A Domain Knowledge Manager for Dialogue Systems

Annika Flycht-Eriksson[1]

Abstract. In order to behave naturally and intelligently a dialogue system often has to gather information from many knowledge sources and perform advanced domain reasoning. These tasks should not be performed by the Dialogue Manager as this makes it very complex and hard to customise to new domains. It is instead suggested that application access and domain knowledge reasoning should be clearly separated from dialogue management and performed by a separate module. In this paper such a module, called the Domain Knowledge Manager and the knowledge structures used by it is presented.

1 Introduction

Much effort have been spent on natural language understanding, dialogue management, and generation in dialogue systems for simple services, i.e. dialogue systems that can provide information given a set of parameters collected from the user [6]. The communication with and the use of external resources containing domain knowledge and application information, in most cases a database, are in general not discussed. This is probably due to the fact that access of such resources are considered rather straightforward. This is however not necessarily the case. Some problems related to communication with external resources and applications are [11]: the vocabulary of the dialogue and the application might differ, thus a mapping between them is necessary; the user can have false presuppositions about the information in the application, which can result in a request and an answer that will be misinterpreted; the retrieved information can be ambiguous or indeterminate.

Furthermore, as information services and domains become more complex, the complexity of the dialogues that the dialogue system has to handle increases. The system must become more intelligent in order to understand and fulfil the user's requests. A consequence of this is that dialogue systems need more and more domain knowledge, and that the domain reasoning mechanisms have to become more sophisticated. The domain knowledge can be distributed over several sources, which means that a system has to access, reason about, and integrate domain knowledge from various sources. Since a system cannot possibly have knowledge of everything related to the domain it also has to have meta knowledge about its own limitations and other possible resources, in order to give as helpful responses as possible.

[1] Department of Computer and Information Science, SE-581 83, Linköping University, Sweden, email: annfl@ida.liu.se

The dialogue in figure 1, which shows how a user can interact with a dialogue system that provides bus timetable information, illustrates some of these issues. The temporal expression in utterance U1 is vague and differs from the way time is represented in a bus timetable. Utterance U2 is an example of the need for specialised domain knowledge, in this case the ability to reason about geographical objects and their relations. Taken together, U1 to S3 show that knowledge from several resources have to be integrated in order to perform a task, e.g. temporal knowledge, geographical knowledge and knowledge of the bus timetable are needed in order to give information on a specific bus route. Finally, U4 and S4 exemplify how meta knowledge about the systems capabilities can be used to provide a helpful and intelligent dialogue.

To solve the problems presented above we propose that a separate module, a Domain Knowledge Manager, devoted to access of application data and reasoning about domain knowledge should be introduced. The new module co-operates with the Dialogue Manager with the aim of achieving a natural and intuitive dialogue. In this paper such a module, the tasks it performs, and its knowledge structures are presented.

2 Domain Knowledge Management

Dialogue systems for information services often have a modular architecture with processing modules for natural language interpretation, dialogue management, access of the application, and natural language generation. The role of the *Interpreter* is to analyse the user input and deliver a meaning representation to the Dialogue Manager. The *Dialogue Manager's* primary tasks are to handle the dialogue with the user and to keep track of the interaction. Managing the interaction with the user involves deciding whether a user request is clear enough to access the application, or if it is not, to initiate a clarification sub-dialogue. As a basis for these tasks the Dialogue Manager uses a dialogue model, a dialogue history, and sometimes system task models. The *Generator* is responsible for realisation of question and answers from the Dialogue Manager.

Co-ordination and control of the different modules differ between systems. Some dialogue system architectures consider the modules as equal and communication and co-ordination is achieved by the use of a separate module called a hub [14] or facilitator [10] that passes messages between the modules. Another type of architecture is based on the use of blackboards where different modules can read and write information from and to a common source [5]. A third alternative is to let the

U1: I want buses to Askeby in the evening Monday to Thursday
S1: Where do you get on?
U2: Is there a bus stop near the railway station in Linköping?
S2: Yes, it is called the Travel centre
U3: Then I want to go from there
S3: Let see, at 17.13 bus 539 leaves from the travel centre and arrives at Askeby shop 17.35
U4: How much is it?
S4: I cannot give information about prices, you have to contact a human operator at 020-211010

Figure 1. A hypothetical dialogue between a user and a dialogue system, which illustrates the need for domain and meta knowledge. For utterance U1 temporal reasoning is required to resolve the expression "evening Monday to Thursday", to answer U2 the system must perform geographical reasoning, and finally S4 shows that the system needs meta information about its own knowledge.

Dialogue Manager have the control of the system and decide when and which of the different modules should perform a specific task [8].

