Interoperability of home automation systems as a critical challenge for IoT

Vittorio Miori
Institute of Information Science and Technologies “A. Faedo” (ISTI)
National Research Council of Italy (CNR)
Pisa, Italy
vittorio.miori@isti.cnr.it

Dario Russo
Institute of Information Science and Technologies “A. Faedo” (ISTI)
National Research Council of Italy (CNR)
Pisa, Italy
dario.russo@isti.cnr.it

Luca Ferrucci
Institute of Information Science and Technologies “A. Faedo” (ISTI)
National Research Council of Italy (CNR)
Pisa, Italy
luca.ferrucci@isti.cnr.it

Abstract— **The spread of enabling technologies for the Internet of Things allows the creation of new scenarios in which home automation plays a significant role. Platforms for smart cities and communities, which must include applications for energy efficiency, health, mobility, security, etc., cannot ignore the use of data gathered directly from homes. In order to implement such scenarios, all the related technological infrastructure and home systems must be able to understand each other and exchange information. In this work, we present a new platform that achieves interoperability between heterogeneous home automation systems. It allows different, incompatible technologies to cooperate inside and outside the home, thus creating a single ecosystem. In order to achieve tins goal, specific problems need to be solved to be able to construct "bridges" between the various home automation networks currently in use. In this regard, some specific solutions adopted for integrating two different technologies (KNX and MyHome) within a home automation platform are illustrated.**

Keywords— Domotics, IoT, Interoperability, Home Automation, KNX, MyHome, Home Gateway

I. Introduction

With the spread of increasingly fast and reliable communication technologies such as *5G*, the number and functionalities of the components available in *Internet of Things (IoT)* solutions will soon grow exponentially [1]. New real-time applications and smart city solutions will be able to exploit these technologies to create a large new coherent infrastructure able to connect people [2], homes, buildings, and entire living areas [3]. More and more, Ambient Intelligence (AmI), as envisioned by Mark Weiser, [4] is becoming a reality. The technology surrounding people will become invisible, and new innovative human-machine interactions with autonomous and intelligent entities will act in a fully interoperable main system to improve the everyday life of humans unobtrusively and transparently.

In order to realise such a scenario, apart from the need for a fast reliable channel such as that offered by 5G technology, all devices and services must be able to understand each other so that they can exchange information. To this end there are currently two main options: (a) using the same language and same transmission protocols to effect the communication between all devices and services; (b) create an interoperability mechanism to enable the exchange of data regardless of the underlying incompatible communication protocols. With the former, the use of a single language and protocol limits implementation choices to devices belonging to a single (often proprietary) technology. The latter solution overcomes this limit by means of the interoperability feature [5], which enables two or more systems (in the case at hand, home automation systems) interact with each other to exchange information originally gathered, stored and processed in different formats.

There is moreover a need for special entities, such as software agents or other intelligent systems [6], able to coordinate the sharing of data generated by the various devices in the overall system. The source of their ‘intelligence’ may be on board the devices themselves, on *Edge* [7] or the *Cloud* [8].

The scientific community has proposed a number of different frameworks for realising such an *IoT* scenario by creating both mechanisms for information exchange between devices and solutions for data processing. Most of these enable systems to read values from sensors and to store them in the cloud for future analysis, but rarely include native support for various domotic technologies [9]. The European Community has also funded many projects on *IoT* development, especially with the *Seventh Framework* and the subsequent *Horizon 2020 Research* programmes. However, in these cases, the result was the creation of some *IoT* frameworks specific for vertical areas of competence and scenarios such as transport and logistics, mobile health and wellness, smart grid and smart factories. Most of these *IoT* solutions implement proprietary communication mechanisms, devices, and resource control protocols, using a single communication technology. Thus, these frameworks are poorly interoperable with each other.

Domotics [9] is the first application domain for *IoT.* Underlying domotic technologies have been steadily been moving beyond the mere control and connection of appliances in an isolated home, to a global connection and integration of the house with the services available around the world. Even *Google Inc.* [10], *Amazon Inc.* [11] and *Apple Inc.* [12] have launched products in the home automation field, and have become big players in this market nowadays. Integration of the home automation world with the global *Internet of Things* still presents a specific research challenge: there is a lack of standardization in communication systems and protocols. This lack is due to the persistence of business models towards closed proprietary solutions that strongly limit the possibility of integrating devices and services belonging to different home automation systems.

