Domotic House Gateway

Paolo Pellegrino  
Politecnico di Torino  
C.so Duca degli Abruzzi, 24  
10129 Torino, Italy  
+390112276108  
paolo.pellegrino@polito.it

Dario Bonino  
Politecnico di Torino  
C.so Duca degli Abruzzi, 24  
10129 Torino, Italy  
+390112276107  
dario.bonino@polito.it

Fulvio Corno  
Politecnico di Torino  
C.so Duca degli Abruzzi, 24  
10129 Torino, Italy  
+390112276105  
fulvio.corno@polito.it

ABSTRACT
This paper presents a domotic house gateway capable of seamlessly interacting with different devices from heterogeneous domotic systems and appliances. Such a gateway also provides the possibility to automate device cooperation through an embedded rule-based engine, which can be dynamically and automatically updated to accommodate necessities and anticipate users’ actions. Some practical applications will show the effectiveness of the system.

Categories and Subject Descriptors
D.2.10 [Software Engineering]: Design;
D.2.11 [Software Engineering]: Software Architectures – domain-specific architectures;
D.2.12 [Software Engineering]: Interoperability.

General Terms
Management, Design, Experimentation, Human Factors.

Keywords
Domotics, Ambient Intelligence, Ubiquitous computing, House, Gateway, Rules.

1. INTRODUCTION
In the last few years ubiquitous computing has grown in popularity. The recent advances and integration of computing technologies in everyday life activities push the desire for even quicker and seamless interaction with people and resources. From mobile phones, to vehicle GPS navigators, from digital television to domestic appliances, the information technologies are silently pervading our daily routines.

In this context, part of the research interest is focused on ambient intelligence and domotic systems, where, even if at high costs, well established solutions can be easily found. Here, however, interoperability and communication between devices play a fundamental role, yet currently evolving from its early stages. Nowadays, different standards are being used to connect devices, either intra- or inter-domotic systems, via wireless communication or through cables. Although a single standard would be strongly desirable, we should rather get used to the coexistence of multiple communication interfaces, as each of them might uniquely meet specific demands. This is, in effect, a shared point of view in most of the domotic-related systems, where device connectivity is still emerging with the diffusion of mobile and smart devices, but little is being done for connectivity between domotic systems. So far, the OSGi Alliance [16], which already involves numerous partners, provides the most complete open source solution to generic interconnectivity through a Service Platform for networked devices. However, besides communication issues, intelligent and articulate approaches to automatic device management are also under study, in addition to the simpler specific solutions for single domotic systems. It would be interesting if a household environment were able of understanding the habits of its hosts, so forecasting actions or even dangerous situations, without limits to the exploitation of the available technologies [8].

Therefore, this paper presents a framework capable of seamlessly and easily integrating heterogeneous devices in a domotic system, so operating as a gateway for information exchange between devices. In addition to providing intercommunication channels, the authors’ framework also offers the possibility to automate the device cooperation and activity through an embedded rule-based engine. The events generated by the various devices can be logged and used either to trigger existing rules, e.g. for the reproduction of recorded events, or to automatically infer new rules, compatibly with the existing rules and situations.

To demonstrate the sustainability of the approach some devices and applications will be presented, with a particular contribution in the context of assistive technologies.

This paper is organized as follows: Section 2 provides an overview of existing domotic system and ambient intelligence solutions. Section 3 introduces the design principles adopted in this work with a description of its general architecture. Section 4 shows the testing environment and interfaces. Section 5 presents some experimental results and eventually Section 6 draws conclusions and proposes future works.

