Fpt partie III

Validation

*Winners compare their achievements with their goals, while losers compare their achievements with those of other people.*

Nido Qubein

The contribution of this thesis has been validated by several elements.

Firstly, EnTiMid has been tested on a realistic use-case scenario, in which all properties previously listed have been stressed. This scenario, defined in collaboration with partners of an AAL project, is presented in chapter .

A second validation has been realized by bringing EnTiMid in front of a bundle of end-users, namely elderly people. This study had two goals. The first goal was the improvement of the gui widgets presented to end-users, in order to make them as intuitive as possible. The second was the verification that people are able to act on, and get information from the home, according to a pre-define scenario. These two points are detailed in chapter **Erreur ! Source du renvoi introuvable.**.

**Chapitre** **7**

# Validation in the context of an AAL project

As one of the IDA project’s partners, we proposed EnTiMid as an integration tool, for the different equipments offered by industrials. Our initial wish was to come to an integrated, but very flexible, solution for elderly people. Even if this wish did not realize, EnTiMid had been considered and evaluated as a possible perspective for the future of aal. This evaluation was based on a scenario, collaboratively defined with partners of the project. It has been designed to stress the properties required for such a kind of system.

The first section of this chapter introduces to the project and its context, and details the scenario. Sections  to describe the evaluation of a set of properties : the environment setup, the procedures, and the results, for each property. Section  lists some threats to the validity of this study. The conclusion of these experiments is presented in section .

## 7.1 Context of the study : the IDA project

The first phase of the ida [[1]](#footnote--1) project took place from June 2008 to June 2010, in the Rennes Metropolis(France) community. This local aal project was funded by the Brittany region, to investigate on issues resulting from the ageing of the population, and its socio-economic impact. More precisely, ida conducted an inquiry about the use of ict to help elderly people in their everyday life at home. To this end, the project involved :

**Association for cares at home** ASSAD du Pays de Rennes

**Industrials** Custos, Delta Dore, Urmet Captive, Spartime, i–Pocarte, Domtis, Ordimemo, Intervox Systems, Laudren, Solem

**Social housing company** Archipel Habitat

**Installers** Marsollier Domotique, Lepage Electronique

**Project ownership assistance** ARELIA

**Research institutes** INRIA, University of Rennes 1, University of Rennes 2, LOUSTIC, IETR, CRPCC

**Public administrations** Rennes Metropole, Conseil General d’Ille et Vilaine, Ville de Rennes, CCI Rennes, Critt Santé, MEITO

**Elderly people** Anonymous individuals for tests, obviously

The conclusions of the project have been compiled in [**Erreur ! Source du renvoi introuvable.**].

The "ASSAD du Pays de Rennes" Association, leader of this project, employs 450 persons among which, nurses, life assistants, technicians, etc. The association helps more than 3,000 persons, split in 17 towns in the south and east of Rennes.

Although IDA was not funded by the European aal programme, the assisted(elderly) person was project-centric, and the six dimensions described in section **Erreur ! Source du renvoi introuvable.** appeared in watermark all along the project. The ASSAD association took care of this point.

EnTiMid had been presented as a system for the integration of the several devices evaluated in the project. Indeed, the objective of this project was to measure, how ict could foster the autonomy of elderly people at home, and support the activity of carers in this context. The idea was to offer an integration platform, able to deal with various devices and services, to promote the deployment of solutions adapted to each person’s needs.

But rapidly, this idea was given up because every product tested in real homes, with real elderly persons, had to be left in the home. As a prototype of laboratory, EnTiMid could not offer sufficient guaranties of maintenance to be really deployed *in situ*.

Still interested by the proposition, a collaborative work had been engaged within the project. With the help of the ASSAD, Delta Dore, Custos, Arelia, and the LOUSTIC, for the most actives, a scenario has been defined to put EnTiMid in situation.

This case study was designed to be as close as possible to real life conditions. The story has been introduced by the ASSAD association. Products have been proposed by Delta Dore. The evolutions have been gathered from past experiences of the ASSAD, Custos and the LOUSTIC. This scenario was setup to evaluate EnTiMid on a realistic case, with the same devices as those actually deployed. It stresses several issues to measure how EnTiMid cope with them.

The following of this section presents the story, and the context of evaluation of EnTiMid. Issues are highlighted and for each, the way EnTiMid addressed the problems is described.

## 7.2 Use case and issues to address

The scenario used for the evaluation of EnTiMid is presented in this section. It involves an elderly person called Mrs P., several members of its family, and different devices selected in collaboration with other partners, for the evaluation to be realized with real products, and in the closest conditions to reality.

The scenario is presented here with names of persons and real products for the sake of clarity.

The scenario is about Mrs P. She is seventy-eight. She has two children, and five grandchildren. Mrs P. begins to experience some difficulties to walk and move. To improve her safety, her daughter offered her to move to an equipped flat.

Among other equipments, the flat is basically equipped with an alert system, which triggers a voice call to a care centre when a button is pressed on a remote control. But Mrs P. is not comfortable with this equipment, and would prefer a more generic remote control. She also would like the alert to be sent to her daughter, instead of the care centre.