To date, this lack of interoperability is still an open issue that severely limits the full potential of *Smart Home* technologies within the *IoT* world.

II. RELATED WORKS

The literature proposes several definitions of *IoT* interoperability. Generally, from a technical point of view, interoperability is mainly classified into three levels [13]:

- *basic connectivity interoperability*: allows for the exchange of data between two or more heterogeneous systems by establishing a communication link. It requires common agreement on the means of data transmission, low-level data encoding, and access rules;
- *network interoperability*: enables the exchange of messages between systems deployed in different networks. It defines the agreement on the transfer of information between heterogeneous systems through multiple communication links;
- *syntactic interoperability*: defines rules to manage the format and structure of the encoded information exchanged between two or more heterogeneous intelligent systems to make them understandable by all the components involved, translating data from one format to another.

There are currently some *open source* interoperability frameworks with explicit support for domotic technologies that implement interoperability up to the *syntactic* level. *Freedomotic* [14] is an event-based framework with a decentralised peer-to-peer network. It furnishes support replication of services across different servers exploiting a load balance mechanism. *DomoNet* [15] is an event-based framework leveraging *SOA* [16]. It can manage different instances of the framework located in different places at the same time, and it assigns *IPv6* addresses to each domotic device, even if they do not possess native Internet addressing capabilities. *Dog gateway* is an *OSGi* semantic framework based on an ontology named *DogOnt* [17], which taxonomizes devices and environments and defines their functions and characteristics. *IoTSys* [18] is a *Seventh Framework Programme* project. It allows *IPv6* addresses to be mapped to objects and it moreover addresses security, discovery, and scalability issues. It exploits *oBIX* [19] at the *API* level to discover services and devices, their setup and invocation.

This work proposes a solution which, unlike other existing frameworks, aims to create an *IoT* framework for domotic interoperability that has been designed and developed to be:

- *modular* – the system enables choosing the functional modules that are included as part of each instance of the system according to users' needs and the computing capacity of the hosting machine(s);
- *highly scalable* (both downwards and upwards) – the solution's modular implementation makes it possible to deploy the software framework on a wide variety of electronic systems, from those with even limited computing capacity up to powerful servers and workstations;
- *flexible* – the modularity and scalability of the system provide a solution suitable for: (a) small facilities (e.g. a single room, an apartment, a single house, etc.); (b) large facilities (e.g., a house or a building consisting of several apartments, etc.); (c) industrial settings (e.g., one or more industrial buildings of a single organization); (d) mission-critical situations (e.g., hospitals or facilities where malfunctions are unacceptable, even temporarily).

III. THE SHELL FRAMEWORK

Shell is an open, free and accessible interoperability *IoT* framework. It acts as the backbone that enables tools for creating vertical solutions for managing energy efficiency, thermo-hygrometric comfort, security and safety, and the sharing of data generated by the home with the *IoT*.

In this way, the home is set to become a functional and interoperable node of a larger *Smart Community*, open to the new opportunities of *Smart Cities*.

The *Shell* framework implements a conceptual model aiming at:

- addressing a variety of smart ambient automation scenarios, ranging from houses to large buildings and smart cities;
- allowing the integration of heterogeneous appliances and devices in a uniform data space where the functionalities of devices are defined at a high level of abstraction, independently of the actual technologies used (e.g., communication protocols, hardware, etc.);
- being general, open, and compatible with existing smart-home models and systems.

A. Shell Architecture

Each node of the *Shell* framework is called a *host*. A *host* (Figure 1) is an embedded geolocalized system having an *IP* address and connected both to a private *LAN,* where the domotic devices it manages reside, and to the Internet, to allow communication with the outside world. The *host* contains an internal representation of its domotic devices (*Shell devices*) that can access other *host*s or external services.

