How to communicate smartly with your house?

José Rouillard* and Jean-Claude Tarby

LIFL Laboratory,
University of Lille1,
59655 Villeneuve d’Ascq cedex, France
E-mail: Jose.Rouillard@univ-lille1.fr
E-mail: Jean-Claude.Tarby@univ-lille1.fr
*Corresponding author

Abstract: We are involved in conceiving and implementing man–machine interfaces allowing multimodal and situated dialogue in order to interact with smart digital environments. Several methods are presented depending on how and what the user can say. We describe our approach and some tracks to permit to end-users to communicate more naturally with their house, by speech or by gesture, for instance. Some applications based on workflow description are presented to show how designers are able to model their own man–machine dialogue. Finally, this work is complemented with a Model Driven Engineering approach using Statecharts for a smart digital home.

Keywords: man–machine dialogue; smart digital home; voice interaction; workflow; statechart.

Reference to this paper should be made as follows: Rouillard, J. and Tarby, J-C. (2011) ‘How to communicate smartly with your house?’, Int. J. Ad Hoc and Ubiquitous Computing, Vol. 7, No. 3, pp.155–162.

Biographical notes: José Rouillard is an Associate Professor in Computer Science at the LIFL Laboratory of the University of Lille1. He obtained his PhD in 2000 from the University of Grenoble (France) in the field of Human–Computer Interfaces and his Habilitation to Direct Research in 2008. He is interested in HCI plasticity, multimodality and multichannel interfaces. He has written more than 70 papers and is now engaged in research on mobility and pervasive/ubiquitous computing.

Jean-Claude Tarby received his PhD in Computer Science from the University of Toulouse I in 1993. Since 1995, he is an Associate Professor at the LIFL Laboratory of University Lille1. His research is focused on the modelling and the design of cooperative applications based on user’s tasks. His research interests include multimodality, meta-modelling, user interface design methods, groupware and traces of activities.

1 Introduction

Historically, home automation has been presented some 20 years ago as to revolutionise people lives. However, blockers factors, mainly due to cost, not reliable solutions, but also to the difficulty to adapt a generic system to each dwelling, have led, if not to a failure, at least to delay in this area (ISTAG, 2001). A very few part of our homes have automated systems that can handle lighting, systems of opening/closing blinds, doors, garages and even controlling the heating and air conditioning.

Currently, these systems are mostly based on clock mechanisms. Some allow remote manipulations, by phone or PDA, and recent commercial solutions, such as HAL (Automatedliving.com, 2009) software, allow using voice command. It seems that a new step will be taken in this area if the building is considered as a unit forming part of a communicating network, and not as a mere receptacle. The building remains at the heart of this smart system because it has the longest lifetime. As and when they appear/disappear, other components will connect seamlessly, ideally without human intervention, and present to users, through introspection, their ability to be used in a ‘natural’ way. The gestures and voices are privileged in our work because they allow more easily communication with the system without special learning. Current interfaces for home automation mainly try to meet the needs of users in an action/reaction mode. They are largely used to command connected devices (lamps, fans, etc.) and are often limited to simple actions without taking into account the changing context (Abowd and Mynatt, 2000; Dey et al., 2001). For a long time, users were mostly ‘forced’ to use graphical interfaces (keyboard/mouse).

More recently, haptic interfaces have emerged in the context of home automation. In fact, they exist since many years, but it becomes easier to integrate them in projects, thanks to multitouch toolkits, table surface SDK, touch phones, etc. The main idea of this paper is to explore the possibility of improving the communication between users and their house by focusing on man machine dialogue in
natural language. This dialogue is often using spoken, written or gestured modalities. Natural gestures are captured by a camera and instrumented gestures are controlled by using remote devices such as Wiimotes for instance.