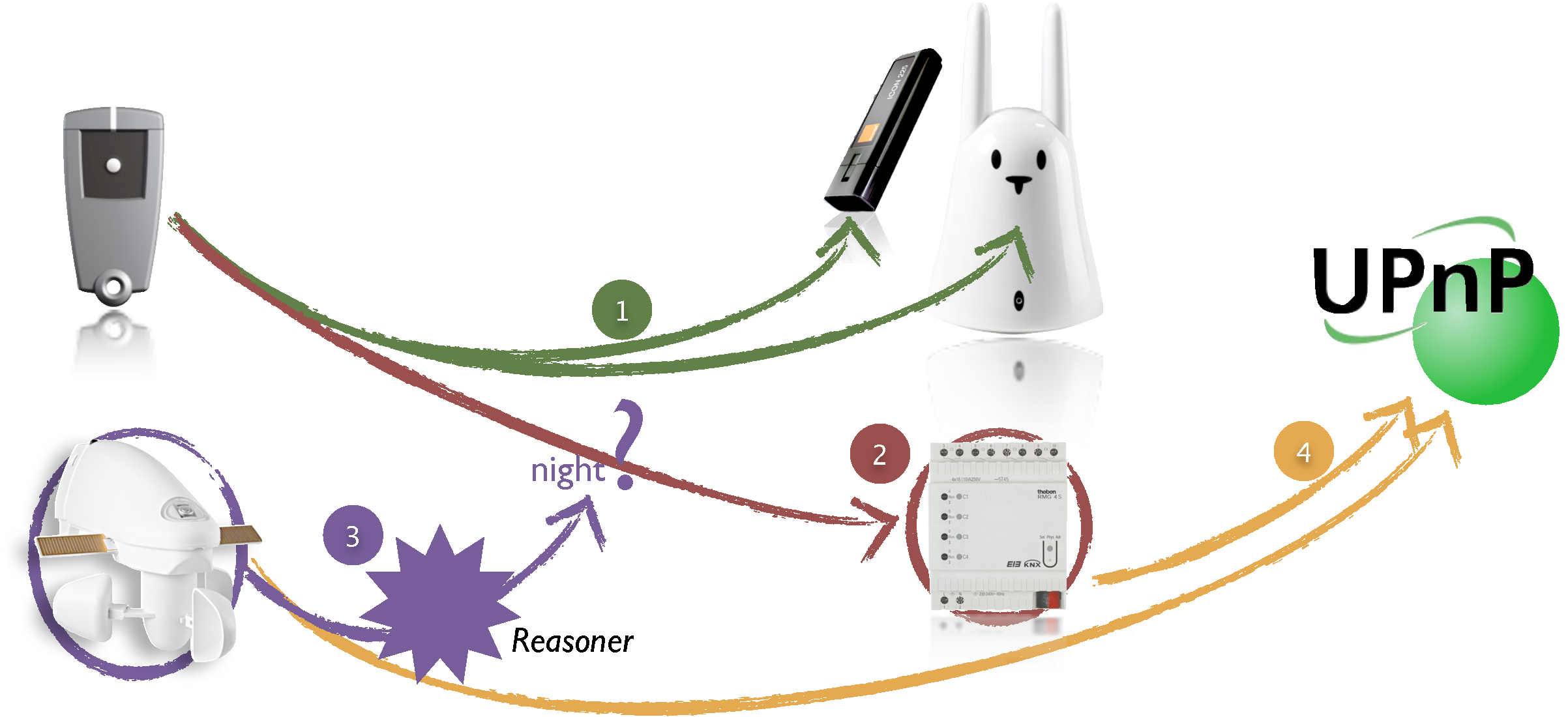


Fig. 7.1: Solution elements for Mrs P.

The remote control, on the top left corner of figure , is a one-button command from Delta Dore (a French manufacturer of home automation devices). This remote control has been designed to be universal for any receiver product, from the Delta Dore catalogue. A 3G-communication stick (Icon 225) is used to send alerts to Mrs P.’s daughter. Then a Nabaztag rabbit helpfully provides feedback to Mrs P. when she asks for help from her daughter.

The connection of these three items raises **interoperability** problems (symbolized by the bullet number one), which are detailed and answered in section .

After some months of use, Mrs P. asks for the system to automatically switch on the lights, when an alert is sent. Luckily, the flat is already equipped with devices that enable the control of lights. The light control is made available by a *RMG4S* device from Theben (on the bottom right of figure , working on a KNX network. Section  presents how EnTiMid enabled this **evolution**.

Following this evolution, the behavior of the lights was sub-optimal, because lights were switched on whatever the period of the day. To get rid of that issue, an **adaptation** mechanism(bullet 3), described in section , has been deployed. The sensing of daylight is realized by a KNX weather station outside the house. This weather station is visible on the bottom left corner of figure .

One day, her son came with a new device. This touch screen would offer Mrs P. to easily access to Internet, and have video calls with her children and grandchildren. The touch screen also has abilities for controlling devices over upnp and dpws networks. Since then, the deployed solution is required to be compatible and so, is required to export all available devices on upnp and dpws. This requirement stresses the need for **openness**. Section  elaborates about the mechanisms used to answer the fourth bullet of figure .

In order to evaluate the answers of EnTiMid in this scenario, and since a real deployment can not be realized, a test environment with real devices has been set up. Just before the description of the solutions offered by EnTiMid, the different elements composing this environment are presented in section .

## 7.3 Experimental setup

The test environment of this study has been realized with equipments provided by industrial partners of the IDA project for a part, and funded by the HID platform, financed by a European Found for Regional Developments, for the other part. It is mainly composed of two home mock-up, one with exclusively Delta Dore devices, the other with KNX devices only, and a MSI Top touch-screen PC, as visible in figure .



Fig. 7.2: Equipments available for the study

### 7.3.1 Delta Dore equipments

The Delta Dore mock-up, visible on the left side of picture , has been set up with devices from heating, alarm, security and automatism functional domains. Here is the description of all the elements.

