

Generic dialogue modeling for multi-application dialogue systems

Trung H. Bui, Job Zwiers, Anton Nijholt, and Mannes Poel

Human Media Interaction, Department of Computer Science, University of Twente
Postbox 217,
7500 AE Enschede, The Netherlands
{buih, zwiers, anijholt, mpoel}@cs.utwente.nl

Abstract. We present a novel approach to developing interfaces for multi-application dialogue systems. The targeted interfaces allow transparent switching between a large number of applications within one system. The approach, based on the Rapid Dialogue Prototyping Methodology (RDPM) and the Vector Space Model techniques, is composed of three main steps: (1) producing finalized dialogue models for applications using the RDPM, (2) designing an application interaction hierarchy, and (3) navigating between the applications based on the user's application of interest.

1 Introduction

A multi-application dialogue system is defined as a dialogue system allowing the user to navigate between a set of applications. Applications considered range from simple tasks such as operating a home device or booking a flight to more complex tasks such as controlling a smart-room or managing the (road) traffic.

To date, due to the complexity of the management of language interfaces and their strong dependence on the interaction context, a really generic approach for multi-application dialogue design does not yet exist; each application or a set of applications requires the development of a specific model. Multi-application dialogue model prototyping therefore represents a significant part in the development process of multi-application interactive systems. However, most current prototyping methods are limited to the development of dialogue systems working on a single application or a small set of applications [3], [8], [10], [12].

In this perspective, we aim at developing a generic dialogue modeling methodology for the efficient production of interfaces for multi-application dialogue systems. The targeted interface allows transparent switching between a large number of applications within one system. The approach, based on the Rapid Dialogue Prototyping Methodology (RDPM¹) [1] and the Vector Space Model (VSM) techniques, is composed of three main steps: (1) producing finalized dialogue models for applications using the RDPM, (2) designing an application

¹ a methodology allowing a quick production of frame-based dialogue models

interaction hierarchy based on VSM techniques, and (3) navigating between the applications based on the user’s application of interest.

These steps are described in sections 2, 3, and 4 respectively. A scenario example for producing a dialogue system accessing 10 applications in the ICIS domain ² is presented in section 5. Finally, in sections 6 and 7 we summarize the main points of the paper and possible further extensions of the methodology respectively.

2 Producing finalized dialogue models for applications using the RDPM

The finalized dialogue model for each application can be quickly produced using the RDPM.

The general idea underlying the RDPM is that the dialogue model is a frame-based model that can be quite easily and systematically derived from a relational representation of the application itself, hereafter called the task model. More precisely, the RDPM consists of five main consecutive steps, namely: (1) *producing a task model* for the targeted application; (2) *automatically deriving an initial dialogue model* from the produced task model; (3) using the generated interface to *carry out Wizard-of-Oz experiments* (i.e. dialogue simulations) to improve the initial dialogue model; (4) *carrying out an internal field test* to further refine the dialogue model (reformulation of system messages, improved feedback, etc.), and to validate the evaluation procedure (coherence, understandability); and (5) *carrying out an external field test* to evaluate the final dialogue model according to the evaluation procedure defined during the internal field test. Steps 1 and 2 are briefly described in the next sections in the context of producing finalized dialogue models for applications, the remaining steps are described in detail in [1].

2.1 Task model

In the RDPM, an application is seen as a set of functions the user can invoke through a multimodal interface to perform the various functionalities provided by the application. In this perspective, an application is modeled as a *solution table* [1], where the rows correspond to the possible functions (also called “solutions” or “targets”) and the columns are the attributes needed to uniquely identify each of the functions, and to invoke it. In other words, the values of the attributes in a row of the solution table (also referred to as canonical values) correspond to the values of the arguments of the function, the call of which results in the fulfillment of the corresponding application functionality. For example, in the ICIS domain, the task model for the patient search can reduce to a single generic function `select_patient(name, age, address, ...)`, the attributes of which identify the selection features available for the patient search. Therefore, the task

² ICIS stands for Interactive Collaborative Information Systems, a Dutch research project which aims at developing intelligent collaborative information systems technology in order to reduce risks and damages in chaotic complex environments.

model of the patient search is a solution table with as many columns as there are attributes, the rows of which are the various value combinations corresponding to patients. At the computational level, the calls to the `select_patient()` function are implemented in the form of SQL queries to the solution table containing the required information.

