Talking Agents: A distributed architecture for interactive artistic installations

José M. Fernández and Juan Pavón∗
Department Ingeniería del Software e Inteligencia Artificial, Facultad de Informática, Universidad Complutense de Madrid, Madrid, Spain

Abstract. Recent advances in Artificial Intelligence imply challenges and opportunities to explore new kinds of artistic experiences in interaction with the spectator of an artistic installation. In this context, a wide range of elements, such as sensors and speech recognition and synthesis, has been considered to establish an intelligent environment with an artistic purpose. The coordination of these elements to create an interactive environment where the spectator may feel faced with some kind of human-like behavior (the use of speech contributes to this goal) requires the integration of different Artificial Intelligence techniques. The definition of Talking Agents as reusable components for managing coordination of this diversity of elements and the interaction with the spectator facilitates building and setting different artistic scenarios. This paper describes the architecture of Talking Agents and their implementation in a multi-agent framework. This has been used in an experiment with an artistic installation called ORACULOS, and the results have been the basis for considering the evolution and application of the Talking Agent architecture in Ambient Assisted Living scenarios.

Keywords: Talking Agent, ambient assisted living, agent architecture

1. Introduction

The context of this work is the development of reusable components for building artistic installations that are able to interact with the spectator using different media, especially speech but also including different types of sensors and actuators in an intelligent environment. These components should also be able to integrate different Artificial Intelligence techniques for performing a series of interactions with the spectator throughout a sequence of situations, each one a product of a specific context in which the spectator participates. In order to facilitate such integration in a distributed setting, an agent-oriented approach has been followed. The artistic installation is conceived as a set of basic elements, each of them with a physical representation (a golden head sculpture, for instance), and a computational entity, which is called a Talking Agent. A Talking Agent is a reusable software component, which is characterized as having autonomous and social behavior. It has the ability to interact with humans using speech or other communication channels, and to work in a distributed computing environment where it can communicate with other agents and manage resources. In most scenarios, Talking Agents use heterogeneous resources, which vary depending on the needs of each particular artistic installation. For example, in the ORACULOS installation, which is described later, speech recognition, processing and synthesis resources are needed, as well as a variety of sensors in the environment, in order to obtain knowledge of what the spectator is saying and doing.

Talking Agents in an installation are organized as a society of agents that can cooperate and interact among themselves. Depending on the installation, the spectator will interact with each Talking Agent one by one, or with several at the same time. Communication among Talking Agents can help them to obtain more information about the context (e.g., what the spectator is doing and has said) and help each other to better manage the interaction with the spectator. Also, each Talking Agent...
Agent may implement a different strategy for conversation with the spectator, and the result of the spectator experience will emerge from the combination of the responses given by the Talking Agents participating in the installation. It is important also, from a conceptual viewpoint, to explore social issues (Talking Agents as a society with emergent behavior) and the role of the human spectator in such virtual societies.

The main focus of this research is not the development of the actual resources used in each scenario, such as speech recognition and synthesis techniques, which have been implemented using existing software components. Instead, research has been done in the design and implementation of the agent architecture that facilitates the evolution of Talking Agent capabilities, since they should be easily configurable to use different kinds of resources as needed for each specific scenario and to integrate various Artificial Intelligence techniques. In this sense, the Talking Agents are reusable components for building different artistic scenarios.

Several agent frameworks were considered for implementing Talking Agents, and finally ICARO [30] was adopted. It is an open source project that promotes the development of distributed applications as organizations of agents and resources. In comparison to more popular agent frameworks (e.g. Jade, Jack), ICARO has the advantage of promoting an organization-based view of the multi-agent system, being flexible in the communication mechanisms and protocols used among agents (not forcing, for instance, the use of FIPA standards). In addition, ICARO provides support for managing a multi-agent system configuration in a similar way to component-based frameworks, which is useful in terms of its ability to cope with our requirement on the flexibility of resource configuration of Talking Agents. Another interesting facility of ICARO is the provision of two agent patterns, one for reactive agents, based on finite state machines, and another for cognitive agents, to facilitate the specification of agents with reasoning and planning capabilities. The first pattern was used as a basis to define the Talking Agent, which is itself a kind of pattern, as it can be configured for different systems.

The rest of the paper is structured as follows. Section 2 discusses the similarities of Talking Agents and dialogue systems, by taking into consideration the trends in this field and explaining the activities that are involved in speech interaction. This establishes the ground for the architecture of a Talking Agent that is described in Section 3. Section 4 presents the multi-agent architecture setting with the ICARO platform, and identifies other types of agents that help Talking Agents in their configuration and coordination. This architecture has been used in an experiment with the ORACULOS installation, which is described in Section 5. The last section presents some conclusions derived from the experimentation and considers the evolution and application of the Talking Agent architecture for Ambient Assisted Living scenarios.

2. Dialogue systems and Talking Agents

The main concern of the research with Talking Agents is the development of a flexible multi-agent system architecture that facilitates different kinds of interaction with humans, taking advantage of previous agents’ experience and information from the environment (e.g., from sensors). The research is thus similar in several aspects to the multimodal dialogue systems field. This section briefly reviews developments in this area, with a focus on representative dialogue system architectures, and finishes with a summary of the main concepts and associated technologies. This will serve to explain the motivation and main challenges addressed by the Talking Agent architecture and its possible implementations with respect to other solutions.