– A *DRIVER 210 CPL + TYDOM 520* heating controller, on the bottom left corner of the mock-up, controls two heater receivers : a *TC51089* plc receiver, and a *RF660FP* radio receiver.

– The big black-box on the top right corner of the mock-up is a *TYXAL CSX40* alarm, which collects information from several sensors. A *DOFX* smoke sensor, and two *MiniCOX* door sensors, visible on the left side of the alarm. The last sensor is a *DFX* water leak sensor (green thing on the table, down the mock-up).

– One *TYXIA 442* light dimming transceiver, and a *TYXIA 411* timed power switch, both hidden behind the rabbit.

– Not visible on the picture, two remote controls. One *TYXIA 110* with a single ON/OFF button, and a *TYXIA 141* with four.

All these elements are using the X2D protocol, property of Delta Dore, to communicate on both PLC and radio media. Since the protocol is not public, Delta Dore lent us a research and development product, able to communicate in both ways (listen and act) on the X2D network.

### 7.3.2 KNX equipments

The second mock-up, on the right side of the picture, is made of KNX compatible products only, from mainly Theben manufacturer, and a bit of Siemens.

– On the top right corner of the KNX mock-up is an outdoor weather station. This weather station gives information about the wind, the rain, the temperature, and the enlightenment value. This information can be provided whether periodically, or when a value roughly changes.

– Just under the weather station, on the left, the *LUNA 113* is a light sensor for the outside. On its right, a *AMUN 7160* provides information about temperature, humidity and rates inside the house.

– Going down the right side comes the *VARIA 826 WH KNX*, which is an ambient controller. It allows for changing the heating regulation values, reading information from the weather station, and many other appliances.

– The four switches, in the bottom right corner of the mock-up, are controlled by a *TA 4*.

– The electric panel includes three other devices. A *RMG4s* device controls the four power sockets, at the very bottom of the panel, with a On/Off behavior only. Up this device, a DMG 2 controls the dimming of the two sockets on its right.

– Hardly visible on the top of the panel, a Siemens *EIB/IP N148/21* gateway makes it possible to access the KNX network through a IP connection. Helped by the Calimero [[2]](#footnote-0) framework, it enables a programmatic control on all the devices, and allows to listen for event on the KNX network.

### 7.3.3 Other equipments

The link between all of these devices is made by EnTiMid, but it still require an environment for its execution. Several other devices are available in this experimental setup.

– A all-in-one PC *MSI Wind Top*, with a touch-screen, an Intel Atom 230@1.6GHz CPU, 1Gb RAM and Ubuntu 9.10 (Linux kernel : 2.6.31-17-generic) for the operating system.

– An *Icon 225* 3G usb modem, used only for sending short text messages

– A Nabaztag :tag, the big rabbit on the picture, able to synthesize a voice from a text, and used to provide feedback to the user. A Nanoztag with a Mir :ror (grey little rabbit and blue circled base) are used as respectively a rfid tag and an RFID reader.

– An Ethernet router to connect the KNX mock-up and the touch-screen

## 7.4 Interoperability issue

The first problem that the story of Mrs. P emphasis, is the connection of three heterogeneous devices.

### 7.4.1 Environment of the test

To evaluate EnTiMid on this question, we made use of the Tyxia 110 remote control from the Delta Dore mock-up, the 3G modem to send short text messages, and the Nabaztag :Tag rabbit to provide feedback to the end-user.

Nothing in EnTiMid was already available to access these products.

The MSI Top touch-screen has been used for the deployment of the test.

### 7.4.2 Protocol of resolution

**Drivers creation**

Each product used in this test is from a different manufacturer. Thus, three drivers had to be created, as described in section **Erreur ! Source du renvoi introuvable.** of the contribution.

The driver enabling the use of the Tyxia 110 makes use of the gateway offered by the manufacturer, making it possible to listen on the X2D network. This gateway was delivered with a Java API, which simplified the creation of the driver. Indeed, the driver just consists in the creation of a listener, and of a class to handle the implementation and the model of the remote control device. The Tyxia 110 component has a unique output port *pressed*, and can be customized to specify the parameters to be sent through the output port when the button is pressed.

Second element, the 3G modem has been considered as a simple modem. The sequence of *AT* commands to be sent to the modem, to send a short text message, has been collected from the modem documentation. With the help of a serial communication library in Java (RxTx), the component has been implemented, and decorated with modeling annotations. The Icon 225 component representative offers a unique input port *send*. This port admits one parameter : the text of the message to send. The receiver’s phone number is given in parameter to the component.

The Nabaztag :Tag rabbit is the last element in this test, and is used to provide feedback to the final user. The web-service API of the Nabaztag rabbit provides a Text-To-Speech facility that can generate and return a MP3 file. Indeed, the rabbit is able to whether, directly synthesize a voice from a text, or generate a file containing the voice synthesis, for it to be played later by the rabbit. The component standing for the rabbit, thus proposes a *generate* input port. The action of this port is to call the text-to-speech facility with the text passed through the port as a parameter. The generated MP3 file is then returned through an output port called *generated*. A second input port, *play*, can be used to ask the rabbit to read a text or an MP3. If a text is given in parameter, the synthesis is made on the fly.



Fig. 7.3: Components used in the interoperability experiment

**Connection of the elements**

An instance of each element of the assembly is created in the model. The Tyxia 110 is customized to send two parameters through its output port on activation : the text "Your request has been sent to your daughter." for the rabbit, and "Your mother asks for a call from you." for the Icon 225.

This message with the two parameters is forwarded to a dispatcher, which triggers in parallel the *send* input port of the Icon 225 instance, and the *play* input port of the Nabaztag. Each component collects the parameter it is interested in, and realizes the required action.

