From Meta-modeling to Automatic Generation of Multimodal Interfaces for Ambient Computing

José Rouillard
LIFL Laboratory – University of Lille 1
59655 Villeneuve d'Ascq Cedex - France
jose.rouillard@univ-lille1.fr

Jean-Claude Tarby
LIFL Laboratory – University of Lille 1
59655 Villeneuve d'Ascq Cedex – France
jean-claude.tarby@univ-lille1.fr

Xavier Le Pallec
LIFL Laboratory – University of Lille 1
59655 Villeneuve d'Ascq Cedex - France
xavier.le-pallec@univ-lille1.fr

Raphaël Marvie
LIFL Laboratory – University of Lille 1
59655 Villeneuve d'Ascq Cedex - France
raphael.marvie@univ-lille1.fr

***Abstract* — This paper presents our approach to design multichannel and multimodal applications as part of ambient intelligence. Computers are increasingly present in our environments, whether at work (computers, photocopiers), at home (video player, hi-fi, microwave), in our cars, etc. They are more adaptable and context-sensitive (e.g., the car radio that lowers the volume when the mobile phone rings). Unfortunately, while they should provide smart services by combining their skills, they are not yet designed to communicate together. Our results, mainly based on the use of a software bus and a workflow, show that different devices (such as Wiimote, multi-touch screen, telephone, etc.) can be coordinated in order to activate real things (such as lamp, fan, robot, webcam, etc.). A smart digital home case study illustrates how using our approach to design with ease some parts of the ambient system and to redesign them during runtime.**

Keywords* — *Pervasive computing; ubiquitous computing; ambient intelligence; multi-channel interaction; multimodality.

I. INTRODUCTION

Ambient computing is one of the most significant recent advances in Human-Computer Interaction (HCI). Due to the arising of pervasive and ubiquitous computing, the design of HCI has to take into account the context of interactions. The objective is to allow users to interact with a smart system with low constraints through the use of multiple modalities, channels, and devices. In the future, with the availability of new devices and smart objects, ambient computing will allow the definition of services seamlessly interacting with both environment and users.

Our current work takes place in this context of ambient computing. In order to support dynamic unplanned interactions with the user, services have to adapt themselves to their mutating environment – resulting from the user mobility and the variability of her/his context. This requires (a) the availability of distributed devices such as PDA (Personal Digital Assistant), laptops, smartphones, robots, probes, and (b) easing the discovery of these devices.

Currently, development tools that enable us to easily generate and integrate ambient services are lacking. Each piece of software is developed on its own, and then integrated in the system. This introduces additional costs as well as misconfiguration risks. This paper focuses on the design of multi-channel interfaces relying on a workflow engine in order to ease the realization of ambient systems.

This document is an extended version of our previous paper [1]. It is structured as follows. Section two presents related works. Section three explains the background and motivation of this project. Section four gives an overview of our conceptual approach in order to tackle the emerging problems encountered. Section five explains in details our approach from an implementation point of view. Section six describes a case study around the smart digital home thematic and presents the benefits of our approach for the design and generation of multimodal and multichannel interactive systems. Then, a conclusion gives our roadmap for future work.

II. RELATED WORK

Computer frameworks and languages have been proposed specifically to facilitate the development of multimodal interfaces. In the World Wide Web Consortium (W3C) MultiModal Interaction (MMI) framework [2], the interaction manager invokes specific application functions and accesses information in a dynamic processing module. The interaction manager presents the result to the user via one or more output components. Obviously, the interaction manager of this framework is very important because it coordinates data and manages execution flow among various input and output components. It also responds to inputs from the input components, updates the interaction state and the application context, and initiates output to one or more output components. Developers use several approaches to implement interaction managers, including: Traditional programming languages such as C or C++;

Speech Application Language Tags (SALT), which extends HTML by adding a handful of HTML tags to support speech recognition, speech synthesis, audio file replay, and audio capture; XHTML plus Voice (often referred as "X+V"), in which the VoiceXML 2.0 [3] voice dialog control language is partitioned into modules that are embedded into HTML; Formal specification techniques such as state transition diagrams and Harel Statecharts [4].

The OpenInterface project [5] is dedicated to multimodal interaction. In this project, everyday objects can take part in the interaction in ubiquitous computing (including an augmented table for instance) and the user can freely switch from one modality to another according to her/his context: running in the street, at home, in front of a big screen in an airport, etc. This project aims at the design and development of an open source framework for multimodal interaction: the OpenInterface framework.

Those kinds of projects are mainly devoted to the study of multimodal interactions, allowing the usage of more than one device or modality at the same time in order to interact with a main system connected to Internet. Ambient computing increases complexity because related applications are not supposed to manage only devices and modalities, but also channels (cf. Section III.A) in order to allow intelligent and context-aware communications. Our research activity takes place in ambient computing area.

III. BACKGROUND

This background section is divided in three parts: multimodality, user activity, and connection with the ambient environment.

A. Multimodality

Our work tackles the ability of ambient computing to permit context-aware interactions between humans and machines. To do so, we rely on the use of multimodal and multi-channel interfaces in various fields of application such as coaching [6], learning, health care diagnosis, or in-situ marketing. For Frohlich, a channel is defined as an interface that makes a transformation of energy [7]. From a user's point of view, he distinguished voice and movement channels, and from the system's point of view he mentioned audio, visual, and haptic channels.

In the Human–Computer Interaction (HCI) domain, the notion of channel is not used very often and there are very few references to multi-channel research with some exceptions such as the work of [8]: "Often these modalities require specialized channels to allow access modalities such as cameras, microphones, and sensors. A multi-modal multi-channel system faces the challenge of accepting information from any input method and delivering information through the appropriate output methods".

Using a multi-channel approach allows users to interact with several channels choosing the most appropriate one each time in order to exchange with an entity. Such channels could be, for instance, plain paper, e-mail, phone, web site. Using a multimodal approach allows users to employ several modalities in order to interact with a single system. It can be sequential, like first being on the phone then on the web, or synergistic [9], like being on the phone while on the web. This approach implies some synchronization requirements both for the interfaces and knowledge bases used during the interactions.

There are very few tools that support the design and implementation of interfaces having such characteristics [10]. One of our goals is to study and propose infrastructures easing interactions that are both multimodal and multi-channel in an ambient context. In our work, we use the Multi-DMC referential proposed in [11]. It can identify a system based on three criteria: Device (D), Modal (M) and Channel (C). It has two positions (Mono or Multi) for each of the three criteria targeted (DMC). This represents 2^3 (=8) possibilities, which are presented on Figure 1 .

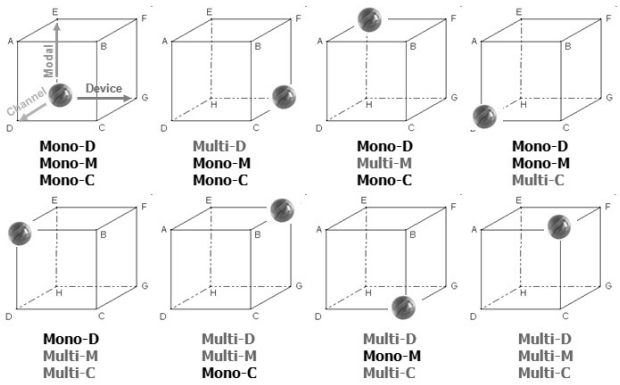

Figure 1. The Multi-DMC referential.

For a given system, one tries to indicate the position of each decisive factor. For example, the system represented on the bottom right of the figure is a multi-device, multimodal, and multi-channel system.