2.1. Evolution of dialogue system architectures

Earlier dialogue systems consisted of simple information pipelining through different abstraction levels, without explicit treatment of the dialogue context. The main phases of this pipeline were: input perception, input analysis to determine speech action, management and decision about interaction, response generation, and response execution. This is basically the approach of Eckert et al. [19], which is often used as an example because of its simplicity. This has been extended in [3], with new elements such as discourse context manager, reference manager, and content planner. And improved later in [4], which makes the organization clearer by distinguishing three sections: interpretation, generation and behavior.

A further step in structuring dialogue systems is the explicit distinction between interaction-level communication and content-level communication [28]. The former refers to the communication itself and it is used to manage certain parameters such as talking speed, turn, and form of the communication. The content-level communication is the main communication and refers to the objective of the communication.
Other approaches take the pipeline architecture as a basis and introduce improvements in the different phases for some specific processing. For instance, Casell [15] proposes managing the interaction-level communication using an improved interpretation module that is able to discriminate between interaction and content events.

Other architectures take into account more aspects than dialogue management, such as distribution of components so that they can be used as services. This idea is commented in [23] and it is considered in Talking Agents as well.

The Olympus Architecture [10] also follows the pipeline pattern, with components for processing the information in each phase, which are independent of each other and are developed separately. This separation of phases and the independence of the modules are similarities with the Talking Agents architecture. However, Olympus focuses on verbal and text-based communication, leaving other types of interaction to internal applications. With Talking Agents, the interpretation of the inputs may vary depending on many types of stimuli, including speech, so they should be considered all together in the pipeline. Another difference with Talking Agents is that Olympus does not consider collaboration among distinct agents in the system.

The TRINDI architecture [27] follows the Information State Update paradigm, which considers the definition of rule modules (application components) of two different types: the Dialogue Move Engine modules, and the others. All the rule modules can read information from the global information state and write their results there. The Dialogue Move Engine is responsible for triggering the corresponding update rules of each other module when the necessary information is available, in order to coordinate the processing of the modules and decide on the actions. This approach permits the addition of new components dynamically, and it is not as focused in verbal and textual communication as Olympus, since it permits the addition of new arbitrary modules that interact with the global state, and the corresponding Dialogue Move Engine modules to coordinate them, without altering the rest of the system. However, one aspect that is not considered in this architecture is collaboration among different agents in a greater system, which the Talking Agents system does consider.

2.2. Associated concepts

When reviewing the different dialogue system architectures, there are common activities for information processing. Each of these can exploit distinct technologies, depending on the actual requirements of the system. These activities are the following:

2.2.1. Input gathering

This activity processes the signals coming from the environment and generates messages or events to the system. These are examples of technologies used:

- **Speech recognition**: MS Speech API [18], Java Speech API [36], Sphinx-4 [14], TalkingJava [16].
- **Visual speech recognition**: several experimental results are described in [40] and [29].
- **Graphical user interfaces**: Java Swing, AWT, acm.gui [1].
- **Physical-magnitude sensors**: Phidgets APIs [33].

2.2.2. Input processing

This activity processes the information that results from the previous activity and determines a domain-specific meaning for the input or sequence of inputs, possibly making use of the context of the interaction, created through previous inputs. These are examples of technologies used:

- **Natural language analyzers**: FreeLing [5], Stanford Parser [26] (only analyzes the syntax). Phoenix [39] and SPIN [21] are able to work with semantics within a certain domain.
- **Confidence annotation of recognition results**: Helios [8].
- **Case-based reasoning for previous identified input cases**: JColibri2 [2].
- **Rule-based reasoning for mapping inputs into acts**: Drools [6].

2.2.3. Control

This activity is present during the whole process. It manages the information flow during the interaction. Its definition is one of the main differences among different architectures.

- **Dialog system middleware**: DIPPER [11].

2.2.4. Dialogue management

A dialogue is a prolonged interaction in which references to previous information and context are made. In this activity the system decides the answers depending on the dialogue being maintained. To make this decision, information from the previous interactions is accessed and updated according to the new situations. This activity also contributes to the differences among different architectures. These are examples of paradigms used:

- **Information state update (ISU)**: TRINDI [27].
- **Expectation agenda**: RavenClaw [9].
2.2.5. Output generation

This activity takes the text generated by the dialogue manager and decides how to coordinate the available resources in order to produce the response. These are examples of technologies used:

*Case-based reasoning:* JColibri2 [2]. In [20] a way of using this type of technology is described for dialogue systems and in [17] for deliberative agents.

*Rule-based reasoning:* Drools [6] is a rule execution engine. TRINDI [27] is an example of the use of this technology.

*Task decomposition:* RavenClaw [9].

*Context-aware processing:* Used for adaptive services [22,31].

2.2.6. Output handling

This activity makes the specific requests to the resources in order to produce the outputs. These are examples of technologies used:

*Speech synthesizers:* Microsoft Speech API [18], Java Speech API [36], FreeTTS [35], TextSound [12].

*Graphical interfaces:* Java Swing, AWT, acm.gui [1].

Taking into account the diversity of technologies a dialogue system can include, it is important that the software architecture be flexible enough for its components to be changed. Furthermore, some of these components are demanding of computing resources, so it is interesting to consider the ability to distribute the parts of the dialogue system to adapt to the capabilities of devices.

