Hybrid Knowledge Modeling for Ambient Intelligence

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

[1] L[2]F INESC-ID – Spoken Language Systems Lab, 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 research in enhance everyday devices as a solution to adapt Spoken Dialogue Systems (SDS) within ambient intelligence. In this context, a SDS enables universal access to ambient intelligence for anyone, anywhere at anytime, allowing the access to any device through any media or language. The main problem that we want to address is the spontaneous configuration of SDS to deal with a set of arbitrary plug and play devices. Such problem is resumed as a portability feature and is a critical research issue. We propose a hybrid approach to design ubiquitous domain models to allow the SDS to recognize on-the-fly the available devices and tasks they provide. When a device is activated or deactivated, a broker's knowledge model is updated from device's knowledge model using a knowledge integration process. This process was tested in the home environment represented by a set of devices.

Keywords: Ambient Intelligence, Spoken Dialogue System.

1 Introduction

Ambient Intelligence (AmI) [1] is a vision that expresses a recent paradigm in information technology. It can be defined as the fusion of two important trends: ubiquitous computing and social user interfaces. It is supported by advanced networking technologies, which allow robust, ad-hoc networks to be formed by a broad range of mobile devices. These context aware systems combine ubiquitous information, communication, and entertainment with enhanced personalization, natural interaction and intelligence. This kind of environment is characterized by the following basic elements: ubiquity, awareness, intelligence, and natural interaction.

Ubiquity refers to a situation in which we are surrounded by a large amount of interconnected embedded systems, which are invisible and moved into the background of our environment. Awareness refers to the ability of the system to locate and recognize objects and people, and their intentions. Intelligence refers to the fact that the digital surrounding is able to analyze the context, adapt itself to the people that live in it, learn from their behavior, and eventually to recognize as well as show emotion. Natural interaction finally refers to advanced modalities like natural speech and gesture recognition, as well as speech synthesis, which will allow a much more human like communication with the digital environment than is possible today.

Ubiquitous computing [2] or pervasive computing are emerging disciplines bringing together elements from distributed systems, mobile computing, embedded systems, human computer interaction, computer vision and many other fields. This means integration of microprocessors into everyday devices like household appliances, furniture, clothing, toys and even paint. Networked computing devices will proliferate in this landscape, and users will no longer be tethered to a single computing device. People on the move will become networks on the move as the devices they carry network together and with the different networks around them.

The nature of devices will change to form augmented environments in which the physical world is sensed and controlled in such a way that it merges with the virtual world [3]. This massive use of such devices will fill the gap between the cyberspace and the real world.

Typically, within a pervasive computing environment two key concepts are designated by: "*device*" and "*service*". Devices include conventional computers, small handheld computers (PDAs), printers, and more specialized network devices, such as a thermometer or a household appliance. Services include any sort of network service that might be available. In fact, most devices are represented on the network by one or more services. Furthermore, a single network attached device may implement several services, e.g., a network printer may provide printing and fax (and who knows what else), all in a single device. In this context, devices and services are considered essentially equivalent and interchangeable. Together, these will be termed "*entities*" or "*resource*s" on the network.

Ideally, entities should interoperate with other entities without pre-existing knowledge. This is a key aspect of spontaneous configuration.

A pervasive computing environment offers us an interesting starting point of discussion regarding the goal of achieving Plug and Play (PnP) functionality and its subsequent application to manage and control devices. There are several imaginable levels of PnP, which can be summarized in the following possibilities:

(i) No PnP – The environment can handle a fixed set of devices that are always connected;
(ii) Weak PnP – The environment can handle a fixed set of devices that can be connected to or disconnected from the network;
(iii) Medium PnP – The environment can handle a fixed set of device classes and only new instances of these device classes can be plug and played;
(iv) Strong PnP – The environment can handle completely new devices of previously unknown classes.

2 Spoken Dialogue Systems

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

The user's request is captured by a microphone, which provides the input for the Speech Recognition component. Next, the Language Understanding component receives the recognized words and builds the related speech acts. The Dialogue

Manager processes the speech acts and then calls the Response Generation component to generate a message. Finally, the message is used by the Speech Output component to produce speech. The response of the SDS is final or is a request for clarification. When everything is acceptable, a final answer is produced based on an external data source, traditionally a relational database.

Fig. 1 shows the logical flow associated to the management of a user's request.

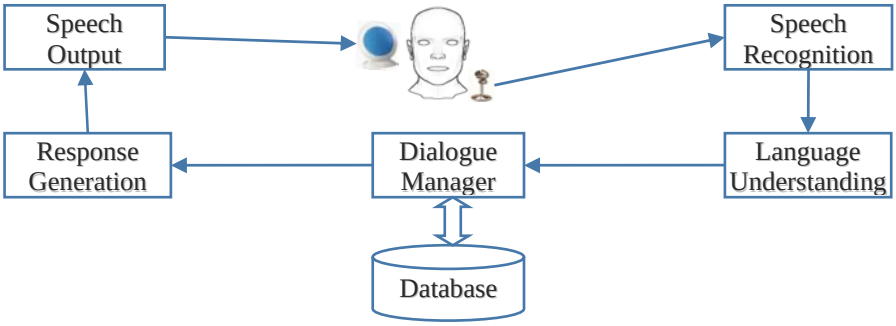

Fig. 1. Logical flow through SDS components.

The origins of SDSs can be traced back to Artificial Intelligence (AI) research in the 1950s concerned with developing conversational interfaces. However, it is only within the last decade or so, with major advances in speech technology, that large-scale working systems have been developed and, in some cases, introduced into commercial environments. The integration of components into a working system is still a key issue [4]. The use of SDS within AmI can only worsen the former problems transforming them into a major challenge.

Since the first SDSs, developed under American and European research projects, issues on portability, extensibility, scalability and reusability still remaining as active research issues. Typically, these issues are addressed through architectures that allow the integration of reusable components. Nevertheless, there is still significant work required to build a new SDS. In practice, most of the work is due to large difficulties in the integration and reutilization of resources or components even when porting from a similar application domain.