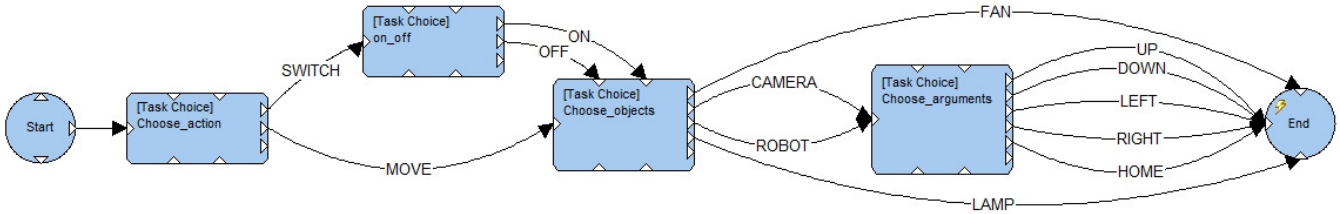

Figure 2. Workflow designed with the Studio Common Knowledge.

B. User activity

In this paper, we are targeting ambient systems, which aim to be user-friendly. Unfortunately, until now such systems are more difficult to conceive and to implement than traditional systems because of the heterogeneity of devices (hardware, software, different locations, etc.). Given its complexity, an ambient system must observe the rules of usability: guidance, low workload, concision, etc. [12]. Therefore, our work is based on concepts identified by HCI domain such as user's activity and logic of use.

The design of interactive systems is based on the notion of tasks and activities, themselves decomposed into sub-tasks/sub-activities whose arrangement is managed by temporal or structural sequences. Among all the approaches used in the design of interactive systems and using these concepts, some are more used such as task models [13][14], Petri nets [15], Statecharts [16], and workflows.

Given all these solutions we have chosen the workflows [17] because they are adapted for non-experts in order to explain their rationale for the use of ambient systems. First, Workflow concepts are as simple as needed to be understood by usual end-users. Second, related modeling languages have been generally designed to be readable by non-(computer)specialists. Finally, they are widespread in information systems and especially in document management systems.

C. Connection with the ambient environment

A major question in pervasive and ubiquitous computing is how to integrate physical objects (screen, chair, coffee machine, etc.) into multimodal applications using technologies such as Radio-frequency identification (RFID), Near field communication (NFC), Barcodes (1D or 2D as QR codes). This will help the users to manipulate freely virtual and real objects with commands like "identify this," "make a copy of that object, here", "move that webcam on the left," etc. We are using the notion of workflow in order to indicate to the user the tasks available at each point of the whole activity flow.

For our work, we are using Common Knowledge [17], which is a cross-platform business rules engine and management system that supports the capture, representation, documentation, maintenance, testing, and deployment of an organization's business rules and application logic. Common Knowledge allows the business logic to be represented in a variety of inter-operable visual formats, including Rete rules, workflows, flowcharts, decision tables, decision trees, decision grids, state maps, and scripts. The engine allows running, testing, and simulating the system behaviors. It can be used through many languages (such as Java, Delphi, VisualBasic, C#, DotNET, etc.) and platforms (Windows, Linux, UNIX).

Figure 2 presents an example of workflow designed graphically using the Studio Common Knowledge tool. It allows following different paths in order to complete a command such as "switch on fan", "move camera down", "switch off lamp", etc.

Figure 3 shows standard and advanced operators used to represent tasks, task choices, split or merge actions, timers, loops, etc. The result is stored using an XML format, in a file with an .aex extension. With our work the resulting system could be used through different modalities of interaction like graphically, vocally, with gesture, RFID, barcodes or a combination of those modalities. Instead of programming applications in an ad hoc fashion, our approach allows to query dynamically the workflow and to propose relevant information to the user while interacting with the system.

The notion of persistence is very important in this context. Indeed, we consider that a global interaction could be the result of many sub-interactions between the system and one or many users. It could also be the result of a sequence of sub-interactions conducted via different kind of channels and modalities.

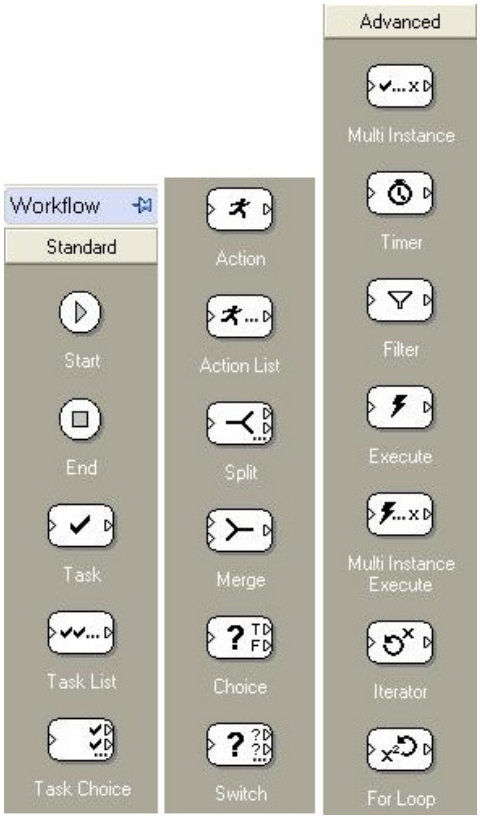

Figure 3. Standard and advanced operators available in Common Knowledge.

The Common Knowledge software supports this persistence feature.

IV. OUR APPROACH

In the context of interaction design based on the DMC referential, we believe, as we explained previously, that meaningful global actions on the system may be the result of a series of sub-actions. These sub-actions can be performed by multiple users cooperating. Several types of devices can be utilized (PC, Smartphone, mobile phone, etc.). Several modalities of interaction, such as direct manipulation (keyboard/mouse), voice, gesture, brain waves, can be employed both in input and output. Finally, multiple communication channels can be exploited such as the telephone or the Internet.

Currently we limit the use to an alternate multimodality (not synergistic). The triggering of a sub-action is based on the FIFO (First In, First Out) principle.

Figure 4 shows our approach based on a software bus. We used for instance the IVY bus [18] and the Web Server Event (WSE) bus (see Section V.B.1) in our experiments, as we will explain later. The "model driven" part mentioned on the figure is used for modeling the activities at a higher level, and mapping resulting models to workflow models, for example. The "engine" part uses an application that queries the generated workflow during the interaction via an Application Programming Interface (API). The "usage" part explains that different kinds of interaction are possible (web client, graphical user interface, vocal user interface, etc.). The "development" part means that the architecture is open in terms of futures applications, technologies and languages. In our approach, the transition from one state to another can be modelized with different tools, such as Petri nets or the usage of workflows for instance. The model driven approach allows working on an abstract level, independently from the chosen technical solution (Petri nets or workflow in our example).

A. Model driven approach

Figure 5 shows that a workflow (middle of the picture) is generated from a high-level model (left of the picture) thanks to a set of model transformation rules. This workflow model is used in order to describe objects and actions that can be applied on those objects using one or more devices in final interfaces (right of the picture).

Our work mainly concerns description of operating and use of multimodal interactions (MMI). The Activity concept is the main notion of our approach. We have experimented a workflow management system (see Section VI) as a support to define the operating logic of MMI and its corresponding execution. However, defining interaction logic may be done at different steps of an application design and so, according to different points of view.

With workflow concepts, we may use complex operators like fork/join, alternatives, variables, composite tasks, to describe some interactions sequences. Using these complex operators corresponds to use software and technical artifacts in order to address functional requirement(s). It may be relevant to define only the interactive requirements without dealing with technical details. The underlying idea is to define a modeling language dedicated to MMI, which contains a minimal set of concepts leaving technical aspects aside in order to easily focus on the interaction concern. With corresponding model transformation rules, the resulting MMI models would be mapped to several technical platforms (other than workflow management system). Thus, operating subtleties underlying the high-level models would be fully described within the generated technical models.