2.2 Finalized dialogue model for a single application

In our approach, a finalized dialogue model is defined as a set of interconnected multimodal Generic Dialogue Nodes (referred to as mGDNs [9]), where each of the dialogue nodes is associated with one of the attributes (also called “slots” or “fields”) in the solution table. In complex applications, these mGDNs are divided into groups, where each group is considered as an object and the mGDNs in the group are attributes of the object. For instance, the *First Name*, *Last Name*, and *Function* mGDNs belong to the *Person* group. For any given slot, the role of the associated mGDN is to perform the simple interaction with the user that is required to obtain a valid value for the associated attribute. In the architecture that we have selected for the implementation of our multimodal dialogue-driven interfaces, the processing of the mGDNs (i.e. the actual interaction with the user according to the specification of the mGDNs) is performed by a specific module called the *local dialogue manager*. However, this is, of course, not sufficient to carry out any real dialogue: some form of global dialogue management also has to be integrated. For example, in addition to the definition of the mGDNs and the specification of the local dialogue manager, some branching logic responsible for the management of the global dialogue flow needs to be specified. In our approach, this branching logic is hard-coded in a specific dialogue management module, called the *global dialogue manager*. The underlying assumption is that the encoded local and global dialogue flow management strategies are indeed application-independent, i.e. that, in most situations, they lead to an acceptable, though not always optimal behavior for the system. Consequently, in our approach, dialogue model design essentially reduces to the application-dependent, declarative specification of the mGDNs, the encoded dialogue management strategies being used without modification for all applications. In short, a finalized dialogue model consists of two main parts: (1) the application-dependent, declarative specification of the mGDNs; and (2) the application-independent (local and global) dialogue flow management strategies encoded in the corresponding (local and global) dialogue manager. Both of these components are described in more detail in [1].

2.3 System Architecture

The general architecture of the dialogue system corresponding to each single application produced by the RDPM is represented in Fig. 1.

Three input modalities: voice, text and pointing can be used independently or simultaneously depending on the configuration of the current active mGDN [9]. These inputs are pre-processed by the Natural Language Understanding (NLU) modules and the Pointer Understanding (PU) module. The outputs from NLU

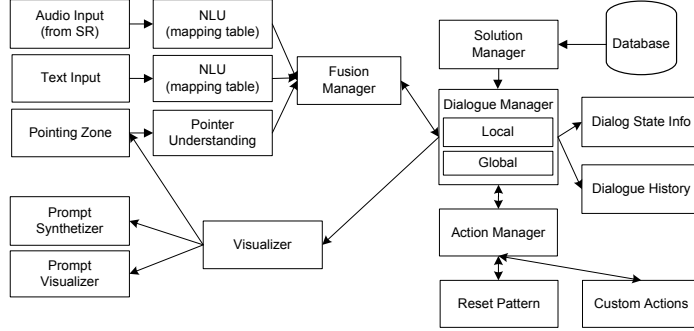


Fig. 1. Architecture of dialogue systems produced by RDPM.

and PU modules are semantic triples (attribute, value, time-stamp). The fusion manager integrates the semantic triples receiving from the NLU and PU modules and sends a set of integrated semantic triples to the dialogue manager. In the current implemented version, the fusion manager simply collects the semantic triples based on their time-stamp relation and forwards them to the dialogue manager.

The dialogue manager encodes the local dialogue flow management strategy and global dialogue management strategy. Therefore, the input to the dialogue manager is first processed by the local dialogue management strategy in which we define five types of generic situations: *OK*, *Request for Repetition*, *Request for Help*, *NoInput*, and *NoMatch* [1].

