Fpt partie IV

Conclusion and Perspectives

*The law of unintended consequences pushes us ceaselessly through the years, permitting no pause for perspective.*

Richard Schickel

This last part wraps up the thesis. The first chapter summarises by coming back on context and requirements, recalls the contribution, and emphasis its adequateness with relation to the requirements. After what, a short section discusses benefits and limitations identified in this thesis.

The second chapter of this part shapes some perspectives of scientific development for this contribution, while the last chapter presents industrial perspectives.

**Chapitre 9**

# Conclusion

This chapter globally synthesises the work realised for this thesis. Starting from the requirements, this chapter walks through the contribution, discusses its appropriateness to the context, and ends by highlighting some benefits and drawbacks.

## 9.1 Recall of context

The ageing of the European population incited the community to search for solutions to support this evolution. In this context, several issues have to be addressed in parallel. First, the domain of health care suffers from a manpower shortage, that could result in a decrease of the health service quality. Then, places in health care centres are not indefinitely extensive, and centres will shortly reach their maximum capacities. Finally, a day spent in health care centres, or hospitals, is quite expensive and funding is limited.

Several projects have been started to try to address these issues. The Ambient Assisted Living(AAL) joint program has been created to promote such kind of projects, and emphasis the interest of Europe in advances in this domain. The *Innovation Domicile Autonomie(IDA)* project, initiated by the metropolis of Rennes, fits into this scheme with an evaluation of how the use of Information and Communication Technologies(ICT) can help to cope with these problems.

After a precise assessment of elderly people’s needs, this project measured the adequateness of several industrial solutions, to help and support elderly people at home. Among others, home automation technologies have been analysed to work out their possible contribution to this problem. Rapidly, the survey demonstrated that a unique solution can not be applied in all cases. Each person has different needs and requirements, which imply the solutions to be adapted for each deployment. Also, manufacturers reach their limits when a device has to be specialized for each user.

Technical solutions designed in this context require some software systems to bridge the gap between, mass market home automation devices, and customized solutions. To meet the needs, these software systems must cope with several requirements.

## 9.2 Summary of requirements

**Interoperability** is the first requirement software systems have to cope with. Indeed, solutions proposed to improve elderly people’s comfort at home may be composed of multiple products, from different manufacturers. Each device taking part in a solution addresses a particular need of the person, and makes the solution closer to the ideal one. Anyway elements of the solution have to communicate with each other to render a global service, but the diversity of manufacturers makes the interoperability of devices a real problem.

The definition of a common communication interface for all components of the system could solve the problem, but it requires all devices to be re-engineered to implement this communication interface. With this approach, all products already available can not be used, because they will never implement this interface. Since the solution must not be limited in terms of usable products, it avoids the definition of a global communication interface.

**Adaptation and Evolution** are the two main concerns to deal with in this domain. Software systems dealing with objects, or services, linked to everyday life actions, have to take into account the environment in which they are executing. They should be able to dynamically adapt to changes while running, in order to maintain a level of services, or functionalities. Obviously, these adaptations should not require any restart of the system that would disable all functionalities for the time of restart.

Needs, uses, protocols and technologies are changing. Some functionalities may finally be required, whereas others can become useless, and need to be uninstalled. Security or communication protocols can be improved, and deployed in new versions that have to be taken into account without needing to re-implement the entire system. Software systems must be ready to accept future and unforeseen evolutions, such as the installation of new services/functionalities.

**Openness and Remote Control** are intended to make all functionalities of the software system available for third party applications. Indeed, the connection of some products may require the system to be accessible through a specific application protocol. This availability offer means to connect to the system with no need to be aware of its internal organization. The only interest is on using the functionalities or devices. It is an open door for new unforeseen appliances, and to external contributions. Each one can take part in adding a smart behaviour, on top of a reliable set of functionalities.

The remote control may be required for a carer to remotely check if everything is doing well. It may also help in supporting a remote assisted management of the home for instance, or to remotely deploy new appliances or maintain the system.

**Variability Management and Distribution** issues are linked to the need for customization of solutions, and to the dispersion of deployments. Since solutions for elderly people have to be customized to best fit to their needs, the set of options of the system may become hardly manageable. Some deployments may require high computation power, whereas others may run on tiny execution platforms. Moreover, evolutions of services or protocols may not be applied to all running systems at the same time, making even more complex the management of versions. All these elements lead to a huge number of configurations, and to a complex variability management.