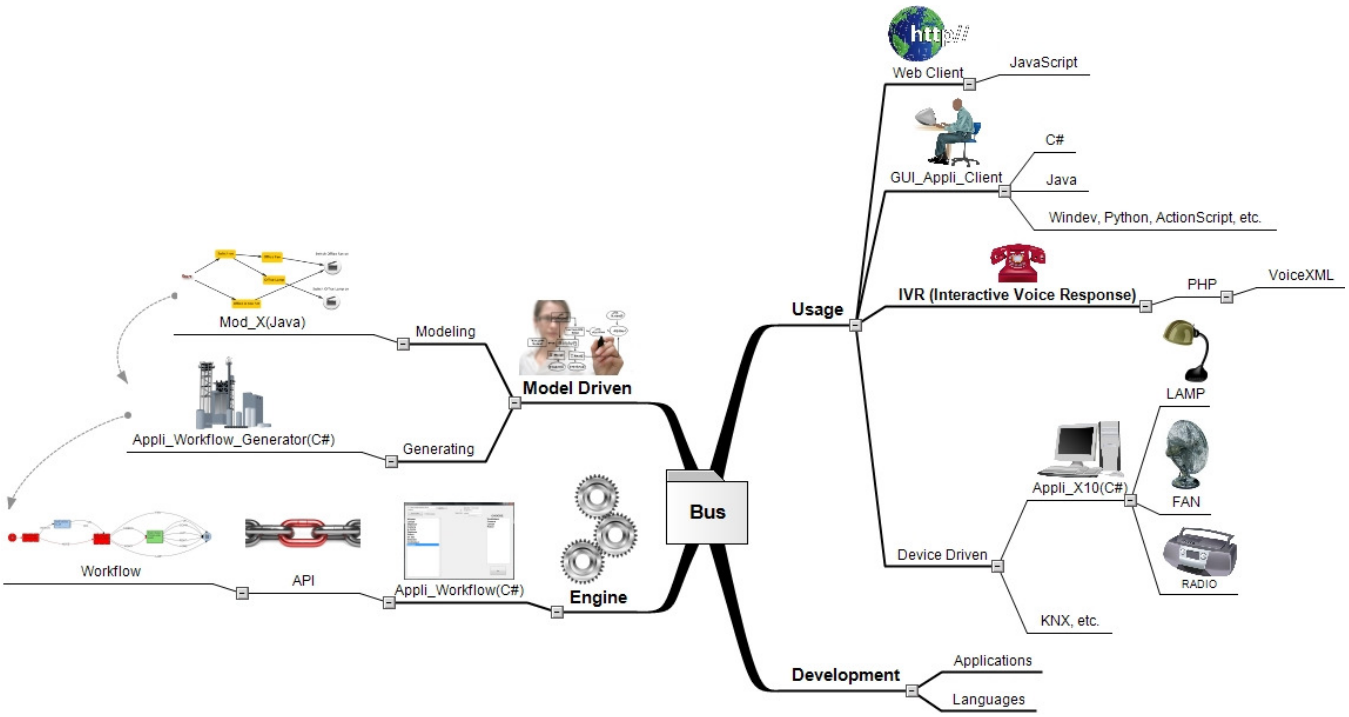

Figure 4. Our approach from meta-modeling to automatic generation of code.

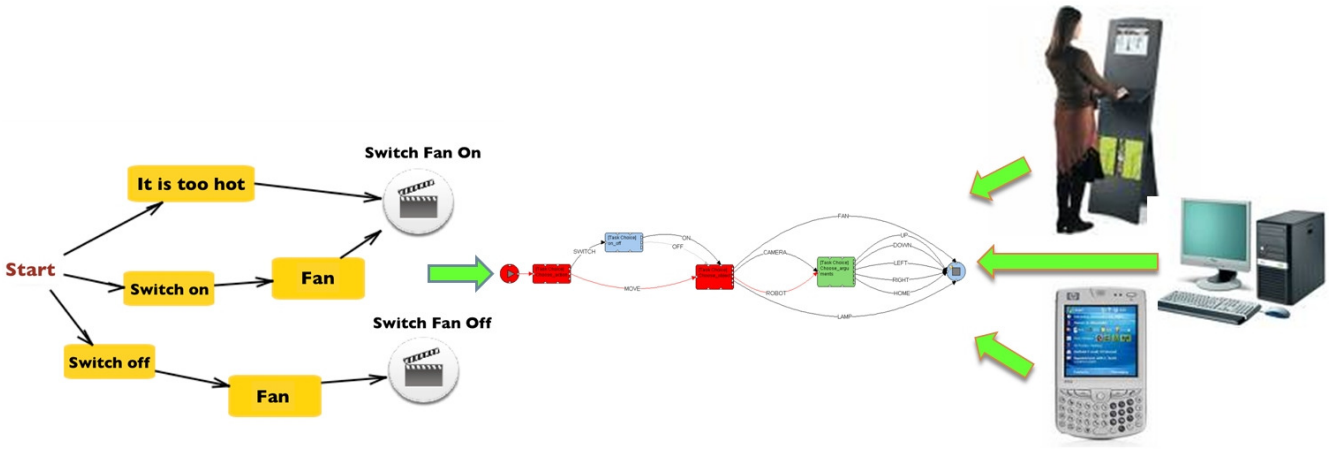

Figure 5. From meta modeling to final interfaces (via a workflow in this case).

This abstraction operation may be repeated in order to propose a simpler modeling language dedicated to end users with some technical skills (like persons who install home automation systems). Finally, we have currently chosen home automation as application domain of our work.

Our approach would have to be tested with other domains like healthcare, e-learning or tourism domains. Indeed, we cannot state that such previous high-level modeling language will still be adapted. In this perspective, we think that domain-oriented modeling languages will be useful in order to better contextualize MMI and to get finer mapping to technical platforms.

For all these reasons, we decided to adopt a Model-Driven Engineering approach, particularly the Object Management Group - Model Driven Architecture (OMG-MDA) declination (abstract towards concrete). We currently focus on an abstract meta-model and a mapping to workflow one. Figure 6 represents what we plan to do and what we have already done (gray rectangle).

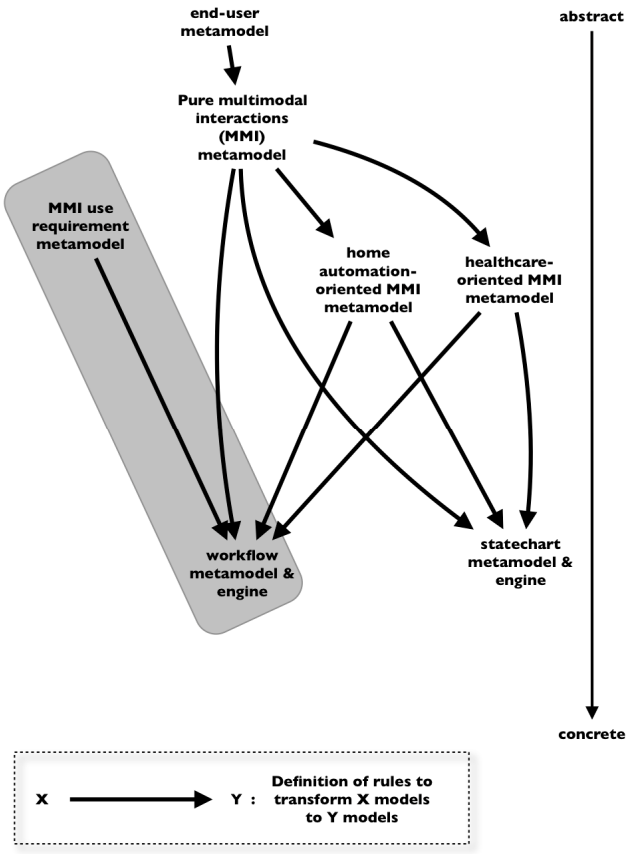

Figure 6. Our Model Driven Engineering (MDE) approach.