This paper is structured as follows. First, Section 2 will introduce natural ways of communication in a smart digital home. Then, in Section 3, we will see how to talk to house with static or dynamic grammars, and with large vocabulary. Section 4 will be a presentation of our work through an MDE approach. An implementation of an interaction engine will be presented. Finally, we close with the main conclusions and suggestions for future research.

2 More natural ways of communication in smart digital home

Thanks to advances in touch and gesture technologies, new ways of communication with smart digital home are becoming possible. In the next paragraphs, we are presenting briefly some interfaces developed in our laboratory and based on touch and instrumented gesture.

We can see in Figure 1 that (a) a rotation movement on the right is executed on a touch screen, running under Microsoft Windows 7, to switch on a fan, and (b) a two fingers translation, from inside to outside the screen, is used to switch on a lamp. This category of interaction is easy to propose across different devices in a house.

Figure 1 Using touch technology to command appliances (see online version for colours)

One of the scientific locks that researchers are trying to solve is how to bring a large variety of movements with mono or multitouch screen that stay natural and easy to perform. Obviously, this kind of tactile gesture needs a close interaction between users and devices. Another possibility to use gesture is to employ instruments such as remote control.

For various kinds of people (elderly people, younger, etc.), it could be a manner to interact smoothly with the system, without the need of Windows, Icons, Menus and Pointing device (WIMP) paradigm.

Figure 2 presents an example of implementation made in a house, where the TV set can be used to control appliances (lamp, fan, radio, etc.). Interacting with a Wiimote controller allows the user not to be very close to the screen. The Opera browser channel of the Wii (Nintendo, 2006) is employed to access a web server running on a PC. Information is broadcasted on a software bus named IVY (CENA, 1996). The control of appliances can be done with a smartphone and other mobile devices with internet connection. Our current work is oriented towards Universal Plug and Play (UPnP) protocol to discover on the fly new devices and services (Ran, 2003) that could be presented on the TV screen of the user, connected to his or her Wii.

Figure 2 Using a Wiimote to switch on a lamp of the house (see online version for colours)

3 Talk to house

Talking is one of the best ways to interact intuitively with intelligent ambient systems. Speech and sound recognition are mainly used in smart homes for two kinds of applications: recognition of particular situations and home automation. For the first aspect, where the user does not interact consciously with the system, some recent work show that sound and speech recognition can be used for many applications inside the home, such as quantification of water use or detection of distress situations. For instance, mixing detected sound and vibration of the floor allowed detecting fall of the occupant of the room (Popescu et al., 2008) cited by Vacher et al. (2010). For the second aspect, home automation, human speech is used intentionally to interact with the system. This can be done across a microphone that the user wears or by a set of microphones integrated into the ceiling.

In the DomoEsi project, researchers are providing UPnP interface that offers the possibility to use voice recognition and text to speech components, available in Microsoft Windows operating systems, as a possible manner to interact with the house:

“These services do not have economical impact for the installation when the PC operating system is Windows Vista or Windows XP. If it is not the case it is not possible to use this application.” (Maestre and Camacho, 2009)
Allen and colleagues demonstrated a prototype system, named Chester, which can assist elderly people, in house of retired, for instance, by dialogue using with them. It is an intelligent assistant that interacts with its users via conversational natural spoken language to provide them with information and advice regarding their prescribed medications (Allen et al., 2006).

We will explain in the next paragraphs how to use speech recognition on PC installed in a house, or via a telephone conversational interface.

3.1 Static speech recognition

When the vocabulary is known before the interaction, it is classical to use a static grammar stored in an XML-based format. Figure 3 shows an example of speech grammar (ongoing_task_grammar.grxml) used for a voice control on a PC running under Microsoft Windows 7.

3.2 Dynamic speech recognition

Speech recognition for smart digital home can be used statically, with simple or complex grammars, based on Backus-Naur Form (BNF) grammars (Naur, 1960). It is a good method (quick and robust) when the vocabulary is known before the interaction. The words and possible sentences are stored in a specific grammar file, written by the designer, as previously explained. But sometimes, it can be interesting to use dynamic grammars that allow users to pronounce words not necessarily envisaged at design time. One method consists of creating dynamically a grammar and to use a static code that executes it at runtime. Another method generates both the grammar and the accompanying code on the fly.