### 7.4.3 Results

At first sight, the connection of a generic remote control, an electronic rabbit, and a short message modem is not that obvious. Industrial partners of the project were baffled by this requirement, because it implied for them to develop an ad-hoc device. This is not viable when targeting a provision of specialized solutions for each person’s needs. EnTiMid makes this problem quite easy to solve.

In this example, three drivers had to be developed to be able to get components in the model. Each driver can now be augmented to provide more products from each manufacturer. The development of drivers, once done, is never to be done again. Components can also be reused in other context, because of their independence, and the avoidance of direct connections. For future applications, this signifies a great gain in time.

## 7.5 Evolution issue

The evolution in this use-case is due to a change in Mrs P.’s needs. She wants the lights to be automatically switched on, when she presses the remote control. This requirement is related to a feeling of safety when light are on at night.

### 7.5.1 Environment of the test

This evaluation makes use of the previous devices involved in the interoperability evaluation.

In addition, the RMG4s of the KNX mock-up is integrated in the application.

### 7.5.2 Protocol of resolution

No KNX product had been used for the moment. So, the first task was to create the driver to control KNX equipments, and get a model representative for the RMG4s (presented in figure **Erreur ! Source du renvoi introuvable.** a few pages backward). Once the component ready, it has to be deployed.

Thanks to the Model@Runtime layer, the technician responsible for the addition of the new functionality, retrieves the current model of the running application, using a TCP/IP remote access. He then adds an instance of RMG4s, and connects all *on* input ports to the dispatcher already present. In this configuration, all lights controlled by the RMG4s are enlightened when the Tyxia 110 button is pressed, in addition to the text sent, and the rabbit speech.



Fig. 7.4: Components used in the evolution experiment

Last step, the technician sends back the model to the runtime of EnTiMid in the home. Once all checks passed, the runtime downloads the newly created component type and its driver, in order to create and connect the instance. All this procedure is transparent for Mrs P., who is just called before and after the operation to keep her informed. The OSGi runtime prevents from any service interruption.

### 7.5.3 Results

Again, a driver had to be created, and a component to handle the RMG4s too. Thanks to the model@runtime engine, the technician was able to collect the model and modify it. He then asked the runtime for the deployment of the new model. Realised through a TCP/IP connection, this evolution could have been realized remotely. This ability reduces the disturbance for the helped person, and can reduce the time from the query to the realization.

For this experiment, the communication with the runtime used a TCP/IP connection, which may not be that easy in real life.

## 7.6 Adaptation issue

The behavior of the lights was sub-optimal after the deployment of the evolution. Indeed, lights were switched on, whatever the period of the day, every time the remote control was pushed. To get rid of that issue, an adaptation mechanism(presented in [**Erreur ! Source du renvoi introuvable.**]) has been deployed. This mechanism changes the configuration of the system, according to the external enlightenment value sensed by a KNX outdoor weather station.

### 7.6.1 Environment of the test

In addition to the previous equipments, selected for interoperability experiments, and evolution concerns, the outdoor KNX weather station is now included in the configuration.

### 7.6.2 Protocol of resolution

The driver for KNX products already exists. The unique thing to implement is a component representative for the weather station.

The Model@Runtime engine offers means to connect reasoners. A reasoner is able to modify (or ask for modification of) the running system configuration. The decision is taken locally according to some contextual information. This information can be as simple as a value change, or as complex as an aggregation of events. For this evaluation, a reasoner has been created to modify the connections of the *RMG4S* according to the period of the day.

The system is currently composed of five components. A *TYXIA\_110* remote control, a dispatcher connected to a 3G usb stick to send texts, and to a Nabaztag rabbit to inform Mrs P. In addition, the RMG4S makes it possible to control the lights. **At night**, this configuration is the one required. **During the day**, the RMG4S should not be activated, and the connections with the dispatcher have to be removed.