Many implementations of dialogue managers perform input interpretation, output generation, and domain specific tasks. These tasks are usually domain dependent because their 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 [5]. Some progresses can be seen in [6], [7], [8], [9], [10], [11].

3 Spoken Dialogue Systems at Home

In this paper, we consider a SDS as a computational entity that allows universal access to AmI. Under the major topic that is AmI, we are mostly interested in home

environments as a particular example of other spaces such as the office, the car or public spaces. Devices throughout the house can be in constant contact with each other, making the AmI home responsive to its inhabitant's needs. These devices must be easily installed and personalized according to the user's wishes. The AmI home is also energy-conscious, able to intelligently manage the use of heat, light and other resources depending on the occupant's requirements.

The exponential drop in microprocessor cost over time has enabled appliance manufacturers to pack increasingly complex feature sets into appliances such as video recorders, refrigerators, washing machines, air conditioners, 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. SDSs provide 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.

According to [12] it is demonstrated that interactions with computers and new technologies are similar to real social relationships and to the navigation of real physical spaces. In this context, it is reasonable to disclose, for instance, that people will talk naturally with a microwave oven.

At the moment, it is unrealistic to consider the existence of an autonomous SDS embedded into each device, due to hardware limitations. While the coordination and collaboration between a set of autonomous SDS is, per se, another challenge.

The use of a SDS admits non-technical users, i.e., with no a priori knowledge of the environment. During the interaction, the SDS has to define the devices that have to be operated, and must inform the user about the tasks and options he/she has available.

The use of simulated characters is frequent in multimodal SDSs [13], [14]. Nevertheless, we believe that in an increasing number of cases, these characters will be replaced by real devices with intelligent behavior.

Dealing with isolated devices is a first important step, but we consider that the real challenge is to deal with tasks involving the collaboration between several devices. Within a very sophisticated home environment, "*turning on*" the light through a voice command is not an important feature, since the lights will be turned on just by the user's presence. However, it will be very useful to control the room's luminosity, when the order "*more light*" is given, as the SDS (automatically at day) may change the transparency of the window, built with electrochromic materials [15], instead of acting over an artificial source of light. The SDS might also take the initiative: asking if the user wants the lights on when leaving home.

We think that a mixed-initiative dialogue will emerge not only from the isolated devices but also from their collaboration. For instance, a sensor in a window sends an alarm "*open window*", simultaneously to the security system, to the sound system and to the control environment system (to turn off the room air conditioner). In this context, the SDS must be aware of any alarm, measure (i.e. temperature, wind) or command that may determine the behavior of any device (or peripheral system) in order to suggest actions (adding not predicted content to the discourse) or just to provide answers when asked.

4 Hybrid Approach

We use a hybrid approach, which was introduced by Filipe and Mamede in [16], for modeling the Domain Knowledge (DK), combining a top-down approach applied to acquire Global Knowledge (GK) and a bottom-up approach applied to acquire Local Knowledge (LK). Both approaches, applied at design time, converge to reach middle-level concepts directly associated to the device interface.
Fig. 2 shows a schema of hybrid approach, where X_n represents a generic device. This schema represents the DK that includes the GK and LK under a device-centered perspective.

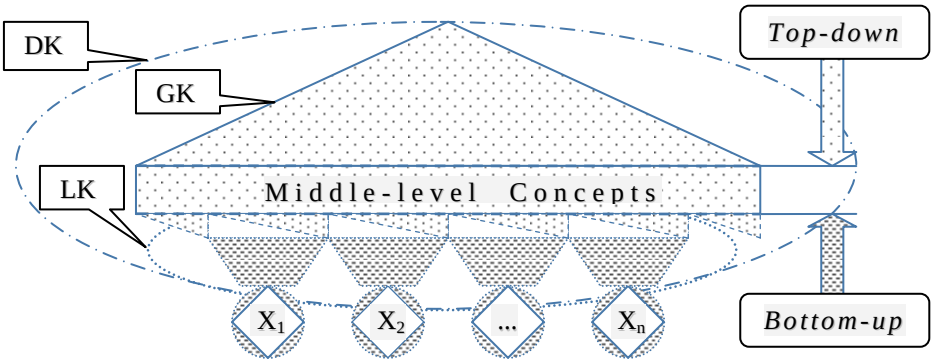

Fig. 2. Schema of the hybrid approach.

The SDS deals with two kinds of knowledge (GK and LK), that are viewed at runtime as a single centralized knowledge model, which is managed by a broker.

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 a device that was not previously known?*".

In this scenario, the SDS does not know which capabilities will be found on a device. In order to address this problem, the capabilities of the device must be described at knowledge or conceptual level [17]. Each device, provided with its own knowledge model, will assist the dialogue manager to understand its functional and structural features.

4.1 Local Knowledge Modeling (bottom-up)

The LK (local knowledge) is defined considering all the available devices (belonging to the surrounded environment) and is modeled using a bottom-up approach applied to adapt the native device interface.

The integration process of a device in a SDS is prepared building a set of three layers, which would potentially cover all the relevant device features:

(i) Communication layer: provides an elementary abstraction of the device expressing its primitive capabilities. For instance, if the device is a door we

must be able, to open or to close the door and to ask about its state (opened/closed);

(ii) Adaptation layer: transforms the first layer into a more convenient interface, considering the device class. For instance, the adapter might transform labels into infrared command codes;

(iii) Operation layer: includes particular features of the device, bearing in mind, for instance, variations of the device commercial model. This layer is personalized to cover the SDS needs. For each capability of this layer, we must define a correspondent descriptor in device knowledge model.

Fig. 3 shows the layers involved in device adaptation. This adaptation allows the SDS to manipulate the device. However, this is not for exclusive use of the SDS because it can be also freely use by other concurrent systems.