2. RELATED WORKS
In the context of domotics, solutions to device connectivity are driven by local commercial leaders. North America, Europe and Japan are three main areas oriented to different wiring technologies and protocols, and the picture gets even more confused when looking at Nation-wide level.
However, three main approaches can be identified: the installation of a specific and separate wired network is a common approach and is either based on proprietary solutions (EIB/KNX [5], LonWorks [13], etc.) or on more diffuse technologies (e.g., Ethernet, Firewire), the latter being preferred for computer related networks and wide-bandwidth requirements. On top of these, various protocols are implemented (X10, CEBus, BatiBus, HBS, just to name a few) and none of them has yet prevailed on a global scale. The Konnex open standard [12], involving EIB, EHS and BatiBus, is one of the major initiatives in Europe aiming at global interoperability. Another common approach is based on the reuse of the existing wirings, such as power or phone lines (the former being more frequently used as a carrier, as in EHS, X10, Homeplug). However, these solutions generally present higher noise levels than dedicated wirings and are therefore less versatile. Lastly, wireless technologies are also becoming more and more attractive, and mostly based on either infrared technology, or radio link (e.g. IEEE802.11b). The latter is usually adopted for guaranteeing connectivity at higher distances. As each of these alternatives is better suited in a given context, they are still far from convergence into a unique solution, even because of strong commercial influences. Instead, in a home one can frequently find different technologies, such and Ethernet for multimedia connectivity and Bluetooth for personal connectivity, while special buses reliably allow home automation tasks. However, as processing complexity is increasingly coming at lower costs, technologies that are well established for common PCs are being transferred to numerous devices. It is not therefore surprising that simple computer systems are being used to bridge this interconnection gap, especially for what concerns ambient intelligence. In this context, most of the efforts are divided into limited domains: each domotic system vendor usually proposes intelligent approaches to the management of the supported devices, and often remote control is also offered through the Internet or with a phone call. However, the intelligence provided to the system is generally basic, unless enhanced using a more complex and versatile device, such as a computer. In effect, some programming software can be found, but is generally only oriented to the control of one specific technology, and offer little support to ambient intelligence techniques (ETS [6], commercial; EIB/KNX [15], open source).

Furthermore, smart devices or information appliances (like PDAs, set top boxes, etc.) are not usually produced by domotic system vendors, and are only recently facing the intercommunication and standardization issues related to domotic systems [7]. It becomes therefore desirable a “neutral” device capable of interfacing all these parts. In this context, the OSGi Alliance has already proposed a rather complete and complex specification for a residential gateway (and alike). Here, protocol and integration issues, as well as intelligence related behaviors, are demanded to third-party “bundles”, that can be dynamically plugged into the OSGi framework at run-time. An additional effort is therefore required to enhance the system with a compact yet general approach to intelligent and automatic interoperability among the involved devices.

Other researches, in the context of ambient intelligence, include the GAS (Gadgetware architectural Style) project, which proposes a general architecture supporting peer-to-peer networking among computationally enabled everyday objects [11]; the event-based iRoom Operating System (iROS) [10], which is mainly focused on communications about devices in a room, and does not take in particular consideration the possibility of a general and proactive interaction of the environment with the user. The one.world project [8] offers a framework for building pervasive applications in changing environments, while in this paper the environment configuration is not expected to change frequently. Other researchers propose a fuzzy and distributed approach to device interaction, by exploiting a special rule-based system [1]: special purpose agents mutually collect, process and exchange information, although at the expense of higher complexity, especially in terms of adaptation of device operations to the agent system requirements.

3. ARCHITECTURE

In this section an overview of the Domotic House Gateway (DHG) is presented, followed by a more detailed description of its components. The proposed system has been designed with the intent of supporting intelligent information exchange among heterogeneous domotic systems which would not otherwise natively cooperate in the same household environment. The concept of event is used to exchange messages between a device and the DHG. As we will see, these low-level events are converted to logical events internally to the DHG so as to clearly separate the actual physical issues from the semantics that lies beyond the devices and their role in the house. In this way, it is also possible to abstract a coherent and human-comprehensible view of the household environment.

The general architecture of the DHG is deployed as shown in Figure 1, and can be divided into three main layers: the topmost layer involves all the devices that can be connected to the DHG and their software drivers, which adapt the device technologies to the gateway requirements. The central layer is the main responsible for routing low-level event messages to and from the various devices and the DHG, and also includes the generation of special system events to guarantee platform stability.
The last layer is the actual core of intelligence of the system. It is devoted to event management at logical (or semantic) level and to this purpose it includes a rule-based core which can be dynamically adjusted either by the system itself or manually through external interfaces. The rules define the expected reactions to incoming events, which can be either generated by the house, let’s say the “door is opening” as an example, or by an application: “open the door”.

Complex scenarios may involve the rule engine: for instance, a rule might generate a “switch on the light in room x” event if two events occur: the room x is dark and a sensor revealed the presence of somebody in that room. Additionally, an automatic rule learning algorithm is under study: the logged events are processed to infer common event patterns that are very likely to be repeated in the future. New rules are firstly proposed to the households and then possibly accepted and added to the existing rule set. Some interfaces permit to check, modify, add or delete each of the rules. Different rule sets may eventually be used. In the next sections, a detailed description is presented for each of the rules. Different rule sets may eventually be used.

