DomoBuilder: A MultiAgent Architecture for Home Automation

Andrea Addis
Dept. of Electrical and Electronical Engineering
University of Cagliari
Email: addis@diee.unica.it

Giuliano Armano
Dept. of Electrical and Electronical Engineering
University of Cagliari
Email: armano@diee.unica.it

***Abstract*—Current technologies permit people to make use of various systems able to fulfill most of their needs while being at home. However, their use is often not intuitive and they are also difficult to integrate. In this paper we propose a solution to these issues, together with a pragmatical demonstration of its effectiveness. The architectural solution we devised, called DomoBuilder, is aimed at abstracting hardware (i.e., electronic devices) and software (i.e., applications, systems), with specific emphasis on the ability of simplifying human interaction while combining heterogeneous devices within the same application. In so doing, system integration is promoted, making it easier to devise complex devices that implement new behaviors while preserving ease of use. A case study has also been devised and implemented, which highlights the great potential of the DomoBuilder architecture.**

I. Introduction

The new technological era has given birth to more and more powerful automatic devices for home automation, providing a huge amount of innovative devices and services. However, such systems are often not intuitive, or impossible to use, without a training phase (e.g., using a TV set typically requires to access decoders, recorders, remote controls, and configuration panels). Even a simple MP3 reader typically requires reading a short manual before beign able to use it.

Human beings feel more comfortable with describing an object throughout its *properties*; in fact, problems that occur with new devices and their use are often due to an imperfect or missing description of the corresponding properties.

In principle a property can be read, set, or modified. However, properties can be different in their meaning and usage. For instance, some are required to describe the internal state of a device without the need of exporting them to the user (e.g., different parameters can be checked by a car for security purposes, but the driver does not need to know them), some are made available to the user in "read-only" mode (e.g., the temperature shown by a thermostat), and others are directly changeable by the user (e.g., the state of a lamp).

In the Object Oriented paradigm, the scope of a property can be controlled with access modifiers. The visual development approach, in which complex components export just the features required for their usage, uses the so-called *Properties, Methods, Events* (PME) model[9]. In PME, properties describe a component, methods allow to use it (modifying properties and/or its state), and events allow the system to be informed about occurring changes. Properties can be made read-only to protect them by accidental access, while methods can be used to change them, or a collection of them, including the state of the component in hand.

The simplest way to describe a device consists of exporting and clearly depicting the set of properties deemed useful for the final user [7]. Encapsulating and combining those properties allows to build new interfaces and components, so that the system can be enriched with more powerful functionalities.

From a software engineering point of view, the Agent-Oriented Software Engineering (AOSE) paradigm [13] allows to embed heterogeneous devices in the same architecture, also providing useful supports for communication, persistence, pro-activeness, and mobility. In particular, there are many evidences of the advantages in using MultiAgent Systems (MAS) for Home Automation (aka Domotics) and Ambient Intelligence (AmI).

Sánchez et al. [11] point out that AmI investigates ubiquitous computer-based services, based on a variety of objects and devices, so that their intelligent and intuitive interfaces act as mediators through which people can interact with the ambient environment. The authors also highlight that research in context-aware systems has been moving towards reusable and adaptable architectures for managing more advanced human-computer interfaces.

Acampora and Loia [1] highlight the capability of AmI to deal with a new world, where computing devices are spread everywhere to improve the quality of the interaction between human beings and information technology and to put together a dynamic computational ecosystem capable of satisfying the user requirements. They show how the design of AmI systems depends upon psychological- and social-science aspects, able to describe and analyze the status of a human being during the process of decision making performed by a system.

Also device coordination during the execution of services in AmI systems is of paramount importance, as focused in [6], where planning capabilities for MAS in the AmI context are analyzed.

According to Bergenti and Poggi [3], a real load of works presents the usefulness of the AmI in the health care context. These applications can take outstanding advantage of the intrinsic characteristics of MAS thanks to notable features that most healthcare applications share: (i) they are composed of loosely coupled (complex) systems; (ii) they are realized in terms of heterogeneous components and legacy systems; (iii) they dynamically manage distributed data and resources; and (iv) they are often accessed by remote users in (synchronous) collaboration.

