate the main functionality of the different components of the system. The scenario will be revised in terms of *implicit organizations* later.

When the PDA "senses" a new infrared code from a beamer, a SensorPosition message is sent to the VisitorAssistant (VA) component (actually, in HIPS the message was sent every 2 seconds even if the code was not changed in order to distinguish between a motionless user and a lost connection).

First of all, the VA notifies the sensors codes to the Physical Organization KB Manager (POKB), and receives back the visitor's position in terms of the area in which the visitor is located (eg. in

"Sala del Mappamondo" close to the "Maestà") and the list of exhibits that she is possibly looking at (both in terms of line of sight and distance with respect to the exhibits' size).

The visitor's position is then communicated to both the User Modeller (UM) and the Input Analyzer (IA). The first just records the position and makes whatever inference about the user (for example, following Veronand Levasseur's classification of visiting styles [34], it can classify the visitor as an "ant" or a "fish" etc.). The UM does not initiate any communication, it just stores info, makes inferences and replies to queries about visitors' characteristics.

The Input Analyzer is a rule-based system that for each visitor's movement decides whether it is "communicative" or not, and consequently plans the system behavior. Basically, it can decide that the current presentation has to be stopped (if there is one playing), that a new exhibit has to be presented, or that nothing has to be done because the movement was meaningless with respect to the interaction with HIPS. In taking this decision, the IA usually queries the UM. In the diagram, the IA/UM interaction has been synthesized in a single conversation in which a "bundle" of information is exchanged. Actually, many different conversations on specific visitor's characteristics take places while the IA is rumbling what to do.

Then, the IA communicates to the VA its decision. If the current presentation has to be stopped, the VA just notifies to the PDA; if a new presentation has to be prepared, the Presentation Planner (PP) is activated.

The PP can in turn start a number of conversations (not depicted in the diagram) with the UM and the POKB to adapt the presentation to the visitor's interests, visiting styles and her current physical position, as well as to what s/he has already seen.

6. SUPPORTING CONTEXTUAL INFORMA- TION

One of the greatest issues in ubiquitous computing is capturing the current *context* of the user. An interesting discussion on the topic of requirement analysis and design of context-aware systems is [2]; the overall process is summarized in [14] (page 22). What this work highlights, however, is that the definition of context is still highly application-specific. Even general context theories such as Giunchiglia's [20, 3], which provide useful logical frameworks and languages, do not help in understanding *what* a context is. In practice, any information collected via any sensor, plus any background knowledge on the user and the environment, may become part of a context useful for some purpose; consider also that *history* can play an essential role (e.g., knowing the last movements of the user in a room for predicting her direction).

Somebody envisages a set of *context services* (see Hohl [12, 24]) that collect and store information from sensors, and make them available to applications. This has the double advantage of freeing the latter from the burden of collecting the information themselves, and of supporting whatever kind of sensors are available in a certain environment.

As mentioned earlier, we are interested in supporting context services; indeed, this is one of the reasons that led to the requirement of observable communication. We envisage collections of "context gatherers" (including what Dey calls "context widgets" and "context aggregators" [14]), distinguished by their "focus of attention"

– for instance, a specific location (e.g., what happens in a room), or a specific user (e.g., a profiler that keeps track of what a museum visitor has been doing recently). Some of these context gatherers may fuse information coming from different sensors or other gatherers, to obtain a higher-level interpretation of what is happening; they may even be started solely to support the context definition of a specific application. In general, we expect to see the emergence of "pipelines" of context gatherers, each processing information of a certain type into a different view that in turn becomes the input of other gatherers, and so on. Discovery and communication among applications and gatherers will happen by means of the general facilities provided by our role-based communication, as discussed in the next sections.

7. THE CENTRAL ROLE OF COMMUNICATION

In order to meet the requirements of active environments, we have chosen to adopt a multicasting technique called "channeled multicast" [5]. Channeled multicast is based on the concept of *channel*, defined as a stream of messages (typically corresponding to speech acts, as in ACLs) that may be listened to by many agents simultaneously. A message is addressed to a specific destination, which may be one agent or a group of them; a group may have a name, or be identified by an expression in an application-specific language that says what are its characteristics. Many channels can co-exists, and new ones can be created on the fly. Channels are identified by a name or, more interestingly, by a topic and a set of arguments. An agent can discover which channels exist, "tune" on as many as it likes, and listen to all exchanges happening on them, no matter whether it is the intended destination of messages or not.