Regardless of whether a hub, blackboard or Dialogue Manager architecture is used, there are several problems related to the communication with applications as stated in the introduction.

Ambiguous and indeterminate knowledge have been dealt with to some extent, often by adding specialised components to the dialogue system. For example, vague temporal expressions must be resolved in dialogue systems in the ATIS and train travel domain ([3, 7]). In most cases these components extend the functionality of the interpreter and are called before a request is delivered to the Dialogue Manager.

A solution to the problem of mapping between the vocabulary used by the dialogue manager and a suitable representation for the external database has been proposed by Whittaker and Attwater [15]. A new component, called the Information Manager, that interacts with the application database is introduced. The Information Manager is responsible for translation between high level requests and a set of operations on the database. To perform its task it utilises a data model consisting of several vocabulary models, e.g. spelt out vocabulary and spoken vocabulary, and it also has knowledge on synonyms and homophones.

Similarly a separate module, the Action Manager, for communication with external resources has been proposed [13]. In this architecture the Dialogue Manager is responsible for the translation between the requests and the actions carried out by the Action Manager. When a requested action is delivered to the Action Manager it has to decide how the action should be executed and which source to access.

Even if some of the presented architectures include a separate module for access of the application or reasoning with domain knowledge, they deal only with one of the problems or a monolithic application. None of the present approaches deal with all the issues and the integration of several domain knowledge sources. A more sophisticated module is needed to handle all these aspects. In the following sections we describe how such a module, called the Domain Knowledge Manager, has been realised and how it can used in a dialogue system.

3 The Domain Knowledge Manager

The primary responsibility of the Domain Knowledge Manager is to provide domain and application specific information when the Dialogue Manager has produced a fully specified request. The Domain Knowledge Manager reasons about where and how the information should be retrieved and how information from different domain or application knowledge sources should be integrated. It also has to provide helpful answers if the requests can not be fulfilled, that is, to state what the problem is and give some possible solutions.

3.1 Multi-agent framework

The Domain Knowledge Manager consists of several agents: the **Control Agent**, the **Recipe Agent**, the **Integration Agent**, and **Domain Agents**. The agents provide different *services*, typically to retrieve some information given some parameters, and can also request services from each other. Communication and co-operation among the agents are achieved by passing messages.

The Control Agent is a generic domain independent agent that controls the processing of a request. For this purpose it utilises knowledge structures called *recipes*. A recipe (cf. recipe-for-action [12]) consist of a series of services from different agents, which are executed in order to construct an answer to the request.

The Recipe Agent is responsible for the construction of recipes that match the requests. The recipes are application specific, but the agent in itself is domain independent.

The Integration Agent is a domain independent agent that can integrate several response alternatives into one answer, utilising *integration rules*, which contains both domain heuristic and more general principles.

The Domain Agents are responsible for appropriate access of domain knowledge sources and are able to perform sophisticated knowledge reasoning in order to retrieve the information. Thus, each domain agent provides a set of services such as storing, retrieving or constructing a specific type of information. The domain agents are in general application-specific.

The implementation of the Domain Knowledge Manager is based on the Open Agent Architecture, OAA, which is a framework for the development of multi-agent systems [10]. How the agents work together to process a request is described in the following section. To make the presentation less abstract examples from the domain of bus timetable information will be used. The Domain Knowledge Manager is however not restricted to this domain, by substituting the domain agents and providing recipes and integration rules it

Agent	*Service*
Spatial Reasoning Agent	getBusStops(From.BusStop, From.Place, From.Street, From.Area, From.Town, FromBusStops)
Spatial Reasoning Agent	getBusStops(To.BusStop, To.Place, To.Street, To.Area, To.Town, ToBusStops)
Temporal Reasoning Agent	getDate(Time.Date, Date)
Temporal Reasoning Agent	getTime(Time.Time, ArrTime)
Timetable Agent	getBusRoutes(FromBusStops, ToBusStops, Date, ArrTime, Routes)

Figure 2. An example of an uninstantiated recipe for trip information.

can be adapted to any domain in which several domain knowledge sources are utilised, for example, tourist information or interactive news services with background information.

3.2 Processing of requests

Processing of a request from the Dialogue Manager in general involves three steps. First the Domain Knowledge Manager has to decide how to treat the request, i.e. to produce one or more recipes. In most cases one recipe is enough, but sometimes the user has provided ambiguous information that cannot be resolved by the interpreter or the Dialogue Manager, in which cases several recipes are needed. The next step is to process the recipe(s). The processing must be carefully monitored and aborted if an error occurs. Finally alternatives must be inspected and integrated into one answer that can be sent back to the Dialogue Manager.