Software systems have to be aware of these problems, and offer tools in order to help system designers and technicians. Decision helping tools should be created to support the design of software systems based on requirements, and available devices.

**Safety & Security** is a very important concern for home automation systems. It is even more important when the system tends to improve the quality of life of dependent people. A minimum service level has to be guaranteed, for inhabitants not to stay stuck in the house in case of emergency for instance. Moreover, accesses to the system have to be secured to disallow anonymous controls, but not annoying for authorized people in a normal use case.

**Acceptability & Accessibility** are issues that must be addressed, particularly when a software system takes responsibility for a part of the home management. The aal domain is a complex environment in which solutions must support the activity of elderly people, and help carers in their work. For both of them, the system must not be perceived as a new constraint, or considered as stigmatizing. People must accept the solution deployed in their home, and have to be ensured to keep a hand on things that can happen.

## 9.3 Survey of existing approached

Among all requirements listed in section , the survey of existing approaches concentrated on interoperability, adaptation, evolution, openness and variability management.

The scientific literature abounds with proposals using different approaches to cope with interoperability, adaptation or remote control concerns in several applications. Generally, service-based propositions sound helpful in targeting interoperability of devices, but clearly lack of descriptions of the running application once deployed. They also bring essential ideas to properly handle the sporadic apparition of elements, since a service can be started and stopped at any time.

Component-based architectures provide an ideal abstraction level to meet the requirements for a virtual representative of home automation devices. However, components’ ports are often defined by an API. This strict definition may disallow some connection unforeseen at design time, without the help of *ad-hoc* connectors.

Components for SOA is certainly the best approach for our concerns, since the benefits from the first balance the drawbacks of the other.

Transversally to any approach, model driven engineering methods and techniques, come with a lot of tools for virtual elements manipulations. They seem handy for runtime management of devices, for the description of software systems, and for the variability management.

All the approaches, tools, and frameworks considered in this survey have been reported in table . This table synthesizes strengths and weaknesses of each approach with relation to the requirements identified.

## 9.4 Outline of the contribution

Inspired by electronics’ achievements this thesis contributes in improving the flexibility of software systems, while maintaining a high level of reliability. The contribution is threefold.

(1) A new component model, which improves flexibility to enable connection of heterogeneous components.

(2) Tools from model driven engineering, to create, edit, simulate and validate the structure and behavior of component assemblies, prior to their (re-)deployment.

(3) A runtime environment built on top of a service-based architecture, to support evolutions, adaptations and openness required by the proposed component model.

The implementation of this contribution called EnTiMid is composed of several elements. Each of them, presented as a layer, addresses a particular concern of the global problem.

**Device Interoperability** takes responsibility for the communication with real devices, and between their virtual representatives in the *Component Model* layer. A mix of drivers (to connect the real to the virtual world), and asynchronous message-based communications, enables the connection of components previously marked as not compatible.

The **Component Model** layer brings up the necessary structures and methods to handle the virtual representation of real devices. It provides a unified description of possible actions, and available information, using the paradigm of ports. In this model, ports can be of two kinds : synchronous (service ports) or asynchronous (message ports). This component model helps in having a detailed view of components, with precise information for the *Model@Runtime* layer to work properly. Tools have been made available to ensure the synchronization of models and implementation codes of components.

**Model@Runtime & Checkers** layer entails necessary tools to ease the management of the system. Specificities of components’ implementations are invisible at this level, thanks to the *Component Model* layer. Simulations and checks can be safely performed at this level of abstraction, with no consequences on the running application, since the model view is synchronized but independent from the execution. *Model@Runtime & Checkers* contribute in enabling management of the system at runtime, in offering tools for checks and validations, in improving the safety of the solution, and help in dealing with the variability of the system.

**Wrappers** layer takes responsibility for publishing the devices, present in the system, on application level networks. This ability opens our solution to existing and future protocols. Often too heavy to be embedded in basic devices, this layer offers all devices to be available on application level protocols for free.