A first, multicast IP-based implementation called LoudVoice is described in [5]. LoudVoice supports a simple ACL, with a predefined set of performatives. Rather than having a generic syntax plus ontologies as in FIPA or KQML, the format of the payload is application-specific and encoded using XML Schema; these choices seem to give a balance between universality and the need for high performance and light-weightiness, especially since we have to support systems with limited computational power. A second version of LoudVoice is under development, which adds some structuring to the stream of exchanges on a channel. Indeed, messages can be sent either *extemporarily* – i.e., unrelated to a specific, short-term objective; typically, event notifications – or within *conversations*. Conversations start with a *header*, which is composed by a topic and a set of arguments, and end with a termination message. Typically, a conversation is created with the aim of performing some short-term task involving two or more agents, such as obtaining a service. An agent can select which conversations to listen to, and can ignore the others; also, message-level filtering criteria can be given, in order to minimize the burden on the application due to the processing of irrelevant messages.

Being based on multicast IP, LoudVoice is inherently unreliable. Some care is taken in order to minimize the losses of service messages (such as conversation headers). Note that message loss is just one of the many reasons for a destination not to be reached; indeed, in our domain, agents must deal in real-time with all sorts of issues, for instance device and power failures. Thus, an agent would have to handle timeouts and apply recovery techniques even if message transport were guaranteed; we then chose to privilege performance and light-weightiness, as provided by multicast IP, rather than robustness against message loss.

A fundamental consequence of dealing with unreliability is that designing and developing an agent's communication logic may become a fairly complex business. This is not always the case; see, for instance, the English Auction protocol described in [5], which is actually *simpler* than using reliable, point-to-point transport, thanks to the possibility of overhearing conversations. We plan to research on computational models and languages that simplify dealing with unreliability. Lots of relevant work has been done in the area of formal modeling of conversations; for instance, [11] presents a technique for the analysis of agent conversation protocols based on Coloured Petri Nets, and provides links to relevant literature. However, we are not aware of works dealing with multicasting and its implications (such as partial reacheability, or recovery from a loss by observing following message exchanges). Moreover, we do not restrict conversations to predefined interactions only, since we allow for overhearing to take place and for unforeseeable message exchanges to happen as a consequence; this adds a further dimension of complexity that can be properly dealt with only at run-time.

Overhearing offers some opportunities that are very significant for our objectives. For instance, it is an additional mechanism, and often an alternative, to matchmaking, brokering, and other sorts of middle agents (a discussion on middle agents can be found in [25]). Moreover, overhearing allows for context collection without interference, i.e. without requiring explicit support from sensors and other agents (as long as the information to be collected is somehow transmitted on a channel). Similarly, overhearing trivially enables unobtrusive monitoring of agents and their activities.

Of course, LoudVoice is not usable in wide-area network environments, for a number of reasons including security and performance; indeed, we envisage its usage within well-defined physical environments such as buildings, while extensions to the outside world should happen via proxies or agents acting as gateways. In future, we plan to explore the integration of LoudVoice with peer-to-peer technologies, both to provide additional discovery mechanisms (usable, in particular, by portable devices entering a new space), and to explore the idea of "virtual active environments", created by aggregating independent LoudVoice installations in different locations.

8. ROLE-BASED INTERACTION AND IMPLICIT ORGANIZATIONS

To move in the direction indicated in "General architectural approach", we exploit the features of LoudVoice for two different, but correlated, objectives: moving from agent-based to role-based interaction; and, supporting what we call *implicit organizations*.

For this discussion, we call "role" a communication-based API, or abstract agent interface (AAI), i.e. one or more protocols aimed at obtaining a cohesive set of functions from an agent. A simple example is mentioned in [5], an auction system with two main roles: the auctioneer (which calls for bids, collects them and declare the winner) and the bidder (which answers to the calls and commits to perform whatever transaction is requested when winning an auction). An agent may play more than one role, simultaneously or at different times depending on its capabilities and the context.

The cornerstone of our approach is that messages sent through LoudVoice have to be addressed to their destination role, rather than a specific agent (or group of agents). It is left to the receivers (that is, everybody tuned on a LoudVoice channel) to decide whether a message is intended for them at the time and in the context of reception.

An appropriate definition of topics and arguments for conversation headers and communication channels simplifies the problem of addressing the correct audience; for instance, the current channel discovery mechanisms support matches with respect to a taxonomy of topics.