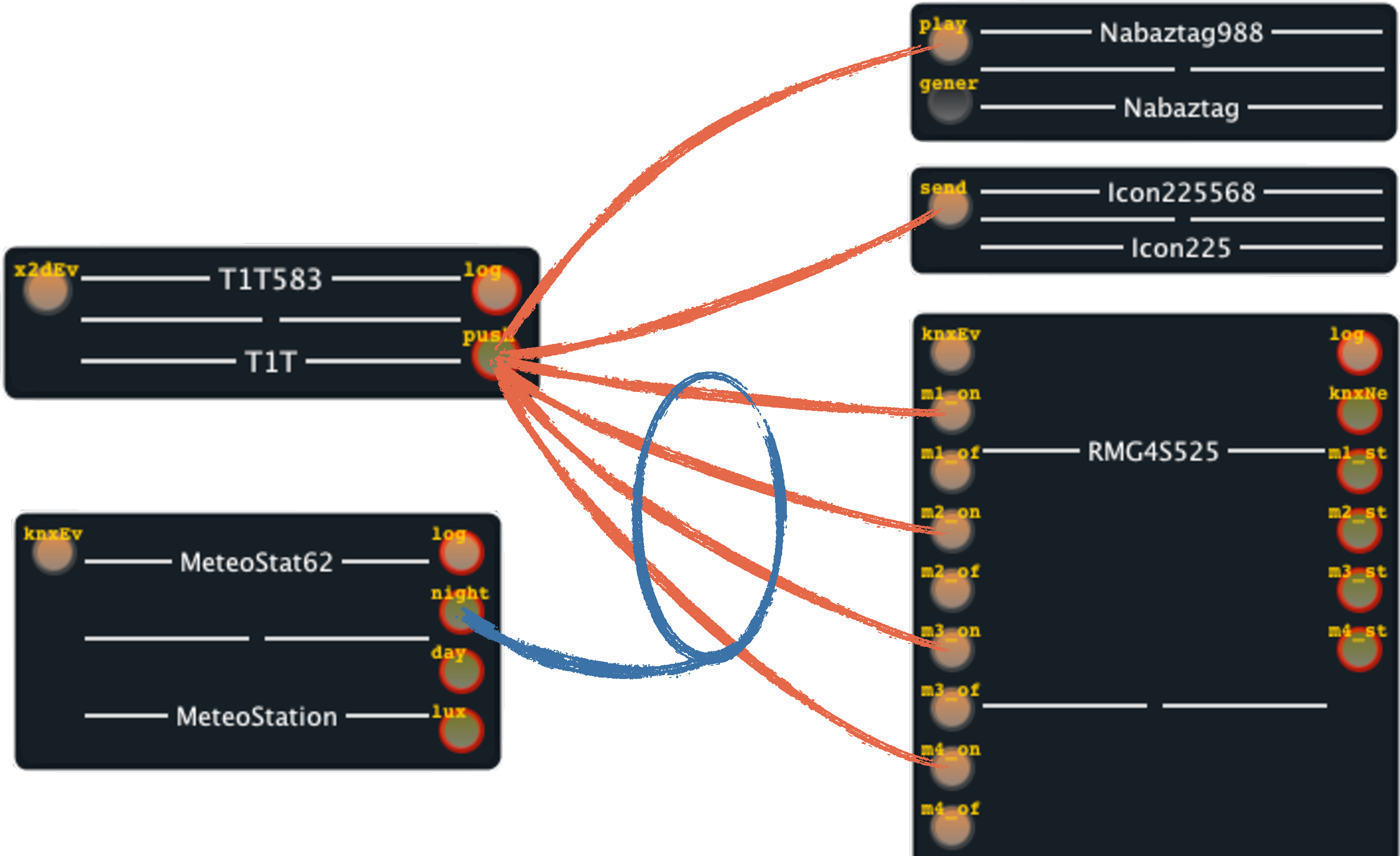


Fig. 7.5: Components used in the adaptation experiment

The weather station has been added to the system as described in section .

To be able to perform execution time test benches, the experiment including the reasoner has been deployed from scratch. The MSI Top was cleaned of any previous binary element, a restarted. The deployment was then made as follow.

**Initial Deployment** The initial deployment is realised during the day. The model deploys only the remote control, the rabbit, the usb stick, the dispatcher, and the weather station. In this configuration, the elderly person can ask for help by pressing the Tyxia 110 button, just as before, but no light is switched on.

**At night** The reasoner adapts the system to the new conditions. It changes the model by adding an RMG4S instance and all necessary connections to the dispatcher.

**Day** When the day rises again, the reasoner removes the connections and the instance of RMG4S.

### 7.6.3 Results

The figure  presents the execution times measured while a sequence of reconfigurations of the system was run. This sequence consisted in five steps. After the initial deployment (*State 1*), the scenario iterates night states (*State 2* and *State 4*) and day states (*State 3* and *State 5*) during the next two days.