Producing the recipes The first step towards fulfilling a request is to find a suitable recipe that describes what information is needed and how the information should be retrieved. For this purpose the **Recipe Agent** provides a makeRecipe service. As a basis for production of a new recipe a recipe library that contains recipe templates for different types of requests is utilised. The recipe agent maps the request to a suitable recipe template and instantiates it with the values from the request. An example of a recipe template for trip information requests is presented in figure 2.

Since the information given by the Dialogue Manager can be ambiguous, the Recipe Agent in some cases have to find one or more unambiguous interpretations of the request. For example the utterance "I want to go by bus to Berga centrum" is ambiguous since "Berga centrum" is the name of both a bus stop and a place. Does the user mean that (s)he wants to go to the exact bus stop called "Berga centrum" or more losely that (s)he wants to go to the place "Berga cetrum"? If the system only considered the bus stop "Berga centrum" it would not find bus routes stopping at the bus stop "Berga söderleden" nearby "Berga centrum".

The disambiguation can be approached in several ways, one is to ask the user for clarification, but this can result in a cumbersome and unnatural dialogue. Our approach is instead to generate several interpretations originating from the ambiguous information, and produce a recipe for each and process them in parallel. The alternative interpretations can then be evaluated depending on the result, and in the end be integrated into one answer.

Processing of recipes As mentioned above the main purpose of the Domain Knowledge Manager is to collect and integrate information from various domain agents. In the bus timetable information domain, four different domain agents are present. The **Temporal Reasoning Agent** contains a calendar and reasons about temporal expressions. The **Spatial Reasoning Agent** consists of a Geographical Information System and a reasoning mechanism used to deduce the relations between geographical objects [4]. The **Timetable Agent** access an information source on the Internet, which contains the timetables for local bus and train traffic. There is also a **System Information Agent** with system information, like references to human operators for questions outside the scope of timetable information, for example on lost property.

Information is gathered by calling specific services, for example: busStopNear?(BusStop, Landmark, Y/N) checks if the BusStop is near the Landmark and provides a yes/no answer, and getTime(PartOfDay, Time) maps a temporal expression PartOfDay to a precise time interval Time.

A complex request involves access of many different domain agents. To fulfil a request for trip information three different agents have to be involved. The Spatial Reasoning Agent must provide a set of bus stops for departure and arrival, respectively. The Temporal Reasoning Agent must provide the time and date. Then the Timetable Agent can access the timetable database in order to retrieve the trip information. Which of the agents services and in what order they should be called are specified in the recipes for retrieval of trip information, see figure 2.

The **Control Agent** supervises the process and interrupts it if errors are encountered. An error can, for example, occur if the user has provided vague information. If (s)he says that (s)he wants to travel to Mjölby by train, Mjölby is a precise arrival location since it can be mapped to one railway station, however, if (s)he says that (s)he wants to travel to Mjölby by bus, Mjölby is to vague since it can be mapped onto to many bus stops. The provided information can also be erroneous, such as June 31 or the railway station in a city with no railway. If a user has assumed there is a railway station in a city and there is no one the Dialogue Knowledge Manager will detect this and an explanation will be sent to the Dialogue Manager.

Integrating the alternatives Since ambiguous requests result in a set of alternative recipes which are processed independently the results must be integrated before an answer is transferred to the Dialogue Manager. This is performed by the **Integration Agent** utilising integration rules. Some of the rules are general and domain independent while others are heuristic and specialised for a specific domain. An example of a general rule is: When all but one recipe have failed the successful one is taken as the correct interpretation of the request and returned to the dialogue manager. When more than one of the interpretations provide a possible answer it is more difficult to integrate them, and domain specific heuristics are needed.

4 Example from the MALIN system

The Domain Knowledge Manager is used in the MALIN (Multimodal Application of LINLIN) dialogue system[2], primarily in an application for timetable information for local bus traffic. In this section we exemplify how the Domain Knowledge Manager interact with the MALIN Dialogue Manager to achieve a natural dialogue.

In MALIN the Dialogue Manager has a prominent role since it controls both the interaction with the user and the interaction between modules in the system. It can handle the dialogue for various dialogue systems, and is customised for each new application [9].

The dialogue model in MALIN is based on a dialogue grammar structured in terms of initiatives and responses. Incorporated in the dialogue grammar are system task models, called Information Specification Forms (ISF), which contain information about what parameters are required for different types of requests (for more details see [2]). Focal information is recorded in a dialogue tree, which constitutes the dialogue history.