### 3.1 Application Interfaces, Hardware and Appliances

In a domotic house, a person interacts with many devices that may be connected to the house gateway, from a simple light switch to a set top box, from a mobile phone to a computer. The type of connection as well as the configuration for each device may vary and often depends on it: low-cost wired domotic buses are well suited for controlling simple devices and smart appliances; wireless connections facilitate a continuous intercommunication between different locations but may be disturbed and not always reliable. Ethernet connections offer a high bandwidth which can be used to transmit videos, etc. Therefore, in order to uniformly control all the devices it necessary to offer a common point of aggregation, the DHG. Domotic systems usually include a control device which permits to manage all the devices connected to the domotic bus. In this case this single device can be connected to the DHG. Other recent smart appliances and devices (digital TVs, handhelds, etc.) can be accessed by modern domotic installations, but are often also accessible via computer oriented connections, as they generally embed simple computer systems. In general, any device may communicate with the DHG using its preferred protocol, as the information exchange is handled by a specific driver for each type of device. So, a web application running on a common PC could use the SOAP protocol; a simple terminal might use raw sockets, and so on.

The configuration of each device is based on a simple and generic structured model of the house (which could actually be extended to more general environments), as depicted in Figure 2.

```xml
<house>
  <room name="kitchen">
    <device name="light" devID="11" devType="Light" driver="BTicino"/>
  </room>
  <room name="living_room">
    <device name="set_tob_box" devID="192.168.1.33" devType="STB" driver="STB"/>
  </room>
</house>
```

Figure 3: Example of device configuration

Configurations are provided by XML files, possibly specifying predefined sets of rules that can enhance the use of the devices. The device drivers are dynamically plugged into the DHG in order to support the most diverse house configurations, especially if only temporary (e.g., in case of guest mobile devices) or changing from time to time (e.g., a new computer has been added).

### 3.2 Device Drivers

The device drivers in the DHG are responsible for translating low level or hardware states and activities of the devices (a light switch, a door, a software application, etc.) into events. As mentioned above, each device may need to use a specific protocol to communicate with the DHG. Therefore, it is necessary to develop specific drivers for each type of device. To this purpose, some simple guidelines are provided for the development and integration of new drivers: when plugged into to the DHG, each new driver must:

- register itself with a unique identifier;
- for each supported device:
  - if the device type (class) does not exist, register it as a subclass of an existent type (e.g., root);
  - register the new device with the associated type;
  - correctly handle (receive and possibly send) events for each device according to its type (and “supertypes”).

It should be noted that the registration of a new device type should imply the extension of an existent type (class), whose events are inherited, and may also involve the registration of new events (Figure 4).

A number of predefined types of devices are initially provided, as well as a list of supported events for each device type. This information will become part of the house knowledge base (or House Model), as explained in Section 3.5, as should facilitate the design of drivers and devices especially in terms of interoperability.

![Figure 2: Sample device instances in a House Model](image-url)
3.3 Communication Layer

The main task of the Communication Layer consists in routing low-level events to the correct destination: events coming from a device are sent to the Event Handler, whereas events from the EH must be sent to the correct device driver for further processing (e.g., to switch on the correct light). The association between devices and drivers is created whenever a new device is instantiated by its corresponding driver. Additionally, a special instance property maintains the “address” (or ID) which identifies the device within the scope of the driver. For instance, a driver controlling a domotic system may use a unique single number for each of the devices which are under its control, whereas a driver that interfaces computer applications may use IP addresses or URLs.

The Communication Layer is also responsible for the management of some driver related issues, such as loading configurations or handling possible driver errors. For instance, it may generate system events which are sent to the Event Handler for further logging or error recovery, thanks to special rules.

3.4 Event Handler

The EH translates low-level events into logical events according to the house model, and viceversa. The main task of the EH is the conversion of the device driver addressing (the instance ID) to a high-level name, which correlates the device to its function or role in the house. In this way, it is possible to hide specific device addressing details to the Domotic Intelligence System (logic or rules), which allows the DHG to autonomously and automatically perform actions on the connected devices.

The uniqueness of a device instance is guaranteed by its logical name, which includes the location (even if fictitious) and a unique name within such location, consistently with the house model.

E.g.: 
{driver “BTicino”, devID “11”, event “…”} \(\iff\) 
{room “kitchen”, devName “light”, event “…”}

3.5 House Model

As already mentioned, the house model represents, with a structured and logical scheme, the house devices and the events they support. The house environment is subdivided in a collection of rooms and for each room the corresponding devices are specified as instances of a supported device type. This representation facilitates the design and configuration of the various devices, as well as their utilization, even through the definition of scenarios, which coordinate the control of multiple devices through a single action (e.g., pressing a button).

Beside the house configuration, the types of devices are also structured in a hierarchy in order to explicitly correlate devices as specific subclasses of simpler devices: for example, a light dimmer is a specific case of light. Each device type is linked to the supported events, and these links are automatically inherited by the descendant device subtypes (or subclasses...), to guarantee that specific devices can always be controlled as a simpler ancestor. So, for instance, it should always be possible to use a dimmer as a simple light bulb, supporting “switch on” and “switch off” events (Figure 4).