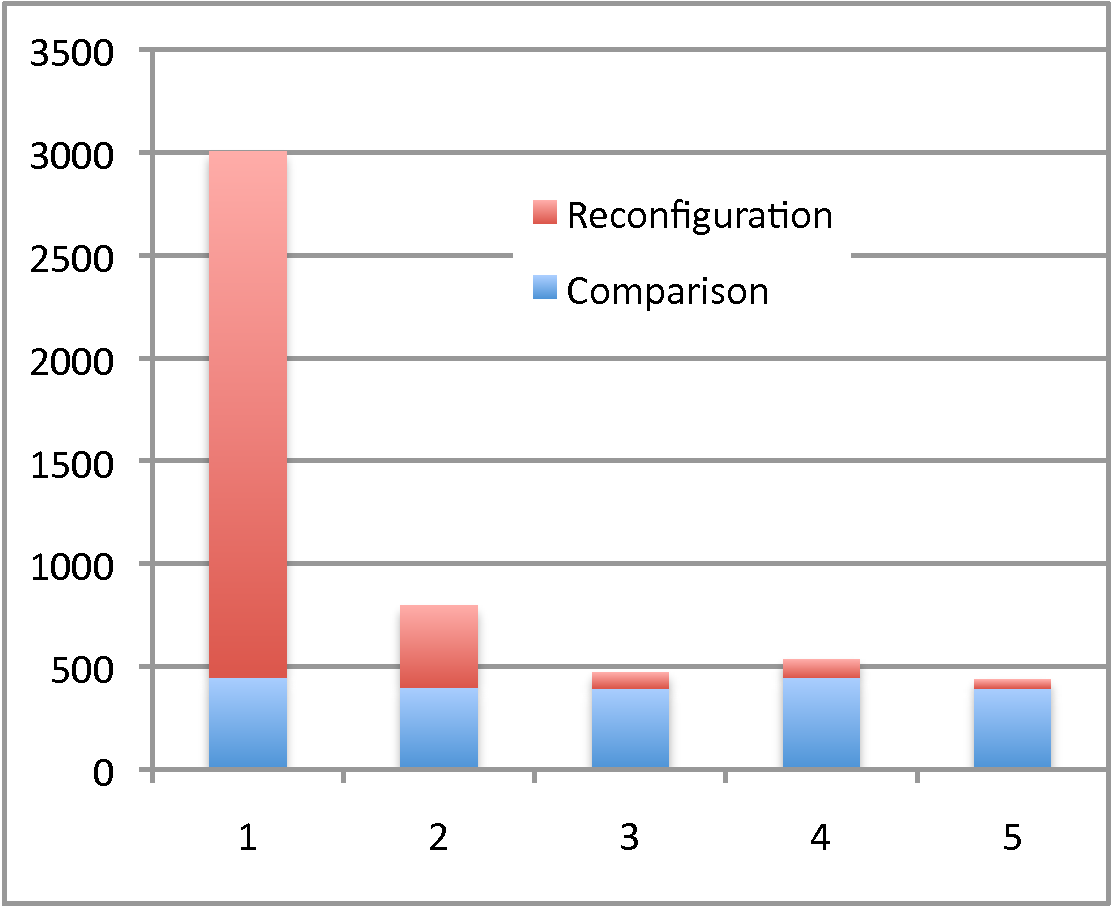


Fig. 7.6: Time (in ms) spent in Configuration Comparison and Actual Reconfiguration

As the scenario has been considered in the worst case, the system is initially empty. Starting from scratch, all components need to be deployed during the initial configuration. In particular, all the component types have to be downloaded, and checks have to be performed on the entire model. It explains the rather long reconfiguration time of step 1 : 2.5 seconds.

The first reconfiguration (day night, step 2) implied the deployment of an instance of RMG4S and the creation of bindings. As this component has never been used before, its component type is not present and has to be downloaded. All other components already deployed are reused. The downloading and deployment of the component type, plus the instance creation and its bindings to the dispatcher, are realized in less than 400 ms.

The next 3 reconfigurations (night day night) are much faster. Step 3 simply consists in unbinding and removing the RMG4S component. Step 4 is similar to step 2. Component types are not uninstalled. The RMG4S component type is thus immediately available for an instance creation. Step 5 is similar to step 4. The actual reconfiguration time of these steps is less than 100 ms.

For each reconfiguration, a model comparison is performed by the model@runtime engine, prior to the real deployment. This comparison detects changes, and creates the commands’ sequence for the transition. This model comparison takes an almost constant time of 400 ms. Executed before the actual reconfiguration, the comparison delays the reconfiguration of the system, but does not impact the duration of dynamic reconfiguration.

## 7.7 Openness issue

As presented in section , the system was required to expose the devices on both upnp and dpws networks. The next two sections describe how components have been automatically wrapped to be available on these networks. They report an actualized version of the work presented in [**Erreur ! Source du renvoi introuvable.**].

### 7.7.1 UPnP export

UPnP [**Erreur ! Source du renvoi introuvable.**] is based on a discovery-search mechanism. As a UPnP-Device joins the UPnP network, it sends an XML description file to all UPnP-ControlPoints. This file presents the device with information such as manufacturer, device type, device model, or its uuid. Most of times a UPnP-Device is self-contained.

It is able to describe itself, and the services it publishes on the network. The description structure, visible on the upper part of figure , organizes as follow.

UPnP specifications allow devices to contain other devices (called embedded devices). In this case, the container (called the *rootDevice*) takes the responsibility for publishing information about itself, and each device is embeds.

Each service a device can offer has to be described in a separated file. This file characterizes all the UPnP-Actions the service render, and all the UPnP-StateVariables used by these actions. UPnP-Actions can admit parameters. These parameters have a direction (in or out), a name, and a related StateVar. UPnP-StateVars handle information such as value types, or lists of allowed values for a parameters.

#### Environment of the test

This experiment required some devices to be deployed on the runtime for them to be exported. We choose to fix the system in the night state of the previous experiment, thus with a maximum of devices present in the system.

This test also required a third party tool to act as a UPnP external control point, as the touch-screen brought by Mrs P.’s son. Since EnTiMid has been deployed on the MSI Top, we made use of a toolkit from Intel : "Intel® Tools for UPnP Technologies (Build 2777)". This toolkit is no longer maintained, and only available for Windows.

#### Protocol of resolution

**Mapping UPnP devices to EnTiMid devices**

Not too far one to each other, EnTiMid devices and UPnP devices are not exactly aligned in their structures. However, the mapping (blue arrows in figure ) has been quite easy to conceive. EnTiMid devices have naturally been mapped on UPnP devices.

EnTiMid devices can provide two kinds of ports : services and messages ports. *NB : only input ports are considered here*. Services ports are composed of operations. This kind of ports has been associated to their UPnP equivalent, namely Service for the port, and Action for the operations. The closest UPnP element, to handle message ports from EnTiMid, is the concept of *Actions*. Indeed, a message port provides only one service/action. As a consequence, as many services as there are message ports are created. Each service proposes a single action, which connects to the message port.