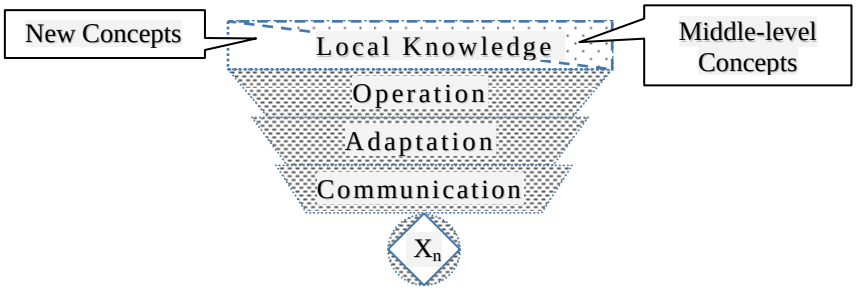

Fig. 3. Device adaptation layers.

According to the presented knowledge model, the knowledge about a device includes concept declarations, class declarations, and task declarations.

A local concept declaration can be about a completely new concept or is a forward declaration, in other words, is a reference to a concept globally (and previously) declared, that is a middle-level concept (see Fig. 2). Therefore, a forward concept declaration is realized indicating only a concept identifier.

When we are filling a task descriptor, using concept identifiers, if we do not know the identifier of a concept (because it is not previously declared as global) it must be declared as local in order to obtain the needed identifier.

4.2 Global Knowledge Modeling (top-down)

The GK (global knowledge) is modeled using a top-down approach. Knowledge modeling is a creative process. We can consider many ways to model the knowledge of a device or either of a domain. The best solution usually depends on the application that we have in mind and the extensions that we anticipate.

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.

Fig. 4 shows an example of a type hierarchy.

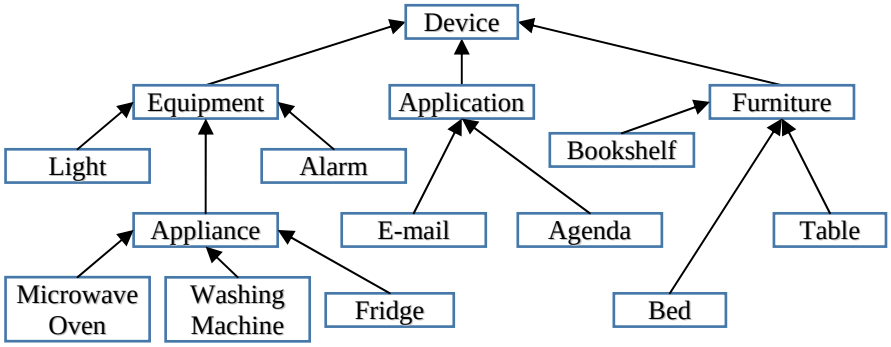

Fig. 4. Example of a type hierarchy.

Following a top-down approach, we start modeling 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 devices that we should consider. Then, we have to identify which are the collections of concepts (middle-level concepts) associated with the selected device 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?*".

5 Knowledge Model

In order to support the knowledge representation we propose a *Knowledge Model* composed by three independent components: a *Discourse Model*, a *Task Model*, and a *World Model*. These components encapsulate the descriptors of the entities that can be mentioned by the user.

A general class diagram of the Knowledge Model is presented in Fig. 5.

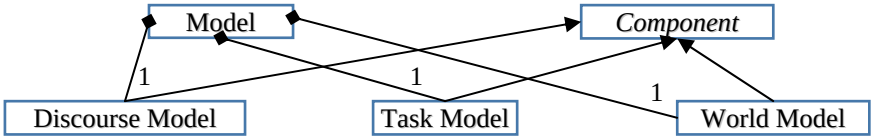

Fig. 5. Knowledge model main components.

The design of this model was adapted from Unified Problem-solving Method Development Language (UPML) [18].

Fig. 6 presents an illustration of the knowledge that can be represented in the proposed knowledge model.

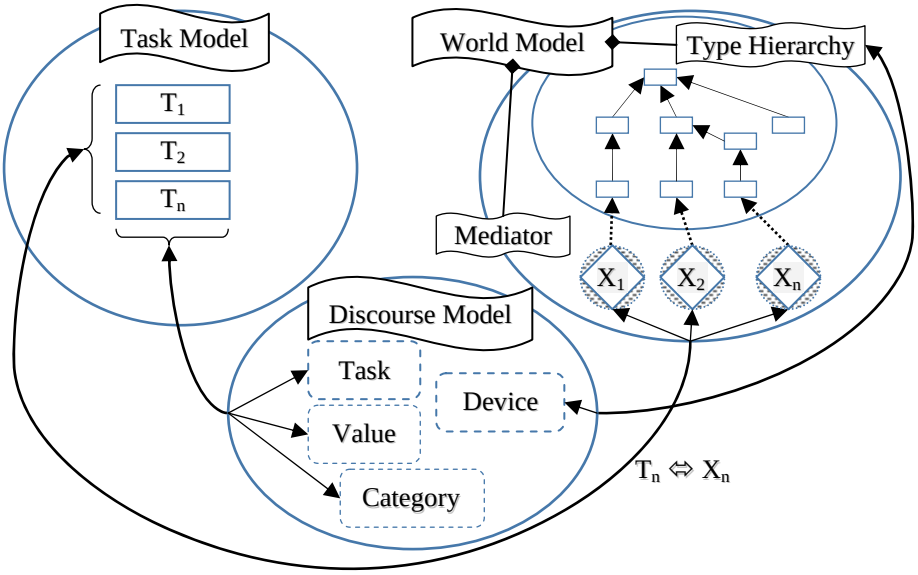

Fig. 6. Knowledge representation illustration.

5.1 Discourse Model

The *Discourse Model* defines a conceptual support, grouping a set of concept definitions used to describe devices and the tasks they provide. A *Concept* is an atomic knowledge unit that has a unique identifier (*ID*). Concepts are not included for complex terms (word root or stem) unless necessary.

a) Concept Declaration

In order to guarantee the existence of the vocabulary needed to designate a concept the concept declarations include linguistic resources. This approach tries to reach the ubiquitous essence of natural language. Although, the coverage of handmade resources such as WordNet [19] in general is impressive, coverage problems remain for applications involving specific domains or multiple languages.