An initial proposal of a specification methodology for roles, conversations, channels topics and their arguments, is being tested [4]. It adopts a variant of the UML sequence diagrams for describing role-based conversation protocols, which caters for the unique features of our chosen communication style. For instance, a message sent to a role is intended for all agents playing that role simultaneously, but it may also be required that other roles process it (e.g., to update their beliefs); in certain situations, more than one message of the same type may be prompted by a certain role at a certain time (e.g., in the auction example, all interested bidders reply simultaneously to a single call for bid); conversely, it may be required that only and exactly one message is prompted by a role even when it is expected that more than one agent may play that role.

The last example is where the concept of implicit organization comes into play. We take from Tidhar [33] the idea of *organizations* as teams of agents augmented with *command, control, and communication* predicates describing their social relationships concerning goals, intentions, and beliefs respectively. In our setting, a number of factors contribute to identify a group of agents to which coordinated behavior is required as reaction to a message; these factors include the channel the message is broadcasted on, the role it is sent to, the header of the conversation it is part of, but also the contents of the message, the state of the agents, and other contextual information outside of the direct control of LoudVoice. We call such a group an "implicit organization", because we require that its members agree on a control relationships (i.e., on a way to decide how to achieve a goal commanded by means of a message sent to the organization's role), but – differently from formal organizations, that can be unambiguously named and have an clearly identifiable formation phase – membership is a context-sensitive function M depending on the parameters highlighted above.

Observe that many different implicit organizations may be active for the same role on a single channel. For instance, in our museum setting, a "presentation planner" role on a channel concerning "presentations" is played by many agents; these agents may have partially different specializations, but they will also be redundant in order to allow a prompt reaction to simultaneous requests. Thus, when a request for preparing a presentation for an exhibit E is sent to the "presentation planner" role, the implicit organization being addressed is formed by those agents currently listening on the channel that know about E and that are not already busy preparing something else.

Implicit organizations are the solution we propose for some of the specificities of active environments. By the informal definition given above, each agent of an implicit organization is able to play the organization's role when standing alone. Thus, with respect to the three types of relationship among roles and agents identified earlier, implicit organizations address the cases of redundancy and partitioning; coalitions may be dealt with appropriate extensions that we leave to future work. More specifically, for the current stage of research we restrict the coordinated behavior of implicit organizations to the following aspects:

- deciding which agents of the organization should actually

commit to achieving a goal commanded though a message;

- if more than one agent commits, deciding when the goal is considered achieved by the organization (e.g., when either one or all have achieved the goal);

- in the same situation, deciding on how to communicate the result of the goal, that is, what to send back (the result obtained from a specific agent rather than a synthesis of all) and who sends it.

A set of procedures and protocols for taking the decision mentioned above is called a *control policy*.

An implicit organization requires *social awareness* to its members; that is, they must know who else can play their same role in a certain context, and negotiate how to manage the organization. To this end, our architecture dedicates a LoudVoice channel (called "control") to social awareness, used by a software component (also called "control package"), still under development, that will be used by all our agents. The issues dealt with by the control package are discussed in next section.

9. MANAGING IMPLICIT ORGANIZATIONS: A PRELIMINARY PROPOSAL

To have an implicit organization running, with the restrictions specified above, the main issues to be solved are:

1. understanding when coordinated behavior is required to an implicit organization, that is, which messages carry commands that have to be satisfied by coordinating the members of the underlying implicit organization;

2. defining a control policy;

3. understanding who are the members of the organization at the time of delivery of a commanding message, and applying the control policy;

4. finally, performing the required coordinated behavior.

The following is a general discussion of how we plan to tackle the issues mentioned above. A preliminary solution is currently under experimentation; details can be found in [6].

Identifying coordination requirements

A general, but unduly restrictive approach for finding out when coordination is required is by *message performative*, i.e. by analysing the expected post-conditions of a message.[2] With reference to the set of performatives defined by FIPA [19], it seems obvious that, for instance, an "inform" does not commit the destination role to a goal, thus no coordination is required. A "call-for-proposal" commits its addressees to reply if they are interested to the object of an auction, but does not require any coordination since bidders are effectively competing against each other. A "query-if" and a "request" commit their destination, too; if we now require that only one agent reacts and only one answer (including "not-understood") is sent back, then the implicit organization needs to coordinate in

---

[2]We assume what we call a "deontological commitment" to achieving any goal specified within a message by its intended audience; this is to say, whenever a role is required to do something, it automatically commits to perform it.

order to delegate a single agent to perform the actions and send the answer.

