An Independent Domain Dialogue System Through A Service Manager*

Márcio Mourão, Renato Cassaca, and Nuno Mamede
{*Marcio.Mourao, Renato.Cassaca, Nuno.Mamede*}*@l2f.inesc-id.pt*

L^{2}F - Spoken Language Systems Lab
INESC-ID Lisboa / IST, R. Alves Redol 9, 1000-029 Lisboa, Portugal
http://www.l2f.inesc-id.pt

Abstract. Building a generalized platform for having dialogues is a hard problem in the topic of dialogue systems. The problem becomes still more difficult if there is the possibility of having a conversation about more than one domain at the same time. To help solving this problem, we have built a component that deals with everything related to the domains. In this paper, we describe this component, named Service Manager, and explain what and how it passes all the information that a dialogue manager needs to conduct a dialogue.

1 Introduction

Only recently Spoken Dialogue Systems (SDS) started emerging as a practical alternative for a conversational computer interface, mainly due to the progress in the technology of speech recognition and understanding[1]. Building a generalized platform for having dialogues is a hard problem. Our main goal is to have a system easely adaptable to new dialogue domains, without changing any code or structures that lie behind the system[2].

When we turned our efforts to implement a natural language dialog system, we selected an architecture that could satisfy the requirements of adaptability to different domains. Our dialog system follows the TRIPS architecture, and consists of four main modules: Interpretation Manager (IM), Discourse Context (DC), Behavioural Agent (BA), and Generation Manager (GM)[3].

Faced with a multiplicity of possible dialogues, we restricted our manager to task-oriented dialogues in order to reduce the information acquisition needs. For each domain, it is necessary to define the actions available to the user. The dialog manager's task is to determine the action intended by the user and request the information needed to perform it.

We use frames to represent both the domain and the information collected during the interaction with the users [4]. Each domain handled by the dialogue system is internally

* This research has been partially supported by Fundação para a Ciência e Tecnologia under project number POSI/PLP/41319/2001 (POSI and FEDER)

represented by a frame, which is composed by slots and rules. Slots define domain data relationships, and rules define the system behaviour. Rules are composed by operators (logical, conditional, and relational) and by functions.

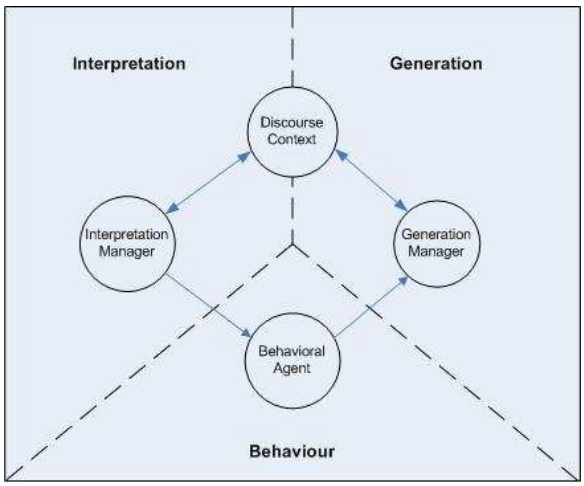

Fig. 1. *Architecture of the Dialogue Manager.*

To keep the filling of the frame slots consistent, it is necessary to indicate the set of values with which a slot can be instantiated. To avoid invalid combination of values, we have defined a meta-language to express the constraints that must be satisfied. So, each frame definition includes a set of recognition rules, used to identify the objects that come with an user utterance and to specify the set of values that each slot may hold; a set of validation rules, to express a set of domain restrictions, i.e., invalid combination of slot values; and classification rules, used to specify the actions that must be performed when some conditions are satisfied, i.e., the valid combinations of values. This approach has the advantage of making the dialogue control independent of the domain.

We have designed a Service Manager (SM) as the interface between the spoken dialogue platform and a set of heterogeneous devices. The role of the SM is to provide the dialogue manager with everything that is related to a domain, which includes representation, access restrictions and, most important of all, dialogue independence. More than one device can be part of a single domain. The SM module determines what can a user really do or execute in a task oriented dialogue system, that is, the services. Inside a domain, a set of services is grouped in one or more states. The services returned by the SM to the dialogue manager will be restricted to the current state, not allowing users doing things that they are not supposed to do, like turning on the lights when the lights are already turned on.

Our spoken dialog platform has been used to produce a spoken dialog system called

“house of the future”, and it is also being used on a demonstration system, accessible by telephone, where people can ask for weather conditions, stock hold information, bus trip information and cinema schedules.