The vocabulary is organized by language, allowing a multi-lingual definition of concepts. Each concept declaration has its own linguistic resources included into a *Linguistic Descriptor*. For example, an item, such as "*kitchen light*", should be modeled as an instance of a "*light*", having the location "*kitchen*" without creating a new concept "*kitchen light*". A linguistic descriptor holds a list of terms, or more generically a list of *Multi-Word Unit* (MWU), referring linguistic variations associated with the concept, such as synonymous or acronyms. Each *Word* (or term), has a part of speech tag, such as a noun, a verb, an adjective or an adverb; a language tag, such as "pt", "br", "uk" or "us"; and some phonetic transcriptions. For instance, if the language tag of a word is "pt" its phonetic transcription is encoded using the

Speech Assessment Methods Phonetic Alphabet (SAMPA) for European Portuguese [20].

Fig. 7 shows the relations between the classes involved in a concept declaration.

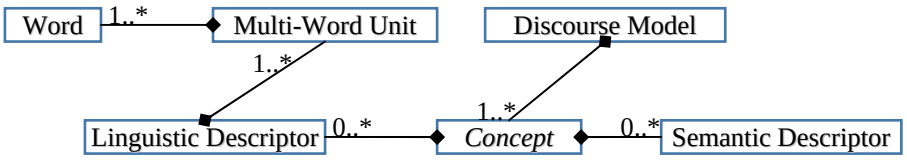

Fig. 7. Concept declaration.

Optionally, each concept can also have semantic resources references represented by a *Semantic Descriptor*. A *Semantic Descriptor* has references to other external knowledge sources, for instance, an ontology [21] or a lexical database, such as WordNet. The knowledge sources references in the knowledge model must be unique.

The references to knowledge sources must be encoded using a data format allowing a unique identification of the concept in the knowledge source. The syntax of the knowledge source references do not need to be universal it is enough to keep the same syntax for a particular knowledge source. We recommend the use of a generic Uniform Resource Identifier (URI) format to encode the knowledge sources references. In particular, could be used a Uniform Resource Locator (URL) or a Uniform Resource Name (URN). For instance, the declaration of the concept "*device*" could have the reference "*URN:WordNet21:device:noun:1*" meaning: the concept "*device*" is linked to the first sense of the noun "*device*" in WordNet 2.1, where it is described by "*an instrumentality invented for a particular purpose*" (in WordNet 2.1 this noun has five senses).

The *Discourse Model* organizes the concepts by classes and subclasses as shown in Fig. 8. The most relevant classes are *Device* and *Task*. The other classes are used to represent task roles.

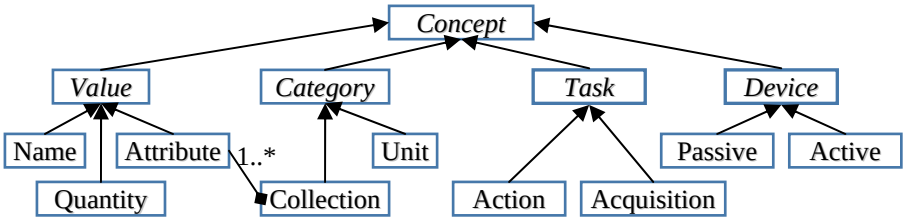

Fig. 8. Classes of concepts.

b) Device and Task Representation

A concept, representing a *Task* name, can be an instance of two subclasses: *Action*, when the task can modify the device state; or *Acquisition*, when it never modifies the device state. A device name is a concept of the class *Name*. A device class can be an instance of two subclasses of *Device*: *Active*, when providing at least one action task; or *Passive*, when providing only acquisition tasks. For instance, an oven is an active

device and a thermometer is a passive device. These device and task classifications are important to prevent task execution conflicts.

c) Representation of Task Roles

The representation of task roles involves two different aspects: role default *Value* and role *Category* used to establish the role range. The default value can be an instance of two subclasses: *Quantity*, to represent default numbers, for instance, zero; or *Attribute*, to represent default attributes, for instance, the black color. The role category can be an instance of two subclasses: *Collection*, when a role has a limited set of attributes, for instance, the rainbow colors; or *Unit*, when a role represents a physical dimension, for instance, the distance in meters, miles or inches.

5.2 Task Model

The *Task Model* represents the set of available tasks provided by existing devices. Typically, before performing a task, it is necessary to express the argument values or parameters that establish the execution conditions. It is mandatory for these values to be represented in the *Discourse Model*. The *Task Model* is also used to check the state of the world before and after a task execution.

a) State of the World

The state of the world is represented by the set of individual states of each device. The state of each device is obtained by calling the adequate acquisition tasks. For instance, when the task is "*switch on the light*", we have to check if the "*light*" is not already "*switched on*" and after the task execution, we have to check if the "*light*" has really been "*switched on*".

b) Task Descriptor

A *Task Descriptor* is used to represent a task in the *Task Model* and is composed by a unique identifier (*ID*), a name (*Acquisition*, *Action*), two optional lists (*In List*, *Out List*) of roles that describe in and out task parameters and two optional rules (*Final*, *Initial*) applicable to the state of the world.

Fig. 9 shows the class diagram of the *Task Descriptor*.

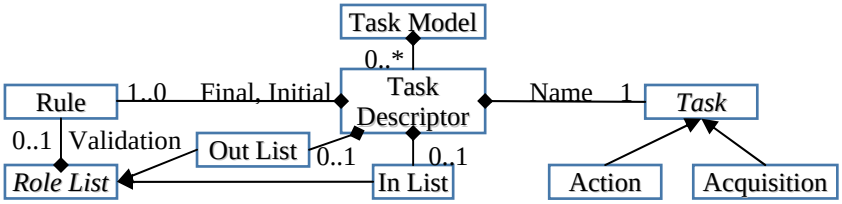

Fig. 9. Task representation.