Further approaches for service-oriented architectures [12], independent from a specific environment [5], or aggregators of technologies, for domestic health purposes [8], are also proposed in the literature.

This work presents DomoBuilder, a multiagent architecture for home automation, together with a case study aimed at showing how a complex system for AmI purposes can be easily built with it.

A major issue with this work is to provide really simple human interfaces. In fact, in the 21th century, human beings are expected to communicate with devices and systems mimicking the way they communicate with each other, e.g., throughout natural speech and/or gestures. People that are clueless when it comes to computer technology should be taken as target, our goal being to make the system usable also to people that are not familiar even to simple operations like selecting, dragging and clicking items with a mouse. For this reason, DomoBuilder represents every single component of the system by its properties and shows in natural language a description of the operations it can perform.

The rest of the paper is organized as follows: Section II illustrates the DomoBuilder architecture and Section III describes its "internals". The case study is then presented in Section IV. Conclusions and future work (Section V) end the paper.

II. DomoBuilder Building Blocks

DomoBuilder is a multiagent architecture for home automation, which promotes the abstraction of any software or hardware device. Furthermore, it allows to centralize their control and to make uniform their interface, so that interacting with them becomes very easy, also thanks to the integration of special devices explicitly devised for handling human interfaces. According to [10], in DomoBuilder each device is intended as the building block of a system –i.e., a resource or tool that has describable properties and can encapsulate some kind of functions. Moreover, thanks to a centralized control, it is possible to connect devices for building complex systems.

A. Devices

Let us take as examples of device a light bulb and an mp3 player. The former has one property that describes its state (i.e., on/off), and a method that allows to turn on and off (i.e., toggle) the light, whereas the latter can: (i) store the track loaded, (ii) store its state (i.e., it is playing/stopped/paused), (iii) play a particular track, (iv) search for a track, and (v) trigger an event –e.g., when a track or a playlist is finished. Other examples of common devices used in Home Automation are a thermostat, a sensor, a microwave, and a washing-machine.

The ability of mashing up heterogeneous devices and of combining their functionalities permits to give rise to complex systems. Let us consider some trivial –though clarifying– examples that highlight this concept: (1) "when the room temperature is low and the sensor detects a movement, turn on the heat pump" and (2) "when the current mp3 playlist has been played, turn lights on". To this end, DomoBuilder provides a support for defining devices, for combining events, and for triggering method calls from a device to another. As a result, devices can be connected together to implement new behaviors.

B. Interfaces

Interfaces can be seen as a particular kind of device, aimed at permitting the user to interact with the system. Due to a reflective behavior performed on top of devices, interfaces in DomoBuilder can dynamically retrieve information about features of other devices. Furthermore, each device can store visual information about its appearance and its position in the environment, thus permitting to build a 3D-like visualization of the structure of a system in terms of its embedded devices.

This approach makes it also easier to ask a device how it can be useful for our purposes. [1]

C. Agents

Since every device hosted by DomoBuilder must be autonomous and must natively integrate social abilities to interact with the other devices, the AOSE paradigm has been adopted, as Domobuilder has been built upon the agent framework JADE [2].

In particular, DomoBuilder devices are in fact Jade agents. To define a specific kind of device one should extend the *Device* class of DomoBuilder, setting name, description and properties. This is the only information required to allow the user and (instances of) other devices to communicate with it (i.e., with an instance of the device being defined).

A special agent called *Kernel* (see section III-C) has been defined –aimed at controlling the life cycle of devices and at handling system events. In fact, according to[4], it is really useful to delegate a particular kind of agent to centralize the knowledge about the network of components embedded by a system. A GUI called *Panel* is also available when a DomoBuilder system starts, together with a *Clock* device useful to temporize actions. In DomoBuilder, devices can be added and activated at run-time. The Kernel handles the life on the system and its persistence.

D. Containers

In general, a device wraps and controls the corresponding hardware or software device. It can communicate within the

[1] See for example http://www.alice.org/. Alice is an innovative 3D programming environment that makes it easy to create an animation for telling a story, playing an interactive game, or a video to share on the web. Alice is a teaching tool for introductory computing. It uses 3D graphics and a drag-and-drop interface to facilitate a more engaging, less frustrating first programming experience.

system and move across *JADE containers*. The containers belonging to the main platform are two: (i) the "Core" container, which hosts the Kernel, and (ii) the "Devices" container, which hosts the *Panel* and other devices. When additional machines want to connect to the system, they must connect their own JADE container to the main platform[2] (see Figure 1).