B. Conceptual architecture

From a conceptual point of view, our approach is based on the concept of message diffusion between the different actors in our system (an actor can be a user, an application or a device). When an actor wants to do something (for example the user wants to switch on a lamp, or the RFID reader will notify that it has decoded an RFID chip), it sends a message that is then received by all actors. Then the actors have the freedom to perform an action based on this message or not, depending on their needs.

Among the actors, the interpreter of messages has a special significance. It is the 'brain' of the system. Each time it receives a message, it processes it and tries to combine it with previously received messages to produce a higher level of abstraction message. For example, if the RFID reader has sent the message 'FAN chip decoded' and the interpreter has previously received the 'switch on' message, then the interpreter will combine the two for the final message 'switch on the fan'. This message will then in turn be sent to other actors. Among them, the application charged to operate the fan will send the X10 command to switch on the fan.

1) Communication bus

For the actors, several solutions are possible to communicate, such as:

- Pushing information, i.e., send messages to actors, such as broadcasting (sending messages to everyone), multicasting (sending messages only to certain actors), and so on.
- Pulling information. In this case, actors must request information themselves, for example by consulting a database or by consulting an actor responsible for managing the overall ambient system, etc.
- Using a distributed approach such as a multi-agent system.
- Using a centralized approach, such as a communication bus.

We chose to use a communication bus, whose function is to receive the messages and distribute them to all connected actors. This type of solution leaves considerable freedom in the implementation as we shall see later.

2) Device access layer

A communication bus is a relevant component in order to develop applications using interactive devices located in a room among several terminals. Sending a command to/from a remote device or listening/reacting to its events refers to marshalling/unmarshalling mechanisms. Its implementation is time-consuming and decreases code readability.

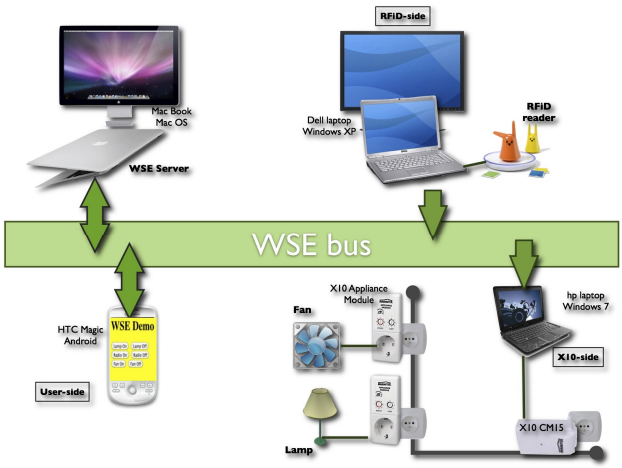

Figure 7. An example of configuration.

Figure 7 illustrates the device access layer based on the following example. A web page is loaded on an Android

mobile phone and can switch on a lamp or a fan through X10. The X10 manager (CM15 module) is connected to a PC (Windows) where a software adapter translates particular messages coming from the communication bus in X10 switch on/off orders. A RFID reader is connected to another PC: when a RFID tag is laid down on the reader, the background of previous web page changes to red, and when the RFID tag is picked up, the background is becomes green. These RFID reactions are possible thanks to a software adapter located to the related PC: for each RFID action, this adapter sends corresponding message on the communication bus.

We call terminals, the android mobile, X10-PC and RFID-PC. Web page and software adapters are called processes. To locate process, we usually mention user-side (Android mobile), X10-side or RFID-side.

To implement the previous example, we may program all processes as following. When user-side sends a "switch fan on" command to X10-side, the related process (i.e., web page) constructs a specific message and sends it through the message bus. The X10-side process receives it, detects it as a X10 order and acts in consequence. When a tag is laid down from the RFID reader, the related adapter reacts by constructing a message and sending it. The user-side process receives the message, detects it as a RFID event and sets the background to red if it is a lay-down event or to green if it is a pick-up one.

Constructing, sending, receiving and detecting messages is a tedious task (long and repetitive) and corresponding code blurs the whole implementation. For this reason it is highly recommended to use an additional software layer that hides messages bus stuff and therefore ease the implementation of MMI application.

V. OUR APPROACH: IMPLEMENTATION

This implementation section is divided in two parts. The first is about model driven engineering, and the second presents the implementation details of our conceptual architecture.

A. Model Driven Engineering

1) Towards a high-level MMI meta-model

As we previously mentioned in Section IV.A, we have adopted a model driven approach (MDA) to get a better separation of concerns (for example, by defining multimodal interactions in dedicated models) and to address the problem of platform heterogeneity.

We use ModX [19] as model framework. ModX is a MOF-tool [20] that we have implemented in 2004. It allows defining abstract and concrete syntaxes; it means meta-models and associated visual representations. ModX-users can create and edit models according to concrete syntaxes. ModX proposes a Javascript API for model transformations.

We have defined a meta-model to describe multimodal interactions requirement. We wanted this meta-model very simple: there is no notion about activity, merge, condition, etc. The main notion of this meta-model is the sentence. A sentence is a sequence of interactions and causes an action/reaction of the ambient system. A term is an interaction that refers to what a user wants to transmit (rather than focusing on the device s/he uses). The meta-model contains 3 concepts (see Figure 8): Start, Term and Action. Start and Action are ways to define the beginning and the end of a sentence. Action is also used to indicate the reaction of the system.

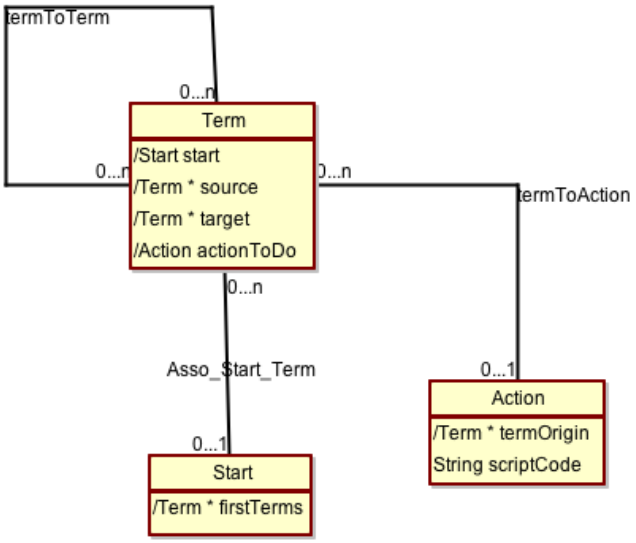

Figure 8. MMI use requirement meta-model.

A Term may be a word or a long expression, and it can be transmitted through different devices. For example, the Term 'Fan' may be indicated through speech recognition, a RFID tag, a QR code, etc. A sentence split into N Terms refers to a sentence with X different interactions.

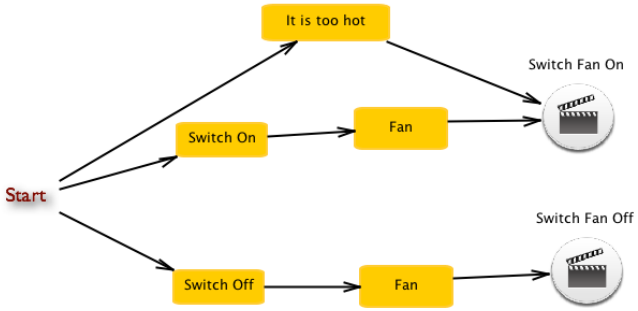

Figure 9. A sample model of multimodal interactions about home automation.

This is illustrated by the model in Figure 9 (Terms and Actions are respectively represented by rounded rectangles and cinema clap in circles).