3. The Talking Agents architecture

Talking Agents are not isolated entities, as they are part of a multi-agent system. This is one of the main characteristics with respect to typical dialogue systems, as Talking Agents cooperate together and with other agents to achieve their goals. The organization of the multi-agent system determines the global system architecture, which is described in Section 4. Also, each individual agent has its own architecture, with well-defined component interfaces. This section describes the architecture of the Talking Agent, first by showing the static view of the system, i.e., the relationships among its components, and next the dynamic view, to show the interactions among components in order to achieve the Talking Agent objectives.

3.1. The Talking Agents structure

Figure 1 is a simple diagram that shows the components that make up a single Talking Agent as a conversational agent: perception, interpretation, planning and execution components, and the Talking Agent core. The diagram shows the components used in the ORACULOS scenario (including speech recognition, sensors and speech synthesis). Each component implements a well-defined generic interface, depending on the previously identified activity it performs, thus enabling the Talking Agent to manage an interaction of any kind (multimodal), in a generic manner, by customizing the respective components. The Talking Agent core is responsible for deciding (using a rule engine) the appropriate intention given certain knowledge, and for coordinating the available resources, using a reactive control model (from the ICARO platform, as described later).

3.1.1. Perception (input gathering)

A perception component is a thread that generates events with some information, either when it detects certain situations for which it is prepared or in a periodic manner, depending on the type of stimuli and the way it expects to obtain the information.

The first type is called *eventual perception*, where a single state change provides meaningful information. The *on demand* perception (i.e., the information that is accessed when the agent makes a request for it, in the execution phase) and the communications from external agents are considered to be of this type.

The second type is called *periodical perception*, where the perception does not offer enough information in just one state change, so it is necessary to observe several state changes during a certain period in order to have enough information to process. The list of state changes notified over time is considered a block for interpretation purposes.

An example of an eventual perception element used in the ORACLES installation is the Speech Recognition, which is implemented in Talking Agents using TalkingJava [16], which notifies an event when it recognizes a speech, and it attaches the recognized text to it. An example of a periodical perception element is a Sonar Sensor, with the Phidgets API [33], which periodically reports the value detected, generating sequences of values, to enable the system to determine if someone is approaching or moving away.

Any perception component must implement an interface that has the following methods (only the main operations are shown):
AddSubscription(Subscription): adds a subscription object to the component. This object contains the information necessary for the component to notify a subscriber component when a change has occurred in a certain subject in the context of the perception component.

RemoveSubscription(SubscriptionId): removes a subscription object on the component.

NotifySubscribers(Subject, Info): notifies the subscribers when a change has been perceived in a certain subject, with associated information, by sending the AcceptPerceptionEvent signal to the subscribers, as will be described later.

The Subscription object contains the following information:

SubscriptionId: a code identifying the subscription.
SubscriberId: a code identifying the subscriber component, in order to be able to send it the event object.
Subject: a code representing the subject for which the subscription is made.
NotificationPeriod: the amount of time between notifications, in the case of periodical perception.
WindowSize: the number of grouped events sent at the same time, in case of periodical perception (used to interpret sequences of events).

3.1.2. Interpretation (input processing)

The interpretation elements take the events generated by the perception elements and try to extract the meaning of the information. The interpretation process is divided into different dimensions: specialization and memory. In terms of the specialization, interpretations can be:

Local: interprets events coming from a specific source, ignoring the rest.
Global: interprets the local interpretations from various sources (i.e. constructs a higher level interpretation based on lower ones).

And, in terms of the memory, interpretations can consider:

Current events: interprets only the current events in the system, ignoring previous ones. This interpretation is made for eventual perception events.
Sequence of events: interprets a sequence of events within a time window. This interpretation is made for periodical perception events.
Previous interpretations: when taking into consideration interpretations of previous events or, in general, the current mental state of the agent (i.e. the information representing the knowledge available to the agent and the objectives it is currently pursuing).
The result of the interpretation is a certain amount of knowledge the agent can use in its decision process. The knowledge is represented in a symbolic manner so that it can be processed by a reasoning engine such as a rule-based reasoning engine.

The interpretation of a heterogeneous set of situations fits well with the idea of case-based reasoning, because this kind of technology can identify previous cases and associated interpretations from complex amounts of information, and adapt those interpretations to the current case. For this reason, a generic pattern for global interpretation using previous interpretations has been developed using this paradigm. This pattern is implemented using the JColibri2 tool [2] and it is used in the ORACULOS implementation.

The local interpretations for current events or sequences of events must be implemented using specific processing for each kind of perception element. For instance, events in the Speech Recognition resource are interpreted using natural language understanding technologies like FreeLing [5], Phoenix [39], or SPIN [21].

Any interpretation component must implement an interface that has the following methods. However, it is possible to customize the existing JColibri2-based [2] pattern with the appropriate cases in order to adapt its behavior to the desired global interpretation.

- **UpdateMentalState(MentalState,AgentId)**: updates the mental state associated with an agent in the interpreter component, in order to include the last perceived events and other available knowledge, and triggers a new interpretation based on the new information.
- **SendInterpretation(AgentId)**: sends the UpdateMentalState signal to the agent, if the interpretation process has changed the mental state. Then changes to the execution phase.

The MentalState object contains the following information:

- **EventsBuffer**: a buffer containing the non-processed perception events.
- **Beliefs**: the collection of agent’s beliefs, including certain facts (e.g., configuration parameters). These objects are used to trigger the decision rules.
- **Intentions**: the set of an agent’s intentions that are not yet carried out.
- **ActionsBuffer**: a buffer containing the pending actions.