A major question in pervasive and ubiquitous computing is how to integrate physical object of the world (screen, chair, coffee machine …) into multimodal applications thanks to technology such as RFID, 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. In our work, we are using the notion of workflow to indicate to the user the tasks available at each point of the whole activity flow (Rouillard, 2009).

With the grammar and the application described earlier, it is possible for a user to pronounce sentences such as ‘Switch on Lamp’ or ‘Switch off Microwave Oven’. Thus, simple vocal orders can be used loudly in a house, with or without (if microphones are fixed in the ceiling, for instance) headset.

Figure 3 Example of speech grammar used for voice control (ongoing_task_grammar.grxml)

```xml
<grammar
xmllang="en-US" tag-format="properties-ms/1.0"
version="1.0" mode="voice"
xmlns="http://www.w3.org/2001/06/grammar">
  <rule id="Actions" scope="public">
    <one-of>
      <item repeat="0-1">SWITCH</item>
      <item>ON</item>
      <item>OFF</item>
    </one-of>
  </rule>
</grammar>
```

Figure 4 is a code snippet written in C# that uses the speech grammar file (ongoing_task_grammar.grxml) presented in Figure 3.

```csharp
using System.Speech.Recognition;
...
private string path = "C:\\grammar\";
private string file_grammar = "ongoing_task_grammar.grxml";
...
// Load "Actions" rule
SR.LoadGrammar(new Grammar(path + file_grammar, "Actions"));
...
// This event is fired after the SR completed the recognition
SR.RecognizeCompleted += new EventHandler<RecognizeCompletedEventArgs>(sr_RecognizeCompleted);
```

Figure 5 is an example of a VoiceXML (W3C, 2007) code that integrates an inline grammar. This code is dynamically generated at runtime, while the user is talking on the phone with his or her house to execute some commands. We will see that the words of the grammar come from a workflow designed graphically (see Figure 6).

Figure 6 presents an example of workflow designed graphically with the Common Knowledge Studio (CKS) tool (ObjectConnections, 2009). It allows following different paths to complete a command such as ‘switch on fan’, ‘move camera down’ and ‘switch off lamp’. The final home automation system could be used through different modalities of interaction: graphically, vocally, with gesture, RFID, barcodes or a combination of those modalities. Instead of programming ad hoc applications, our approach allows querying dynamically the workflow and proposing, on the fly, relevant information to the user, during the interaction with the system.

If the designer decides to add a possible new direction, he or she can do it graphically, on the workflow, by adding an arc (called ‘home’ for example), near the up/down/left/right already available. With no addition of code, a new possible path is potentially usable in the workflow. Consequently, one can then pronounce a sentence like ‘move camera home’, to physically make the webcam move. The notion of persistence is very
important in this context. Indeed, we consider that a main interaction could be the result of many subinteractions between the system and one or many users. It could also be the result of a sequence of subinteractions conducted via different kinds of channels and modalities. The Common Knowledge software that we are using for our research supports this persistence feature. To do this, we use Common Knowledge, a cross-platform business rules engine and management system that supports the capture, representation, documentation, maintenance, testing and deployment of an organisation’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, flow charts, decision tables, decision trees, decision grids, state maps, and scripts. The engine allows running, testing and simulating the system behaviours. It can be used through many languages (such as Java, Delphi, Visual Basic, C# and DotNET) and platforms (Windows, Linux, UNIX). Standard and advanced operators can be used graphically to represent tasks, task choices, split or merge actions, timers, loops, etc.