The house model is (re)populated whenever a driver is plugged into the DHG, and new device types may be registered as well as new events that they may support. A minimal set of device types and of events is provided through a built-in fictitious device driver, which may be used for testing.

The house model is also used to map the existent devices, room by room, to the correct driver and to a registered device type and events according to simple XML configuration files. In addition, it may also be exploited into the Domotic Intelligence System to improve its intelligence and dependability, though at the cost of increased complexity.

3.6 Domotic Intelligence System

The DIS permits to generate new (logical) events at run-time basing either on events coming from the house through the Event Handler or on predefined or inferred “rules” that may, for instance, act at a specified time.

The current implementation of the DIS is based on a run-time engine that processes rules according to the events received by the Event Handler. When certain conditions are met, new events may be generated and sent back to the Event Handler that routes them to the correct devices through the Communication Layer. The rules can be preloaded or added either manually via external interfaces, such as a simple console, or through the Rule Miner, which examines the event log (see the Event Logger) to infer new rules. At this moment, some rule control mechanisms are being studied to prevent annoying or dangerous situations. At the very least, it is possible to save the rules and the status of the DIS at any time for future reloading.

3.7 Event Logger

The event logger receives events from the Event Handler and saves them in a file in order to facilitate the identification of possible erratic behaviors. Both logical events and system events can be logged, and through external interfaces it is possible to specify filters in order to limit the amount of stored data. The Event Logger is also used by the Rule Miner to (semi-automatically) generate rules for the Domotic Intelligence System.

3.8 Rule Miner

The Rule Miner tries to infer new rules for the Domotic Intelligence System by reading and analyzing the event log. The idea is to identify common event patterns so as to forecast and automatically generate events according to such patterns. However, this is still work in progress, as it is also necessary to keep into account dangerous situation as well as possible conflicting actions, which should actually prevent the execution of certain inferred rules. To this purpose the Rule Miner might exploit the House Model or other sources of information as knowledge base to achieve a more consistent understanding of what the event represent, and therefore of what is happening in the house.
3.9 Run-Time Engine
The Run-Time Engine receives events at run-time from the Event Handler, parses them according to a rule-based system and, if such input events meet certain conditions, it generates new events, which are then sent to the Event Handler and consequently to the Communication Layer for routing to the devices. The event processing and generation mechanisms could actually be implemented using any technique (generally related to Artificial Intelligence). However, according to the authors, techniques like artificial neural networks or genetic algorithms render rather difficult to precisely and securely control events. Therefore, as mentioned since the beginning, current implementation of the RTE is based on a rule system, nominally the open source Drools framework [4].

In this way, the status of the engine can be easily initialized by loading a configuration file. Additional configuration files or user interfaces (such as a simple command-line parser or even interfaces based on Natural Language Processing techniques) permit to modify, restore or save the RTE status at run-time in order to adjust or fix improper or erratic settings manually.

3.10 RTE User Interfaces
A number of different interfaces (a console, a Natural Language Processing interface, other graphical user interfaces) can be provided to permit the manual configuration of the Run-Time Engine so as to fully keep under user's control the Domotic Intelligence System. At now, a simple command-line interface is available, but a NLP-based interface is under study to facilitate the interaction with non expert users.

4. TESTING ENVIRONMENT
In order to test the actual feasibility of the proposed system, which has been developed in Java mainly for portability issues, a number of different devices and interfaces have been used. They are briefly presented in the following paragraphs and explained in more details in subsequent sections, followed by the explanation of the actual experimental setup.

The most relevant element is the domotic house near the authors' laboratories and it is equipped with a home automation system produced by BTicino, a leading Italian industry for house electrical equipment. The house is part of a scientific and technological park maintained by C.E.T.A.D. and dedicated to promotion, development and diffusion of technologies and innovative services for rehabilitation and social integration of elderly and disabled people.

To complete the picture of the tested devices, two additional elements are to be cited. The first one is a simple parallel port connected to eight small led and driven by a common PC running Linux. The second and last one is the MServ open source program [14], again running under Linux, capable of remotely choosing and playing music files.

Some results will be presented in Section 5 after the description of the aforementioned experimental setup.

4.1 BTicino MyHome System
The MyHome system developed by BTicino [2] is a home automation system able to provide several functionalities as requested by the increasing needs of users for smart and intelligent houses. These functionalities cover several aspects of domotics such as comfort configurations, security issues, energy saving, remote communication and control. The common framework in which every available device is deployed is based on a proprietary bus called “digital bus” that conveys messages among the connected devices and that provides them the required electrical power.