In the case of the *OK* situation, control is handed back to the global dialogue manager which applies the global dialogue management strategy for the activation of the next mGDN. The dialogue state information (e.g. the current dialogue state, the active mGDN, etc.) and the recognized semantic triples are updated to the dialogue state info module and the dialogue history module respectively. When the dialogue manager gathers enough constraints³, it sends the request to the action manager, the application connected with this module performs the task and sends the feedbacks to the action manager, the action manager then forwards these feedbacks to the dialogue manager. In addition, functions related with user modeling and system customization have been integrated such as *Reset Patterns* and *Custom Actions*. *Reset Patterns* allows the system to adapt to the behavior of a specific user or population of users by anticipating their next decisions. The idea is to develop an intelligent reset algorithm that estimates the most probable values for some mGDNs slots in a new dialogue session according to the previous interactions with the user. *Custom Actions* allows the users to dynamically associate sequences of solutions with a single new solution. The

³ this happens when the number of solutions (extracted from the solution manger) satisfying the current constraints is smaller than or equal to a pre-defined solution threshold.

main goal of these two functions is to reduce the time to perform a task with the interface. The hypothesis is that these functions will indeed increase the quality of the interaction as perceived by the user. These two functions are described in detail in [2].

The outputs from the dialogue manager to the visualizer are multimedia prompts containing messages and a pointing zone update content. The messages are visualized in the user interface (Prompt Visualizer) and/or uttered by the mGDN during the interaction (Prompt Synthesizer). The messages are combined with the pointing zone update content (the content is a map, a calendar or a table depending on the nature of the mGDN) to allow the user to provide the desired values using keyboard, microphone or mouse click/touchscreen.

3 Designing an application interaction hierarchy

In section 2, we showed that it is possible to produce n finalized dialogue models M_0, M_1, \dots, M_{n-1} from n applications A_0, A_1, \dots, A_{n-1} using the RDPM ⁴, the question is how to integrate these applications in one unique system (i.e. multi-application dialogue system).

Vrugt and Portele [12] introduced a dialogue system accessing multiple applications with a dynamic setup that can be changed at run-time. Their goal is achieved by application-independent knowledge processing inside the dialogue system based on modular ontological descriptions. They also define a clear interface between a dialogue system and applications by realizing a generic dialogue functionality on top of the application independent knowledge processing. This approach assumes that the user knows exactly which application he is going to interact with and therefore it is not scalable to the development of dialogue systems with a large number of applications.

Carroll and Carpenter [3] developed a call-routing dialogue system using the VSM techniques. The system allows routing the user's telephone call to the right department. Two main modules in the system are the routing module and the disambiguation module. When the routing module returns more than one candidate applications, the disambiguation module is invoked. The disambiguation module determines the number of terms relevant to the user's request (say n) and uses a YN-question ($n = 1$) or a WH-question ($n > 1$) to identify the desired application (i.e. the department) or transfers the call to the operator ($n = 0$). The authors do not view each application as a finalized dialogue model, therefore no further interaction happens when an application is identified.

We organize applications in a hierarchy since it allows flexible dealing with a large number of applications [4]. The hierarchy can be created manually or automatically. When the number of application is large (hundreds, thousands, or more ⁵), it is difficult to create the hierarchy manually, therefore an automatic

⁴ each application can have its own set of input modalities as described in section 2.3

⁵ we assume that each application is described by an associated textual document and the main goal is to find out the user's application of interest

process is suitable for this case. In our approach, the hierarchy is produced automatically using VSM techniques and an hierarchical clustering algorithm.

3.1 Application interaction hierarchy

An application interaction hierarchy is an m levels hierarchy of n finalized dialogue models consisting of three types of nodes:

1. Root (level: $m-1$): unique node on the top of the hierarchy.
2. Internal nodes (level: from $m-2$ to 1): each internal node consists of at least two child nodes, a child node can be an internal node or a leaf. The hierarchy accepts lattice nodes (i.e. internal nodes, each of them has more than one father node).
3. Leaves(level: 0): correspond to n applications.

An application interaction hierarchy ($n = 10$) is represented in Fig. 4.

3.2 Vector space model for the finalized dialogue models

We assume each finalized dialogue model which the production is described in section 2 is characterized by a textual description of the associated application. The textual description can be extracted from the mapping tables (cf. Fig. 1). We represent these descriptions by k -dimension vectors d_0, d_1, \dots, d_{n-1} using the VSM techniques.