This paper is divided into six sections. The next section describes the Dialogue Manager, and its actual four main components. Section three describes Service Manager, and its main role to achieve dialogue domain independence. In section four we present a dialogue example, showing the fluxes between the dialogue manager and the Service Manager. Next to last, we describe our real experience in the implementation of our dialogue system in controlling parts of the ”house of the future”. Finally, we present some concluding remarks and future work.

2 Dialogue Manager

Our Dialogue Manager is composed by four main modules: (i) Interpretation Manager; (ii) Discourse Context; (iii) Behavioral Agent; and (iv) Generation Manager;

The Interpretation Manager (IM) receives a set of speech acts and generates the correspondent interpretations and discourse obligations[5][6][7]. Interpretations are frame instantiations that represent possible combinations of speech acts and the meaning associated to each object it contains[1]. To select the most promising interpretation two scores are computed. The recognition score to evaluate the rule requirements already accomplished, and the answer score, a measure of the consistency of the data already provided by the user. A more detailed description of this process can be found in [8].

The Discourse Context (DC) manages all knowledge about the discourse, including the discourse stack, turn-taking information, and discourse obligations.

The Behavioral Agent (BA) enables the system to be mixed-initiative: regardless of what the user says, the BA has its own priorities and intentions. When a new speech act includes objects belonging to a domain that is not being considered, the BA assumes the user wants to introduce a new dialog topic: the old topic is put on hold, and priority is given to the new topic. Whenever the system recognizes that the user is changing domains, it first verifies if some previous conversation has already taken place.

The Generation Manager (GA) is a very important component in a dialog manager. To communicate with the user, it has to transform the system intentions in natural language utterances. It receives discourse obligations from the Behavioral Agent, and transforms them into text, using template files. Each domain has a unique template file. The Generation Manager uses another template file to produce questions that are not domain specific. For example, domain desambiguation questions, used to decide proceeding a dialogue between two or more distinct domains, or to clarify questions, are defined in this file.

3 Service Manager

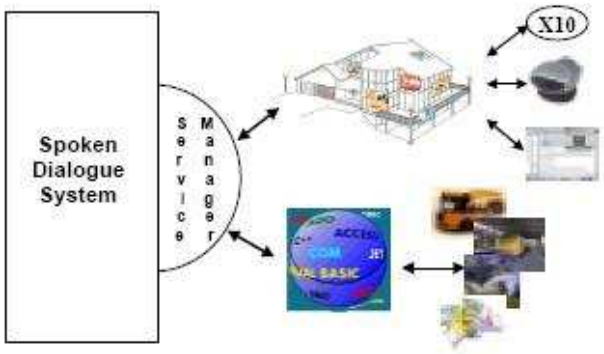

Fig. 2. *Interaction with the Service Manager.*

We designed the Service Manager (SM) as the interface between the spoken dialogue platform and a set of heterogeneous domains[2]. A domain is a representation of a set of devices that share the same description. This description is composed by slots and rules. Slots define domain data relationships, and rules define the system behaviour. A rule or service represents an user possible action. The spoken dialogue platform has been applied to several different types of devices. They are splitted into two big sets: (i) in a home environment; and (ii) on a database retrieval, see Figure 2. In the first, based on X10 and IRDA protocols, we control a wide range of home devices, such as, lights, air conditioning, hi-fi, and TV. We can extend the application to include any infra-red controllable device or whose control functions may be programmed by the X10 protocol. The second type of application, information retrieval via voice from remote databases, has been tested with weather information, cinema schedules and bus trip information. This type of application can easily be extended to other domains.

3.1 Interface Operations

The Service Manager component of the Spoken Dialogue System provides the following main set of operations:

- Domain Description Insertion: this operation inserts a new domain in the Service Manager. It receives as argument a path to a file that contains a XML description of a domain. After the completion of this operation, a Service Manager internal representation of the domain is maintained, for future requests.

- Objects Recognition: to become possible the construction of interpretations and discourse obligations by the Interpretation Manager, an identification of every speech

utterance object is crutial. Whenever this identification is needed, this operation will be invoked, using as arguments: (i) a list with the name of all objects to be recognized; and (ii) the current user. This operation returns a list with the description of the identified objects. Objects belonging to domains where the current user does not have access are not checked. If an object is not recognized, the name of the object will be returned without a description.

- Domain Services Return: a service represents an user possible action. This operation returns the set of services associated with the specified domain(s). In a task oriented dialogue system, these services are used to determine the system behavior: an answer or the execution of a procedure. The operation receives as arguments: (i) the domains; and (ii) the current user. It is possible to ask for all domain services, or ask for services from specific domains. The current user is used to check if the user has access to the services.