The most salient characteristic of the BTicino system is what they call the control sub-system, i.e., the ability to supervise and to manage a home by using a PC, a standard phone or a mobile phone. The control in the BTicino system can be either local through a LAN connection, as experimented in this paper, or remote through an Internet connection or a telephonic network. Through a proprietary protocol, it permits to manage all the devices of the house, e.g. lights, doors, shutters, phones, etc. This component permitted us to interface the BTicino system to our DHG by simply exploiting its features, instead of connecting each device to the DHG.

Therefore, a single driver has been prepared to handle the communication between the control and the DHG. Additionally, a number of specific modules has been provided to manage basic devices such as lights, doors, shutters, and alike.

It should also be noted that this driver must poll the control to check the status of the domotic devices and convert the returned information into events for the DHG. This is due to the fact that the BTicino system installed in our environment does not support events natively.

4.2 Parallel port and LEDs
Eight Light Emitting Diodes have been wired to the data lines of a standard IEEE 1284 parallel port connected to a PC running Linux. The driver for the DHG is based on a simple TCP/IP server that drives the parallel port as a generic parallel device using Linux-specific calls, so that the DHG is enable to control each of the eight lights. They have been used to test the rule system, as well as to demonstrate the flexibility of the proposed work. Legacy or special purpose devices are in fact still based on this simple technology (and the standard RS-232 is another example).

4.3 Music Server (MServ)
This open source Linux application is basically a music player that can be controlled remotely, therefore acting as a server. It exposes a TCP/IP connection to receive commands such as play, next song, stop, etc., and to provide its status (e.g., which song is being played, as it is normally randomly chosen). MServ is normally accessible through a simple telnet application, but also HTTP CGI and several GUI applications are available for a more easy interaction with the system.

The DHG driver for MServ is therefore rather simple even in this case. In fact, it only needs to exchange simple strings through a TCP/IP connection and to parse them appropriately (some informative message may be related to the status change caused by other connected clients, etc.). Events like play and stop are accepted as commands, while relevant status messages are sent to the DHG for logging purposes as, at now, no visual interface has been provided.

4.4 Experimental Setup
The DHG has been installed on a common PC, running a Java Virtual Machine (JVM), on an AMD 1800+ processor with 512MByte of RAM (the DHG is expected to work fine on fairly less performing PCs). Two other similar PCs have been used for the parallel port and the MServ application respectively. A simple
Ethernet switch served as physical connection for the three computers and the BTicino control server, which also exposes a standard RJ45 connector and supports TCP/IP connections. Once all the systems were ready and functional, the three drivers mentioned in the previous sections have been plugged into the DHG. The house model has been provided as a simple XML configuration file, specifying the actual device instances viewable through each of the drivers. So, for instance, each of the LEDs has been identified through a number from 0 to 7, as this is sufficient for its driver, while their type is a light, as all of them can only be switched on or off.

Some basic rules have also been added to the DHG, mainly to test the rule system and to understand if any unexpected issue may arise in practice. So, for example, whenever the entrance door is opened, a rule is activated and generates an event that makes the LED 4 to switch on. Conversely, when the door is closed, another rule sends a “switch off” event to the same LED.

5. RESULTS

Two aspects have been considered: the device drivers and the rule engine.

In the first case, all the drivers have been created and with rather little effort (a few man-hours for each of them, including debug), and the interactivity with the DHG proved to be very stable as no crashes have ever been registered.

The simple rules inserted into the system were also appropriately executed. So, for instance, the light was automatically switched on after the windows in the same room where all shuttered, the shutters being controllable domotic devices. However, particular attention is necessary when inserting rules into the system. In fact, referring to the previous example, if the shutters remain closed one must still be able to switch off the light, without causing the system to switch it on shortly thereafter. These types of issues, as well as cases regarding more critical devices such as the oven which may contain improper items, are to be considered with great care, especially when designing an automatic way to infer new rules.

6. CONCLUSIONS

This paper presented the architecture of a domotic house gateway that supports the integration of heterogeneous devices into an enhanced rule-based run-time engine. It allows the interaction of systems that would not otherwise natively cooperate, not only in terms of simple information exchange but also of intelligent and seamless collaboration with the users and the surrounding environment. In particular, special attention has been dedicated to uses of the DHG as aid for elderly or disabled people, as they may receive significant benefits by automating repetitive action on either common or specific devices. Thus, the DHG presented in this paper will also serving as a base for further studies on the automatic generation of rules, with particular focus on cases which may become critical for the users.

7. REFERENCES