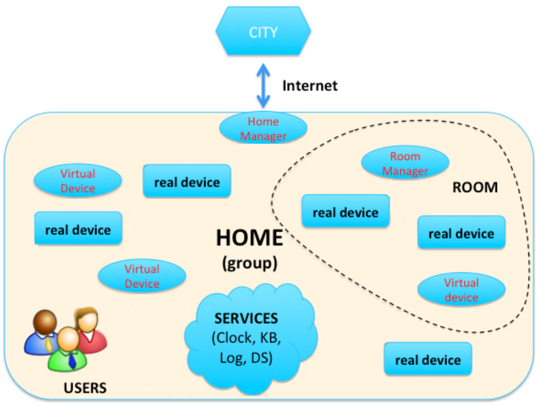

Figure 1: Shell architecture

*Shell device*s can be *real* or *virtual*. A *real* device is the Shell's representation of a smart object equipped with specific hardware. Instead, a *virtual* device is a representation of a set of software functionalities (e.g., a weather service on the Web) without any hardware. Every *device* has an *id* (i.e., *serial_number.model.producer*) for identification and addressing.

To accomplish communication between devices, each *host* implements a naming service system that translates and maps each *id* to the corresponding *IP* address. *Device*s are included in environments called *rooms*. In turn, *Rooms* can also be *real* (e.g., a geolocalized place such as a living room, a bathroom, etc.), or *virtual* (e.g., the room called "lamps" consist of a group of objects with similar functions related to lighting). Each *group* is controlled by a special *device* called a *device manager*. It determines the functionality and purpose of the *group* and manages its devices. A *device* can belong to one or more *group*s. There is one predefined *group* called *Home* that includes all the devices of the *host*. The manager of the *Home* group is aptly named *Home Manager* and supervises all the activities of the *Host*. Each *host* offers a set of special *service*s to the *device*s. By way of example, some of these services are: (a) *Clock*: synchronizes devices to allow them to perform actions simultaneously (e.g., at prefixed time intervals get the status of all devices at the same instant); (b) *KnowledgeBase* (*KB*): implements a database that classifies devices according to type and their functionalities. In particular, it subdivides the Shell commands into *functional classe*s that are then associated with each type of device. Moreover, it defines the messages that the devices generate each time they change state. For example, the device *light* has three commands belonging to the functional class *binaryLight*: *on*, *off* and *getStatus*. When switched on or off, a *light* device sends a message setting the *lightState* parameter with the new value. (c) *System Logger* (*Log*): keeps the sequence of events occurring throughout the system up to date, and offers the possibility to query and retrieve information; (d) *Discovery Service* (*DS*): when a host starts, it announces to its external network both its presence and the list and status of its devices. This service thus broadcasts that a new host has become available, and each *host* has a defined set of *user*s who can access it at the same time. The framework defines four levels of users: (a) *superuser*: has access to all system functions; (b) *technician*: has access to all configuration and installation functions; (c) *power user*: has access only to some configuration functions; (d) *user*: has access only to basic system functions (e.g. user interface to pilot devices).

Attribute	Description
id	Unique identification of the message.
ts	The timestamp associated with the message.
to	Unique identification of the recipient device.
from	Unique identifier of the sender device.
toIP	IP address of the recipient device. Alternative for *to*
fromIP	IP address of the sender device. Alternative for *from*
type	Type of message.
cmd	Command to execute, a signal or a notification.
params	List of the parameters expressed in the form of key-values required in order to correctly execute the request.
auth	Reserved field. Used for possible implementations of security systems (e.g. message signature etc.).

Table 1: JSON message fields

The choice of using the addressing system based on standardised and well-established protocols such as *IP* and *HTTP* guarantees *Shell* interoperability both at the *basic* and *network connectivity* levels. The *host* directly manages devices with *Ethernet* and/or *wifi* connectivity and configures them using the *Avahi/Bonjour Zero-conf* mechanism [20]. The Shell moreover provides *syntactic interoperability* based on a lightweight, expressive and flexible messaging system implemented in *JSON* format [21].

Table 1 shows the fields that compose a *Shell JSON* message. Table 2 describes the possible values for the *type* attribute.

Message type	Description
Exec	A command to be executed synchronously.
Dexec	A command to be executed asynchronously.
Signal	A signal (e.g., a notification or an event).
Response	The answer to a request.