- Execution of Services: this operation allows for execution of services that belong to a domain in the Service Manager. More than one service can be ordered to be executed at the same time.

The use of the Service Manager with these operations provides the following features:

- A dialogue independent component that offers all the information related with domains. The Service Manager becomes the only "door" by which the SDS components gets information about a domain.

- There is no mixture between the linguistic parts of the SDS and information about the domains.

- Because information about the domains is concentrated, it is more easier to define layers of reasoning using that information and extract knowledge from it.

- As we are working with different types of devices there was the need to define an abstract representation to create an uniform characterization of domains. The definition of this abstract description allows for an equal treatment of the domains.

- A new domain is more easily introduced.

3.2 Architecture

Figure 3 shows the service manager architecture. It is composed by five modules: (i) Service Manager Galaxy Server; (ii) Device Manager; (iii) Access Manager; (iv) Object Recognition Manager; and (v) Device Proxy.

The Service Manager Galaxy Server defines the interface of the service manager. So, it is the unique point of communication from where all requests and responses are made.

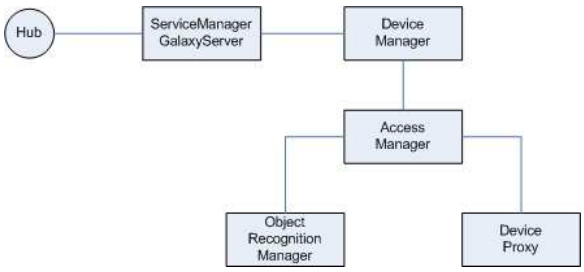

Fig. 3. *Service Manager Architecture.*

All the operations in the interface that were explained before are defined here. This module follows a design pattern named “facade”. This pattern provides a simplified, standardized interface to a large set of objects. This avoids the problem of having to reference many different, complicated interfaces to each object of that set.

The Device Manager is the service manager main module. It is the business layer of the architecture. It has the main methods by which the service manager works, including the ones that deal with the service requests.

The Access Manager component main function is to serve as a filter. This filter is applied for all requests of services. It restricts the available services to the current user. For doing that, this module manages a device restrictions file, and checks whether or not that user has permission to access to specific devices.

The Object Recognition Manager component is responsible for the identification of all objects. This component manages the XML description object files for each domain available in the system. Figure 4 shows part of a lights domain recognition objects file. Two actions are available, turn-on and turn-off. This means that these two objects are identified as being actions in the lights domain.

```
<objects>
 <action>
  <item> turn-on </item>
  <item> turn-off </item>
 </action>
</objects>
```

Fig. 4. Part of a recognition objects file.

The Device Proxy module keeps the connection to the real device (the one that implements services). For some devices, it is impossible to obtain its state directly from it. For example, it is difficult to check if a televison is turn-on or turn-off. Most part of the televisions do not have sensors to do this job. By this reason, this module is also capable

of creating and managing a device state. This device state can be changed only when a service is executed. The system only uses this capability when no state from the device can be obtained, else, it goes directly to the device to check the current state.

The representation of the services is based on a XML description, validated by a generic DTD. DOM trees, resulting from the XML processing, are used as the internal representation of each device. A device might have more than one state. A state can include more than one service. At a given instant, it is only possible to have access to the hierarchy of services available in the actual state of the device.

```
<service name="turn-on-light">
 <label>Turns on the bedroom light</label>
 <params>
  <param action value="turn-on"></param>
  <or>
   <param action value="turn-on"></param>
   <param zone></param>
  </or>
 </params>
 <execute>
  <name>turn-on</name>
  <success>on</sucess>
  <failure/>
 </execute>
</service>
```

Fig. 5. Description of a service.

Figure 5 contains a description of a service. It is only part of a large set of services of a domotic domain we have implemented. It shows a description of a service that is used to turn on the lights of two different zones of a house. Nested to the XML tag “params”, there are explicit the preconditions for the service execution. By default, the logical operator “and” is used. In this case, there are two possible ways of executing a service: (i) provide only an action with value equals to “turn-on”. This means that a user can only say “turn-on” to the system to turn the lights on; (ii) provide an action with value equals to “turn-on”, and the zone where the user wants to act.

The tag named “execute” defines the behavior for the execution of the service. The tag “name” indicates the function code name that is executed when the service is selected. The tag “sucess” indicates the new state of the device. This new state: “on”, does not include, naturally, the “turn-on-light” service, but it includes a “turn-off-light” service. Theoretically, if we turn on a light we cannot turn on the same light again, but we are able to turning it off.