The *Initial* rule is checked before the task call and must produce the logical value true before the task execution. The *Final* rule is checked after the task execution and

must produce the logical value true when a successful task execution occurs. A rule is expressed using relational operators ('<', '>', '=', '<>', '<=', '>=') and logical operators ('Or', 'And'). When we need to specify a task argument value in a rule expression, the ID, of the concept that is the role *Name*, must be between square parenthesis ('[',']'). When we need to specify a concept, its ID must be between braces ('{','}'). For instance, we can write the rule "{1} > [2]" to denote that. Each task argument is represented in its *Task Descriptor* by a *Role* that is associated with other classes (see Fig. 10).

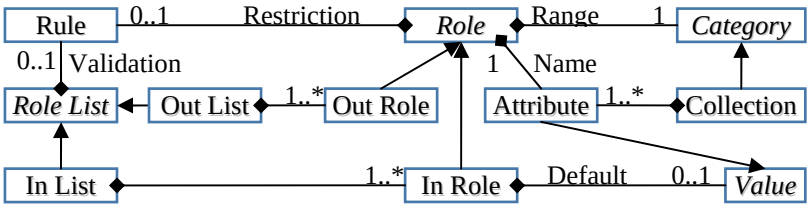

Fig. 10. Task role representation.

A role describes the possible values that can be used to instantiate a task argument. An *In Role* describes the values that can be used as input parameters. An *Out Role* describes the values that can be used as output parameters. All roles have a name (*Attribute*) and a range (*Category*). Each role has an optional *Restriction* rule to check the parameters, for instance, the parameter must be positive. An *In Role* may have an optional *Default Value* (*Quantity*, *Attribute*).

When the *Default Value* is not present, the task parameter is mandatory. Task roles are organized in two role lists (*In Role* and *Out Role*) that have a *Validation* rule to check its parameters. The validation rule of an *In Role* list is evaluated before the task execution. The validation rule of an *Out Role* list is evaluated after the task execution.

5.3 World Model

The *World Model* represents the devices that are part of the world and integrates two components: a *Type Hierarchy* and a *Mediator*.

Figure 11 presents a class diagram of the World Model components.

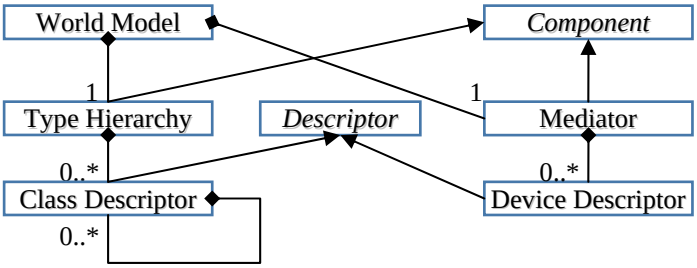

Fig. 11. World model components and descriptors.

5.4 Coupling the Descriptors

The knowledge representation is essentially based on descriptors (*Task Descriptor*, *Device Descriptor*, *Class Descriptor*) that are coupled using bridges. After the definition of the needed concepts belonging to the *Discourse Model*, we can fill the descriptors without obligation to follow a predefined sequence. Then, we can introduce the instances of the *Bridge* class that associate task descriptors to device descriptors (*Bridge T*) and device descriptors to class descriptors (Bridge C).

Figure 12 shows the knowledge model descriptors and bridges involved in the knowledge representation.

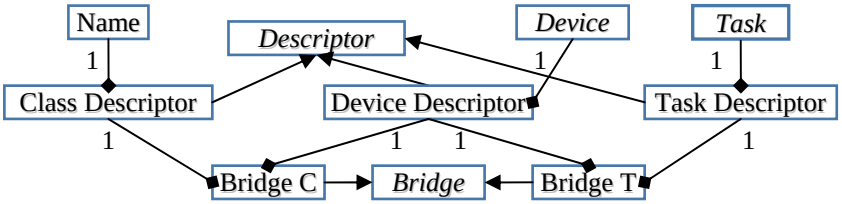

Fig. 12. Model descriptors and bridges.

A device knowledge model has only one device descriptor that describes the device itself and must have at least one task descriptor. The broker's knowledge model does not include any device descriptors, because the device descriptors are added only at runtime.

6 Knowledge-based Broker

The broker is a proposed SDS component that assumes two responsibilities: performs the domain knowledge management and maintains the predefined global knowledge. This component allows the customization of the principal SDS component, the dialogue manager, which should only be concerned with phenomena related to the user's dialogue. Fig. 13 shows the SDS reference architecture with a broker, where X_n is a generic device and A, B, C, and D are other components of the SDS.

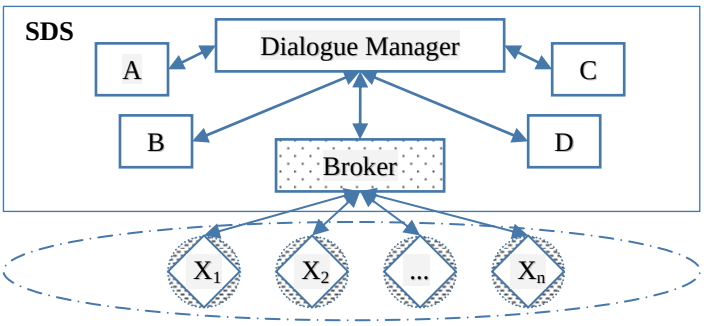

Fig. 13. Adaptation of the SDS to AmI.

The main goal of the broker is to support the communication interoperability between the SDS and the devices in the pervasive application domain. The use of a broker 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 devices.

6.1 Broker's Knowledge Integration Process

The goal of the Knowledge Integration Process (KIP) is to update automatically the DK (Domain Knowledge), integrating the GK (Global Knowledge) included in the broker and the LK (Local Knowledge) included in the domain's devices.
Two other processes compose the KIP: the device attachment process and the device detachment process.

a) Similar Concepts

We consider that two concepts are similar when: its identifiers are equal, one of its semantic descriptors is equal or its linguistic descriptors are equal. In special cases, two concepts can be considered as similar by other convenient similarity criteria. In this context, two similar concepts cannot exist in the same *Discourse Model*.