Table 2: Values for "type" attribute

Figure 2 shows an example of *JSON* message of *exec* type (line 8) where a *device manager* (line 5) orders a light (line 4) to switch on (line 9). Each command has an unique identifier expressed as: <*f-class*>.<*cmd*>, where: *f-class* is the name of the functional class of the command and *cmd* is the command itself. In the example in figure 2, the command *binaryLight.on* (line 9) means the light is to be switched on.

```
1. {
2.     "id": "198800003467190012378-76630133-31191013",
3.     "ts": 14559594800,
4.     "to": "100000000002.light.semedia",
5.     "from": "100000000001.manager.semedia",
6.     "toIP": "192.168.65.181",
7.     "fromIP": "192.168.65.180",
8.     "type": "exec",
9.     "cmd": "binaryLight.on",
10.    "params": {},
11.    "auth": ""
12. }
```

Figure 2: Example message

However, if specific models of smart objects have functions that do not have a match in any *f-classe*s, the system allows extending the *Shell* representation of the devices by personalising them.

B. *Interoperability*

To implement interoperability, we introduce the concept of *Signal*. A *signal* is a special type of *Shell* message that provides notification of the occurrence of an event related to a device. In order to classify and manage them, the *Knowledge Base* in the *host* defines the *signal*s for each type of device. A typical example of a *signal* message is a change in the state of a light. When such a state change occurs, the system emits a *signal* regarding the specific "light" device in question. The configuration file of the device to which the *signal* refers specifies the modalities for implementation of interoperability between devices.

In the example in Figure 3, when the device in question emits a *signal* message which sets the *lightState* parameter to *true* (line 9), then, for a dimmable light, the system creates a new *exec* message (line 12) which sets *lightDimmerState* to 100%.

```
{
 "actions":[
   {
     "conditions":[
       {
         "conditionType":"OUTPUT_PARAM_VALUE",
         "paramName":"lightState",
         "operator":"EQUAL",
         "value":true
       }
     ],
     "actionType":"EXEC",
     "name": "DimmerableOnlyLight.setLightDimmerState",
     "to":"aeon.dimmerable-light.001",
     "params":{
       "lightDimmerState":100
     }
   }
 ]
}
```

Figure 3: Example of interoperability configuration

IV. Shell Drivers

The *Shell* allows incorporating technologies involving devices that use protocols and languages which are not natively compatible with the *Shell* framework. This can be achieved by using special software that provides gateway functions, named *driver*s. Thus, there is a *driver* for any and every "external" domotic system that one might want to include. Each *driver* is structured in two parts: the first acts directly on the *Shell,* the other physically interfaces the devices belonging to the external network. Each *Driver* performs the following operations: (a) creating the physical connection with the domotic bus; (b) auto-configuration according to the device to be managed; (c) ensuring correspondence between external devices and *Shell devices*; (d) translating *Shell message*s into messages that comply with the external domotic protocol; (e) and notification of events that occur within the external system by creating appropriate *Shell messages* of type *signal*.

Shell drivers also provide the *Shell* framework with access to every single feature of each device it manages, thereby making such features utilizable. Both the discovery and implementation of services are different for each driver, as they are strictly dependent on the technology of the different external systems. Any smart object is addressed from inside and outside the framework using the unique *id* assigned to each *Shell device*. In order to maintain consistency between the different addressing spaces of the different external subnetworks, a global routing mechanism associates each unique *Shell device identifier* with each actual device,.

By way of example, the drivers named *KNXDriver* and *MyHomeDriver, which* allow integrating within the system two existing commercial domotic technologies, *KNX* [22] and *MyHome* [23],, are described below.

A. KNXDriver

KNX main features are: (a) coexistence of products and applications from different manufacturers under the same umbrella; (b) good reliability – the *KNX Association* regularly checks manufacturers' production and certifies its quality; (c) standardized functionality – the *KNX Association* integrates different application profiles for both home and building automation into its domotic system.

The KNX architecture provides for distributing the onboard intelligence of the individual devices. Wiring is simple and economical, and the probability of failure low. On the other hand, this involves rather high costs per device.

KNXDriver exploits three main technology features: the *individual address*, the *group address* and the *datapoint*. *KNX* uses an *individual address* both to identify a device and to indicate its position in the network topology. It consists of three dot-separated numbers (e.g., *1.1.1*). A *group address* identifies a *KNX* domotic function available to the network (e.g., switching off a light). It involves at least two devices and makes it possible to establish a logical connection between them. It consists of three slash-separated numbers (e.g. *1/1/1*). A *datapoint* is represented by a variable indicating the value of a domotic function implemented in the network. *Group address* and *datapoint* are two strictly interrelated features, that is, each *group address* has its *datapoint*. Note that the *datapoint* defines how to code the value of the variable. Variables are standardised, which is fundamental in order to allow devices from different *KNX* manufacturers to communicate with each other.