3.3 Speech recognition with large vocabulary

An interesting challenge for researchers in smart digital home domain concerns the possibility to discover new services, during the interaction, thanks to UPNP protocol for instance. But, if technically some solutions become available and secure, it is not so easy to bring to end-users the ability to say words or sentences that have never been envisaged by designers.

In front of new situations, users will certainly use different strategies, according to the context. For example, to switch on a specific light in the house, people can describe it according to different criteria (on the table, new, outside, old fashion, red, of my room, fragile, etc.). One possible solution that we propose is to let users pronounce whatever he or she wants, and to exploit the obtained speech recognition with natural language specialised tools. We implemented such a solution by using the SpinVox API technology (Spinvox, 2005). Figure 7 shows how we sent a speech audio file to the SpinVox API and obtained the translated sentence pronounced by the user: “Show me the result on the screen please”. This method is very robust but need a certain time to be performed (around 30 s for this example, depending on the quality and the size of the audio file).

After that step, it is possible to refine the result by using tools such as Antelope, provided by Proxem, for instance. Proxem is an innovative company dedicated to Natural Language Processing and the Semantic Web (Proxem, 2007). Antelope stands for Advanced Natural Language Object-oriented Processing Environment. This framework facilitates the development of Natural Language Processing software. Antelope, currently in version 0.8.7, is designed for the Microsoft .NET framework (version 2.0 or above). Therefore, it is possible to use it with C#, Visual
How to communicate smartly with your house?

Basic.NET, Delphi.NET and many other .NET compliant languages. Antelope includes the following features: Access to many part-of-speech taggers, a chunker with ‘divide and conquer’ strategies, access to the Link Grammar dependency parser, access to the Stanford Parser 1.6 (no need of a Java VM), a full lexicon, with rich relations, based on WordNet 3.0 data, a very versatile user-defined lexicon facility, a 300,000 synsets extension with a mapping to Wikipedia, a syntax/semantic layer, based on VerbNet 1.5, an anaphora resolver, a context extraction module, etc. Thus, it is interesting for a home automation system to ‘understand’ in a sentence that a ‘microwave oven’ is a kind of ‘appliance’, for example. This can be the basis of a smart and rich dialogue between the house and the users, to determine what to do and to learn it for the next sessions.

4 Contributions of MDE for our modelling work

We use an MDE approach in our work, currently only through modelling; code generation and model transformation are being studied. However, this approach has two major advantages:

- it allows modelling the application and therefore to be disconnected of technical constraints and programming languages, thus bringing the application in various physical environments and software
- it allows the generation of code, which implies consistency between models and the application code.

Modelling is used in two stages: the application domain modelling (i.e., the home automation), and modelling of interactions. We detail below these two stages.

4.1 Modelling of home automation

Figure 8 shows our meta-model of a digital house. This meta-model contains the three basic entities: objects, places and actions. Each of these entities possesses properties, which have a name, a type, a value and a mode (‘mandatory’ or ‘optional’). This implies the total generality of these properties; thus, each of the three basic entities possesses properties, which may be different from other properties entities. With this meta-model, we can represent that a digital house contains ‘communicating’/‘interactive’ objects, which are located into rooms (e.g., the lamp in the living room), or more generally defined by places in the house (e.g., the lamp near the living room window). These objects are subjected to actions from the inhabitants of the house (e.g., switch on/off the lamp near the living room), but they can also provide actions (e.g., a robot could move an object in the room) or services (e.g., a camera could send images).

Our meta-model allows the generation of an interaction model as Statecharts. On the MDE perspective, this generation is a model-to-model transformation, and is currently manually but we are working on implementing this model transformation.

4.2 Modelling the interaction with Statecharts

All actions that can make the inhabitants of the digital house are modelled in an overall Statechart (see Figure 9). The main function of this Statechart is to retrieve information from user interactions, then to construct messages (complete if possible) from the information and finally to send these messages to the execution engine, which will in turn spread these messages for the household objects react to the order of the inhabitants.