### 3.1.3. Decision: Talking Agent core

This is the part in which the agent coordinates its managed modules and its functional cycle and decides which immediate objectives to pursue, depending on the knowledge of its own state and the environment (the agent’s beliefs). The decision is performed at two different levels: control and intention.

At the control level, the agent only decides which phase of the information flow is going to be processed next. This decision process is modeled as a state machine following the reactive agent behavior model (from the ICARO framework), which controls each of the functions performed by the agent at every moment. This kind of state machine reacts to certain received signals, which are generated by the resources, indicating the completion of each processing. Although it is recommended that the state machine provided be used in order to implement the Talking Agent behavior, one may modify it to address certain concrete requirements.

The main signals accepted by this state machine are the following:

- **AcceptPerceptionEvent(PerceptionEvent)**: in the perception phase, it includes the new perception event in the events buffer of its MentalState object and calls the UpdateMentalState method of its interpretation component. Then changes to the interpretation phase.
- **AcceptInterpretation(NewMentalState)**: in the interpretation phase, updates the mental state of the agent and runs the rule engine, as we will see later. Then changes to the decision phase.
- **DecisionFinalized**: in the decision phase, this signal indicates that the rule engine has ended its processing. The UpdateMentalState and GeneratePlan methods of the planning components are called. Then changes to the planning phase.
- **AcceptPlan(Action)**: in the planning phase, includes the new action in the actions buffer of its MentalState object and then calls the Execute method of the action, as we will see later. Then changes to the execution phase.
- **ActionFinalized**: in the execution phase, this signal indicates that the action has been executed. The ObtainResults method of the action is called, updating the mental state. Then changes to the perception phase.
- **NoMentalStateChange**: in any phase, changes to the perception phase.

At the intention level, the agent decides its next immediate objectives, or intentions. The knowledge used
for the process obviates irrelevant low-level details, such as concrete words used by the user, the frequency of his voice, etc. Similarly, the generated decision also obviates those details, only expressing the agent intentions from an abstract point of view. The concrete details depend on the lower levels. This is implemented using the Drools [6] rule engine, which produces a set of intentions based on a set of input beliefs, corresponding to the specified rules that model the agent behavior. An example of the definition of these rules is shown in the Appendix. With this formalization, the beliefs (which include beliefs and facts), obtained from interpretations, and distinguished by the code field, are put into the when part, thus triggering the then part at the moment they are identified, executing the creation of certain intention objects, which will determine the next plans.

After each execution of the rule engine, a decision-finalized signal is sent to the state machine of the agent if there has been any change in the mental state, or a noMentalStateChange signal otherwise.

3.1.4. Planning (output generation)

The planning elements take the intentions generated in the previous phase and create a set of concrete actions to be carried out.

Basically, the intentions are broken down into actions through several iterations, starting from the more abstract, generic actions to the more specific, concrete actions, which will trigger operations on the associated execution resources.

A general planner makes use of more specific planners that plan the sequence of actions for a particular actuator. This tree-like process fits well with the idea of rule-based reasoning, so a generic planner has been implemented for the Talking Agents architecture using the Drools [6] rule-based reasoning engine. An example of definition of these rules is shown in the Appendix. With this formalization, the intention objects are put into the when part, thus triggering the creation of action objects, which will execute the plan.

Any planner component must implement an interface with the following methods:

UpdateMentalState(MentalState, AgentId): updates the mental state associated with an agent in the planner component, in order to include the last generated intentions and other available knowledge.

GeneratePlan(AgentId): generates a list of actions associated with the specified agent’s mental state and sends the results to the agent.

The actions generated are objects of any of the following types:

Atomic actions: actions that call operations on execution resources directly. Sending a signal to an external agent is considered an atomic action.

Specialized actions: actions that create a Helper Agent as a result. A Helper Agent is a special kind of resource, which has its own initiative and can communicate the progress of its task to the parent.

Script of actions: any arbitrary script of some of the other types of actions.

An action object has the following methods:

Execute: either triggers the execution of an operation on an associated execution resource, creates a Helper Agent, or executes a sequence of actions, depending on what type of action the instance is.

ObtainResults: retrieves the initial feedback of the action.

3.1.5. Execution (output handling)

This part is responsible for performing the previously generated actions. The immediate results of the execution of the actions generate an initial feedback that can be used as knowledge in the decision process.

An example of an execution element used is the Speech Synthesis, which is implemented using TextSound [12] as explained in Section 4.2.3.

An execution component may implement any arbitrary operations, which will be called by a corresponding Action object.

3.1.5.1. Helper Agents

Sometimes, certain actions require continuous reception of feedback from the environment and successive tuning or some special treatment that would consume the attention of the agent. For example: complex communication protocol handling with another agent, or a specific sub-interaction with a user.

For these cases, Helper Agents can be used. Helper Agents are like normal agents with two differences:

- Their objectives are very concrete: they come into life with a very specific mission that must be accomplished.
- Their life is limited: when their objective is achieved, they self-destruct, informing their creators.
An example of a Helper Agent that is being developed is one whose objective is to guide a spectator toward a certain point of the room in order to begin the interaction. This agent would use the sensors of the room to guess the user position and the speech synthesis to give instructions until the user reaches the correct position.