However, there are many situations where the one-reaction / one-answer criterion given above is excessively limiting, or insufficient to determine a control policy. For instance, an agent may be interested in getting all possible answers to a query (e.g., all known user profile information); or, it may be appropriate that more than one agent performs the action simultaneously (e.g., all screens visible to a user in a dangerous situation may show the same warning message). A potential but unsatisfactory solution is augmenting the set of admitted performatives with "multiple reactions / answers expected" cases (e.g. "query-all" vs "query"). Going this way, however, means to enter that muddle of rather arbitrary and open-ended extensions that has been plaguing KQML; worse, it moves part of the burden of deciding a control policy to the message sender, while reducing the ability of an implicit organization of exploiting the capabilities of its members.

Thus, we prefer to derive coordination requirements from a role's specification, rather than from messages taken in isolation. As mentioned earlier, we are developing a methodology and a notation for conversation specification that is amenable to automated reasoning. From a simple inspection of a specification, it is possible to spot the points where coordination is required to an implicit organization; at the very least, these are all the cases where a single message is prompted by a role, either as a reaction to a solicitation (e.g., answers to queries) or unsolicited (e.g., event notifications such as state changes) [3].

Defining a control policy

There are various parts to this problem, that have to be solved at different times: designing a set of coordination protocols; deciding which types of coordination are appropriate for a certain role; and, agreeing on which one to use in a specific context. Before discussing policies, it should be noted that an agent developer has to isolate organizational control issues from the code that actually performs actions; we have defined a set of guidelines on how to do this and delegate control to our package.

Rather than imposing a control policy as part of a role's specification, or equivalently using a team-based approach that forces a top-down perspective, we leave developers free of specifying which policies are supported by their agents, and let an organization agree on which one to adopt at run-time.

Thus, having separated control from action, an agent developer still has to decide which policies are appropriate, and which are not, when achieving a certain goal. For instance, computationally cheap and safe actions such as querying an agent's own beliefs are unharmful no matter which policy is adopted; for expensive or state-changing goals, a designer may want that her agent is involved only when it is the best available as determined, for instance, by running an auction; long operations that can be easily aborted in the middle (e.g., long searches on databases or the Web) can tolerate policies that exploit parallelism; goals that have to be attained in the shortest possible time (e.g., reacting to a danger) call for minimal decision time, e.g. by having a master deciding for the organization.

In an ideal world where computational power and communication

---

[3] However, as mentioned in the previous section, in the first version implicit organization will coordinate only when reacting to solicitations.

speed were infinite, a policy should be negotiated among the members of an implicit organization at every command received. This is clearly unfeasible, so we have taken a few architectural decisions that we are implementing in our control package; these choices may be relaxed or changed in future, after sufficient experimentation. First, we are defining a set of possible policies and related coordination protocols. Policies include master-slave (where the master, elected or imposed from the outside e.g. via configuration, decides who is in charge for achieving a goal), selection via auctioning, parallel execution (concluded when all agents terminate executing), competition (like parallel execution, but concluded when the first agent terminates), fixed scheduling (e.g. round-robin), and so on. Second, a developer specifies which policies are admitted, in order of preference, and which ones are refused for each of the roles supported by her agent. Third, a policy is negotiated on the control channel among all the agents that play a certain role on a specific channel. This negotiation is performed every time an agent decides to play a role (typically when it starts executing), and boils down to a constraint satisfaction problem for determining a set of candidate policies sorted in order of their global preference; if this set is empty, the newly joining agent is forbidden from playing the role (our package reports an exception to the user, and prevents any further control operation concerning that role) while the others revert to the previous policy. The special case of only one agent playing a role on a channel is trivially solved by our control package by adopting a "green light" policy.

It is important to note that agreeing on a policy for a role on a channel is not enough for its application, since some details can be filled up only when the implicit organization is in place; for instance, deciding who is the master in a master-slave policy, or who runs an auction.

From this discussion, it clearly follows that the choice of channels is as sensitive as selecting coordination protocols. We are outlining a set of guidelines on channels configuration, which need experimentation to show their effectiveness.