To ensure a physical connection with the *KNX bus*, *KNXDriver* depends on a special system component named *IP Gateway*. Dedicated software, called *ETS* (*Engineering Tool Software*) [24], is used to program the devices and supervise the *bus*. *KNXDriver* exploits the *Calimero 2 API* [25] libraries to interact with a *KNX* network. It enables read and write operations and the *datapoint*s management.

The driver auto-configures at startup time. It associates the instances of *Shell device*s with the real devices, using the information found in the device configuration files. Figure 4 shows an example of the device configuration file for a light.

```
 1.{
 2.  "id": "shell.knx-binary-light.0001",
 3.  "friendlyName": "Simple KNX Light 1",
 4.  "description": "KNX Simple Light",
 5.  "location": "Bedroom Room",
 6.  "internal": true,
 7.  "classes": ["BinaryLight"],
 8.  "drivers": [{
 9.    "driver": "KNX",
10.    "driverParams":{
11.      "knx_individualAddress": "1.1.1",
12.      "BinaryLight_getLightState_knx_groupAddress": "0/0/1",
13.      "BinaryLight_getLightState_knx_datapoint": "DPTXlatorBoolean.DPT_SWITCH",
14.      "BinaryLight_getLightState_shell_output_name": "lightState",
15.      "BinaryLight_on_knx_groupAddress": "0/0/1",
16.      "BinaryLight_on_knx_input_value": "on",
17.      "BinaryLight_on_knx_datapoint": "DPTXlatorBoolean.DPT_SWITCH",
18.      "BinaryLight_on_shell_output_name": "lightState",
19.      "BinaryLight_on_shell_output_value": "on",
20.      "BinaryLight_off_knx_groupAddress": "0/0/1",
21.      "BinaryLight_off_knx_input_value": "off",
22.      "BinaryLight_off_knx_datapoint": "DPTXlatorBoolean.DPT_SWITCH",
23.      "BinaryLight_off_shell_output_name": "lightState",
24.      "BinaryLight_off_shell_output_value": "off",
25.      "BinaryLight_lightStateChanged_knx_groupAddress": "0/0/1",
26.      "BinaryLight_lightStateChanged_knx_datapoint": "DPTXlatorBoolean.DPT_SWITCH",
27.      "BinaryLight_lightStateChanged_shell_output_name": "lightState"
28.      },
29.    "commands": [],
30.    "signals":[]
31.  }]
32. }
```

Figure 4: Example configuration for a KNX device

In the example in the figure, the *driver* field specifies the following: the name of the *driver* managing the device, that is KNX (line 9); that the *Shell* device is a *BinaryLight* type, as defined in the *Shell* classification (line 7). In the example, we ensure correspondence between *Shell device* called *shell.knx-binary-light.0001* (line 2)*,* with the *KNX device* with *individual address 1.1.1* (line 11). To translate *Shell messages* into *KNX* messages and vice versa, we use the information provided in *driverParam* (line 10), where there are a number of parameters needed to map *Shell* messages to *KNX* compliant messages and vice-versa. The *KNXDriver* implements the functionality to convert a *KNX* value to a *SHELL* value and vice versa, according to the respective *datatype* and *datapoint*. One example of this is the representation of a scale of values: in *Shell* a percentage scale has a range from 0 to 100, while in *KNX* its range is from 0 to 255. Since the *Shell* value *90* does not correspond semantically to the *KNX* value *90*, it is necessary to convert it appropriately using a proportion (resulting in a KNX value of *229)*.

The *driver* implements a *listener* to capture events occurring on the *KNX* bus. The events are embodied in two types of messages: (a) a synchronous message, as a reply message to a *Shell* request (e.g., the request *binaryLight.getLightState*, to obtain the state of a light as a reply), or (b) an asynchronous message, as notification that an event has occurred in the domotic bus and must be translated into a *Shell* message of type *signal* to achieve interoperability.

B. MyHomeDriver

MyHome is a domotic system produced by *BTicino* (Legrand group), a manufacturer of electrical components whose brand name is well-known in more than 60 countries around the world. *MyHome* is based on a distributed intelligence system.

