A Framework to Integrate Ubiquitous Knowledge Modeling

Porfírio Filipe [1, 2], Nuno Mamede [1, 3]

[1] L[2]F INESC-ID – Spoken Languages Systems Laboratory, Lisbon, Portugal
{porfirio.filipe, nuno.mamede}@l2f.inesc-id.pt
[2] ISEL – Instituto Superior de Engenharia de Lisboa, Lisbon, Portugal
[3] IST – Instituto Superior Técnico, Lisbon, Portugal

Abstract

This paper describes our contribution to let end users configure mixed-initiative spoken dialogue systems to suit their personalized goals. The main problem that we want to address is the reconfiguration of spoken language dialogue systems to deal with generic plug and play artifacts. Such reconfiguration can be seen as a portability problem and is a critical research issue. In order to solve this problem we describe a hybrid approach to design ubiquitous domain models that allows the dialogue system to perform recognition of available tasks on the fly. Our approach considers two kinds of domain knowledge: the global knowledge and the local knowledge. The global knowledge, that is modeled using a top-down approach, is associated at design time with the dialogue system itself. The local knowledge, that is modeled using a bottom-up approach, is defined with each one of the artifacts. When an artifact is activated or deactivated, a bilateral process, supported by a broker, updates the domain knowledge considering the artifact local knowledge. We assume that everyday artifacts are augmented with computational capabilities and semantic descriptions supported by their own knowledge model. A case study focusing a microwave oven is depicted.

1. Introduction

A Spoken Dialogue System (SDS) should be a computational entity that allows access to any device by anyone, anywhere, at anytime, through any media, allowing its user to focus on the task, not on the tool.

This paper describes our research in designing knowledge-based everyday devices that can be dynamically managed by a SDS. We are mostly interested on home environments as a test bed of other spaces such as the office, the car or public spaces. The devices throughout the house can be in constant contact with each other. These devices must be easily installed and personalized according to the users' wishes.

Only in the last decade, with major advances in speech technology, have large-scale working systems been developed and, in some cases, introduced into commercial environments (McTear, 2002). Nevertheless, many implementations of dialogue managers perform input interpretation, output generation, and domain specific tasks. These tasks are usually domain dependent because its designs consider particular requirements of each domain.

This approach may easily lead to situations in which the dialogue manager is a monolithic component. Monolithic components make it harder to build modular, distributed systems, and reusable components (O'Neill & McTear, 2000). Typically, these issues are addressed through architectures that integrate reusable components. Recent progresses can be seen in (Bohus & Rudnicky, 2003; O'Neill *et al*., 2003; Pakucs, 2003; Polifroni & Chung 2002; Neto *et al*., 2003; Turunen & Hakulinen, 2003).

The exponential drop in microprocessor cost over time has enabled appliance manufacturers to pack increasingly complex feature sets into appliances such as air conditioners, refrigerators, washing machines, and more. As household appliances grow in complexity and sophistication, they become harder and harder to use, particularly because of theirs tiny display screens and limited keyboards. In fact, this can be seen in the growing amount of information on manuals and inscriptions or symbols on the appliance itself. The SDSs have here an opportunity to handle this amount of technical information and help users to directly invoke tasks as a way to solve the interface problems in an effortless style.

2. Knowledge-based Approach

The problem that we want to address is the reconfiguration of a SDS to work with generic plug and play artifacts. Such reconfiguration can be seen as a portability problem (Zue & Glass, 2000). Our proposal is not about plug and play environments but about an important issue around them: the agree on meaning.

The main question is: "*Is it possible for a SDS to deal with an artifact that was not previously known?*". In this scenario, the SDS does not know which capabilities will be found on a generic artifact. In order to address this problem, the capabilities of the artifact must be described at a knowledge or conceptual level (Newell, 1982). Each artifact, provided with a semantic interface, will assist the dialogue manager to understand its functional and structural features.

Knowledge modeling is a creative process. There is no correct way to model the knowledge of a domain. The best solution usually depends on the application that one has in mind and the extensions that one anticipates. Before we start to represent a domain, we have to decide the use of the knowledge will have and, consequently, how detailed or general the model is going to be. We need to determine which representation would simplify the algorithms, be more intuitive, more extensible, and more maintainable.