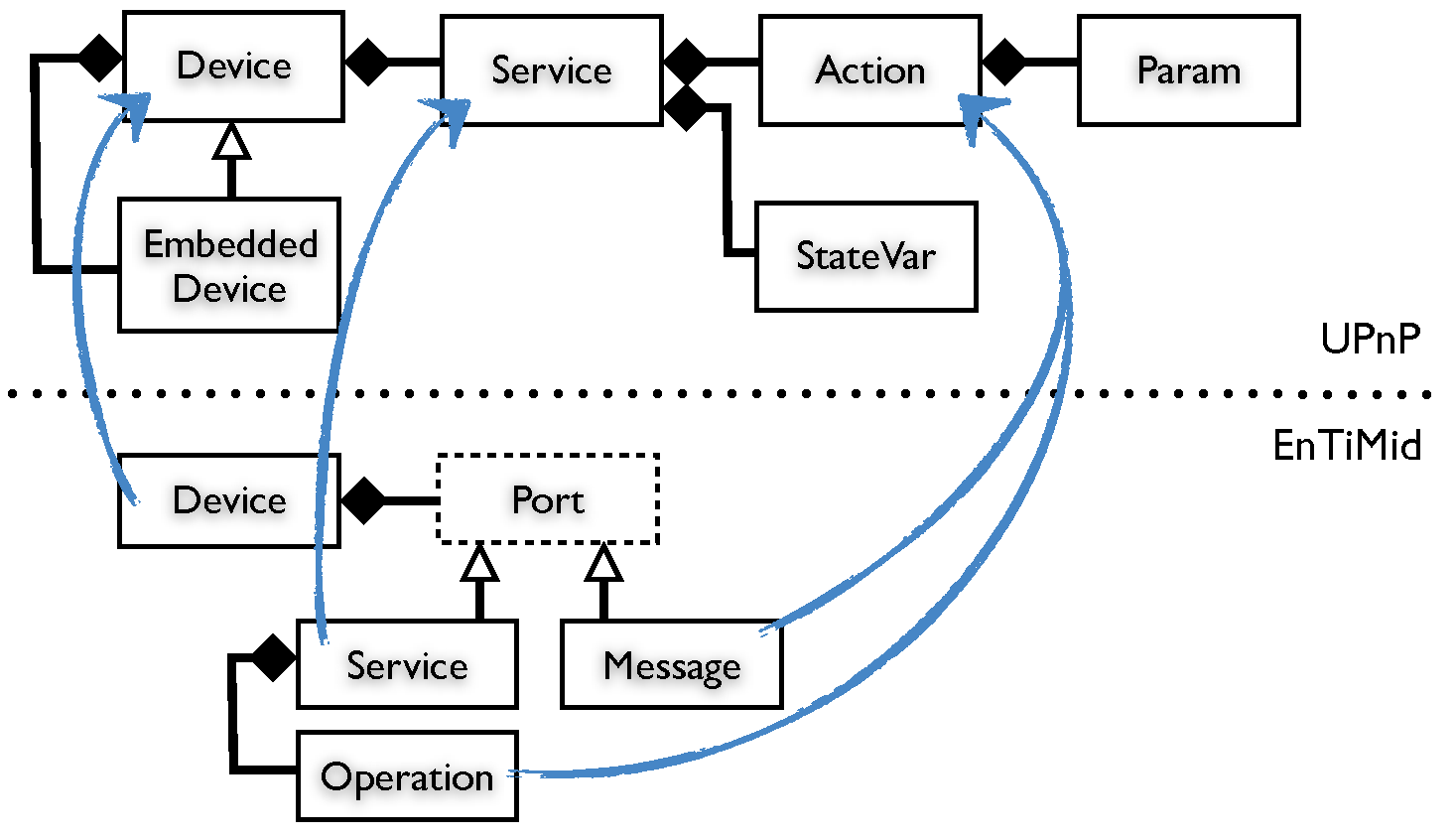


Fig. 7.7: Mapping UPnP-EnTiMid

**Generation of description files**

In EnTiMid, each component is described by a model. The model is a graph of objects at runtime, and is serialized in an XML file. The generation of the XML description file for UPnP is thus made really intuitive and simple.

The services descriptions are the first files to be generated. For each EnTiMid service, a new file is created. Operations of the service are then described in the UPnP formalism. Not all operations are described, due to a sometimes complex translation between EnTiMid and UPnP runtimes. Anyway, all simple methods are included in the description. In the same way, a new service file is created for each message port. The service takes the name of "<port>on<component>" for the services to be differentiated. Each service of this kind is composed of a unique action which name is identical to the one of the port.

In a second time, a separate description file is created for each component available on the system. The description of a device in UPnP is threefold. The first part provides general information about the product like the name of the manufacturer, a brief description, the model type, the model number, or its serial id. If not available in the model, these elements are automatically generated, or completed with default values. The second part contains the list of the services a device provides. This description makes reference to the service description files previously generated. The last part contains information about the embedded devices. For each of them, the two previous parts have to be specified.

When a UPnP-ControlPoint starts, it broadcasts a query for descriptions of devices available on the network. The UPnP wrapper the sends in response the description files of all devices and services provided.

**Runtime elements**

The generation of description files is necessary, but not sufficient for UPnP queries to be forwarded to the real device. Indeed, no connection has been realized between the real runtime component standing for the devices, and the UPnP network. To cope with this question, virtual abstract EnTiMid components are created at runtime. Each real device exported on the UPnP network is linked to an abstract component responsible for the handling of communications between the real device, and the network requests. These abstract components only have output ports. *ie* : an output port is especially created on the abstract device to be connected to each input port of the real device exported on the UPnP network.

Then, queries are routed by the UPnP exporter to the abstract component in charge of the concerned device. The abstract component activates the input port of the real device according to the request.

#### Results

Once the wrapper deployed on the MSI Top, the Nabaztag rabbit, the 3G usb stick and the RMG4S were all made available through the UPnP network this way. Indeed, we have been able to view and act on these devices from a remote PC equipped with Windows, and the Intel UPnP toolkit.

The poor number of tools that accept the discovery of self-describing UPnP devices can be a limitation for the use of this wrapper.

### 7.7.2 DPWS export

The dpws [**Erreur ! Source du renvoi introuvable.**] defines a minimal set of implementation constraints, to enable secure Web Service messaging, discovery, description, and eventing on resource-constrained devices. Its objectives are similar to those of upnp. The difference is that DPWS is fully aligned with Web Services technology, and is designed to work upon a web-service transportation protocol. It also includes numerous extension points, to allow for seamless integration of device-provided services in enterprise-wide application scenarios.