The sentence "Switch On Fan" (from the Term sequence "Switch On" and "Fan") launches the "switch fan on" action. This sentence refers to two successive interactions and so can use a maximum of two different devices. The same Action is caused by the sentence 'it is too hot', which contains only one Term, so only one interaction that can be performed with one RFID tag or one QR code for the whole expression, for instance. Such a definition also means that "it is too hot" refers to an 'only-one interaction': the previous sentence cannot be constructed by an interaction for "it is too" and another interaction for "hot".

2) Model transformation

To map each MMI use requirement model on Common Knowledge platform, we have defined a set of model transformation rules, implemented as following:

```
1. Create a workflow model

2. Create a starting node

3. Create a taskChoice (STC) connected to the
   previous starting node.

4. For each term (T) connected to the start
     If T is bound to an action (A)
       Create a EndNode (EN)
         EN.caption = A.name
       Associate it with STC,
         association.caption = T.name
     Else
       Create a taskChoice (TC)
         TC.id = T.id
       Associate it with STC
         association.caption = T.name

5. For each term (TA)
     For each its connected term (T)
       If T is bound to an action (A)
         Create a EndNode (EN)
           EN.caption = A.name
         Associate it with TA
           association.caption = T.name
       Else
         Create a taskChoice (TC),
           TC.id = T.id
         Associate it with TA
           association.caption = T.name
```

To summarize these rules, a Term corresponds to a link, i.e., a choice that is done. When a Term is the last of a sequence (and cause an action), an `EndNode` is also created. If the Term points out to other possible choices, a `taskChoice` is created instead.

Object Connections provides a C# API for its workflow engine. It allows creating and editing workflow models. We have implemented a software adapter of this API for our communication bus, called WSE (see below). In this way, the Javascript code (in ModX) corresponding to the previous model transformation, sends WSE messages in order to create elements of workflow model.

B. Implementation of our conceptual architecture

1) Communication bus: WSE

The implementation of a communication bus can be done in several ways, e.g., with the IVY bus as we demonstrated in a previous paper [1] or a multi-agent system [21]. Unfortunately IVY does not work through the web, while using the web is one of our requirements. Therefore we decided to implement our own communication bus, called WSE (Web Server Event).

WSE is the core of our architecture and the central point of traffic. All messages, i.e., user interactions but also actions requested to devices, are carried by WSE (see Figure 11).

WSE is an HTTP-based message bus, like COMET [22]. Such buses are generally dedicated to web pages. Because we focus on interactive devices whose drivers are generally not accessible with JavaScript, we also provide an API in Java and C#. Only a web server supporting PHP scripts, for instance EasyPHP or WAMP (Windows, Apache, MySQL, PHP) is required to install WSE. We choose not to create a standalone WSE server in order to avoid conflict on port 80 with a possible existing web server. Finally we choose to use PHP scripts because of the popularity of this language. Thus, WSE should be installable on most existing / running web servers.

The immediate benefits of this web server-based solution are:

- Multi-OS: if an operating system can access the web, it can use WSE. WSE is therefore compatible with Mac OS, Windows, Android, and Linux.

- Multi-platform: the previous point implies that WSE is running on computers, smartphones, tablets, etc.

- Multi-browser: each operating system has dedicated web browsers. Because we are multi-platform and multi-OS, we are also multi-browser. Thus WSE can be used by Internet Explorer, Firefox, Chrome, Safari, Opera, and so on, as long as they support JavaScript.

- Multi-network: the web access can be done via wired connections, Wi-Fi, 3G. WSE can be used by

all these different modes of connection without restriction. As long as people have access to the web (port 80 is open), they can use WSE. We are therefore not blocked by firewalls. We also tested successfully WSE in our University that offers two different internet accesses, a network dedicated to the staff (teachers, researchers, administration) and a network with a proxy for students.

a) Features of WSE

WSE is multi-languages. Programming a Web application that uses simultaneously a Wiimote [23], a RFID reader and X10 adapters requires handling several programming languages. It can be for instance Java for Wiimote, C# for mir:ror [24] (RFID reader), Javascript for Web application. Currently WSE can be managed with C#, Java, JavaScript, and soon with ActionScript (Flex/Flash) and Python. The only two constraints for languages are to be able to process JSON (JavaScript Object Notation) and support HTTP requests, which can be implemented in any language if necessary.

Installing WSE is very simple. It consists in copying a directory ("Miny/WSE/PutOnWebServer_Root") from the ZIP file available at http://www.lifl.fr/miny, and to place this file in the root of the web server.

WSE provides basically a mechanism for trace. Traces are very interesting for an interactive system, e.g., to do debug, to support the "Undo" command or to analyze user's activities.

Messages routed by WSE are JSON objects. This implies that each message must respect a JSON structure, for instance {"param1":"value1", "param2":"value2", "param3":"value3"}. The advantage is no message format is required. Thus messages like {"action":"open"} or {"whatToDo":"open"} are acceptable. Consequently, each developer can write her/his own message format dedicated to her/his application. For our MINY project, we use the following format: {"action":"…", "actionParams":"…", "object":"…", "objectParams":"…", "location":"…", "locationParams":"…", "fromWhere":"…", "fromWhom":"…"}.

b) Using WSE

To use WSE, simply connect to a session or create one, then send and receive messages. WSE is session-based. All messages within a session are stored in a file, which is named as the session (http://server_url/WSE/traces_files/name). Below is an example in JavaScript and C# for the three steps, connect, send and receive (equivalent code in Java can be found on our web site).

i. Connect to a session

To connect to a session, the user only needs to provide the session name. If the session already exists, WSE connects to it, otherwise the session is automatically created and the connection is established.

JavaScript code:

```
<script LANGUAGE="JavaScript" src="wse.js"/>
…
wse.joinSession("mySession");
```

C# code:

```
using Newtonsoft.Json.Linq;
using Wse;
…
private Wse.Bus myWSEBus;
String serverUrl =
"http://xxx.xxx.xxx.xxx/WSE/traceSession.php";
String sessionName = "mySession";
myWSEBus = new Bus(serverUrl, sessionName);
```

ii. Send a message

To send a message, simply send a JSON object.

JavaScript code:

```
wse.sendMessage
({"action":"switchOn","object":"lamp"});
```

C# code:

```
JObject myMessage = new JObject();
myMessage.Add ({"action":"switchOn", "object":"lamp"});
myWSEBus.SendBusMessage(myMessage);
```

iii. Receive a message

To receive a message it is necessary to declare a listener for messages traveling on the bus. Each time a message is transmitted on the bus, the listener is notified and performs the associated function (Observer pattern). Then the function can extract all the needed information for the application.

JavaScript code:

```
myListener = {};
myListener.newMessageReceive = function
(message)
   { alert("A message has been received: " +
            message);
   };
wse.addListener(myListener);
```

C# code:

```
public class MyListener : IListener
{
   public void NewMessageReceive(string source,
JObject jObject)
   {
      MessageBox.Show("A message has been
received: " + jObject.ToString());
   }
}
…
MyListener myListener = new MyListener();
myWSEBus.AddListener(myListener);
```

2) Device access: Proxy/Stub generator

a) Principles

As explained before, constructing, sending, receiving and detecting WSE messages is a tedious task. For this reason, we have developed a code generator that produces a WSE-based software layer, which handles WSE message operations. With this layer, a programmer uses a remote device as a local device. The following Javascript code shows how to switch on a fan with the devices layer on the example of Section IV.B.2.

```
manager = new Manager("IJAIS2010");

X10 = manager.getX10("328", "Xavier", "Lamp");
// Param 1 : for the office number 328
// Param 2 : around the desk of xavier
// Param 3 : this X10 adapter is dedicated to a
// lamp

X10.switchOn();
```

Here is the code related to RFID events (still in Javascript).

```
rfid = manager.getRFiD("328","all");
// Param 1 : for the office number 328
// Param 2 : for all the office


rfid.layDown = function (stamp) {
     document.body.bgColor = "red";
}
// lay down a RFid tag will set the
// background color of page to red


rfid.pickUp = function (stamp) {
     document.body.bgColor = "green";
}
// pick up a RFid tag will set the
// background color of page to green
```

b) Generator

The code generator produces `userSide.X10` and `userSide.RFiDReader` classes to allow developers to focus on functional/interactive concerns without worrying about remote access. Production of such a class is done from a description of actions (called methods) and events of related devices. The description is JSON formatted and therefore does not imply to use another language.