We use a hybrid approach, which was introduced in (Filipe & Mamede, 2004), to model the domain knowledge, combining a top-down (applied to acquire global knowledge) and bottom-up (applied to acquire local knowledge) approaches. Both approaches converge

to reach middle-level concepts directly associated to the artifact semantic interface.

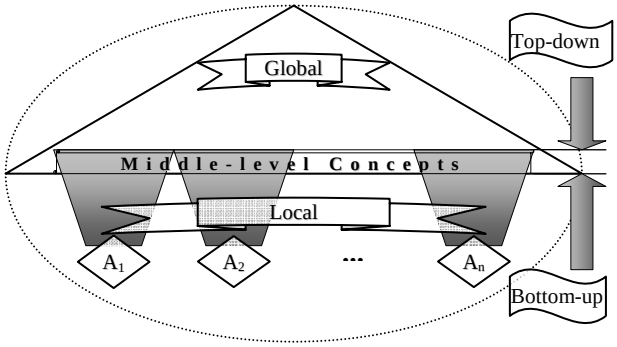

Figure 1: Schema of the Knowledge-based Approach

Figure 1 shows a schema of the knowledge-based approach where A_n is a generic artifact. This schema divides the domain knowledge in global and local knowledge under an artifact-centered perspective. These two kinds of knowledge are viewed, by the SDS, as a centralized knowledge model, which is managed by a broker.

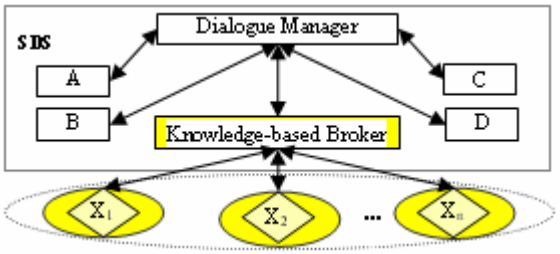

Figure 2: Adaptation of the SDS to the Domain

Figure 2 shows an architecture schema where X_n is a generic artifact and A, B, C, and D are the other components of the SDS.

The knowledge of the domain is represented by concepts that map unique IDentifiers (ID), which are alphanumeric codes, into a list of terms or more generically into a list of Multi-Word Unit (MWU) (Daille *et al*. 1994).

A MWU list contains linguistic variations associated with each concept, such as synonymous or acronyms (or other multilingual equivalent). To declare a concept, at global or local level, one has to bind an ID to a MWU list (see Table 2).

The concepts are organized in predefined groups: *general*, *task*, *quantity*, *attribute*, and *artifact*. A group contains collections of concepts that can be seen as a controlled vocabulary.

2.1. Local Knowledge Modeling

The local domain knowledge is defined considering all the available artifacts (belonging to the surrounded environment) and is modeled using a bottom-up approach. The integration process of an artifact in a SDS is achieved by building a set of three layers, which would potentially cover all the relevant artifact features:

(i) The first layer is an artifact driver that provides an elementary abstraction of the device expressing the primitive capabilities. For instance, if the artifact is a door we must be able, through the artifact driver, to open or to close the door and to ask about its state (opened/closed);

(ii) The second layer is an adapter that transforms the first layer into a more convenient interface, considering the artifact class. For instance, the adapter might transform labels into infrared command codes;

(iii) The third layer includes particular features of the artifact, bearing in mind, for instance, variations of the artifact commercial model.

The third layer is personalized to the SDS needs. For each capability of the third layer, we must define an artifact task descriptor.

We consider two kinds of tasks: *action* and *perception* tasks. A perception task cannot modify the state of an artifact and an action task can.

A task descriptor is a semantic representation of an artifact capability and has a name and optionally an input list, an output list, and assumptions. A name is a concept from the predefined *task* group. Table 1 presents a task descriptor where the "*" means mandatory fulfilling.

slot			value
name*			task
input list	**input role**	**name**	attribute
		range*	attribute or quantity
		restriction	rule
		default	attribute or quantity
	other input roles …		
	pre-condition		rule
output list	**output role**	**name**	attribute
		range*	attribute or quantity
	other output roles …		
	pos-condition		rule
assumptions	**initial condition**		rule
	final condition		rule

Table 1: Artifact Task Descriptor

The input list, that describes all input parameters, has a set of optional input roles. An input role, that describes one input parameter, has a name, a mandatory range, a restriction, and a default. The name is member of the attribute group and is optional. The range is member of the attribute or quantity groups. The restriction is a rule that is materialized as a logical formula. For instance, if the range is a positive integer (quantity) and we want to assure that the parameter is lower than 10, then we must indicate the restriction rule: "name < 10". The default of the input role is a member of the attribute or quantity groups. When the default is not provided the input role must be filled.