3.2. Dynamic view

As can be deduced from the previous section, the information flow established between system and user runs through different abstraction levels, as shown in Fig. 2. The information exchange generated is controlled by the Decision component at the control level, using signal sending. The abstraction levels are named as:

- **Physical**: the physical form of the environment information: sound waves or electromagnetic signals.
- **Machine**: the information handled by certain software relative to input and output events.
- **Knowledge**: the machine information condensed for representing certain useful concepts for decision-making.

These abstraction levels are crossed in both directions, depending on the direction of the information flow. When an agent makes the decision to communicate something, it generates the knowledge in the form of an intention. That intention only contains information on what to do, but not on how to do it. The planning module transforms the intention into a set of actions, which specifies the way to achieve the immediate objective. Finally, the execution module executes the sequence of actions in the world.

In the other direction, when an agent receives physical information, e.g., sound waves, a module takes that information and transforms it into machine information, which can be handled by the agent. Later, the interpretation module obtains the relevant knowledge of the machine information. That knowledge can now be used in the decision making process of the agent.

Figure 3 shows the complete processing sequence performed by the Talking Agent, which coordinates the information flow through the different resources.

4. Multi-agent system architecture

Talking Agents collaborate among themselves and with other agents in the ICARO agent framework [30], which provides communication and management services that facilitate the distribution, configuration and monitoring of the multi-agent system. The use of this framework implies some constraints in the way Talking Agents are built and deployed, with the advantage of relieving the developer of implementing certain distribu-
tion and management concerns. In order to understand the organization of the Talking Agents system, first we present the ICARO framework and then how Talking Agents are part of an ICARO agent organization.

4.1. The ICARO framework

ICARO is a framework for building distributed applications, which are conceived as organizations of two types of entities: agents and resources. ICARO provides services for such an organization, in order to facilitate its distribution, configuration and monitoring. A system implemented with ICARO consists of the following types of entities:

**Agent**: represents an entity that can manage information flows. It can send commands to resources, receive their information and distribute it to other agents and resources. Each agent has a behavior model, which defines the way it will act in each circumstance. Each behavior model provides a pattern for specification and control of an agent’s behavior. Currently there is a reactive model, which is based on a finite-state automaton, and a cognitive model, based on the definition of a set of rules and a knowledge base.

**Resource**: represents an entity that performs operations on demand by some other entity, an agent or other resource. These operations usually satisfy the functional requirements of the application.

**Information**: represents an entity that is part of the application’s domain model. Information entities are used either to structure information, to store information, or to perform certain business tasks. They are managed by both agents and resources and they basically make up the information flow of the system.

**Description**: represents an entity that describes how the agents and resources are organized in the system, i.e., their dependencies and deployment.

In addition, ICARO provides management facilities, which establish that every agent or resource within the system is a manageable element. A manageable element offers an interface to perform management operations on it, such as start, shutdown, pause or testing. They are used to maintain the system’s integrity and have nothing to do with application’s functional requirements. Because of this, the management operations are usually handled by a set of pre-defined agents called **managers**. These agents have the responsibility of putting the system to work in the first place, i.e. allocating necessary resources, instantiating agents and resources; checking and maintaining the integrity of the system; and possibly stopping the system.

4.2. Talking Agents in an ICARO agent organization

The Talking Agents system, as a society composed of several autonomous entities, has been designed as a multi-agent system in which each agent has the capa-
bility of interacting with humans, as a conversational agent, but also with the other agents, as in a typical multi-agent system. This inter-agent communication is necessary for coordination purposes since agents may collaborate to achieve certain objectives. In the use case considered, the objective is to create the illusion for the spectator that they are within a society.

The Talking Agents architecture follows the ICARO organization paradigm, which already manages the distribution of components and their dependencies. With this paradigm, each component is considered an agent if it has a certain level of autonomy or a resource if it only reacts to operation requests. Following this approach, Perception, Interpretation, Planning and Execution modules are considered as resources, and the coordination and orchestration of these resources are done in the Talking Agents, which use Decision modules. In addition, these components coexist with others that are predefined within the framework, the above-mentioned managers, which are responsible for management tasks in the system. The system view for the ORACULOS scenario is shown in Fig. 4. The dependencies among the components are specified using organization description artifacts from the ICARO framework.

The architecture can be divided into three types of entities, which are explained below.

4.2.1. Manager agents

**Organization manager:** this agent is responsible for configuring, initializing and monitoring the whole system. It is able to detect problems in the initialization process due to poor functioning of components or configuration. It delegates to Agent Manager to manage the agents and to Resource manager to manage the resources.

**Agent manager:** This agent is responsible for configuring, initializing and monitoring the application agents. It is able to detect problems in these processes and to notify the Organization Manager of them.

**Resource manager:** This agent is responsible for configuring, initializing and monitoring the application resources. It is able to detect problems in these processes and to notify the Organization Manager of them.

4.2.2. Application agents

**Talking Agent:** This is the main agent of the system. It is implemented using the reactive agent pattern defined in ICARO, so its control model is defined as a state machine, sensitive to certain signals, which coordinates the information processing through the different resources and communication with other agents. There are three instances of this type of agent running on the ORACULOS system. Its implementation is based on the Decision component.