This overall Statechart is constructed from the three basic entities (i.e., action, object and place). For this, we create a state containing three concurrent sub-states, one per basic entity. This overall state resets either periodically (after a fixed delay) or as soon as these sub-states are completed. The first case corresponds to the situation where the user has not provided the information for lack of time or lack of information (e.g., she or he does not know the name of an action) or else by laziness (she or he hopes the system will guess what is missing in her or his order, for example, ‘light the lamp’). The second case occurs when the user gives all the information necessary for a complete message can be performed, for example, “open a little the blinds of all bedrooms”.

Each sub-state manages a basic entity that is one sub-state for the actions, one for the places and one for the objects. Since the basic entities have properties, we decompose the sub-states in two concurrent sub-states, one for the entity itself and for its properties. Thus, for the Action entity, we get an Actions sub-state that contains a sub-state, which manages the names of actions, and another that manages the parameters of actions (see Figure 10). This principle applies even to places (see Figure 11) and objects (see Figure 12).
When a sub-state runs, it fills a message structure (see Figure 13). This structure is also derived from the model of home automation. This structure contains the basic entities and their properties, such as “action (name) + action (parameters)”, and also cells specific to interaction such as “who has called for action” and “where is the person requesting action”.

If the message structure is completely filled before the overall Statechart does rerun, then the Statechart sends the complete message to the execution engine that processes the message. Otherwise, the Statechart sends the message with its ‘incomplete’ contents when the Statechart resets. In this case, the execution engine chooses an option from:

- performing the message as it is because it is not necessary that all boxes are filled for this message
- completing itself the message because it can deduce the unfilled boxes
- requesting additional information to the user (e.g., to remove ambiguity)
- not sending the message because it believes it is inconsistent.

Figure 13 shows two examples. The first one corresponds to an order sent from the parents’ bedroom and means “open the blinds in this room”. The second example corresponds to an order from John who is in the living room, and means “From the living room, John asks to open the blinds in parents’ bedroom”.

Note also that thanks to the properties of Statecharts, all sub-states run in parallel. Thus, the user can create a message with the words in the order she or he wants. For example, “living room lamp switch on” shall have the same result as “switch on lamp living room”. The message is created by filling the right cells (the Actions sub-state performs the actions cell, etc.), but the message elements come in an unpredictable order.

### 4.3 Implementation of the interaction engine

To exploit our Statechart modelling, we chose two solutions: SCXML and CKS. These two solutions provide functionalities to perform the Statechart and to interact with it from external programmes.

#### 4.3.1 Solution with SCXML

SCXML, or the “State Chart extensible Markup Language” (W3C, 2009), provides a generic state-machine-based execution environment based on XML and Harel Statecharts (Harel, 1987). With SCXML, a Statechart is translated into an XML file. To be used, this SCXML file must be parsed by an external programme (in Java for example) that will interact with this file. For that, some tools exist. Apache (Apache, 2009) for instance provides Java classes to parse,
manage and interact with SCXML files. Figure 14 shows a snippet code of our modelling in SCXML. We wrote Java code to exploit this SCXML file as the interaction engine as explained previously.

This solution has pros and cons:

- **Pros:** (1) it is very easy to translate a Statechart into an SCXML file. All major Statechart concepts currently exist in SCXML; (2) with SCXML, one can easily separate Statechart logic and concepts of their performing. For instance, if a transition must be changed, one has only to change one line in the SCXML file without modifying the Java code.

- **Cons:** (1) translating a large Statechart produces a lot of XML lines that are difficult to decode for a programmer; (2) modifying a Statechart implies to rewrite (or regenerate) the SCXML file; (3) to totally exploit SCXML, we are obliged to write programming code (Java, Python, C#…), and N states in the Statechart involve to write at least N methods even if there is nothing to do inside these states. So, if we need to modify the Statechart, we are obliged to modify the programming code too.