At its starting point, KIP puts side by side the concept definitions in DK and the concept definitions in LK, which are going to be merged. KIP uses a Conversion Concept Table (CCT), linked to each *Device Descriptor*, to switch identifiers of similar concepts.

b) Device Attachment Process

When a device is attached (activated), it searches for the broker component of the SDS. After establishing the initial communication, the broker leads the device attachment process following the next nine steps, in order to update its *Knowledge Model*:

(i) A new *Device Descriptor* is added to the broker's *Mediator*;
(ii) An empty CCT is linked to the new *Device Descriptor*;
(iii) The concepts in the device's *Discourse Model* fill the first column of the CCT;
(iv) Each concept in the first column of the CCT, with a similar concept in the broker's *Discourse Model*, is associated with its similar, filling the second column of the CCT;
(v) Other concepts in the first column of the CCT (without a similar concept in the broker's *Discourse Model*) are added to broker's *Discourse Model*;
(vi) Each new device *Task Descriptor* is added to the broker's *Task Model* and its concepts identifiers are replaced by the existing similar concepts identifiers, using the CCT;

(vii) Each *Class Descriptor* in the device's *Type Hierarchy* is integrated in the broker's *Type Hierarchy* and its concepts identifiers are replaced by the existing similar concepts identifiers, using the CCT;
(viii) The new *Device Descriptor* is associated with its *Class Descriptor* using the appropriate bridge (Bridge C);
(ix) The new *Device Descriptor* is associated with its *Task Descriptors* using the appropriate bridge (*Bridge T*).

c) Device Detachment Process

When the broker detects that a device has been detached (deactivated), it follows the next five steps, in order to update its Knowledge Model:

(i) *Task Descriptors* exclusively associated with the current (detached) *Device Descriptor* are removed from the broker's *Task Model*;
(ii) *Class Descriptors* exclusively associated (in a *Bridge* or in a CCT) with the current *Device Descriptor* are removed from the broker's *Type Hierarchy*;
(iii) Concepts that appear only in the CCT are removed from the broker's *Discourse Model*;
(iv) *Bridges* associated to current *Device Descriptor* are removed from the broker's Knowledge Model;
(v) Current *Device Descriptor* is removed from broker's *Mediator*.

6.2 Broker's Recognizer

This section describes the broker's recognizer service, proposed to recognize the domain's concepts from a natural language request.

The recognizer service receives a request and split its words into groups, trying to obtain a match against the linguistic descriptors in the broker's *Domain Model*. The words in the request are grouped from the left to the right. Each group of words is processed in several interactions.

First iteration uses a group of W words. Second iteration uses a candidate group of W-1 words, and so on. The maximum length of a group of words, represented in the *Domain Model*, determines the value of W.

When a group of words matches a MWU (multi-word unit) in a concept *Linguistic Descriptor*, the concept is recognized and these words are removed form the original request. When the candidate group as only one word this word is removed form the request. The recognition process ends when the request is empty.

An improved version of the recognizer service can also accept annotated natural language requests, including part of speech tags and specific phonetic transcriptions of some words, in order to solve potential linguistic ambiguities.

The list of recognized concepts can be directly used by the SDS *Dialogue Manager* to fill the list of pivot concepts indicated as input to the broker's advisor service. However, the *Dialogue Manager* can remove or add other concepts into the pivot concept list, according to its own dialogue strategies.

6.3 Broker's Advisor

This section describes the broker's advisor service, proposed to suggest the best task-device pairs to satisfy a request formalized in a list of domain's concepts. The ideas behind this service are based on the relative weight of each concept that figures in the request.

Fig. 14 illustrates the processing of the request "*defrost codfish in microwave oven*" that combines the use of the recognizer service with the use of the advisor service.

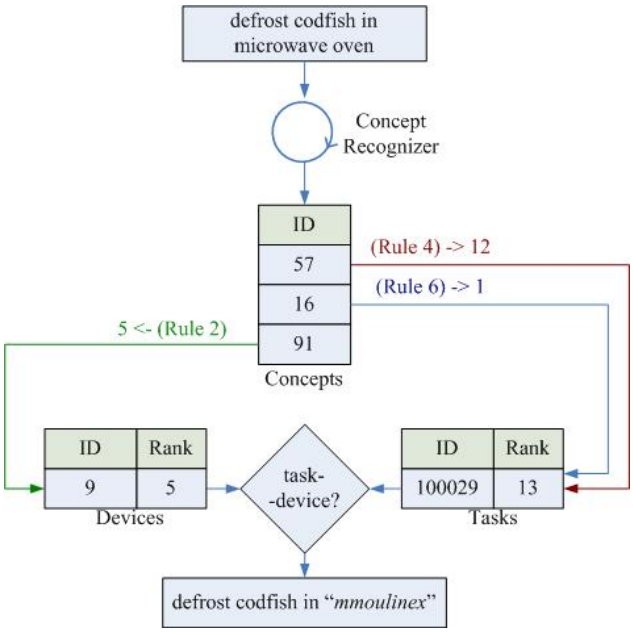

Fig. 14. Illustration about the recognizer service.

In Fig. 14 example, the concept recognizer service identifies a list with three concepts: ID=57: "*defrost*", ID=16: "*codfish*", and ID=91: "*microwave oven*". In order to determine the best task-device pair, the advisor service applies its rules to each one of the pivot concepts. Rule 4 applied to concept, with the ID=57, adds 12 points to the rank of the task, with the ID=00029. Rule 6 applied to concept, with the ID=6, adds 1 point in the rank of the same task. Finally, Rule 2 applied to concept, with the ID=91, adds 5 points in the rank of the device, with the ID=9. The reference to the device class "*microwave oven*" becomes a reference to the device with the name "*mmoulinex*", because we have only this microwave oven represented in the *Domain Model*.

Two independent ranking, for tasks and devices, support the suggestion for the best task-device pairs. The ranking points are determined considering three heuristic values: *nABase*, *nTBase*, and *nTUnit*. The *nABase* value is determined by the maximum height of the domain model type hierarchy plus (1) one. The *nTBase* value is determined by the maximum number of task roles (arguments) plus (1) one. The

nTUnit value is constant and equal to 3 (three) that are the number of ways to reference a task role (by *name*, *range*, or *value*).