It offers services for: (a) comfort (lighting management, energy loads, sound diffusion and scenarios management); (b) security (technical alarms and anti-theft protection); (c) audio-video communication (video door entry system and video control); (d) local and remote equipment control.

The protocol used to exchange data and send commands between a remote unit and the *MyHome BTicino* system is named *OpenWebNet* (*OWN*). The *OWN* language [26] allows communication independently of the physical means of communication used.

The characters allowed in an *OpenWebNet message* belong to the following set *Sym = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *, #}*. An *OpenWebNet message* starts with * and ends with *##*. The fields making it up are of variable length and separated by the special characters "*". Taking into account for simplicity only the command messages, these fields are *WHO*, *WHAT*, *WHERE*, *DIMENSION*, *and VALUE*. *WHO* contains the domotic function the message refers to (e.g., 1 for the lighting system). *WHAT* identifies the type of action to be carried out (e.g., lights ON, lights OFF, dimmer 20%, shutters UP, set programme 1 in the temperature control unit, etc.).

WHERE contains the objects addressed by the message (single device or a group of them, such as the lights in group 1). *DIMENSION* is used to request information about the value of a parameter belonging to a single device, a set of devices, or the entire system. The server responds to the size request by sending one or more messages containing one or more *VALUE* fields. Both fields are also used to modify the values of the parameters belonging to the devices.

For every *WHO* message, there is a specific *DIMENSION* table containing the range of allowable *VALUE*s (this applies only to functionalities for which they are required).

OpenWebNet has four types of messages: *Command/Status*, *Status Request*, *Request/Read/Write Dimension*, *Acknowledge*.

To establish a connection with the *MyHome* bus, *MyHomeDriver* uses a particular device named *Open Server*. *OpenServer* is an *OpenWebNet gateway* that sees to interfacing with the *MyHome* system using the *OWN* language. *MyHomeDriver* communicates with the *Open Server* through the use of a customised version of the *btcommlib* library [26].

MyHomeDriver initialises at startup using the configuration files related for *Shell devices* it manages. Figure 5 shows an example of the configuration for a light.

```
 1.{
 2. "id":"shell.bticino-binary-light.0001",
 3. "friendlyName":"Simple BTicino Light 1",
 4. "description":"BTicino Simple Light",
 5. "location":"Bedroom Room",
 6. "internal": true,
 7. "classes": ["BinaryLight"],
 8. "drivers": [{
 9.   "driver": "MyHome",
10.   "driverParams":{
11.     "classes": ["BinaryLight"],
12.     "ambient":"1",
13.     "light_point":"1"
14.   },
15.   "commands": [],
16.   "signals":[]
17. }],
18. "mainImplementationJarFile" : "",
19. "mainImplementationJavaClass" : ""
20.}
```

Figure 5: Example configuration for a MyHome device

Line 9 shows the driver that controls the *MyHome* device and line 11 specifies the *Shell device* type. By concatenating values in lines 12 and 13 we obtain the contents of the field *WHERE* of an *OWN command*, thus providing identification of that device within the *MyHome* system. To ensure a match between the *Shell device* and the real device, the driver associates these values with the *Shell device* identifier (line 2). Figure 6 shows an extract of the configuration file used to specify the other fields of an *OWN command*. They are independent of the single device, but strictly related to the functionalities. For example, to invoke the *Shell command* named *BinaryLight.on* (lines 2 and 4), the driver fills in the *WHO* and *WHAT* fields with the value 1 (line 6 and 8).

```
{
 "BinaryLight": {
  "commands": {
   "on": {
    "msg_type": "Request",
    "who": "1",
    "whatlist": [{
     "what": "1"
    }]
   },
   "off": {
    "msg_type": "Request",
     "who": "1",
     "whatlist": [{
      "what": "0"
     }]
   },
   "getLightState": {
    "msg_type": "RequestStatus",
    "who": "1"
   }
  }
 }
}
```

Figure 6: Configuration to map BinaryLight commands

To send commands to *MyHome* bus and receive notifications from it, the driver creates two special threads to connect both the *command* and *monitor* sessions of the *OWN* gateway. The *command session* sees to sending commands, status requests and dimension requests.