The SM can also restrict the access of certain users to the services. There are three

types of devices: (i) public, (ii) private, and (iii) restricted devices. The public devices are available for all users. Whenever the SM looks for services, these ones are immediately considered. The private devices are only avaliable for those users that have gained privilege access, and therefore, have inserted a key or password in the system. The restricted devices are even more specific than the private ones. They demand particular users, and no one else can access those devices besides them. Figure 6 shows part of a file defining the access information related to three devices. The first device named "Future House Bedroom Light Controller", is declared public, so everyone has access to it. Underneath is a private device named Future House Bedroom TV Controller, meaning that users who want to play with the TV controller must have privilege access. At the bottom is a restricted device. In this case, only the user named "marcio" can access the device.

```
<device hashcode="589654695" type="public">
 <name>Future House Bedroom Light Controller</name>
 <first_access>1083357587921</first_access>
 <last_access>1084211071109</last_access>
 <users/>
</device>
<device hashcode="614235759" type="private">
 <name>Future House Bedroom TV Controller</name>
 <first_access>1084204657484</first_access>
 <last_access>1084204716140</last_access>
 <users/>
</device>
<device hashcode="614235761" type="restricted">
 <name>Future House Bedroom Glass Controller</name>
 <first_access>1084206473546</first_access>
 <last_access>1084211071109</last_access>
 <users>
  <user>marcio</user>
 </users>
</device>
```

Fig. 6. Device restriction file content example.

3.3 Addition of a new domain

To add a new dialogue domain, it is necessary to: (i) define a description of the domain in XML. This description includes the new services, see figure 5; (ii) define the implementation code that will real execute the services. This is a kind of device driver; (iii) define a file in XML with the identification of the domain objects. This file will be used to make the recognition of the objects, see Figure 4. (iv) Define a template file in XML of the system domain utterances. The Generation Manager module makes use of this file, but it is currently loaded by the Generation Manager and not requested to the

Service Manager. Figure 7 shows some utterances used by the dialogue system in the lights domain. The tag “requestslot” is used to ask for a specific slot value. When an action must be fullfilled, one of the three utterances inside the scope of the tag “action” is ramdomly choosed. If the system wants to ask for a zone to act, the utterance nested to tag “zone” is sent to user with “`[$action]`” being actually replaced for an action previously provided by the user.

```
<requestslot>
 <action>
  <utt> Do you wish to turn-on or turn-off the light? </utt>
  <utt> What are your intentions? You want the lights on or off? </utt>
  <utt> Do you wish to raise or diminish the light intensity? </utt>
 </action>
 <zone>
  <utt> What light do you want to [$action]? </utt>
 </zone>
</requestslot>
```

Fig. 7. Part of a template system utterances file.

4 Dialogue Flow Example

In this section we present a small example of how our dialogue system works, including some of the messages interchanged between the components. This example makes the assumption that the dialogue system controls the lights and the television of a home division. The system can turn on and turn off the lights as well as the television. The difficulty is extended because there exists two independent light zones, where the user can apply the light operations. It is possible to apply operations to one zone independent of the other. For example, turn-on the lights of the A zone and after, turn-off the lights of the B zone.

1. **The user starts by saying ”turn-on”.** The Interpretation Manager (IM) component receives the speech act associated with this phrase. To build the interpretations, the IM needs to know what it means in the available domains.
2. **The IM requests Service Manager (SM) to give information about the ”turn-on” object.** The SM grasps the available and accessible user domains and searches a database for the object with that information.
3. **The SM responds to the IM.** The response gives the characteristics of the object to the IM. In this case, the object is recognized in two different domains. The word is an action that belongs to the lights domain and to the television domain.

   We should note that at any time, what is recognized or not, is totally dependent on SM. So, if for some reason, we decide to pull out the lights device from the domain, turning it inaccessible, the next time recognition is made, the answer would

be positive, but only for the television domain.

4. **The IM creates two interpretations.** Because the object was recognized in two different domains, the IM creates a Discourse Obligation (DO) accordingly. In this case, the IM creates a Domain Desambiguation DO.
5. **The IM sends the DO created to the Behavioral Agent (BA).**
6. **The BA selects this DO as the next system action, and sends it to the Generation Manager (GM).**
7. **The GM uses the domain independent template file to generate a desambiguation question.** In this case the system generates a question like: ”What do you pretend to turn-on, the lights or the tv?”. We should note that the system uses a single independent domain template file to generate all the desambiguation questions. In this one, “turn-on” is the intendent action and, “lights”, or “tv”, are the domain names associated with the Service Manager.