4.2.3. Application resources

**Interface Phidget Kit Resource:** this resource is able to process the continuous signals received from different sensors about the environment in order to periodically generate the corresponding sequences of events containing the sensor values, which will be sent to the subscribing agents. It is implemented using the Phidgets API [33].

**Recognition Utility Resource:** this resource is able to process the sound waves detected by a sound input device that correspond to certain discourse in order to recognize the pronounced words. It then generates the corresponding events containing that information, which will be sent to the subscribing agents. It is implemented using the Microsoft Speech [18] implementation of JSAPI [36].

**Oracles Interpreter Resource:** this resource consists of distinct components: those that can interpret the distinct perception events locally (sensors and discourse), obtaining partial knowledge about the interaction; and those that can consider that partial knowledge globally and, considering the previous interactions and the current state of the agent, can make the correct interpretation. This resource uses the case-based reasoning pattern implemented with JColibri2 [2].

**Oracles Planner Resource:** this resource is able to determine how the intentions of the agent will be performed. In this case, it constructs the text that will be said by the agent in order to express the answer it wants to communicate. This resource uses a case-based reasoning system that contains a base of discourses that are adjusted according to the specific situation of the agent. It is implemented using JColibri2 [2]. This component is specific for the concrete application, in this case for the discourses that the particular installation requires.

**TextSound and SoX Resource:** this resource is able to transform the text provided as input into spoken words, which are played through a particular sound output device. It is implemented using TextSound [12], which is one of the few text-to-speech engines that work properly in the Spanish language.
4.2.4. Inter-agent communication model

The inter-agent communication capability is the main feature that distinguishes the multi-agent architecture view from the single-agent architecture view. This feature has been integrated into the latter in the following way: any agent is able to send an AcceptPerceptionEvent signal to another agent, as if it were a perception resource. This way, the receiver agent is able to process the communication in the same way it processes any other perception event from the environment. The concrete protocols used to establish collaborations are implemented when defining a concrete system using this principle.

5. The ORACULOS installation

ORACULOS is an interactive artistic installation that creates the conditions of an oracle receiving the spectator to give him or her advice on the future. The term oracle designates either the asked divinity, the human mediator who transmits the response, the sacred place or the given response. Each spectator will have to individually face the oracular trial. Three art elements with a golden human head, internally made up of three Talking Agent instances representing the oracles, receive the spectator (only one is allowed to enter at a time). Each element recognizes the spectator’s speech (TalkingJava [16]), generates spoken responses (TextSound [12]), and can detect his or her presence through distance sensors as well (Phidgets API [33]). The spectator moves from one oracle to another (in different rooms), being asked by each one to be more specific or to redefine the inquiry to receive a final response (see Fig. 5). Apart from direct interaction with the spectator, the Talking Agents collaborate among themselves to achieve greater accuracy in guessing the intentions of the spectator, by sharing the information achieved with each question. Thus, each time the spectator is asked, the oracle generates new knowledge to be used by the others, generating multiple feedback stimulation.

Fig. 4. ICARO view of the system.
The main objectives that led the development of this scenario, from a technical point of view, and the approaches to achieve them, were the following:

- Refine the architecture, by following an iterative and incremental process for the development of the Talking Agent architecture.
- Prove the flexibility of the architecture for integrating different resource types and control techniques.
- Analyze the emergent behavior of the Talking Agents, by defining a simple communication protocol among the agents so they can share their interpretations of the dialogue with a user (agents send a message with that information to other agents), in order to observe how an agent develops a behavior due to its coexistence in a multi-agent system that otherwise would be different.
- Fulfill the requirements of the scenario, by implementing each part of the system, especially the interpretation, decision and planning rules.

The achievement of the last three objectives would prove the success of the Talking Agent architecture. The discussion of how these objectives have been achieved is in Section 5.4.

It is important to note that the main research of this work has not been aimed at natural language processing. Consequently, the system, apart from the speech recognition and synthesis tools, performs simplistic processing of the user’s discourses and system responses, using similar principles from chat bots, like Cleverbot [24], or certain videogames, which just detect certain keywords in the discourse to infer some basic knowledge and generate response (in this case, the main topics of the questions). Instead, the objective of the scenario is to prove the success of the architecture by testing its capability to coordinate several different resources (speech recognizer, sensors, and speech synthesizers) as well as the communication and collaboration among the agents.

5.1. Main scenario

This scenario represents the normal functioning of the system. The intention is to make the spectator travel through the installation, and interact with three oracles one by one, in different rooms, as shown in Fig. 5. This scenario shows the conversations of a spectator with the Talking Agents (Oracles 1 to 3):

**Spectator:** (enters the room and activates the sensors, generating an event that is interpreted as a new spectator arrival, provided that the agent is waiting for a spectator)

**Oracle 1:** Speak, human! Would you like to know about health, money or love?

The knowledge of the entrance of a new spectator triggers a rule in the agent that creates the intention to say a welcome message and ask an initial question. This text is generated by the planning resource and later spoken by the speech synthesis resource.

**Spectator:** Umm... let me think... about health

The spectator’s speech is recognized by the speech recognition resource, which generates a speech event. This event is later interpreted by the interpretation resource, which extracts the word “health”, generating the knowledge of the topic of conversation.

**Oracle 1:** Speak, human! Would you like to know about health, money or love?

The knowledge of the entrance of a new spectator triggers a rule in the agent that creates the intention to say a welcome message and ask an initial question. This text is generated by the planning resource and later spoken by the speech synthesis resource.