From a conceptual point of view, the DPWS structure is close to the UPnP one, described in figure . Consequently, the mechanisms to map EnTiMid devices and their dpws representative, follow the same idea. Nevertheless, the generation process is different. Publications to the DPWS network have been realized thanks to the WS4D project [**Erreur ! Source du renvoi introuvable.**]. In their approach, each DPWS compatible device has to extend an abstract DPWS device, proposed by the framework they provide. The reason is that this abstract component handles all web-service specific communication concerns. The creation of virtual component is not sufficient in this case. Source code has to be generated.

#### Environment of the test

Since UPnP and DPWS are very close in terms of needs for devices to be exported, the same set of devices has been selected for this experiment.

As for UPnP, we made use of an external tool to check that the export of devices has been made properly. The experiment was realised using a second PC equipped with a DPWS explorer[[3]](#footnote-1) to list and act on published devices.

#### Protocol of resolution

**DPWS devices creation**

For each device, service and operation, a Java class has to be generated. According to the element they represent, classes must extend *HostedService* for services, *HostingService* for devices, and *Action* for operations. Parameters are instances of the class Parameter. Luckily, all these classes still can be generated with an automated process. To achieve the code creation, the JET Framework has been used. Templates of DPWS files have been set up, and they are used at runtime to produce Java classes.

More than simple Java classes, the generated files are also implementations of new component types. These types are the wrappers of real devices for DPWS. These components are thus responsible for the direct connection between, model elements, and DPWS controllers.

**Compilation and use**

The generation process produces Java classes, but no binary code. These classes still have to be compiled to be useful at runtime. The decision has been made to embed the JDT compiler provided by Eclipse. As a result, the bundle to export devices through DPWS is a bit heavy. The compilation is also resource consuming for a quite small computer device. Once compiled, these classes are packaged into a bundle, which is then deployed on the OSGi runtime.

Classes are then handled just as classical components. The tool asks for new components to be added in the runtime, and they are bound to the device they export.

#### Results

Obviously, this is not optimal but it works, since we were able to see all devices and act on them using the DPWS Explorer tool. Other solutions involving more powerful servers in charge of the transformation from a model representation to a component type bundle have been proposed, but not yet completely realized. Anyway, the principle is just to extract a tool chain that already works.

## 7.8 Threats to validity

### 7.8.1 Validity of the scenario

The scenario of experimentation, even defined in collaboration with several actors in the domain of AAL, may not consider all cases. The interface between people coming from a technical field, and people coming from social field is quite difficult to find. Because people in social activities are not aware of what is possible to do with technologies, and industrials are not aware of every day problems the dependency of persons can pose, the discussions can rapidly come to a dead end.

The scenario validated for this study was accepted by every part, but may be limited by the comprehension each part had of the problem.

### 7.8.2 Variability management

The variability management, described as being an important problem in home automation for assisted living is not addressed in this experiment. Indeed, the scenario considers a unique deployment, for a single person, in a single home. A second round of definition of a more global scenario may have stressed this requirement for variability management.

Nevertheless, several works have been realised to try to cope with that issue, such as an approach using Aspect Oriented Modeling presented in [**Erreur ! Source du renvoi introuvable.**]. Other perspectives to address this questions are presented in section **Erreur ! Source du renvoi introuvable.**.

### 7.8.3 Scalability

For the same reason, the scenario did not stress any question about distribution, and scalability of the solution for a deployment at the scale of a town, or even of a country. The contribution of this thesis, and its validation, could have been different if a real deployment had been required.

The MSI Wind Top, on which the experiments have been led, may not be the unique platform to test EnTiMid on, and a large-scale vision scenario would have highlighted this.

### 7.8.4 Safety and Security

Voluntarily, we decided to abstract from access security and privacy considerations. Not because they are not essential, but because they impose such heavy constraints that the research of technical solution may have been compromised. Now that the system is clearly designed, and that the proof of concept has been validated, a work for securing communications and data has to be realized prior to any real deployment.

About the safety, our experimentations did not require complex checks on models. Only simple structural checks, to find cycles for instance, have been implemented and used. Many checkers had not been completed since they were related to the business targeted. Their complete definition would have been useless in the context of the experiment. In case of real deployments, they have to be completed to verify that no configuration marked as failing is asked for deployment.

### 7.8.5 Communications with smart devices

Gateways are essential for us to be able to communicate with smart devices. For instance, the bi-directional communication with Delta Dore devices has been made possible by a R&D product. Otherwise, the *TYDOM 350*, an embedded web server, is the only device they commercialize to act on their devices. This one only enables to act on devices through a web page interface. This product does not detect any event on the X2D network, it just acts on devices (with no acknowledgement by the way).

EnTiMid is not able to use any device without means of communication with it.

## 7.9 Conclusion

The validation of the core elements of contribution of this thesis suffered from the lack of a real deployment. However, an experimentation using real devices has been set up to evaluate EnTiMid on a scenario as realistic as the collaborative work of project’s partners could have made it.

EnTiMid successfully passed the main requirements stressed by the scenario, and required for such kind of systems to be deployed one day as integration platforms offering customized solutions for each person’s needs.

Some limitations due to the lack of large scale deployment has been highlighted. These limitations will probably be addressed by the project of company in charge of the promotion of this technology in the industry.

1. http ://www.ida-autonomie.fr/ [↑](#footnote-ref--1)
2. http ://calimero.sourceforge.net [↑](#footnote-ref-0)
3. http ://ws4d.e-technik.uni-rostock.de/dpws-explorer/ [↑](#footnote-ref-1)