The output list, that describes all output parameters, has a set of optional output roles. An output role, that describes one output parameter, is like an input role without restriction rule and default.

The rules of the task descriptor allow three kinds of validation: restriction rule to perform individual parameter validation, pre-condition to check input parameters before task execution, and pos-condition to check output parameters after task execution. A restriction can refer the associated input role, a pre-condition can refer task input role names and a pos-condition can refer output role names.

The assumptions perform state validation: the initial condition (to check the initial state before task execution) and the final condition (to check the final state after task

execution). Assumptions can refer role names and results of perception task calls.

slot		value
concept declaration*	**global**	group*
		collection
		ID*
	local	group*
		collection
		ID*
		MWU list*
other concept declarations …		
class declaration*	**name**	artifact
	class	artifact*
	super class list	artifact list*
task descriptor*		
other task descriptors …		

Table 2: Artifact Semantic Interface Descriptor

Table 2 describes a device semantic interface where mandatory parts are signaled by "*". An artifact has one semantic interface descriptor, which integrates concept declarations, class declarations, and task descriptors. A concept declaration refers a group in which the concept belongs, an optional collection of related concepts, an ID, and a MWU list. A local concept declaration is simultaneously a local concept definition. A global concept declaration is a forward declaration, in other words, a reference to a concept globally declared. Therefore, in a semantic interface descriptor, a global concept declaration does not need a MWU list. The class declaration refers members of the *artifact* group and has optionally a name, a class, and a super class list. Finally, the semantic interface descriptor ends with one or more task descriptors (detailed in Table 1). When one is filling a task descriptor, using concept identifiers, if one does not know the identifier of the current concept (because it is not previously declared as global) it must be declared as local in order to obtain the needed identifier.

2.2. Global Knowledge Modeling

The global domain knowledge is modeled using a top--down approach. Figure 2 shows an example of a type hierarchy.

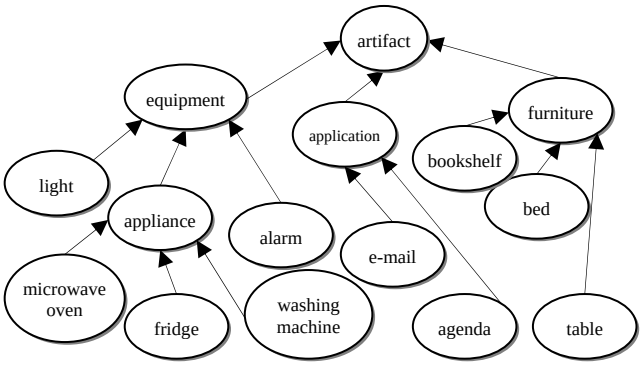

Figure 2: Representation of Top-level Concepts

Following the top-down approach, we start the modeling of the global knowledge by defining its scope or domain, for instance, the home environment. Therefore, we have to decide about the important classes and subclasses of artifacts that we should consider. Then, we have to identify which are the collections of concepts (middle--level concepts) associated with the selected artifact classes. At the end, we can sketch a list of competency questions such as "*Does the model contain enough knowledge to cover the relevant SDS needs?*" or "*Do we have the needed detail level for a particular case study?*".

3. The Knowledge-based Broker

The broker is a SDS component that assumes two responsibilities: performs the domain knowledge management and maintains the predefined global knowledge. This component allows the customization of the dialogue manager that should only be concerned with phenomena related to the user's dialogue.

The main goal of the broker is to support the communication interoperability between the SDS and the devices in the pervasive application domain. To achieve this goal, the broker has an architecture with three independent knowledge components: *discourse model*, *world model*, and *task model*, see Figure 4. This architecture was adapted from Unified Problem-solving Method Development Language (UPML) (Fensel *et al.*, 1999).

The *discourse model* holds the global definition of all concepts in the domain. These concepts are organized in four groups of collections. The *general* group maintains all the collections. The *task* group contains two collections *action* and *perception* that holds the task names. The *quantity* group contains two collections *number* (integer, real, positive, integer, …) and *measure* (time, power, …). The *attribute* group contains collections of concepts that are usually attributes (color, shape, texture, …). The *artifact* group contains the set of artifact classes (artifact, equipment, application, furniture, appliance, …) that can by referred in the *type hierarchy*.