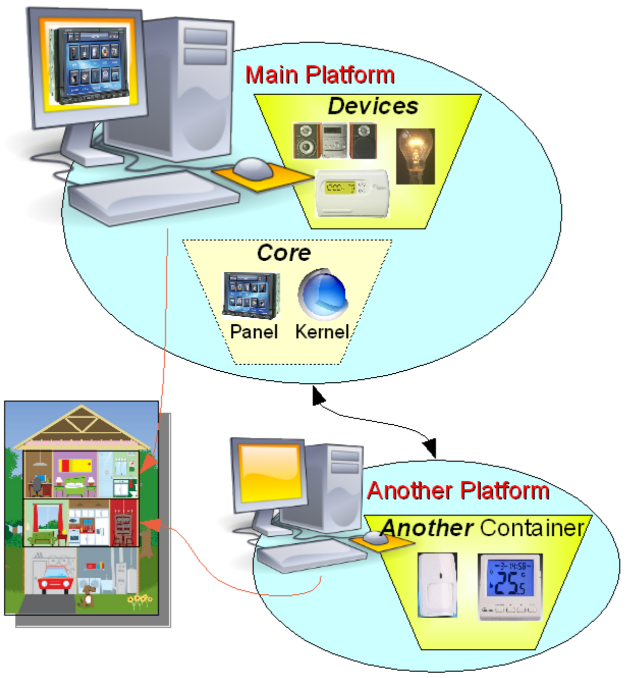


Fig. 1. The system at a glance

E. General-Purpose Transducers

As DomoBuilder is expected to deal with heterogeneous hardware devices, a general-purpose transducer, called *Morettillo*, has been devised and implemented (see Figure 2). Morettillo is a small plug-and-play hardware device equipped with radio switches, which allows to interact with up to 5 output (wired or wireless) and 7 input channels. It is endowed with a transmission channel compatible with switches working on radio frequencies 433.92MHz x 1000W max, which are commonly used and not expensive (about 7 Euros x plug). Connected via USB and auto-powered, Morettillo is automatically recognized by MS Windows.

III. DomoBuilder Internals

At the first start, a default configuration is created, which includes a Panel and a Clock device. The former is a GUI that allows to interact with the system, whereas the latter is useful to create temporized rules (e.g., alarms or time events).

A. Communication

Since devices are JADE agents, it is possible to communicate with them using ACL performatives according to a given ontology.

[2] A batch file is provided to quickly and easily start a new platform.