Here are the two description files corresponding to X10 and et RFID reader devices.

```
{
  name : "X10",
  package : "x10",
  type : "Device",
  constants : {

    object : '"x10"',
    objectParams : null,
    location : null,
    locationParams : null
  },
  methods : {
    switchOn : {},
    switchOff : {}
  }
}
```

```
{
  name : "RFiD",
  package : "rfid",
  type : "Device",
  constants : {
    object : '"RFiDReader"',
    location : null,
    locationParams : null
  },
  events : {
    layDown : {
      stamp : String
    },
    pickUp : {
      stamp : String
    }
  }
}
```

We have defined a generic format inspired from JSON-RPC [25] in order to homogenize the structure of WSE messages that will be exchanged though this devices layer.

This format message protocol is the following:

- action: the expected action (e.g., switchOn) or name of the event (layDown for a RFID reader).
- actionParams: arguments of action or event.
- object: type of device (e.g., X10, RFIDReader).
- objectParams: optional details about the device (for instance X10 has two objectParams: lamp, fan).

Rather than automatic identifiers, we choose to use explicit identifiers, which indicate where the device is.

- location: indicates where the device is (for example: 'Office 328').
- locationParam: details the place in the previous location (e.g., 'Desk of Xavier')

Figure 10 shows a communication between user-side and device-side. Concerning the user-side, the generator produces a proxy class for each description file. For each described method, the proxy (step 2 on Figure 10) contains a corresponding method that consists in creating a WSE message and sending it. If the description defines events, an interface is generated. It contains one method for each event. This interface is associated to the proxy: add/remove listeners methods are added to the proxy while a listener consists in an object implementing the interface. In Javascript, there is no listener interface. The events correspond to methods of the proxy.

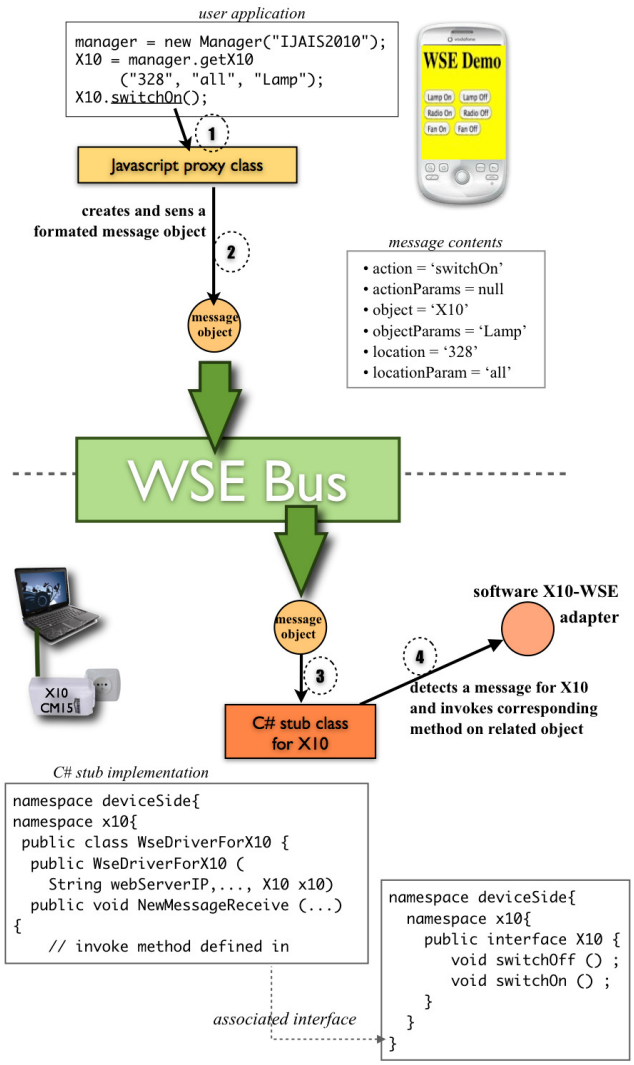

Figure 10. Stub and skeleton on example.

A class `Manager` is also generated and acts as a factory. This class is instantiated with a WSE bus as parameter, and gives access to proxy objects according to location/identification values (step 1 of Figure 10). If a programmer wants to add a new type of device in the device access layer, the code generator can also help her/him by producing code, a stub class, (step 3 of Figure 10) related to WSE stuff. This stub class will have to be connected to another class, a device-WSE adapter (step 4). This one has to interpret a) WSE actions into actions on devices, b) events from device into WSE events. The generation principle is the same as for the user-side but with reversed responsibilities: the skeleton contains a method for each event that device can emit and an associated interface which defines a method for each possible action on the device.

C. Our methodology in a few words

To summarize, here are the major steps to follow to implement our methodology:

1. Identify the devices and the associated actions to use.
2. Define a MMI model to specify the possible interactions that you want to apply through the device actions (cf. Figure 9).
3. Convert this MMI model into a workflow; this step is done automatically in our case.
4. Implement a distributed communications and access to devices. Designers can use the stub/proxy generators (see above), or even can use the already-implemented package we propose for RFID, Androphone, BCI, X10 and IP Camera.
5. Implement the parts that associate interactions to real actions on devices. For instance lay down a specific RFID tag should produce the "It is too hot" interaction.
6. Start the WSE drivers for each device with providing parameters such as IP address, session name, location, etc.
7. Start the workflow engine and the code produced in step 5.

VI. Case Study

This case study section is divided in four parts, which present, respectively, the domain of smart digital home, the architecture of the project, the implementation of this case study and finally, the multimodal aspects of this implementation.

A. Smart digital home

A smart digital home refers to a living space with devices that are connected through wired or wireless networks. The

connected devices may be sensors, actors, consumer electronics, appliances, mobile and PC devices that cooperate transparently for facilitating living and improving usability in the home. Since a variety of devices are present in a smart digital home, convergence and standardization across all the screens of TVs, PCs, appliances and mobile devices, and management of multi-channel interactions is manifestly the key for the success of residential applications.

In our example, several objects are identified in order to be driven remotely: a lamp, a fan, a Rovio robot [26], and a webcam. The possible actions on those objects are the following: move (up, down, left, right, and home) and switch (on/off). As we can see on Figure 2, while the interaction takes place, one of the possible paths of the workflow is followed. Once the final state is reached, a command is sent to the bus.

B. Architecture

For this smart digital home case study, we are using the IVY software bus [18] or our WSE bus, indifferently.

With the IVY bus, a publish/subscribe mechanism is available. Some applications are only subscribers. It means that they need data to prompt information to the user (a synthesized speech for example), to activate appliances (micro-wave oven, washing machine, etc.), or to generate some piece of VoiceXML [3][27] code that will be dynamically generated and used at runtime. Some applications are only sending information to the bus. Others are using the bus to both receive and send data. For instance, the Automatic Speech Recognition (ASR) application usable on a PC needs to receive the different labels corresponding of the speakable words, and oppositely, it sends to the bus the result of the speech recognition engine.