The following paragraph presents the process of producing vectors and computing the similarity between the textual descriptions of the applications using the standard VSM technique (in the implementation phase, a suitable VSM and the number of index terms are selected based on the content of textual descriptions. For example, in case the textual description is a set of sentences, a semantic VSM taking into account the dependence between terms such as [11] is appropriate):

1. Produce index terms from the textual descriptions
We analyze the textual descriptions using Natural Language Processing (NLP) techniques (syntactic analysis, morphological & stop words filtering, term extraction) to produce k index terms: t_1, t_2, \dots, t_k .

2. Construct occurrence matrix F
A description is represented by a lexical profile: $d_i = (w_{i0}, w_{i1}, \dots, w_{ik-1})$. w_{ij} is the weight (or importance) of the j^{th} indexing term t_j in the textual description d_i . w_{ij} is often simply the number of occurrences of t_j in d_i or the inverted occurrence frequency.

The $n \times k$ occurrence matrix F:

$$F = \begin{pmatrix} d_0 \\ d_1 \\ \dots \\ d_{n-1} \end{pmatrix} = \begin{pmatrix} w_{00} & w_{01} & \dots & w_{0k-1} \\ w_{10} & w_{11} & \dots & w_{1k-1} \\ \dots & \dots & \dots & \dots \\ w_{n-10} & w_{n-11} & \dots & w_{n-1k-1} \end{pmatrix}$$

3. Compute the score (or measure the similarity)
The most common similarity measure for the standard VSM is the cosine of the angle between the vectors:

$$sim(d_i, d_j) = \cos(\vec{d}_i, \vec{d}_j) = \frac{\sum_{p=0}^{k-1} (w_{i_p} \times w_{j_p})}{\sqrt{\sum_{p=0}^{k-1} w_{i_p}^2 \times \sum_{p=0}^{k-1} w_{j_p}^2}}.$$

We use this measure to determine the similarity between two applications, i.e. the score between A_i and A_j : $s(A_i, A_j) = sim(d_i, d_j)$.

3.3 Hierarchical clustering algorithm

From the vectors d_0, d_1, \dots, d_{n-1} and their similarity computation produced in 3.2, we use the hierarchical clustering algorithm [7] to produce the application interaction hierarchy:

1. Consider each d_i is a single cluster, we have n clusters, the distances between a pair of clusters i and j (in this step): $D(i, j) = 1 - sim(d_i, d_j)$.
2. Find the most similar pair of clusters (i.e. $\min(D(i, j))$) and merge them into a single cluster, so that we have one cluster less.
3. Compute distances between the new cluster and each of the old clusters.
4. Repeat steps 2 and 3 until all items are clustered into a single cluster of size n .

Step 3 can be done in several ways such as single-linkage, complete-linkage, or average-linkage clustering [6]. Applying the single-linkage, the formula to calculate the distance between two clusters C_1, C_2 :

$$D(C_1, C_2) = \min_{i \in C_1, j \in C_2} [D(i, j)]$$

The output of the presented clustering algorithm is a binary tree (Fig. 3), this tree is transformed to an application interaction hierarchy based on the degree of similarity between the applications ⁶ (Fig. 4).

4 Navigating between applications based on the user's application of interest

The system aims to find out the target application with a minimal number of dialogue turns. Based on the application interaction hierarchy produced in section 3, the preliminary experimented work presented in [5], and the textual content provided by the user, the user-system interaction process is described in detail in the following algorithm:

⁶ for example, if a node N_1 has two child nodes (N_2, N_3) and N_2 has two child nodes N_4, N_5 and $[D(N_2, N_3) - D(N_4, N_5)] \leq \alpha$, α is a predefined threshold, then N_2 is removed; N_4, N_5 become the child nodes of N_1 .

1. Start

The system starts with a generic prompt: “What can I do for you?” (similar to the internal GDN “Start” described in [1]).

2. Active node determination

When receiving a user’s request, the system first represents the request in the form of a vector $q = (q_1, q_2, \dots, q_k)$ using the set of k index terms described in section 3.2, and then determines the active node on the hierarchy by the following steps:

(a) Score computation

Compute the similarity between q and d_0, d_1, \dots, d_{n-1} , we obtain a set of scores s_0, s_1, \dots, s_{n-1} : $s_i = \text{sim}(d_i, q)$.

For example, in Fig. 2 we have $s_0 = 0.85, s_1 = 0.9, \dots, s_9 = 0.15$.