The *monitor session* allows capturing the asynchronous events of the *MyHome* network and decoding them using a grammatical parser *LL(1)* [27]. Then it translates them into *Shell events* using the information available in the configuration files.

V. Results and Conclusions

Figure 7 shows the exchange of messages between *KNX* and *MyHome* binary lights. The messages are the results of the operations implemented to achieve interoperability. Whenever the *MyHome* light is turned on, it sends a power-up message to the *KNX* light, and receives notification of its status change. Then, when the *MyHome* light goes out, it sends a turn off message to the *KNX* light and receives its current status in reply.

This work has illustrated the operating logic of *Shell*, the proposed middleware solution to allow effective non-proprietary exploitation of the Internet of Things paradigm. This software is part of the *Shell* project which aims to create an open, freely accessible "interoperability framework" as a supporting structure and enabling tool for vertical solutions in diversified and multifunctional areas (energy, security, comfort). The project aims to transform the home into a set of shared, interoperable ecosystems, shaping the technology for smart environments that can gather information and produce actions tailored to its inhabitants. Although the *Shell* framework is still under development, its functional core is fully and freely available today.

Thanks to the power of the *Shell* language, the framework implements a software abstraction layer that hides the technological complexities underlying natively incompatible devices.

```
Sending Message >> {"id": "1", "to": "shell.knx-binary-light.0001",
"from": "shell.myhome-binary-light.01", "type": "EXEC", "cmd":
"BinaryLight.on", "ts": 1553104822734, "params": {"state":
"lightState"}}

Process Response << {"id": "1", "to": "shell.myhome-binary-light.01",
"from": "shell.knx-binary-light.0001", "type": "RESPONSE", "rcode":
0, "cmd": " BinaryLight.on", "ts": 1553104822819, "params":
{"lightState": true}}

Sending Message >> {"id": "2", "to": "shell.knx-binary-light.0001",
"from": "shell.myhome-binary-light.01", "type": "EXEC", "cmd":
"BinaryLight.getLightState", "ts": 1553104822740, "params": {"state":
"lightState"}}

Process Response << {"id": "2", "to": "shell.myhome-binary-light.01",
"from": "shell.knx-binary-light.0001", "type": "RESPONSE", "rcode":
0, "cmd": "BinaryLight.getLightState", "ts": 1553104822870,
"params": {"lightState": true}}

Sending Message >> {"id": "3", "to": "shell.knx-binary-light.0001",
"from": "shell.myhome-binary-light.01", "type": "EXEC", "cmd":
"BinaryLight.off", "ts": 1553104822744, "params": {"state":
"lightState"}}

Process Response << {"id": "3", "to": "shell.myhome-binary-light.01",
"from": "shell.knx-binary-light.0001", "type": "RESPONSE", "rcode":
0, "cmd": "BinaryLight.off", "ts": 1553104822905, "params":
{"lightState": false}}

Sending Message >> {"id": "4", "to": "shell.knx-binary-light.0001",
"from": "shell.myhome-binary-light.01", "type": "EXEC", "cmd":
"BinaryLight.getLightState", "ts": 1553104822748, "params": {"state":
"lightState"}}

Process Response << {"id": "4", "to": "shell.myhome-binary-light.01",
"from": "shell.knx-binary-light.0001", "type": "RESPONSE", "rcode":
0, "cmd": "BinaryLight.getLightState", "ts": 1553104822961,
"params": {"lightState": false}}
```

Figure 7: Interaction between a KNX and MyHome devices

Acknowledgment

The *Shell* project ("Shared and interoperable Home Ecosystems for Sustainable, Comfortable and Safe Living Environments" - CTN_00128_111357) of the Italian National TAV Cluster - Technologies for Living Environments (CTN_00128), has been financed by the *MIUR* (Italian Ministry of Education, University and Research) with Directorate Decree no. 1883 of 11/10/2013.

References

[1] L. Shancang, D. X. Li e Z. Shanshan, "5G Internet of Things: A survey" *Journal of Industrial Information Integration,* vol. 10, pp. Pages 1-9, 2018.

[2] J. Miranda, N. Mäkitalo, J. Garcia-Alonso, J. Berrocal, T. Mikkonen, C. Canal e J. M. Murillo, "From the Internet of Things to the Internet of People" *IEEE Internet Computing,* vol. 19, n. 2, pp. 40-47, 2015.