Verifying membership

Assume that an agent is playing a role R on a channel C and a control policy for R on C has been already established. At time T, a message M, addressed to R and carrying a goal G (say, "request do-something"), is sent on C. Thus, at time T, it is finally possible to determine who are the members of the implicit organization that is committed to achieving G. As said above, this means that it is possible to complete the details necessary for the application of the chosen policy, e.g. determining who acts as auctioneer for an auction-based policy, or the master in a master-slave policy. These details and related protocols depend on the chosen policy, and are known to our control package.

Observe that, while some policies impose significant burden (such as selecting the auctioneer, or deciding who is the current master), others are reasonably lightweight. Consider, for instance, a competition policy: all agents can start working on achieving the goal G carried by M as soon as M is received. The first terminating agent could immediately reply to the sender of M, causing at the same time the competitors to know that G has been achieved; however, a race condition may happen if two agents terminate simultaneously. This can be prevented by applying a simple coordination protocol such as sending a "goal achieved" message on the control channel, followed by a short timeout to make sure that either there is no conflict or, otherwise, applying a deterministic selection technique

(e.g., the agent on the host with the lowest IP address wins).

Applying the control policy, and getting the job done
Finally, when everything is in place, the coordination protocols required by the policy can be applied, by adopting the well known patterns mentioned earlier or any new one that needs to be devised. As said above, the coordination protocols are run by our control package; coordination messages are exchanged on the control channel, not on the channel where the goal was posted.

10. FROM THEORY TO PRACTICE

This section discusses how the architectural lines outlined in the previous sections can be applied to the scenario proposed in our example.

A trivial redefinition of the scenario in terms of an agent architecture can be done by simply considering each component as an agent. In this solution, the VisitorAssistant acts as a mediator with respect to the PDA and the rest of the system; in other words, its task is to dispatch requests coming from the PDA to the appropriate agents, and viceversa. By adopting this simplistic approach, however, we face two major issues. First, the VisitorAssistant becomes a bottleneck at the communication level, and a single point of failure for the system. Second, if new agents join the systems, the VisitorAssistant has to be notified and its control policies have to be updated consequently.

A different, better approach is to model the scenario in terms of implicit organizations. Five main roles can be immediately identified from its description:

- **PDA**: the role of the user interfaces on mobile devices. Agents that belong to this role are able to: (i) sense infrared codes and communicate them to the rest of the system; (ii) play presentations (i.e., play audio files and display synchronized images), (iii) communicate to the rest of the system notable events concerning the presentations being played, e.g. start, end, stop or pause requested by the user.

- **POKB**: the role of the Physical Organization Knowledge Base. Agents playing this role are able to map sensor codes to physical positions, and to provide a number of related reasoning services (e.g. retrieve the set of closest frescoes, etc.).

- **UM**: the role of User Modeling. It deals with storing, retrieving and reasoning on information related to the user.

- **IA**: the role of the Input Analyzer, which takes cares of interpreting user movements in terms of inputs to the system. An agent playing this role is able to monitor each visitor's movements or utterances, and to decide whether to stop the current presentation, start a new one or just do nothing.

- **PP**: the role of the Presentation Planner. Agents playing this role assure the composition of a coherent presentation on a given artwork, tailored for a given visitor.

Of course, further refinements to the list given above are possible, for instance by dividing the PDA role into sub-roles each dedicated to a specific aspects.

Looking at the communication, an initial analysis of the conversations easily identifies three main themes, that can potentially become the topics of three different LoudVoice channels:

- **interactions with the PDAs**, concerning all commands and information relevant to PDAs. They include:

    - sensor codes, from a specific PDA to the POKB role;
    - requests to compose presentations on artworks, from IA to PP;
    - commands to stop a current presentation, from the IA role to a specific PDA;
    - commands to load a presentation from an URL, from PP to a specific PDA (the actual presentation is better streamed directly by the PDA rather than on a multicast channel).

- **interactions with knowledge bases**, concerning all those agents that maintain static or semi-static information about the environment. In our simplified example, the only knowledge source is the POKB role.

- **user informations**, concerning all interactions with agents that dynamically collect information about users and their current context; in our case, the agents playing the UM role.

In the scenario proposed, most if not all the roles can be played by exactly one agent (i.e., each agent is nothing else than one of the components described in our example). However, it is easy to see how the system would benefit in terms of efficiency and robustness by using a multi-agent, redundant approach to roles.