The dialogue presented in figure 1 will be used to illustrate how the Dialogue Manager and the Domain Knowledge Manager divide the labour and how the knowledge reasoning tasks are carried out. Consider the first utterance, U1: *I want buses to Askeby in the evening Monday to Thursday.* The Dialogue Manager begins specifying a Trip ISF:

$$\begin{bmatrix} Type: & Trip \\ Arr: & \begin{bmatrix} Town: & Askeby \end{bmatrix} \\ Dep: & req. \\ Time: & \begin{bmatrix} Day: & \{Mon..Thu\} \\ Time: & Evening \\ D/A: & Arrival \end{bmatrix} \\ Bus\#: & opt. \end{bmatrix}$$

This does not form a complete request to the background system since the required field Dep is empty, and thus the Dialogue Manager begins to prompt the user for further information in order to fill it, in this case asking for departure place, S1: *Where do you get on?*. The user, however, does not know the name of the bus stop and instead asks for a bus stop, U2: *Is there a bus stop near the railway station in Linköping?*. This should *not* be added to the current ISF, instead a new request is instantiated:

[2] MALIN is a refinement of the LINLIN-system architecture [1] to handle also multimodal interaction and more advanced applications.

$$\begin{bmatrix} Objects: & \begin{bmatrix} Place: & RailwayStation \\ Town: & Linköping \end{bmatrix} \\ Properties: & \begin{bmatrix} Aspect: & BusStopNear \\ Value: & \end{bmatrix} \end{bmatrix}$$

This request is sent to the Domain Knowledge Manager which asks the Recipe Agent to find and instantiate a suitable recipe. The resulting recipe is very simple, and consist of only one action ('_' denotes a place holder for parameters with no value):

getBusStopsNear/In(_, 'Railway station', _, _, 'Linköping', BusStops)

The service getBusStopsNear/In is requested from the Spatial reasoning Agent. Since there is only one answer, the list {Travel Centre}, it is returned to the Dialogue Manager.

The retrieved information results in the answer S2: *Yes, it is called the Travel centre.* This was obviously information that was useful for the user and (s)he decides to leave from there, U3: *Then I want to go from there.* This in turn updates the Dep: parameter of the previous Trip ISF to:

$[Dep: [BusStop: TravelCentre]]$.

As all required parameters of the ISF are specified, the Dialogue Manager realises that it has enough information and sends this updated Trip ISF to the Domain Knowledge Manager. The Domain Knowledge Manager finds the recipe template corresponding to the task of finding trip information, instantiates it, and processes the recipe requesting services from the Spatial Reasoning Agent, the Temporal Reasoning Agent, and the Timetable Agent. The Timetable Agent delivers a ranked list of possible trips that are returned to the Dialogue Manager and the first alternative is presented to the user, S3: *Let see, at 17.13 bus 539 leaves from the travel centre and arrives at Askeby shop 17.35*

The final exchange U4-S4 illustrates the type of requests where information about the systems abilities are provided. U4: *How much is it?* is a task-related request, and the structure sent to the Domain Knowledge Manager is a complex structure where the trip is the object:

$$\begin{bmatrix} Objects: & \begin{bmatrix} Arr: & AskebyShop \\ Dep: & TravelCentre \\ DepTime: & 17.13 \\ ArrTime: & 17.35 \\ Bus\#: & 539 \end{bmatrix} \\ Properties: & \begin{bmatrix} Aspect: & Price \\ Value: & \end{bmatrix} \end{bmatrix}$$

The Recipe Agent retrieves a recipe consisting of one action:

findSystemInformation('Price', Information)

The recipe is processed and information requested from the System Information Agent. The response to the Dialogue Manager is a response with the content that the system does not have the requested information and further helpful information on where to call retrieved from the System Information Agent, S4: *I cannot give information about prices, you have to contact a human operator at 020-211010.*

5 Conclusions and future work

In this paper an approach to development of intelligent information dialogue systems where a Domain Knowledge Manager and a Dialogue Manager co-operate to achieve natural interaction has been presented. The domain knowledge is represented in a number of domain agents who provide services to the Domain Knowledge Manager. The use of a Domain Knowledge Manager in a dialogue system gives us two major advantages. The first is that dialogue management becomes more focused as it only has to consider dialogue phenomena, while domain specific reasoning is handled by the Domain Knowledge Manager. The second major advantage is that the domain knowledge sources can easily be modified, exchanged, and reused. This in turn facilitates porting of the system to new domains since domain related aspects are included in the domain agents.