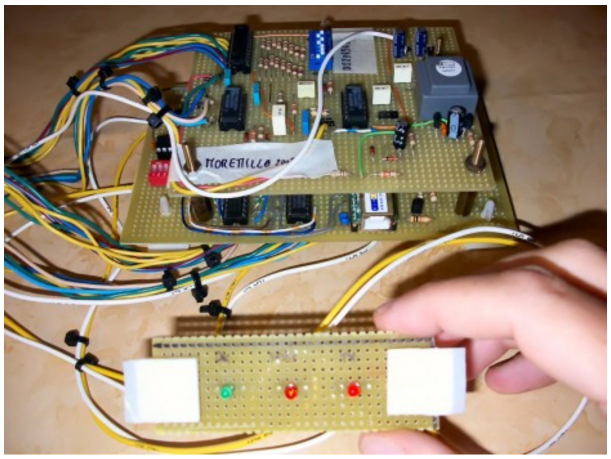


Fig. 2. Morettillo, a panel for controlling electronic devices

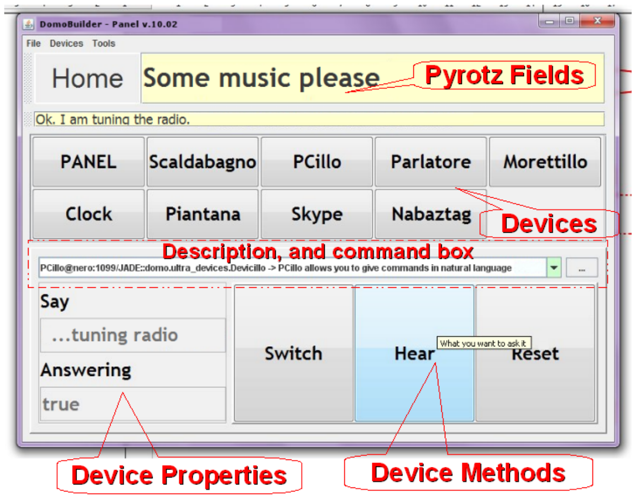


Fig. 3. The Panel Device

The Panel device (see Figure 3) is a very simple graphical user interface, which allow to easily communicate with the other devices, hiding the technical details concerning communication.

B. Events

Events are handled by the Kernel throughout an instance of the class *EventHandler*, which is contained in *UltraUnits*, a general purpose library that has been implemented to provide a suite of tools for handling common data structures, operating system components, data sources, and multimedia. An instance of the *EventHandler* class is generated according to the singleton pattern, yielding the *eventHandler* object, which is used to dispatch asynchronous events if given conditions are reached. A *condition* represents the achieved state of one or more variables. When a condition is reached a corresponding *action* is undertaken. It is also possible to define *time constraints*, and the number of *times* an action can be executed.

As an example let us assume that one wants to create the following rule: "when the thermostat measures less than 18 degrees, the Panel must show the new temperature provided that the light is on. Activate this rule from December 20^{th}, 2010, at 8pm and only on Monday" we must write:

```
<id>Thermostat/Light Rule Example</id>
<conditions>Thermostat.Temperature<<18<&& /> Lamp.light==ON
</conditions>
<action>PANEL SHOW The new temperature is %value%.
</action>
<dateconstraint>START 2010-12-20,DAY_OF_WEEK 2
</dateconstraint>
<times>-1
</times>
```

C. Kernel

The Kernel is aimed at managing (i) the life cycle of the devices populating the system and (ii) the occurring events. In order to do this, the Kernel uses the following commands:

1) Commands to Manage Devices:

- `KERNEL_DEVICE_ADD`: adds a device to the system defining its name (e.g., *Thermostat*), according to the given class (e.g., *domo.devs.Thermostat*), and the name of an existing container (e.g., *Devices*). On failure (e.g., when the container does not exist) a `KERNEL_DEVICE_ERROR` command is issued.
- `KERNEL_DEVICE_REMOVE`: removes a device identified by its name.
- `KERNEL_DEVICE_ADDED`: informs the Kernel that a device has been correctly created. Then the `KERNEL_EVENT_TRIGGERED` command is issued (e.g., `KERNEL.KERNEL_DEVICE_ADDED == Thermostat`).
- `KERNEL_DEVICE_ERROR`: informs the Kernel that an error has been triggered by a device.

2) Commands to Handle Events:

- `KERNEL_EVENT_ADD`: adds an event defining its properties;
- `KERNEL_EVENT_REMOVE`: removes an event identified by its id;
- `KERNEL_EVENT_TRIGGERED`: informs the Kernel that an event has been triggered (i.e., a device changed its state).

D. Devices

As already pointed out, devices are agents wrapping hardware devices into DomoBuilder. Their description is set during their development by calling the methods:

putDeviceDescription(String deviceDescription)
putDeviceDescription(String deviceDescription,
boolean deviceVisible,
BufferedImage deviceImage,
Point3D devicePosition)

where:

- *deviceDescription*: a string describing the device;
- *deviceVisible* (optional): a boolean value indicating whether the device is visible in the control panel. In fact, a device can be removed from the Panel if not used by a user (e.g., a user may not need to interact with a speech engine, whereas it can be used by other devices to communicate with the user);
- *devicePosition* (optional): the physical position of a device in the environment;
- *deviceImage* (optional): the image used to represent the device.

The properties of a device are identified by their name and can store a string value, optionally with the type of the value. They can be set calling the method:

putDeviceProperty (String name, String description,
String type, String format,
String initialValue)

where:

- *name*: the name of the property.
- *description*: a string that describes the property, in natural language;
- *type*: the type of the property (it must be a Java class name);
- *format*: the property in an EBNF-like format;
- *initialValue*: a string representing the initial value (it can be empty).