[3] K. E. Skouby e P. Lynggaard, "Smart home and smart city solutions enabled by 5G, IoT, AAI and CoT services" in *International Conference on Contemporary Computing and Informatics (IC3I)*, Mysore, 2014.

[4] M. Weiser, "The computer for the twenty - first century" *Scientific American,* vol. 265, n. 3, pp. 94-104, 1991.

[5] T. C. I. 184, "ISO 14258:1998 - Industrial automation systems -- Concepts and rules for enterprise models" 2014. [Online]. Available: https://www.iso.org/standard/24020.html.

[6] A. M. Mzahm, M. S. Ahmad e A. Y. C. Tang, "Agents of Things (AoT): An intelligent operational concept of the Internet of Things

(IoT)" in *International Conference of Intelligent Systems and Design and Applications*, Bangi, 2013.

[7] W. Shi, J. Cao, Q. Zhang, Y. Li e L. Xu, "Edge Computing: Vision and Challenges" *IEEE Internet of Things Journal,* vol. 3, n. 5, pp. 637-646, 2016.

[8] M. Aazam, I. Khan, A. A. Alsaffar e E.-N. Huh, "Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved" in *11th International Bhurban Conference on Applied Sciences & Technology*, Islamabad, Pakistan, 2014.

[9] G. M. Toschi, L. B. Campos e C. E. Cugnasca, "Home automation networks: A survey" *Computer Standards & Interfaces,* vol. 50, pp. 42-54, 2017.

[10] P. Dempsey, "The teardown: Google Home personal assistant" *Engineering & Technology,* vol. 12, n. 3, pp. 80-81, 2017.

[11] A. Purington, J. G. Taft, S. Sannon, N. N. Bazarova e S. H. Taylor, "Alexa is My New BFF: Social Roles, User Satisfaction, and Personification of the Amazon Echo" in *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, Denver, Colorado, USA, 2017.

[12] D. Gack, Apple Homekit: The Beginner's Guide, vol. 1, Van Helostein, 2017.

[13] T. Perumal, A. R. Ramli, C. Y. Leong, S. Mansor e K. Samsudin, "Interoperability for smart home environment using web services" *International Journal of Smart Home,* vol. 2, n. 4, pp. 1-16, 2008.

[14] "Freedomotic Open IoT Framework" [Online]. Available: https://www.freedomotic-iot.com.

[15] V. Miori e D. Russo, "Domotic Evolution towards the IoT" in *28th International Conference on Advanced Information Networking and Applications Workshops*, Victoria, BC, 2014.

[16] E. Newcomer e G. Lomow, Understanding SOA with Web services, Addison-Wesley, 2005.

[17] D. Bonino e F. Corno, "DogOnt - Ontology Modeling for Intelligent Domotic Environments" in *The Semantic Web - ISWC 2008*, 2008.

[18] "IoTSyS - Internet of Things integration middleware" [Online]. Available: http://www.iue.tuwien.ac.at/cse/index.php/projects/120-iotsys-internet-of-things-integration-middleware.html.

[19] "oBIX" [Online]. Available: http://www.obix.org/.

[20] K. Wallis e C. Reich, "Secure Zero Configuration of IoT Devices - A Survey" in *W-CAR Symposium on Information and Communication Systems (SInCom)*, Karlsruhe, Germany, 2016.

[21] T. Bray, Bray, Tim. The javascript object notation (json) data interchange format*,* 2017.

[22] S. D. Bruyne, "Finding your way around the KNX Specifications" in *KNX Technoology Tutorial Workshop*, Deggendorf, Germany, 2004.

[23] "Home automation" bticino, [Online]. Available: http://www.bticino.com/solutions/home-automation.

[24] "About ETS - KNX Association" [Online]. Available: https://www2.knx.org/en-us/software/ets/about/index.php.

[25] B. Malinowsky, G. Neugschwandtner e W. Kastner, "Calimero: Next Generation" in *Proceedings Konnex Scientific Conference*, Dusiburg, Germany, 2007.

[26] "Home Page - MyOpen" bticino, [Online]. Available: https://www.myopen-legrandgroup.com.

[27] A. V. Aho, R. Sethi e J. D. Ullman, Compilers, principles, techniques, Addison wesley, 1986.