   Lets suppose the user says that he wants to turn on the lights. In other words, the user answer was ”the light”. With this answer, the desambiguation is over, and IM will deal with only one interpretation.

8. **IM requests SM for services associated with the light domain.** IM needs to know now how to proceed the dialog. To do that, IM requests SM information about all the services associated with that particular domain.
9. **SM processes the IM request.** SM retrieves information about all services associated with the particular domain. As said before, these services define the behavior of the system, and constitutes what can be done by the Dialogue Manager. We are assuming that the user has the necessary privileges to access these services.
10. **The SM returns a response to IM.**
11. **The IM processes the SM answer.** The SM has found at least two services for the light domain: (i) the turn on; and (ii) the turn off service. But because the user has already said that he wants to turn on the lights (the Discourse Context provides this information), only this service is considered.
12. **IM creates a DO for execution of the service.**

5 House of the Future

We recently extended the demo room at our lab to the project known as “Casa do Futuro”. This project consists of a house where are installed some of the latest technology inovations in smart-house. We have installed our system in the main bedroom. In this division there is: (i) one big tv with split screen capabilities; (ii) two independent light zones; and (iii) a glass with the ability to change its state to opaque or lucid. It was not possible to obtain the current state of the television and of the glass. No feedback directly from these devices could then be used. For that reason, except for the lights where we could obtain their intensity, the current state of the television and of the glass

was maintained in the Service Manager, and updated by the execution of operations on the device.

Besides the integration of the recognition and synthesis technologies with the spoken dialogue system, we also developed an animated character with lypsinc capability[9]. So, whenever a visitor wants to control a device, he calls our virtual buttler, and he appears on the tv set, waiting for any order, and giving feedback when something happens.

6 Concluding remarks

The work reported in this paper results from an integration of several components being developed in our lab. This system is the result of a collaborative effort between people working in the different technologies, and it became a common platform where the different components can be associated to produce multiple applications.

The Service Manager (SM) was developed to provide an unique point of contact with the dialogue domain(s). The SM can be modified with the ignorance of the dialogue manager, only reflecting the modifications when a request is made. We achieved independence of the dialogue manager relative to the domains, and we can introduce new domains without having to modify the dialogue manager behavior. Currently, the system utterances template file used by the the dialogue system is loaded in the Generation Manager module. Because this is an activity that envolves domain knowledge, the SM should also provide to the Dialogue Manager information about generation activities. This remains future work in the SM development.

The system has been used in the “house of the future” to control three devices. This work made also use of speech recognition and an animated character. It is also being used on a demonstration system, accessible by telephone, where people can ask for weather conditions, stock hold information, bus trip information and cinema schedules.

We expect to conduct a more formal evaluation of our system in order to quantify the user satisfaction.

References

1. M. McTear, ”Spoken Dialogue Technology: Enabling the Conversational User Interface”, in ACM Computing Surveys, pp. 1-85 (2002).
2. J. Neto, N. Mamede, R. Cassaca, L. Oliveira, The Development of a Multi-purpose Spoken Dialogue System, EUROSPEECH’2003 - 8th European Conference on Speech Communication and Technology, (2003).
3. J. Allen, G. Ferguson, A. Stent: An architecture for more realistic conversational systems, in Proc. of Intelligent User Interfaces (IUI-01), Santa Fe, NM, Jan, 14-17 (2001).
4. P. Madeira, M. Mourão, N. Mamede: STAR FRAMES - A step ahead in the design of conversational systems capable of handling multiple domains, ICEIS, Angers, France (2003).
5. D.R.Traum: Speech Acts for Dialogue Agents. UMIACS, Univ. of Maryland, pp. 17-30 (1999).

6. J. Kreutel, C. Matheson: Modelling Questions and Assertions in Dialogue Using Obligations, 3rd Workshop on the Semantics and Pragmatics of Dialogue. Univ. of Amsterdam (1999).
7. D. R. Traum, J. F. Allen: Discourse Obligations in Dialogue Processing, in Proc. of the 32nd Annual Meeting of the Association for Computational Linguistics, pp. 1-8 (1994).
8. M. Mourão, P. Madeira, N. Mamede, "Interpretations and Discourse Obligations in a Dialog System", in Proc.Propor 2003, Faro, Portugal, pp. 197-200 (2003). Springer-Verlag Berlin Heidelberg 2003.
9. M. Viveiros, "Cara Falante - Uma interface visual para um sistema de diálogo falado", Graduation Thesis, IST, Work in progress.