The methods applicable to a device are identified by their name, the parameters of a method (if any) being embedded into a single string (multiple parameters can also be represented in XML format). A method can be attached to a device as follows:

putDeviceMethod (String name, String description,
String type, String format)

where:

- *name*: the name of the method;
- *description*: the description of the method in natural language;
- *type*: the type of the property, i.e., the name of a Java class;
- *format*: the property in an EBNF-like format.

A device can handle the following commands:

- `DEVICE_METHOD`: executes a method with its parameter;
- `DEVICE_MOVE`: moves the device to another container;
- `DEVICE_DIE`: turns off and remove a device from the list of devices.

E. Requesting Information

Requests on different topics can be made to the Kernel or to a device by sending the message "INFORMATION_REQUEST". The corresponding reply will be given throughout the message "INFORMATION_INFORM".

The list of topics, with the corresponding parameters and answers, follows:

- `TOPIC_DESCRIPTION`: requests the description to a device (`parameters` = *none*, `answer` = *the description of the device in text format*);
- `TOPIC_PROPERTY`: requests a specific property to a device (`parameters` = *the name of the property*, `answer` = *the name of the property and its value*);

- `TOPIC_EVENTS`: requests the list of the events to the Kernel (`parameters` = *none*, `answer` = *the system events in text format*).
- `TOPIC_DEVICELIST`: requests the list of the device names to the Kernel (`parameters` = *none*, `answer` = *the name of the devices in text format, one for each line*).

To give the reader the flavor of how an information request is issued, let us consider the following examples:

> Kernel INFORMATION_REQUEST TOPIC_EVENTS

> Thermostat INFORMATION_REQUEST
TOPIC_PROPERTY Temperature

The answer is issued by the device to which the request has been addressed throughout the overridden method:

onAnswer(String senderLocalName, String topic, String answer).

IV. Case study

The case study (called DomoPro) developed in the field of home automation consists of:

- Panel: A GUI for the system
- HiFi: An mp3 Reader
- Clock: A system clock with alarm
- Pyrotz: A device that can understand commands in natural language
- Movimentio: A movement sensor that exploit a computer webcam
- Nabaztag: A device to control the Nabaztag Tag[3], a Wi-Fi enabled ambient electronic device in the shape of a rabbit
- Skype: A device that interfaces the system with the Skype messenger
- Talker: A text to speech synthesizer
- MailReader: A device able to read e-mails
- Morettillo: A device that controls the homonym general-purpose hardware device
- Piantana: A device that controls a lamp
- Scaldabagno: A device that controls a house water heater

Thanks to the portability of DomoBuilder and to the plug-and-play capability of Morettillo, a new installation of the above set of useful home automation functionalities can be performed in less than 10 minutes.

A. Starting DomoPro

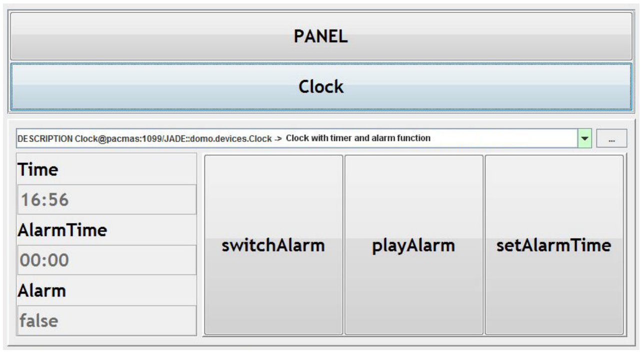
When DomoBuilder starts for the first time, a new configuration file (i.e., *domo.xml*) is created; then the Kernel, the Panel, and the Clock start running. The properties and methods of these devices can be easily inspected throughout the graphical interface of the Panel (see Figure 4). In order to do this, the user must click the button with the name of the device to inspect. Upon clicking, the properties of the device, together with the corresponding values, will be listed on the left, whereas the method buttons will appear on the right side of the Panel.

[3] See Nabaztag on Wikipedia.