The “Workflow_Engine” application is in charge of the connection with the persistent workflow that we use for this project. It exploits a dedicated API to send the choices of the user to the object connection engine, and to receive the next elements to be presented to the user.

C. Implementation

Our global project was conceived to manage various kinds of devices, sensors, effectors and technologies such as keyboard and mouse, voice over telephone or softphone, QR code, multi-touch screen, Wiimote, Mirror [24] / Reflet [28] NanoZtag RFID, motion webcam, X10 protocol, Rovio robot [26], etc.

Our proposition is based on the architecture illustrated in Figure 11. Three types of elements are present: (1) Interactive components that are detectors and/or effectors, (2) Communication bus for message exchange and (3) Workflow engine. This proposal aims at providing developers the ability to associate to her/his application a multimodal dimension concerning its interactive part. Currently, interactions supported are ruled by only one principle, which is "sentences triggering actions". A sentence consists in a sequence of words that can be triggered by any type of modality (voice, QR code, keyboard/mouse, etc.). To facilitate the writing of such sentences for an application, we use the Task Choice concept [17] in order to factorize words. For example, a sentence may begin by "move" and then be divided into 4 sub-sentences (one for each concerned device). This avoids writing four complete sentences.

An example of path may be the following one: the user activates the button "move" from the Windows application (first sub-action), presents in front of a webcam a QR code identifying the robot (second sub-action) and then pronounces on her/his phone the word "left" (third sub-action). This path is completed and the action "move the robot on the left" is triggered.

Once a model is loaded into the workflow engine, it is executed by the engine that starts with the first task choice. Each time the engine points to a new task choice, the list of possible choices is sent to the bus. This is done by a software agent attached to the workflow engine. Thus, interactive components can subscribe to this type of message, in order, for example, to present the list of choices to the user (as graphical buttons, voice prompt, etc.).

Two other software agents were needed and developed. The first one notifies the workflow engine that a sub-action was performed. This type of agent is attached to an interactive component and translates each relevant interaction into a sub-action that is sent to the bus. The second agent allows to be notified that an action is requested (e.g., switch on fan). Such agent aims to be associated to an interactive component that will translate actions into actual commands on the component, using X10 protocol, for instance.

The three software agents previously mentioned have two roles: to subscribe/transmit on the communication bus and to establish a protocol for discussion between the workflow engine and interactive components. This protocol is based on actions, sub-actions and possible actions. Note that in the model associated to smart digital home, we defined paths so user must first specify the command, then identify the device and finally give a possible parameter for command.

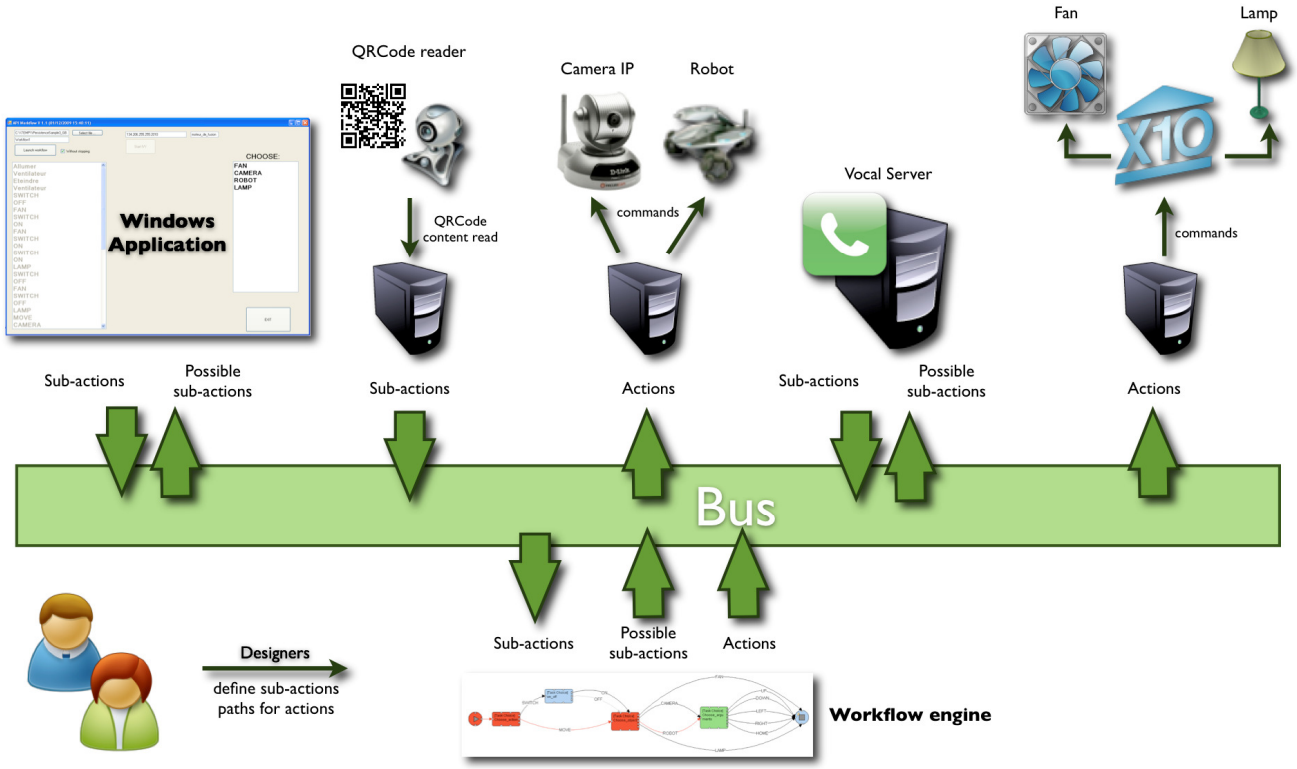

Figure 11. Architecture of our Smart Digital Home project.

The three software agents used the workflow presented in Figure 2 to describe the objects and actions that can be applied on those objects using one or more devices.

D. Multimodality

As previously mentioned, our goal is to provide tools in order to facilitate the design and implementation of multimodal interfaces for ambient computing. Concerning vocal interactions, one big challenge is to provide the designers an easy and robust way to generate code (like VoiceXML [29] for instance) that can integrate grammars related to a particular changing context. Dynamic voice grammars (or entire VoiceXML files) can be generated with our approach, as we can see in Figure 12.