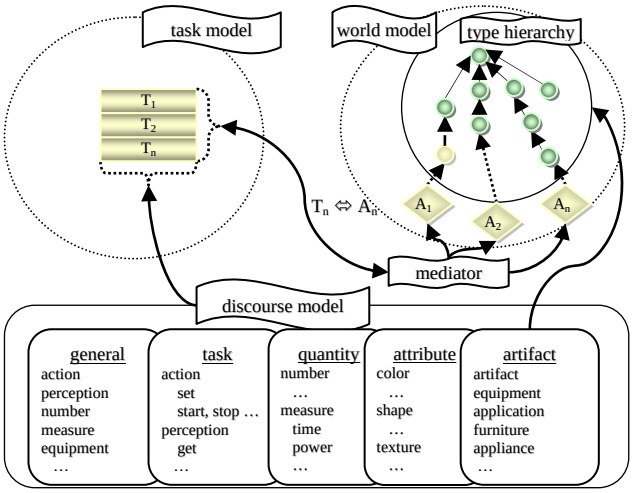

Figure 3: Broker's Knowledge-based Architecture

The *world model* has two components: *type hierarchy* and *mediator*. The *type hierarchy* organizes artifact classes. The *mediator* manages artifacts instances linked to their classes. Each artifact instance has a *registry list* and information to access the network. The *registry list* binds local concepts to equivalent global concepts. Two concepts are equivalent when they have the same ID or the same MWU list. The *task model* contains task descriptors (T_n) that are associated to artifact (A_n) instances through links ($T_n \Leftrightarrow A_n$).

When the broker detects the activation of an artifact its three knowledge components are updated on the fly, processing, the artifact semantic interface descriptor:

(i) The declared concepts are processed. For all local concepts, without an equivalent global concept (check MWU List), a new global concept is defined using the MWU list of the local concept. A reference to a global concept that is not defined produces an error message;

(ii) The class declaration is processed. If the *class* does not exist in the *type hierarchy* the information about the *super classes* is used to determine the missing "is-a" relations between the declared *class* and the *classes* belonging to the *type hierarchy*;

(iii) The task descriptors are processed. The task descriptors that refer undefined concepts are not considered and an error message is reported. For others task descriptors, a link ($T_n \Leftrightarrow A_n$) is established to the respective artifact instance. The global identifiers of the task descriptors replace (using the artifact *registry list*) the local concepts identifiers. All the processed task descriptors are added to the *task model*.

When an artifact is deactivated, the artifact instance and the related task descriptors are removed. The global concepts, referred by the artifact *registry list*, that are not referred by any other *registry list* are removed from the *discourse model* and from the *type hierarchy*.

The use of the described broker's architecture allows successive improvements of the global knowledge by including concepts earlier defined as local. The basic idea is that the global knowledge can evolve, considering that, the improvement of the global knowledge leads to a better and efficient domain model that is able to continue operational with the former artifacts.

4. Case Study: A Microwave Oven

This section describes an example of a semantic interface used to adapt a microwave oven to be controlled through a SDS. Microwave ovens are typically used to heat up and defrost food. However, they are also used for cooking vegetables, fruit, fish and poultry.

Type	Capability	Input	Output
action	start	-	-
action	stop	-	-
action	select	duration	-
action	select	power	-
perception	ask	-	state

Table 3: Capabilities of the Microwave Oven

We assume that the microwave oven has a driver, which allows access to the primitive artifact capabilities shown in Table 3.

Capability	Input	Range	Restriction
select	duration	minute	duration > 0 and duration <= 30
select	power	watt	power >= 100 and power <= 900

Table 4: Restrictions Rules

Table 4 focus on particular microwave restriction rules that depend on the microwave oven commercial model. Now we have to extend the knowledge about power by relating it with cooking notions.

Table 5 presents the relation between power and some cooking actions.

Capability	Power
cook	900 W
reheat	900 W
defrost	500 W
keep	100 W

Table 5: Cooking Actions and Power

Furthermore, we continue to extend the knowledge about cooking using this particular microwave oven introducing some relations between time duration, types of food, cooking actions and a reference amount, see Table 6.

In Table 6, we have to be careful with unit's conversions, because the users can choose between unit systems. Depending on the type of food and its amount, the duration values presented in Table 6 should be adjusted.