B. Building a New Device

To create a device, one must extend the *domo.Device* DomoBuilder class. To describe the device, together with its properties and methods, the *putDeviceDescription*, *putDeviceProperty*, *putDeviceMethod* must be called as described in section III-D.

For the sake of simplicity, let us take a look at the Clock device, which is actually already available in DomoBuilder. As we can see from Figure 4, the final device ought to have three properties and three methods. An example of code follows:


DESCRIPTION Clock@pacmas:1099/JADE::domo.devices.Clock -> Clock with timer and alarm function

Time: 16:56
AlarmTime: 00:00
Alarm: false

Fig. 4. The Panel shows the Clock properties

```
public Clock() {
    putDeviceDescription("A timer with alarm");
    [...]
    putDeviceProperty("Time", "The Actual Time", ...
    [...]
    putDeviceMethod("setAlarmTime", "Set the Alarm Time"', ...
    [...]

    timer.start(999);
}
```

Note that the device timer has been instantiated to run an action each second; i.e., every second the overridden method *onTimer()* will be called. In this particular case, the onTimer() method will (i) update the time of the clock and (ii) check whether the alarm sound has to be played.

At this point, the system knows the description of this device, its readable properties, and its callable methods. To better illustrate what happens when a user or a device in the system calls a method, let us report the following Java code, which overrides the default *onMethod()* method:

```
@Override
public void onMethod(String name, String value) {
    if (name.equals("setAlarmTime")) {
        // Example of changing a property
        set("AlarmTime", value);
      }
    if (name.equals("playAlarm")) {
        // Example of calling an inner method
        soundAlarm();
    }
    [...]
}
```

In so doing, our Clock device is ready.

C. Adding Further Devices to DomoPro

The Kernel can be asked at any time to add further devices. For the sake of brevity, let us describe how to add the Morettillo device.

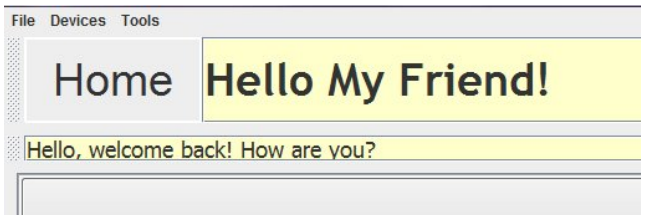
The following command must be sent to the Kernel through the Panel command box[4]:

```
KERNEL KERNEL_DEVICE_ADD
<name>Morettillo</name>
<class>domo.ultra_devices.Morettillo</class>
<container>Devices</container>
```

After that, the Morettillo button will appear on the panel, close to the buttons of other devices. Please note that this operation could be made completely automatic, being the device recognized and activated as soon as its class (e.g., jar file) is added to the program folder.

D. Analysing Some Relevant Devices of DomoPro

1) Pyrotz: We prototyped Pyrotz, a device which permits to communicate with the system using natural language. We can communicate with Pyrotz by sending the command message `DEVICE_METHOD device.user Hear Hello my friend!`[5] as well as writing *Hello my Friend!* in the big yellow text field on the Panel. It will write the answer in the small yellow panel below (see Figure 5).



Hello My Friend!

Hello, welcome back! How are you?

Fig. 5. Saying hello to Pyrotz

Pyrotz can understand natural language, so, it is able to translate sentences like: “turn on the light”, “please could you turn the lights on?”, “it is too dark, make light”, with, `Piantana DEVICE_COMMAND Switch ON`.

Thanks to the event-triggering mechamism, when Pyrotz says something the Talker will speech it.

2) Skype: The system can be interfaced with the Web by wrapping any kind of Internet application. For instance, thanks to the Skype device, we can (i) directly send commands to DomoPro or (ii) chat with Pyrotz. Using an external voice recognition software it is also possible to talk to DomoPro.

In principle, everybody can talk and receive answers from Pyrotz, even if for obvious security reasons, only selected Skype users are allowed to send command to the house in which the system has been installed.

[4] Opening the command box, a list of command templates is given.