(b) Upward propagation

Select the best scores at each level and propagates them upward until the root is reached.

For example, in Fig. 2 we have $s_{0-2} = \max(s_0, s_1, s_2) = 0.9$.

(c) Downward traversal to determine the active node

Start from the root, compute the difference between two highest score child nodes, if this difference is below a certain threshold (we call this threshold the internal node stop threshold $t_s : (0 < t_s \leq 1)$), then stop. If not, go down to the highest score child node and continue to determine the active node.

For example, in Fig. 2, starting from M_{0-9} , we calculate the difference between M_{0-4} and M_{5-9} : $\text{dif}(M_{0-4}, M_{5-9}) = 0.4$, it is greater than $t_s = 0.15$, then we go down to M_{0-4} , we still have $\text{dif}(M_{0-2}, M_{3-4}) = 0.2$ is greater than t_s then we go down to M_{0-2} , we have $\text{dif}(M_0, M_1) = 0.05 < t_s$ then M_{0-2} is the active node.

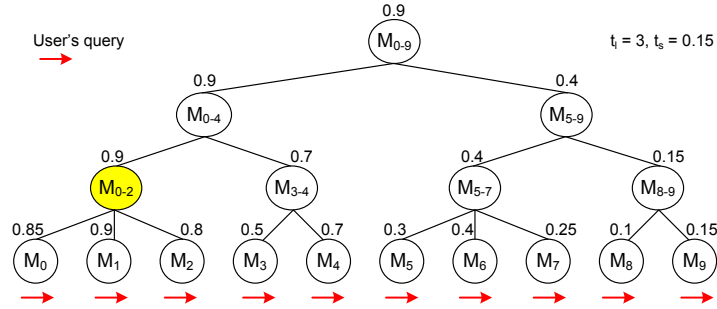


Fig. 2. Determine the active node based on the user’s query

3. Response generation

The active node identified in the previous steps can be a root, an internal node or a leaf. Two types of response depending on the position of the active node are:

- (a) The active node is the root or an internal node
 In this case, the functionality of the active node is similar to the list processing GDN described in [1]. The system shows a list of application candidates belonging to the active node and their score is not below the highest score leaf outside the active node (e.g. in fig. 2, M_4 is the highest score leaf outside the active node M_{0-2}). To avoid showing a bulky list to the user (particularly in case of vocal dialogue), the maximum number of application candidates is limited by a threshold called the list processing threshold t_l , with t_l is a positive interger. The user can determine to go **next** (i.e. show the t_l following application candidates), **previous** (i.e. show the t_l previous application candidates), **stop** (i.e. restart the dialogue), **up** (i.e. move to the upper level on the hierarchy), **down** (i.e. move to the highest score child node), or **select** the desired application. If the user does not change the active node (i.e he does not use the command up or down) and after browsing all the applications (belonging to the active node) he could not find his desired applications, the system temporarily assigns the scores of the browsed leaves to zero and goes back to step 2b.
- (b) The active node is a leaf
 The application takes control and interacts with the user as an application-specific dialogue system. If the user's request is out of the application's domain, go back to step 2.

An example of the algorithm with $n = 10$ and $t_l = 3$ is presented in Fig. 5 and explained in detail in section 5.3.

5 Scenario example

This section illustrates, on the global level, the process of developing a dialogue system accessing 10 applications in the ICIS domain using three steps presented in sections 2, 3, 4. The applications are: car route navigation (A_0), air route navigation (A_1), traffic lanes (A_2), map and fire management (A_3), tunnel sensors management (A_4), weather forecast (A_5), virtual control room (A_6), road surface temperature monitoring (A_7), patient information search (A_8), and medical worker verification (A_9).

5.1 Step 1

Applying the RDPM, we produce the finalized dialogue models: M_0, M_1, \dots, M_9 .

5.2 Step 2

From the finalized dialogue models, we create the application interaction hierarchy (cf. Fig. 4). Finalized dialogue models for the root and internal nodes are the list processing mGDNs produced by the RDPM. The role of each node is to select a subset of the applications belonging to it, for example the role of M_{0-2} is to select a subset of $\{A_0, A_1, A_2\}$.