**Spectator:** Umm... let me think... about health

The spectator’s speech is recognized by the speech recognition resource, which generates a speech event. This event is later interpreted by the interpretation resource, which extracts the word “health”, generating the knowledge of the topic of conversation.
Oracle 1: Everyone was healthy before becoming ill. Will you go ahead into the next room to find out more about your health?

Using the knowledge of the spectator’s previous answer, the agent again triggers a rule to ask a question related to the topic, the text of which is again generated by a planning resource using a predefined text database, and spoken by a speech synthesis resource.

Spectator: Yeah, of course

Oracle 1: Continue, visitor

Spectator: (enters the next room and activates the entrance sensors for that room)

Oracle 2: Tell me, human, what is the problem that threatens your health?

The knowledge of the new spectator arrival triggers a rule in the agent that creates the intention of asking the other agents for knowledge, thus obtaining the information from the previous agent that the subject of the question was Health.

Spectator: It’s... my back, it’s killing me!

Oracle 2: Your body is an imperfect machine. You are not yet prepared to hear our advice. Continue to the next room.

Spectator: (reaches the last room and activates the sensors)

Oracle 3: (after receiving information from the previous Oracle that the problem is “BackProblem” as in the last user discourse) You wasted the opportunity of gaining insight just asking about backs, did you? What is the motivation for that question?

Spectator: Umm... let me think... hmm... I don’t know

In this case, the agent doesn’t give a set of options. The spectator could use any word related with health, money or love in order to determine the topic, but he had initial doubts. With his answer, the interpretation resource can’t extract useful knowledge, so the agent decides to ask again.

Oracle 1: Your ideas are confusing. Try to reformulate your answer. Are you worried about health, money or love?

Spectator: It’s complicated, what am I supposed to say?

Once again, the answer is not useful. The agent decides that it won’t give more opportunities so it finalizes the conversation.

Oracle 1: The art of questioning is not as easy as it seems. It is necessary to know many things to be able to question what is not known... Leave and come back when you are ready.

Spectator: (leaves the room).

5.2. Secondary scenario

In this scenario, the spectator does not pass the first oracle, as the Talking Agent is not able to recognize what the spectator is talking about. This can occur for different reasons: the spectator does not speak correctly, he uses very strange terms, or he simply does not collaborate with the oracles. The Talking Agent therefore generates automatic sentences to ask the spectator to reformulate his question a limited number of times. In the following example, the number of attempts the spectator can make is limited to two. The intention is to prevent a single spectator from monopolizing the installation and to allow other spectators to experience it.

Spectator: (enters the room and activates the sensors, notifying the agent of his presence)

Oracle 1: Speak, human! What is the main topic that worries you?

Spectator: Umm... let me think... hmm... I don’t know

In this case, the agent doesn’t give a set of options. The spectator could use any word related with health, money or love in order to determine the topic, but he had initial doubts. With his answer, the interpretation resource can’t extract useful knowledge, so the agent decides to ask again.

Oracle 1: Your ideas are confusing. Try to reformulate your answer. Are you worried about health, money or love?

Spectator: It’s complicated, what am I supposed to say?

Once again, the answer is not useful. The agent decides that it won’t give more opportunities so it finalizes the conversation.

Oracle 1: The art of questioning is not as easy as it seems. It is necessary to know many things to be able to question what is not known... Leave and come back when you are ready.

Spectator: (leaves the room).

5.3. Alternative scenario

This scenario represents another possible deployment of the Talking Agents and their resources, which is interesting for observing the emergent behavior of the system. In this case, there are three instances of Talking Agents, but only one instance of each resource: distance sensor, speech recognizer and speech synthesis. Thus, there is an agent that can only “see”, another that can only “listen”, and another that can only “speak”. Consequently, the agents must collaborate in order to create a real interaction with the spectator.
**Spectator**: (enters the room and activates the sensor, generating an event that is sent to the “seeing agent”).

**Seeing agent**: (receives the sensor event, and interprets a new spectator arrival. Then it sends this interpretation to the other agents).

**Speaking agent**: Speak, human! Are you worried about health, money or love? (communicates end of speech to the other agents).

Since the Speaking agent has received the interpretation of the arrival of the spectator from the Seeing agent, it is able to act consequently and generate the welcome discourse.

**Spectator**: I am completely in love with my neighbor...

**Listening agent**: (receives the speech of the spectator, and interprets the “Love” topic. Then communicates it to the other agents).

**Speaking agent**: Love is a beautiful flower that lies at the edge of a precipice. Now, leave (communicates end of speech to the other agents).

The knowledge obtained by the Listening agent has been used by the Speaking agent in order to create the corresponding answer for the spectator.

**Spectator**: (leaves the room).

### 5.4. Results discussion

The development, execution, and results analysis of the scenarios have contributed to the achievement of the previously identified objectives in the following ways:

**Refine the architecture**: The architecture definition and especially its implementation have been improved in the iterative process of development. The specific details of this process are not of interest in this paper.

**Prove the flexibility of the architecture**: Speech recognition, distance sensors and speech synthesis resources are uniformly integrated in the system as perception and execution components, since they implement the required interfaces. Considering this, it is possible to change these resources including, for example, a GUI to replace them all without changing any other part, and to obtain different input and output types. A text input element in the interface could act as the speech recognition resource, sending the same events. A simple slider could act as the distance sensor, and a textual output could act as the speech synthesis. As it is possible to do this without altering the rest of the system, the flexibility is proven.