With the use of a Domain Knowledge Manager the problems regarding access of applications are also solved. The different domain agents can map expressions expressed in the vocabulary of the dialogue to expressions that fit the application. False presuppositions about the information in the application are detected by the Dialogue Knowledge Manager when it processes a request and an explanation is delivered to the Dialogue Manager. If the retrieved information is ambiguous or indeterminate the Dialogue Knowledge Manager helps the Dialogue Manager by stating that a clarification is needed and the preferred type of information to ask for. For example, if the user has been to vague when describing a departure location and only given the name of an area, the Dialogue Knowledge Manager explains that the area is to big and asks for more precise information such as a street name.

The proposal of how a Domain Knowledge Manager can be designed has two important features: the mechanisms for managing requests are generic and to some extent domain independent, and the domain knowledge sources has a common well specified interface based on an agent communication protocol. This means that changing from one domain to another for the Domain Knowledge Manager only involves the creation of new recipes, integration rules, and plug-in of new domain agents.

Future challenges are to apply the proposed architecture, utilising a Domain Knowledge Manager, to other domains and types of dialogue systems, such as advisory or tutoring systems. For such systems other knowledge sources like user models and argumentation models are relevant and have to be incorporated in the system architecture. We hope that with some modifications, the framework presented in this paper can be used.

6 Acknowledgements

This work results from a number of projects on development of natural language interfaces supported by The Swedish Transport & Communications Research Board (KFB), and the Centre for Industrial Information Technology (CENIIT). Thanks to Arne Jönsson and Lena Santamarta for comments on previous versions of this paper.

REFERENCES

[1] Lars Ahrenberg, Arne Jönsson, and Nils Dahlbäck, 'Discourse representation and discourse management for natural language interfaces', in *Proceedings of the Second Nordic Conference on Text Comprehension in Man and Machine, Täby, Sweden*, (1990).

[2] Nils Dahlbäck and Arne Jönsson, 'Knowledge sources in spoken dialogue systems', in *Proceedings of Eurospeech'99*, Budapest, Hungary, (1999).

[3] Cristina Dobrin and Peter Boda, 'Resolution of date and time expressions in a www-based dialogue system', in *COST249 10th Management Committee Meeting*, Porto, Portugal, (February 12-13 1998).

[4] Annika Flycht-Eriksson and Arne Jönsson, 'A spoken dialogue system utilizing spatial information', in *Proceedings of International Conference on Spoken Language Processing, ICSLP'98*, p. 1207, Sydney, Australia, (1998).

[5] Brigitte Grau, Grard Sabah, and Anne Vilnat, 'Control in man-machine dialogue', *THINK*, **3**(1), 32–55, (1994).

[6] Philip J. Hayes and D. Raj Reddy, 'Steps toward graceful interaction in spoken and written man-machine communication', *International Journal of Man-Machine Studies*, **19**, 231–284, (1983).

[7] Bernd Hildebrandt, Gernot A. Fink, Franz Kummert, and Gerhard Sagerer, 'Understanding of time constituents in spoken language dialogues', pp. 939–942, (1994).

[8] Arne Jönsson, *Dialogue Management for Natural Language Interfaces*, Ph.D. dissertation, Linköping University, 1993.

[9] Arne Jönsson, 'A model for habitable and efficient dialogue management for natural language interaction', *Natural Language Engineering*, **3**(2/3), 103–122, (1997).

[10] David L. Martin, Adam Cheyer, and Gowang Lo Lee, 'Agent Development Tools for the Open Agent Architecture', in *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, pp. 387–404, London, (April 1996). The Practical Application Company Ltd.

[11] Michael McTear. Spoken dialogue technology: Enabling the conversational user interface. http://www.infj.ulst.ac.uk/~cbdg23/survey/spoken_dialogue_technology.html, 2000.

[12] Martha E. Pollack, 'Plans as complex mental attitudes', in *Intentions in Communication*, chapter 5, MIT Press, (1990).

[13] Dragomir R. Radev, Nanda Kambhatla, Catherine Wolf Yiming Ye, and Wlodek Zadrozny, 'Dsml: A proposal for xml standards for messaging between components of a natural language dialogue system', in *Proceedings of AISB Workshop on Reference Architectures and Data Standards for NLP*, Edinburgh, UK, (April 1999).

[14] Stephanie Seneff, Ed Hurley, Raymond Lau, Christine Pao, Philipp Schmid, and Victor Zue, 'GalaxyII: A reference architecture for conversational system development', in *Proceedings of International Conference on Spoken Language Processing, ICSLP'98*, volume 3, pp. 931–934, Sydney, Australia, (December 1998).

[15] Steve J. Whittaker and David J. Attwater, 'The design of complex telephony applications using large vocabulary speech technology.', in *Proceedings of International Conference on Spoken Language Processing, ICSLP'96*, pp. 705–708, Philadelphia, USA, (October 1996).