If the designer decides to add a possible new direction, s/he can do it graphically, on the workflow, by adding an arc (called “home” for example), near the up/down/left/right already available. Then with no addition of code, a new possible interaction is available through the workflow. Consequently, one can then pronounce a sentence like “move camera home”, in order to physically make the webcam move.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
version="2.0" xml:lang="en-gb">
<form>
<grammar version="1.0" root="GR_VOICE"
mode="voice" tag-format="semantics/1.0">
<rule id="GR_VOICE">
<one-of>
<item>up<tag>out.choice="up";</tag></item>
<item>down<tag>out.choice="down";</tag></item>
<item>left<tag>out.choice="left";</tag></item>
<item>right<tag>out.choice="right";</tag></item>
<item>home<tag>out.choice="home";</tag></item>
</one-of>
</rule>
</grammar>
<field name="choice"><prompt>
Choose among up, down, left, right, home
</prompt>
<filled>
<prompt bargein="false">
The chosen value is: <value expr="choice"/>
</prompt>
</filled>
</field>
</form>
</vxml>
```

Figure 12. Example of VoiceXML code generated by the VoiceXML_Maker agent.

For this case study, we have implemented a multi-device, multimodal, and multi-channel system:

- a Multi-device system because more than one device can be used during the interaction. In our experiments we used many PCs, smartphones and telephones, and a Wii Console.

- a Multi-modal system because more than one modality can be used during the interaction. In our examples we used traditional keyboard/mouse interactions, vocal, gesture and brain computer interaction (BCI). We also used QR codes and RFID tags containing data related to desired actions or objects.

- a Multi-channel system because more than one channel can be use during the interaction. In our smart home case study, it was done across internet and telephone networks.

VII. Conclusion and Future Work

The goal of this paper was to describe how we can facilitate the design of multi-channel and multi-modal interfaces for ambient computing with a model-driven approach. We used a smart digital home case study to explain how to design easily an ambient system using a workflow oriented approach.

Our results show that different devices (such as Wiimote, multi-touch screen, telephone, etc.) can be managed in order to activate real or virtual things. Adding new features (such as appliances, actions, direction, etc.) to an existent system is also very easy and only needs a modification of the workflow.

Our work is orientated toward the production of code generated from model (and meta-model) transformations, and shows that this model-driven approach is encouraging and suitable for the ambient computing domain. With our methodology, a large part of the scripts and applications programs, traditionally coded by developers, can be automatically generated by the ambient system itself.

In the future, this should improve the possibility to detect new objects, persons or possible behaviors dynamically and to respond to them as soon as possible with relevant feature of the ambient system. Thus, it will be challenging to work on the possibility to manage simultaneously different natural languages with a unique model of existing actions.

We will also work on the important point of semantic aspect of the workflow. This will help users for instance when they will not use the commands in the right order. Indeed, a smart system must be able to understand that “move up robot” is the same command as “move robot up”. We are also planning to offer the possibility to dynamically switch from a software bus to another and to manage virtual representation of tangible things (fridge, oven, etc.) in order to allow realistic simulations before real implementation.

VIII. Acknowledgement

The authors would like to thank ObjectConnections, Jaxo Sytem and bcWebCam for providing special tools: Common Knowledge, Cam'A'Bar and bcwebcam.

IX. References

[1] Rouillard, J., Tarby, J.C., Le Pallec, X., and Marvie, R., “Facilitating the Design of Multi-channel Interfaces for Ambient Computing”, The Third International Conferences on Advances in Computer-Human Interactions, ACHI 2010, St. Maarten, Netherlands Antilles, 2010, pp. 95-100.

[2] W3C Multimodal Interaction Activity (MMI), Retrieved January 10, 2011, from http://www.w3.org/2002/mmi/

[3] VoiceXML 2.0., W3C Recommendation (16/03/04), Retrieved January 10, 2011, from http://www.w3.org/TR/voicexml20

[4] Harel, D., “Statecharts: a visual formalism for complex systems”, Science of Computer Programming, Volume 8, Issue 3, pp. 231-274, 1987.

[5] OpenInterface European project. IST Framework 6 STREP funded by the European, Commission (FP6-35182). Retrieved January 10, 2011, from http://www.openinterface.org and http://www.oi-project.org.

[6] Tarby, J.C. and Rouillard, J., “Assistance, advice and guidance with digital coaching”, EAM'06 European Annual Conference on Human Decision-Making and Manual Control Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2006, Valenciennes.

[7] Frohlich, D., “The design space of interfaces, multimedia systems, Interaction and Applications”, 1st Eurographics workshop, Stockholm, Sweden, Springer Verlag, p. 53-69, 1991.

[8] Healey, J., Hosn, R., and Maes, S.H, “Adaptive Content for Device Independent Multi-modal Browser Applications”, Lecture Notes In Computer Science; Vol. 2347, Proceedings of the Second International Conference on Adaptive Hypermedia

and Adaptive Web-Based Systems, pp. 401-405, ISBN: 3-540-43737-1, 2002.

[9] Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J., and Young, R. M., "Four easy pieces for assessing the usability of multimodal interaction: the CARE properties". In INTERACT, pages 115-120. Chapman & Hall, 1995.

[10] Vanderdonckt, J., Grolaux, D., Van Roy, P., Limbourg, Q., Macq, B., and Michel, B., "A Design Space for Context-Sensitive User Interfaces", Proc. of ISCA 14th Int. Conf. on Intelligent and Adaptive Systems and Software Engineering IASSE'2005 (Toronto, 20-22 July 2005), International Society for Computers and their Applications, Toronto, 2005, pp. 207-214.

[11] Rouillard, J., "Multimodal and Multichannel issues in pervasive and ubiquitous computing", Multimodality in Mobile Computing and Mobile Devices: Methods for Adaptable Usability, Idea Group. Inc, Information Science Reference, ISBN: 978-1-60566-978-6, 409 pages, 2009.

[12] Bastien, Ch. and Scapin, D., "Ergonomic Criteria for the Evaluation of Human-Computer Interfaces", J. M., INRIA Technical report N° 156, 1993.

[13] Bourguin, G., Lewandowski, A., and Tarby J-C., "Defining Task Oriented Components, Task Models and Diagrams for User Interface Design", 6th International Workshop, TAMODIA 2007, Toulouse, France, November 7-9, 2007, Marco Winckler, Hilary Johnson, Philippe A. Palanque (Eds.), Lecture Notes in Computer Science 4849 Springer 2007, ISBN 978-3-540-77221-7, pp. 170-183

[14] Tarby, J.C., "One Goal, Many Tasks, Many Devices: From Abstract User Task Specification to User Interfaces" (Chapter 26). In, Diaper, D. and Stanton, N. The handbook of Task Analysis for Human-Computer Interaction. (pp.531-550). Mahwah, New Jersey: Lawrence Erlbaum Associates, 2004.

[15] Palanque P., Bernhaupt, R., Navarre, D., Ould, M., and Winckler, M., "Supporting Usability Evaluation of Multimodal Man-Machine Interfaces for Space Ground Segment Applications Using Petri net Based Formal Specification". In International Conference on Space Operations (SpaceOps 2006), Rome, Italy, 18/06/06-22/06/06, American Institute of Aeronautics and Astronautics (AIAA), 2006.

[16] Horrocks, I., Constructing the User Interface with Statecharts, Addison-Wesley Professional, 272 pages, 1999.

[17] ObjectConnections, Common Knowledge Studio and engine, provided by ObjectConnections. Retrieved January 10, 2011, from http://www.objectconnections.com

[18] IVY Bus, Retrieved January 10, 2011, from http://www2.tls.cena.fr/products/ivy/

[19] ModX MOF modeling tool, Retrieved January 10, 2011, from http://edutechwiki.unige.ch/en/ModX

[20] MOF OMG Meta-Object Facility, Retrieved January 10, 2011, from http://www.omg.org/mof/

[21] Kubera, Y., Mathieu, P. and Picault, S., "Everything can be Agent!", Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010), van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), Toronto, Canada, pp.1547-1548, 2010.

[22] COMET, Retrieved January 10, 2011, from http://en.wikipedia.org/wiki/Comet_(programming)

[23] Nintendo, Wii game console and Wiimote controller, Retrieved January 10, 2011, from http://www.nintendo.fr/

[24] Nabaztag, Mir:ror, Nano:ztag, and Ztamp:s, from Violet, Retrieved January 10, 2011, from http://www.violet.net/index_en.html

[25] JSON-RPC: lightweight remote procedure call protocol, Retrieved January 10, 2011, from http://json-rpc.org/

[26] WowWee Group Limited, Rovio robot, Retrieved January 10, 2011, from http://www.omg.org/mof/http://www.wowwee.com/en/support/rovio

[27] VoiceXML 2.1, Recommendation, (19/06/07), Retrieved January 10, 2011, from http://www.w3.org/TR/voicexml21/

[28] Ref:let, An open-source alternative to mir:ror from Violet, under Windows, Retrieved January 10, 2011, from http://code.google.com/p/reflet-mirror/

[29] VoiceXML 3.0, W3C Working Draft (08/08/2008), Retrieved January 10, 2011, from http://www.w3.org/TR/vxml30reqs/