For example, the Presentation Planner role could be played by two agents: a complex adaptive planner that composes tailored presentations, and a database that indexes "canned" presentation on the artworks. The coordination policy between the two could be implemented as a fixed scheduling as the following: if the complex planner is not able to build a presentation, the simpler one replaces it. This coordination procedure is fairly simple to implement.

A different kind of coordination may be employed by the UM agents. Suppose that the following agents are part of that role:

- UM-1: a clique-based recommender able to infer the visitors' interests on the basis of a set of explicit or implicit ratings on already seen artworks;

- UM-2: a "visiting style" user modeling able to classify visitors with respect their way of moving in the space [34];

- UM-3: a simple recency list that stores the exhibit already seen by each visitor;

- UM-4: a database that stores user personal data.

A typical interaction with the UM role would encompass a logon procedure carried out by UM-4 (which will in turn notify the others of the identifier assigned to the visitor), as well as other requests and notifications carried on by the agents independently (such us the notifications of the ratings by UM-1, for example). Yet, there will also be some requests that necessarily have to be handled by more than one agent: for example, the request of providing the most interesting artwork for a visitor in a given position may require: (i) a coordination between UM-1 and UM-2 to select the artworks more appropriate to both the user models, and (ii) a verification with UM-3 to avoid suggesting already an artwork already seen[4].

---

[4]Of course, the POKB role may be involved in this process and this

11. CONCLUSIONS AND FUTURE WORK

This paper introduces a novel approach to designing large scale, multi-user, adaptable multi-media systems. The key element of this approach is its communication style, that moves from the traditional direct agent-to-agent interaction (possibly mediated by further agents, as in many existing multi-agent systems) to a role-to-role interaction. This change entails some specific coordination requirements; to this end, we have introduced the concept of implicit organization, which aims at structuring dynamic, self-organizing sets of agents playing a specific role in a specific context.

While the basic communication infrastructure is available, work is still in progress on the support of implicit organizations. In the short term, we plan to validate the numerous assumptions described in this paper and the guidelines written so far concerning the definition of channels, the negotiation of policies, and so on. To this end, we are developing examples of context-sensitive applications and context intermediaries, tested in interactive simulations of visits to a digitally reconstructed museum. Some preliminary results are already available [6].

Acknowledgments

12. REFERENCES

[1] Project JXTA. http://www.jxta.org/.

[2] Gregory D. Abowd, Anind K. Dey, Robert Orr, and Jason A. Brotherton. Context-awareness in wearable and ubiquitous computing. *Virtual Reality*, (3):200–211, 1998.

[3] M. Benerecetti, P. Bouquet, and C. Ghidini. Contextual Reasoning Distilled. *Journal of Theoretical and Experimental Artificial Intelligence*, 12(3):279–305, July 2000.

[4] P. Busetta. Software Integration Guidelines for PEACH. Technical Report 0207-17, ITC-IRST, Trento, Italy, 2002.

[5] P. Busetta, A. Doná, 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.

[6] P. Busetta, M. Merzi, S. Rossi, and F. Legras. Intra-role coordination using channeled multicast. Technical Report 0303-02, ITC-IRST, Trento, Italy, 2003.

[7] P. Busetta, L. Serafini, D. Singh, and F. Zini. Extending Multi-Agent Cooperation by Overhearing. In *Proceedings of the Sixth Int. Conf. on Cooperative Information Systems (CoopIS 2001)*, Trento, Italy, 2001.

[8] H. Chen and T. Finin. Beyond Distributed AI, Agent Teamwork in Ubiquitous Computing. In *Workshop on Ubiquitous Agents on embedded, wearable, and mobile devices at the First International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2002)*, Bologna, Italy, July 2002. Proceedings available at http://autonomousagents.org/ubiagents/papers/.

[9] K. Cheverst, N. Davies, K. Mitchell, A. Friday, and C. Efstratiou. Developing a context-aware electronic tourist guide: Some issu es and experiences. In *Proceedings of CHI 2000*, Amsterdam, 2000.

[10] P. R. Cohen and H. J. Levesque. Teamwork. Technical Report 504, AI Center, SRI International, Menlo Park, CA, 1991.

[11] Scott R. Cost, Yannis Labrou, and Tim Finin. Coordinating agents using agent communication languages conversations. In Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf, editors, *Coordination of Internet Agents: Models, Technologies, and Applications*, chapter 7, pages 183–196. Springer-Verlag, March 2001.