[5] The parameter device.user identifies the sender, e.g., domo.Panel. This prevents other devices to communicate with it; “Hear” is the method name, and the rest is the parameter of the command, or the sentence in this case

E. Adding Events to DomoPro

All DomoPro devices can be used in isolation. For instance, the talker could be asked to say “Hello”, the house lights can be turned on and off (even from an Internet computer connected with Skype). More interesting is when DomoPro automatically undertakes decisions and executes complex behaviors (i.e., behaviors that involve different devices). Some examples of complex behaviors follows:

1) When Pyrotz utters or answers something, should Talker synthesize it.
2) When somebody wakes up in the morning, turn on the mp3 reader
3) When somebody gets back home in the afternoon, read the emails
4) When somebody leaves the room, turn off the lights
5) If somebody touches the ears of the Nabaztag, ask him to stop doing it

V. Conclusions and Future Work

In this paper we proposed and illustrated DomoBuilder, an architectural solution aimed at abstracting hardware and software, with specific emphasis on the ability of simplifying human interaction while combining heterogeneous devices within the same application. A case study has also been illustrated, which highlights the great potential of the DomoBuilder architecture. With DomoBuilder, it is very easy to devise, implement, and install easy-to-use and low-cost systems for Home Automation, also thanks to the uderlying multiagent architecture that facilitates the integration of, and the interaction among, heterogenous devices.

Acknowledgments

We would like to thank Marco Lai (aka Moretti) for his help in developing the Morettillo hardware device and Eloisa Vargiu for her useful suggestions about the DomoBuilder architecture.

References

[1] G. Acampora and V. Loia. A dynamical cognitive multi-agent system for enhancing ambient intelligence scenarios. In *FUZZ-IEEE'09: Proceedings of the 18th international conference on Fuzzy Systems*, pages 770–777, Piscataway, NJ, USA, 2009. IEEE Press.

[2] F. Bellifemine, A. Poggi, and G. Rimassa. Developing multi-agent systems with JADE. In *ATAL '00: Proceedings of the 7th International Workshop on Intelligent Agents VII. Agent Theories Architectures and Languages*, pages 89–103, London, UK, 2001. Springer-Verlag.

[3] F. Bergenti and A. Poggi. Multi-agent systems for e-health: Recent projects and initiatives. In *10th Workshop dagli Oggetti agli Agenti (WOA 2009)*, 2009.

[4] W.-H. Chen and W.-S. Tseng. A novel multi-agent framework for the design of home automation. *Information Technology: New Generations, Third International Conference on*, 0:277–281, 2007.

[5] G. Fiol, D. Arellano, F. J. Perales, P. Bassa, and M. Zanlongo. The intelligent butler: a virtual agent for disabled and elderly people assistance. *International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008)*, 2009.

[6] N. Gatti, F. Amigoni, and M. Rolando. Multiagent technology solutions for planning in ambient intelligence. In *WI-IAT '08: Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, pages 286–289, Washington, DC, USA, 2008. IEEE Computer Society.

[7] G. G. Márquez. *Cien años de soledad*. 1967.

[8] C. Munoz, D. Arellano, F. J. Perales, and G. Fontanet. Perceptual and intelligent domotics system for disabled people. In *Proceedings of the Sixth IASTED International Conference*, 2006.

[9] K. Reisdorph and H. Ken. *Borland C++ Builder*. Apogeo, 1997.

[10] A. Ricci, M. Viroli, and A. Omicini. Give agents their artifacts: the A&A approach for engineering working environments in MAS. In E. H. Durfee, M. Yokoo, M. N. Huhns, and O. Shehory, editors, *AAMAS, 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007), Honolulu, Hawaii, USA, May 14-18, 2007*, page 150. IFAAMAS, 2007.

[11] N. Sanchez, E. Mangina, J. Carbs, and J. M. Molina. Multi-agent system (mas) applications in ambient intelligence (ami) environments. *Trends in Practical Applications of Agents and Multiagent Systems*, 2010.

[12] N. I. Spanoudakis and P. Moraitis. An ambient intelligence application integrating agent and service-oriented technologies. In *SGAI Conf.*, pages 393–398, 2007.

[13] F. Zambonelli and A. Omicini. Challenges and research directions in agent-oriented software engineering. *Journal of Autonomous Agents and Multiagent Systems*, 9:253–283, 2004.