The advisor service uses as input a list of pivot concepts. The pivot concepts references, about tasks and devices, are converted following the next six rules into points that are credited to the respective device or task rank.

The rank of a device is modified according to the rules:

(i) If the pivot concept refers a device name, the value *nABase*2* is credited in the respective device rank;

(ii) If the pivot concept refers a device class name, the value *nABase* is credited in the respective device rank;

(iii) If the pivot concept refers a device super-class name, the value *nABase-n* is credited in the respective device rank, where *n* is determined by the number of classes (in the *Type Hierarchy*), between the device class and the referred super-class.

The rank of a task is modified according to the rules:

(iv) If the pivot concept refers a task name, the value *nTBase*nTUnit* is credited in the respective task rank;

(v) If the pivot concept refers a task role name or a task role range, the value *nTUnit/2* is credited in the respective task rank;

(vi) If the pivot concept refers a task parameter, the value *nTUnit/3* is credited in the respective task rank.

Finally, the task-device pairs are composed selecting the tasks with the best rank and the devices, which provide the tasks, also with the best rank.

7 Experimental Evaluation

Our work is based on an environment simulator in which we are testing the proposed approach considering a set of common home devices, described in Table 1, that are normally present in the kitchen.

Table 1. Devices in the environment simulator.

Device Name	# (Concept) (Task) (Argument)	# (Adjective) (Noun) (Verb) => (Term)
Air Conditioner	(63) (24) (25)	(8) (44) (8) => (63)
Freezer	(96) (13) (20)	(19) (79) (3) => (106)
Fryer	(92) (23) (36)	(18) (64) (10) => (96)
Light Source	(62) (20) (21)	(9) (39) (10) => (61)
Microwave Oven	(167) (26) (44)	(17) (117) (18) => (159)
Kitchen Table	(48) (13) (17)	(10) (28) (4) => (44)
Water Faucet	(63) (24) (25)	(8) (44) (8) => (63)
Window	(44) (13) (17)	(8) (26) (4) => (41)
Window Blind	(65) (22) (23)	(10) (39) (10) => (62)
Total	**(700) (178) (228)**	**(107) (480) (75) => (695)**

The 9 (nine) devices are using a total of 700 concepts. Initially the GK (that is equal to DK at design time) is using 261 concepts.

After the attachment of all devices, DK retains 360 concepts (at runtime).

The knowledge integration rate is 360/700*100 = 51%.

Each one of the knowledge models for devices and broker is supported by a relational database with 19 (nineteen) tables.

Fig. 15 shows a screenshot of the environment simulator, developed originally for Portuguese users. On the bottom of the screen, we can see the electrochromatic Table device simulator.

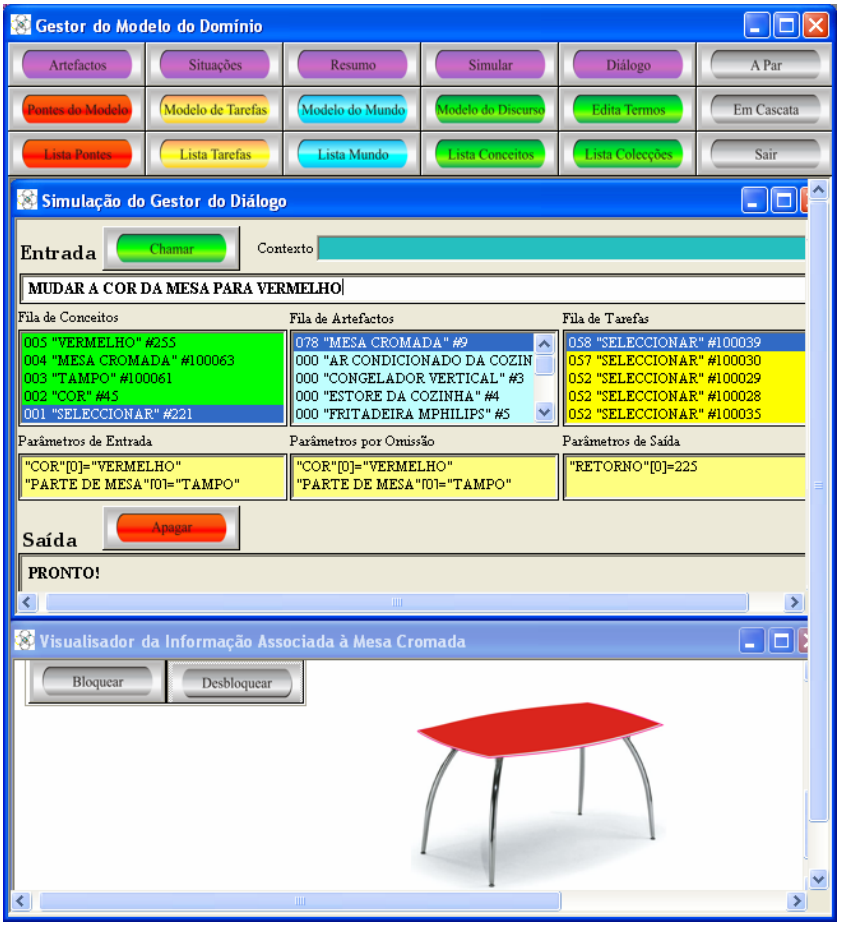

Fig. 15. Screenshot of the environment simulator.

The environment simulator allows the debugging of KIP and the simulation of the interaction made by the dialogue manager. We can attach and detach devices, do requests of tasks, obtain the answers and observe the devices behavior. We can also consult and print data about the several knowledge models and about the task execution progress.

Figure 16 shows the screen of the microwave oven simulator after the execution of the request: "*defrosting carrots*". This picture shows the automatically select power (300 watts – see symbol) and duration (8 minutes) of the defrosting process.

Fig. 16. Screenshot of the microwave oven simulator.