Duration	Food	Capability	Amount
8 min	cod steak	defrost	2*400g
4 min	shelled prawns	defrost	200g
9 min	roast beef	cook	1Kg
8 min	carrot	cook	400g
30 s	rice	reheat	150g
30 s	coffee	reheat	3.5 fl oz

Table 6: Duration, Food, Cooking Actions and Amount

This process might continue, adding all the knowledge we need. Before we can define the semantic interface, we must define the complete set of available capabilities. Consequently, we use the composition of former capabilities to define new capabilities that take advantage of the knowledge include in Table 5 and Table 6.

We present some lines in Table 7, which illustrates the device capabilities that can be used by the SDS. Considering this microwave oven interface when the SDS receives the command "*defrost cod steaks*" the duration is adjusted to 8 minutes and the power is set to 500 watts.

Capability	Composition
cook	select((power) **Table5**('cook'))
reheat	select((power) **Table5**('reheat'))
defrost(food)	select((power) **Table5**('defrost')); select((duration) **Table6**(food, 'defrost'))
keep	select((power) **Table5**('keep'))

Table 7: Composition of Capabilities

Finally, Table 8, Table 9 and Table 10 present the semantic interface of the microwave oven using the structure depicted in Table 2.

Table 8 shows concept declarations. Table 9 shows type hierarchy. Table 10 shows task descriptors.

The tag (global) in column MWU list means that we assume the concept is already defined as global.

concept declarations			
group	collection	ID	MWU list
artifact		id-microwave	(*global*)
artifact	-	id-appliance	(*global*)
artifact	-	id-equipment	(*global*)
attribute	-	id-aliment	(*global*)
attribute	-	id-duration	(*global*)
attribute	-	id-power	(*global*)
attribute	-	id-boolean	(*global*)
attribute	-	id-state	(*global*)
attribute	general	id-food	food, foodstuff, meal
attribute	id-food	id-carrot	carrot
attribute	id-food	id-cod-steak	cod steak
attribute	id-food	id-coffee	coffee, café
attribute	id-food	id-rice	rice
attribute	id-food	id-roast-beef	roast beef
attribute	id-food	id-shelled-prawn	shelled prawn
quantity	-	id-minute	(*global*)
quantity	-	id-watt	(*global*)
task	action	id-close	close, lock
task	action	id-cook	(*global*)
task	action	id-defrost	defrost, unfreeze
task	action	id-keep	keep, hold, maintain
task	action	id-reheat	reheat, warm up
task	action	id-select	select, accept, appoint
task	action	id-start	start, activate, initiate, switch on, turn on
task	action	id-stop	stop, deactivate, terminate, switch off, turn off
task	perception	id-ask	ask, request

Table 8: Concept Declarations

type hierarchy	
class	id-microwave
super class	id-appliance, id-equipment

Table 9: Type Hierarchy

task descriptors			
name	id-ask		
output list	role	name	id-state
		range	id-state
name	id-start		
assumptions	initial condition		stopped
	final condition		started
name	id-stop		
assumptions	initial condition		started
	final condition		stopped
name	id-select		
input list	role	name	id-duration
		range	id-minute
		restriction	id-duration>0 and id-duration<=30
assumptions	initial condition		stopped
name	id-select		
input list	role	name	id-power
		range	id-watt
		restriction	id-power>=100 and id-power <=900
assumptions	initial condition		stopped
name	id-cook		
assumptions	initial condition		stopped
name	id-reheat		
assumptions	initial condition		stopped
name	id-defrost		
input list	role	name	id-aliment
		range	id-food
assumptions	initial condition		stopped
name	id-keep		
assumptions	initial condition		stopped

Table 10: Task Descriptors

5. Testing the Approach

Figure 4 show a screenshot of the domain simulator, developed originally for Portuguese users. On the bottom of the screen, one can see data about the microwave oven.