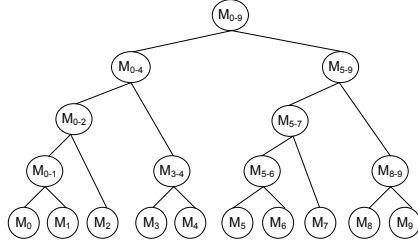


Fig. 3. Binary tree

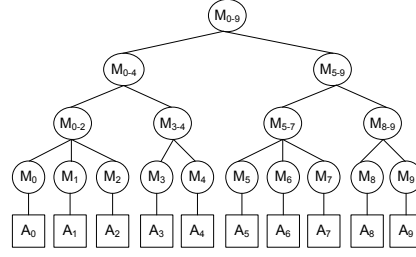


Fig. 4. Application Interaction Hierarchy

5.3 Step 3

An example of the system-user interaction is presented in Fig. 5. The “Start” mGDN sends the system’s prompt S_1 to the user. According to the content of the user’s prompt U_2 , the active node M_{0-2} is determined. M_{0-2} asks the user to select an application from the list $\{A_0, A_1, A_2\}$ (all three applications are shown because $t_l = 3$). Based on the user’s answer in U_4 , M_0 is activated. In steps from 5 to $k - 1$, M_0 interacts with the user as an application-specific dialogue system. In step k , the user’s request U_k is out of M_0 ’s application domain, M_0 then forwards U_k to the system. The system analyzes U_k and activates M_8 . M_8 continues the interaction with the user and processes the out of the application domain case in a similar manner M_0 has done.

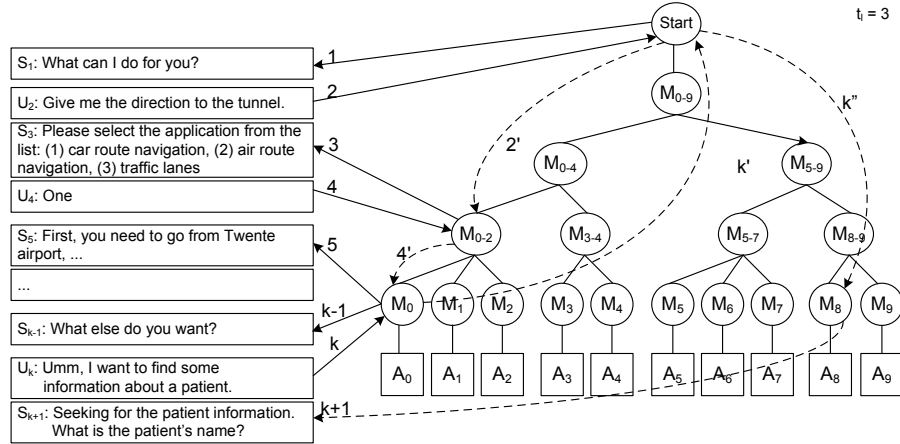


Fig. 5. Navigating between the applications

6 Conclusion

We have presented a framework for the development of interfaces for multi-application dialogue systems. Three important steps in the framework are described and illustrated by a scenario example.

Currently, the RDPM software toolkit is available for the development of finalized dialogue models for single applications. It has been used in three research projects: InfoVox ⁷, INSPIRE ⁸, IM2.MDM ⁹ to validate the principle idea of the methodology, and is being extended for the development of a large number of applications in the ICIS domain. The practical result shows that from a simple application, we can develop an initial dialogue model in several hours. The dialogue manager, the most important part of dialogue prototyping, covers most of the application independent dialogue functionalities (i.e. branching logic, dialogue dead-end management strategy, confirmation strategy, dialogue termination strategy, incoherencies, strategy defining level of initiative, etc.) Therefore, we can re-use the dialogue manager and the other modules described in section 2.3 for the development of multi-application dialogue systems.

Some initial work toward developing the application interaction hierarchy and navigating between the applications (sections 3 and 4) has been analyzed and implemented (e.g. NLP Pre-Processing Tool, VSM). The multi-application dialogue system for ICIS domain presented in section 5 is currently under development.

7 Future work

Two main possible extensions of the generic dialogue modeling methodology we plan to study in the future are *crossing-application* and *task selection*.