Fig. 17 shows the screen of the freezer simulator after the execution of the request: "*asking the amount of carrots*". Table in the picture shows the selected type of food. However, the domain simulator also returns the answer "*1 package with 300 g*" in a text window.

We can also execute requests that evolve relational operators, for instance: "*asking the type of food with amount less than five*".

Fig. 17. Screenshot of the freezer simulator.

The "*carrot*" concept is shared by the microwave oven and by the freezer. We have only one declaration of the "*carrot*" concept in the broker's knowledge model.

8 Discussion

The growth in pervasive computing will require standards in device communication interoperability. These devices must be able to interact and share resources with other existing devices and any future devices across the network.

Our proposal tries do improved the flexibility of the SDSs architectures allowing the independent design of devices. Other proposals are also using a centralized domain knowledge model complemented with linguistic parts [22].

However, our knowledge model, supported by the broker, can be unfilled at design time. We are proposing KIP (knowledge integration process) to update, at runtime, this knowledge model dealing from completely new devices of previously unknown classes and trying to achieve "*Strong PnP*". In addition, we propose the recognizer and advisor services to provide a broker's high-level and easy to use small interface, instead of a conventional interface with several remote procedures/methods.

Current technologies require human interventions to solve environment reconfiguration problems. We believe that these technologies must be improved with more human like ways of interaction that should include spoken natural language support.

9 Concluding Remarks and Future Work

This paper describes our research in enhance everyday devices to be dynamically managed by a SDS. Essentially, we are dealing with a reconfiguration problem that can be seen also as a portability issue [23].

We have described an approach to deal with communication interoperability between a SDS and a set of heterogeneous devices within AmI vision. We have presented a knowledge-based broker component to adapt the SDS to the environment, which provides the recognizer and advisor services to simplify and support the dialogue manager needs.

We have also present a broker's knowledge integration process, trying to reach the ubiquitous essence of natural language, because the coverage of handmade lexical resources is limited, coverage problems remain for applications involving specific domains or involving multiple languages.

The hybrid knowledge modeling approach, supported in the field by a knowledge-based broker, 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 devices that represents a home environment.

As future work, we expect to explore, more deeply, the knowledge integration perspective working with simulated and real devices. In order to include real devices we expect to extend the *Device Descriptor* to support specific data about the different kinds of device access network protocols.

We believe that in the near future, 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.

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

References

1. K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J-C. Burgelman: Scenarios for Ambient Intelligence in 2010. ISTAG 2001. IPTSSeville (Institute for Prospective Technological Studies) (2001)
2. Weiser, M.: The Computer for the Twenty-First Century. Scientific American (1991)
3. Henricksen, K., Indulska, J., Rakotonirainy, A.: Infrastructure for Pervasive Computing: Challenges. Workshop on Pervasive Computing Informatik 01, Viena (2001)
4. McTear, M.: Spoken Dialogue Technology: Enabling the Conversational Interface. ACM Computing Surveys, Vol. 34 (2002)
5. O'Neill, I. and McTear, M.: Object-Oriented Modelling of Spoken Language Dialogue Systems. Natural Language Engineering 6, Cambridge University Press, Cambridge (2000)
6. Bohus, D. and Rudnicky, A.: RavenClaw: Dialog Management Using Hierarchical Task Decomposition and an Expectation Agenda. Eurospeech 2003, Geneva, Switzerland (2003)
7. O'Neill, I., Hanna, P., Liu, X. and McTear, M.: An Object-Oriented Dialogue Manager. Eurospeech 2003, Geneva, Switzerland (2003)
8. Pakucs, B.: Towards Dynamic Multi-Domain Dialogue Processing. Eurospeech 2003, Geneva, Switzerland (2003)
9. Polifroni, J. and Chung, G.: Promoting Portability in Dialogue Management. ICSLP 2002, Denver, Colorado, USA (2003)
10. Neto, J., Mamede, N., Cassaca, R., Oliveira, L.: The Development of a Multi-purpose Spoken Dialogue System. Eurospeech 2003, Geneva, Switzerland (2003)
11. Turunen, M., Hakulinen, J.: JASPIS^2 – An Architecture for Supporting Distributed Spoken Dialogues. Eurospeech 2003, Geneva, Switzerland (2003)
12. Reeves, B., Nass, C.: The Media Equation: How People Treat Computers, Television and New Media Like Real People and Places. Cambridge, Mass: CUPress (1996)
13. Gustafson, J., Lindberg, N., Lundeberg, M.: The August Spoken Dialogue System. Eurospeech 1999 (1999)
14. Gustafson, J., Bell, L., Beskow, J., Boye, J., Carlson, R., Edlund, J., Granström, B., House, D., Wirén, M.: AdApt - A Multimodal Conversational Dialogue System. ICSLP 2000, (2) 134-137, Beijing, China (2000)
15. Wigginton, M.: Glass in Architecture. Phaidon Press Ltd, London (1996)
16. Filipe, P., Mamede, N.: Towards Ubiquitous Task Management. Interspeech 2004, Jeju Island, Korea (2004)
17. Newell, A.: The knowledge level. Artificial Intelligence, 18(1) (1982)
18. Fensel, D.; Benjamins, V.: Motta, E.; Wielinga, B.: UPML: A Framework for Knowledge System Reuse, IJCAI 1999 (1999)
19. Fellbaum, C., (editor): WordNet: An Electronic Lexical Database. MIT Press (1998)
20. SAMPA (Speech Assessment Methods Phonetic Alphabet), Spoken Language Systems Lab (L2F), http://www.l2f.inesc-id.pt/resources/sampa/sampa.html
21. Gruber, T.: Toward Principles for the Design of Ontologies Used for Knowledge Sharing. International Workshop on Formal Ontology, Padova, Italy (1992)
22. Montoro, G., Alamán, X., Haya, P.: A Plug and Play Spoken Dialogue Interface for Smart Environments. CICLing 2004: 360-370 (2004)
23. Zue, V., Glass J.: Conversational Interfaces: Advances and Challenges. IEEE (2000)