[12] ETH Zurich Department of Computer Science. Dagstuhl Seminar on Ubiquitous Computing, September 2001. Proceedings available at http://www.inf.ethz.ch/vs/events/dag2001/.

[13] Dwight Deugo, Michael Weiss, and Elizabeth Kendall. Reusable patterns for agent coordination. In Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf, editors, *Coordination of Internet Agents: Models, Technologies, and Applications*, chapter 14, pages 347–368. Springer-Verlag, March 2001.

[14] Anind K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, College of Computing, Georgia Institute of Technology, Georgia, USA, December 2000.

[15] 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), February 2001. http://www.cordis.lu/ist/.

[16] E. H. Durfee and V. R. Lesser. Negotiating task decomposition and allocation using partial global planning. In L. Gasser and M. N. Huhns, editors, *Distributed Artificial Intelligence, Volume II*, chapter 3, pages 229–244. Morgan Kaufmann Publishers, 1989.

[17] T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In Jeff Bradshaw, editor, *Software Agents*. MIT Press, Cambridge, 1997.

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

[19] Foundation for Intelligent Physical Agents. FIPA Communicative Act Library Specification. http://www.fipa.org/repository/cas.html.

[20] F. Giunchiglia. Contextual reasoning. *Epistemologia, special issue on I Linguaggi e le Macchine*, XVI:345–364, 1993. Short version in Proceedings IJCAI'93 Workshop on Using Knowledge in its Context, Chambery, France, 1993, pp. 39–49. Also IRST-Technical Report 9211-20, IRST, Trento, Italy.

[21] R. E. Grinter, P. M. Aoki, A. Hurst, M. H. Szymanski, J. D. Thornton, and A. Woodruff. Revisiting the visit: Understanding how technology can shape t he museum visit. In *Proceedings of ACM Conf. on Computer Supported Cooperative Wor k*, New Orleans, LA, 2002.

can be regarded as a coordinator among roles. It might be argued than, in this way, the POKB is in some sense part of the UM: using an organization-based approach, it is possible to avoid the pitfall and describe the situation at the proper level of abstraction.

[22] Barbara Grosz, Luke Hunsberger, and Sarit Kraus. Planning and acting together. *AI Magazine*, pages 23–33, Winter 1999.

[23] A. Hodgson, R. Rönnquist, and P. Busetta. Specification of Coordinated Agent Behavior (The SimpleTeam Approach). In *Workshop on Team Modeling and Plan Recognition at the Sixteeth IJCAI*, Stockholm, Sweden, 1999. proceedings available at `http://www.agent-software.com/`.

[24] F. Hohl, L. Mehrmann, and A. Hamdan. A Context System for a Mobile Service Platform. In H. Scheck, T. Ungerer, and L. Wolf, editors, *Proceedings of the International conference on Architecture of Computing Systems (ARCS) 2002*, volume LNCS 2299, Karlsruhe, Germany, April 2002. Springer-Verlag.

[25] Matthias Klusch and Katia Sycara. Brokering and matchmaking for coordination of agent societies: A survey. In Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf, editors, *Coordination of Internet Agents: Models, Technologies, and Applications*, chapter 8, pages 197–224. Springer-Verlag, March 2001.

[26] I. Alfaro M. Zancanaro, O. Stock. Using cinematic techniques in a multimedia museum guide. In *Proceedings of Museums and the Web 2003*, Charlotte, NC, March 2003.

[27] Joseph McCarthy. Active environments: Sensing and responding to groups of people. *Personal and Ubiquitous Computing*, 5(1), 2001. available at `http://www.inf.ethz.ch/vs/events/dag2001/`.

[28] E. Not, D. Petrelli, O. Stock, C. Strapparava, and M. Zancanaro. The environment as a medium: Location-aware generation for cultural visitors. In *Proceedings of the workshop on Coherence in Generated Multimedia at INLG'2000*, Mitze Ramon, Israel, 2000.

[29] 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.

[30] John R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1970.

[31] R. G. Smith. The contract net protocol: High level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, 1980.

[32] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[33] Gil Tidhar. *Organization-Oriented Systems: Theory and Practice*. PhD thesis, Department of Computer Science and Software Engineering, The University of Melbourne, Australia, 1999.

[34] E. Veron and M. Levasseur. *Ethnographie de l'exposition*. Bibliothque publique d'Information, Centre Georges Pompidou Paris, 1983.

[35] Michael J. Wooldridge. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, February 1997.