7.1 Crossing-application

The application interaction hierarchy created in section 3 can be used to manage several concurrent applications (i.e. crossing-application). This extension is significant when the user wants to simultaneously execute several applications in order to achieve his goal in an optimal way. For example, in the scenario presented in section 5, the user's goal is to find out an optimal route for sending a rescue team to the disaster site. Suppose that the system contains two applications, the car root navigation application and the traffic lanes application. Obviously, if the user can interact with both these applications simultaneously, his goal can be more quickly satisfied than he does with each application sequentially.

7.2 Task selection

In the definition of the application interaction hierarchy in the section 3.1, we mentioned that each leaf corresponds to an application. In task-oriented dialogues, each application usually consists of several tasks. We can extend the hierarchy for identifying a task or a set of tasks in an application. To achieve this goal, the hierarchy will be constructed from the set of tasks in the same way we have done for the set of applications. Further work on task sharing (i.e. one task appears in several applications) will be studied.

⁷ <http://liawww.epfl.ch/Research/infobox.html>

⁸ <http://www.knowledge-speech.gr/inspire-project/index.html>

⁹ <http://www.issco.unige.ch/projects/im2/mdm/>

8 Acknowledgements

We would like to thank Martin Rajman for his useful suggestions and comments about the generic dialogue modeling idea presented in this paper, Lynn Packwood for her advices in English writing, and the two anonymous reviewers for their useful comments on the first version of this paper. This work is part of the ICIS program (<http://www.decis.nl/html/icis.html>). ICIS is sponsored by the Dutch government under contract BSIK 03024.

References

1. T.H. Bui, M. Rajman, and M. Melichar. Rapid dialogue prototyping methodology. In *Proceedings of the 7th International Conference on Text, Speech & Dialogue (TSD 2004)*, P. Sojka, I. Kopeček & K. Pala (Eds.), *Lecture Notes in Computer Science*, Springer-Verlag, Berlin Heidelberg New York, pages 579–586, Brno, Czech Republic, September 2004.
2. T.H. Bui, M. Rajman, and S. Quarteroni. Extending the Rapid Dialogue Prototyping Methodology: User Modeling. Technical Report WP4 Deliverable 4.3, Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland, August 2004.
3. J. Chu-Carroll and B. Carpenter. Vector-based natural language call routing. *Computational Linguistics*, 25(3):361–388, 1999.
4. D.R. Cutting, D.R. Karger, J.O. Pedersen, and J.W. Tukey. Scatter/gather: a cluster-based approach to browsing large document collections. In *SIGIR '92: Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 318–329, New York, NY, USA, 1992. ACM Press.
5. E. Forler. *Intelligent user interface for specialized web sites*. Ecole Polytechnique Federale de Lausanne (EPFL), Lausanne, Switzerland, diploma thesis edition, 2000.
6. A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.
7. S.C. Johnson. Hierarchical Clustering Schemes. *Psychometrika*, (2):241–254, 1967.
8. B. Lin, H. Wang, and L. Lee. Consistent dialogue across concurrent topics based on an expert system model. In *Sixth European Conference on Speech Communication and Technology (EUROSPEECH'99)*, Budapest, Hungary, 1999.
9. A. Lisowska, M. Rajman, and T.H. Bui. Archivus: A system for accessing the content of recorded multimodal meetings. In Samy Bengio and Hervé Bourlard, editors, *Machine Learning for Multimodal Interaction, First International Workshop, MLMI 2004, Martigny, Switzerland, June 21-23, 2004, Revised Selected Papers*, volume 3361, pages 291 – 304. Springer.
10. H.I. Ng and K.T. Lua. Dialog Input Ranking in a Multi-Domain Environment Using Transferable Belief Model. In *Proceedings of the 4th SIGdial Workshop on Discourse and Dialogue*, Sapporo, Japan, July 2003.
11. M. Rajman, R. Besanon, and J.-C. Chappelier. Le modèle DSIR : une approche à base de sémantique distributionnelle pour la recherche documentaire. *revue Traitement Automatique des Langues (TAL)*, 41(2), Paris, 2001.
12. J. Vrugt and T. Portele. Application-Independent Knowledge-Processing in a Task-Oriented Speech-Dialog-System. *it - Information Technology*, 46(6):306–314, December 2004. <http://www.it-inftech.de>.