**Analyze the emergent behavior**: The coexistence with other agents gives an agent an additional source of information, thus creating the possibility to take advantage of the other agent’s experience, which depends on its own environment. This way, an agent may take actions based on information that it otherwise would not be able to obtain, which is a form of emergent behavior.

In the main scenario, the absence of this information sharing would make the different rooms of the artistic installation completely independent, and it would not give the sensation of undergoing some kind of “trial”, which is the intention of the scenario.

In the alternative scenario, the individual agents are completely unable to create an interaction with the spectator, but working together, they obtain this new capability that none of them had before. The whole is greater than the sum of its parts.

**Fulfill the requirements of the scenario**: The requirements of the scenario determine the following characteristics:

- The event types the perception resources have to be able to detect. In this case, speech perception events and distance sensor events are detected.
- How a collection of events has to be interpreted by the agent in order to identify a certain interaction type. In this case, the case-based interpretation pattern is customized in order to detect each combination of beliefs and events to determine the correct interpretation.
- How the agent must act, depending on the interpretation of the environment. The Decision module is implemented with the correct rules that indicate the intentions based on the beliefs.
- How the intentions of the agents are broken down into actions in order to obtain the desired effect. The Planning module is implemented with the correct rules to achieve this.

Considering the previous characteristics of the system, it is possible to fulfill the requirements of the desired scenario in the context of the architecture.

It is important, considering the previous point, to clarify that the main application of the Talking Agents architecture is the development of intelligent environments where there is a need to process arbitrary interaction types. Although the proposed scenario does not consider the existence of many concurrent interactions, special attention to synchronization issues may be necessary for other scenarios, as discussed at the end of the conclusions.
6. Conclusions and future work

Although the initial purpose of the Talking Agents architecture is the development of interactive artistic installations, its generic definition also enables it for developing other types of distributed systems that include user-interaction through different channels. The different technologies of each phase of the interaction can be integrated in a generic way so that each component may be replaced or extended in order to change the system behavior. This was one of the main issues to validate with the installation art scenario.

The **ORACULOS** scenario is defined with the intention of making use of several simple resources and the capability of sharing information among the agents, in this case to share the conversation topic detected during the interactions of the spectators with the different talking agents. This information can be used to improve the interpretation made by resources such as speech recognition software. In this sense, the scenario shows the interest of a collaborative architecture. The existence of this inter-agent communication, apart from human-agent communication, expands the possibilities of the multimodal dialogue systems up to a fully qualified multi-agent system, in which agents have special skills to interact with humans outside of their own “virtual world”.

Experimentation results have been the basis for considering the evolution and application of the Talking Agent architecture for Ambient Assisted Living (AAL) scenarios [37]. AAL face similar issues to those addressed here for the interactive art installation: both are electronic environments that are sensitive and responsive to the presence of people, and this is achieved by integrating a variety of sensing, reasoning, acting, communication and interaction means. Talking Agents provide a solution for such integration. A distinguishing feature with respect to typical AAL implementations that has been already explored in the architecture is the interaction with users through speech, as well as controlling different sensors and actuators. This takes advantage of the ability of agents to communicate information on the context that is derived from previous interactions with the user.

Some issues require further work to gain flexibility. For instance, the coordination of concurrent interactions (in the **ORACULOS** installation the user goes sequentially from one Talking Agent to another) and a systematic management of real-time design issues [25]. In this sense, we are currently working with a group of Talking Agents, each one implementing a different strategy, that cooperate at the same time to interpret the user’s acts and speech and provide an agreed answer. This answer is selected from one agent in the group following some strategy (e.g., voting, or the first ready). This can support the satisfaction of timeliness and robustness requirements that are stronger in AAL than in an art installation scenario.

**Acknowledgements**

This work has been done in the context of the project Agent-based Modeling and Simulation of Complex Social Systems (SiCoSSys), supported by Spanish Council for Science and Innovation, with grant TIN2008-06464-C03-01. In addition, we acknowledge support from the Programa de Creación y Consolidación de Grupos de Investigación UCM-Banco Santander with reference GR58/08.

**Appendix**

**Example of definition of Planner**

```java
import icaro.aplicaciones.informacion.dominioClasses.
    aplicacionTalkingAgent.*;
import java.util.*;

global Set intentions;

rule "Ask for subject" salience 100
    when:
        Intention(code="askSubject")
    then:
        Action action = new SpeechPlanner().generatePlan[i];
        actions.add(action);
end
```

**Example of definition of Intention Level**

```java
import icaro.aplicaciones.informacion.dominioClasses.
    aplicacionTalkingAgent.*;
import java.util.*;

global Set intentions;

rule "Ask for subject" salience 100
    when:
        B:Belief(code="newVisitor")
    then:
        Intention intention = new Intention("askSubject");
        intentions.add(intention);
end

rule "Ask for problem" salience 100
    when:
        B:Belief(code="subjectKnown")
    then:
        Intention intention = new Intention("askProblem");
        intentions.add(intention);
end

rule "Answer consultation" salience 100
    when:
        B:Belief(code="problemKnown")
    then:
        Intention intention=new Intention("answerConsult");
        intentions.add(intention);
end
```
References


[38] D. Traum and S. Larsson, The Information State Approach to Dialogue Management, in: Current and New Directions in