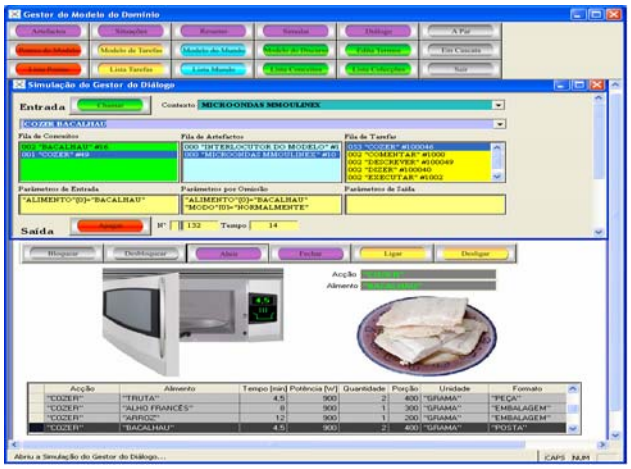

Figure 4: Screenshot of the Domain Simulator

The domain simulator allows the debugging and the simulation of the interaction with the SDS dialogue manager. We can attach and detach artifacts, request the execution of tasks, obtain the answers and observe the artifacts behaviors. We can access the represented knowledge and the task execution progress.

The proposed approach was tested in a home environment domain with common devices and household appliances that are Air Conditioner (63 - concepts), Freezer (96 - concepts), Fryer (92 - concepts), Light Source (62 - concepts), Microwave Oven (167 - concepts), Table (48 - concepts), Water Faucet (63 - concepts), Window (44 - concepts) and a Window Blind (65 - concepts). All the artifacts are using about 700 concepts, that defines N_1=700. Initially the predefined global knowledge is using 261 concepts. After the activation of all artifacts, the broker's knowledge model retains 360 concepts, that defines N_2=360. The knowledge integration rate is N_1/N_2*100=51%. The knowledge modeled for each artifact and for the broker is supported by relational databases with 19 (nineteen) tables.

6. Conclusions

We have described an approach to deal dynamically with communication interoperability between a SDS and a set of heterogeneous artifacts. This approach is a significant contribution to improve the flexibility, and simultaneously the robustness, of the SDS being developed in our lab. The presented ideas have been applied, with success, in a set of artifacts that represents a home environment. As future work, we expect to explore, more deeply, the knowledge integration perspective. We believe that in the near future, the SDSs are not only useful but also easy to use and accommodating, such that users will prefer them over alternative means of managing their needs.

7. Acknowledgements

This paper was partially supported by project POSI/PLP/41319/2001.

8. References

Bohus, D. and Rudnicky, A. (2003). RavenClaw: Dialog Management Using Hierarchical Task Decomposition and an Expectation Agenda. In *Eurospeech 2003*. Geneva, Switzerland.

Daille, B., Gaussier, E., Lange, J. (1994). Towards Automatic Extraction of Monolingual and Bilingual Terminology. *COLING 94* 515-521.

Fensel, D., Benjamins, V., Motta, E., Wielinga, B. (1999). UPML: A Framework for Knowledge System Reuse. In *IJCAI 1999*. Stockholm, Sweden.

Filipe, P. and Mamede, N. (2004). Towards Ubiquitous Task Management. In *Interspeech 2004*. Jeju Island, Korea.

McTear, M. (2002). Spoken Dialogue Technology: Enabling the Conversational Interface. In *ACM Computing Surveys*, Volume 34.

Neto, J., Mamede, N., Cassaca, R. and Oliveira, L. (2003). The Development of a Multi-purpose Spoken Dialogue System. In *Eurospeech 2003*. Geneva, Switzerland.

Newell, A. (1982): The Knowledge Level. *Artificial Intelligence*, 18, pp. 87-127.

O'Neill, I. and McTear, M. (2000). Object-Oriented Modelling of Spoken Language Dialogue Systems. In *Natural Language Engineering 6*, Cambridge University Press, Cambridge, UK.

O'Neill, I., Hanna, P., Liu, X. and McTear, M. (2003). An Object-Oriented Dialogue Manager. In *Eurospeech 2003*. Geneva, Switzerland.

Pakucs, B. (2003). Towards Dynamic Multi-Domain Dialogue Processing. In *Eurospeech 2003*. Geneva, Switzerland.

Polifroni, J. and Chung, G. (2002). Promoting Portability in Dialogue Management. In *ICSLP 2002*. Denver, Colorado, USA.

Turunen, M. and Hakulinen, J. (2003). JASPIS^2 – An Architecture for Supporting Distributed Spoken Dialogues. In *Eurospeech 2003*. Geneva, Switzerland.

Zue, V., Glass J. (2000): Conversational Interfaces: Advances and Challenges. *IEEE*, Vol. 88, No. 8.