**Figure 14** Snippet code of the SCXML file

```
<scxml xmlns="http://www.w3.org/2005/07/scxml"
xmlns:cs="http://commons.apache.org/scxml"
initialstate="domotouch" version="1.0">
  <parallel id="domotouch">
    <parallel id="actions">
      <state id="state_action">
        <initial>
          <transition target="IdleAction"></transition>
        </initial>
        <state id="idleAction">
          <transition event="open" target="actionName"/>
          <transition event="close" target="actionName"/>
        </state>
        <state final="true" id="actionName"></state>
      </state>
      <state id="state_actionParam">
        <initial>
          <transition target="IdleActionParam"></transition>
        </initial>
        <state id="idleActionParam">
          <transition event="a little" target="actionParam"/>
          <transition event="slowly" target="actionParam"/>
        </state>
        <state final="true" id="actionParam"></state>
      </state>
      <parallel id="locations">
        <parallel id="objects">
          <transition cond="messageComplete==true"
            target="actions"/>
          <transition event="t" target="domotouch"/>
        </parallel>
      </parallel>
    </parallel>
  </parallel>
</scxml>
```

4.3.2 **Solution with Common Knowledge Studio**

As said previously, CKS (ObjectConnections, 2009) is an IDE allowing representing, testing and automating business rules and application logic. This is partially represented through workflows that can be executed inside CKS or from external programmes. Figure 15 shows one of our first modelling of the overall Statechart (the complete modelling is too big to be presented in this paper, but the modelling is similar). The main information on this figure is the arc connections, which represent the words that can be said to the system (open, close, camera, oven…).

As SCXML, this solution has pros and cons:

- **Pros:** (1) workflows can be directly drawn inside CKS, which facilitates their reading by programmers; (2) workflows can be generated and performed by external programmes; (3) modifying a transition in the Statechart implies only modifying an arc connection in the workflow. This can be made automatically by an external programme.

- **Cons:** CKS does not support directly all Statechart concepts. The current version obliged us to insert ‘artificial’ states (i.e., states that do not exist in the original Statechart) to obtain a workflow working like the Statechart. Consequently, it is a little bit difficult to read the resulting workflow compared with the original Statechart. This difficulty is the same to generate the workflow.

**Figure 15** A simplified translation of our Statechart in Common Knowledge Studio (see online version for colours)

5 **Conclusion**

In this paper, we presented solutions to communicate with a computing environment like home automation. We showed that many solutions exist, such as voice, gesture and haptic interfaces. Concerning the usage of voice, static, dynamic or even large vocabulary solutions are now available, as we demonstrated. We also presented our solution that uses dynamic workflows driven by external programmes. These workflows are the ‘brain’ of the resulting system because
they contain all the possibilities offered to the user. They can be written directly in the IDE or generated with our MDE approach. Our future works concern:

- The implementation of the meta-model to describe transformations to generate Statecharts from basic entities and their instances corresponding to the real objects, which are present in the house.
- Generation of SCXML files or CSK workflows from the generated Statecharts.
- Personalisation of the resulting home automation, for instance to manage habits of the user or his or her native tongue. In this last case, we will be able to control specific properties of languages like in German where verbs must be placed at the end of the sentences, or in French where adjectives are generally after names, etc.

Some of these works are already in progress. We currently are able to transform instances of the meta-model (only objects and actions for now) into CSK workflows. We can also execute directly these instances as Statecharts.

The implementation of the meta-model to describe transformations to generate Statecharts from basic entities and their instances corresponding to the real objects, which are present in the house.

The authors thank ObjectConnections, Jaxo System, bcwebcam, Spinvox and Proxem for special tools provided.

Acknowledgements

The authors thank ObjectConnections, Jaxo System, bcwebcam, Spinvox and Proxem for special tools provided.

References


Apache (2009) SCXML, URL: http://commons.apache.org/scxml/


CENA (1996) IVY Bus, URL: http://www2.tls.cena.fr/products/ivy/