**Service Oriented Runtime** comes to complete the contribution by providing an execution environment for the new component model. It brings life to the *Model@Runtime* by supporting dynamic *adaptations* and *evolutions* while running.

## 9.5 Adequateness of the contribution

The aal context, the home automation domain description, and the state of art, led to an extraction of a list of requirements. These requirements have been stressed as essential abilities a software system should be able to provide to be used in this context. The table  recalls the table presented in chapter **Erreur ! Source du renvoi introuvable.**, in which all existing approaches were presented, and where their participation with relation to each identified concern had been reported.

The last line completes this table with EnTiMid, and shows that it fits most of these requirements.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Interoperability | Openness | Adaptation | Evolution | Variability Management | Remote Control |
| 25mm |

[Sorry. Ignored \begin{sideways} ... \end{sideways}]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Habitation | + | - | - | - | + | - |
|  | Dia SUite | + | + | - | - | - | + |
| 45mm |

[Sorry. Ignored \begin{sideways} ... \end{sideways}]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Darwin | - | - | + | + | - | - |
|  | Koala | - | - | - | + | + | - |
|  | Fractal | - | - | + | - | - | + |
|  | uMiddle | - | - | - | - | - | + |
| 45mm |

[Sorry. Ignored \begin{sideways} ... \end{sideways}]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Hydra | + | + | + | - | - | + |
|  | JBI | + | + | - | + | - | - |
|  | OSGi | - | - | + | + | - | - |
|  | SOPRANO | - | - | + | + | - | + |
| 45mm |

[Sorry. Ignored \begin{sideways} ... \end{sideways}]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | SCA | - | + | - | + | - | + |
|  | FraSCAti | + | + | + | + | - | + |
|  | iPOJO | - | - | + | + | - | - |
|  | Niagara | + | + | - | - | - | - |
|  | **EnTiMid** | **+** | **+** | **+** | **+** | **-** | **+** |

Tab. 9.1: Adequateness of the contribution

The Device Interoperability layer, helped by the Component Model, addresses the interoperability requirement. The openness is ensured by wrappers, for application level protocols, and by drivers for future manufacturers. Adaptation at runtime, and evolutions, are made available by the use of Model@Runtime techniques and the OSGi runtime for supporting the implementation of this contribution. The variability management is simplified by the presence of a model, but tools are still insufficient to properly cope with this issue. Finally, remote control is possible thanks to 1) the model@runtime for remote adaptations or evolutions, 2) the wrappers for remote application level control.

## 9.6 Conservativeness

During the state of the art survey, some good properties had been identified. The contribution of this thesis is conservative with relation to these properties.

**Reflexive Model** proposed by mde is still available. The model@runtime layer is responsible for this ability, and keeps synchronized an explicit and independent model reflecting the architecture living at runtime. This model allows for the creation of reasoners, able to perform changes on the model, with no immediate impact on the running system, until the model is conform.

**Externalized coupling** is provided by the model@runtime, by enforcing the elicitation of links between components, and by the presence of drivers, which can not presume of any use of the components. This externalized coupling makes it possible for reasoners to dynamically change component connections, and even components directly. The enforcement of the component independence requirement, to allow interoperability, is also taking part in ensuring the externalization from the component implementation.

**Hot deployment** is natively supported by the OSGi runtime execution environment used for EnTiMid. Indeed, for reasoners to be able to adapt the system with new components, or for evolutions to be easily deployed, EnTiMid had to preserve this property in the final solution.

**Close Isolation enforcement** is imposed by the device interoperability layer, and the entire EnTiMid system. Types and instances are handled separately, and all instances have independent life cycles since real devices can evolve independently.

**Openness** was identified as a good property, but also as a key requirement, which has been addressed in this contribution, and explained in the validation.

## 9.7 Immediate benefits

### 9.7.1 Development of components made easier

The tools coming with this contribution are making the development of components easier. The component model forces developers to respect properties such as close isolation, or externalization of component couplings, and makes the maintenance and the evolutions easier. The use of annotations in the component code, and the availability of a code generator, are also simplifying the everyday work of component developers. The code generator and the synchronization mechanism, bring significant gains in terms of prevention of errors, time consumption, and shorten the time elapsed from a new requirement to the solution.

### 9.7.2 Simple creation of applications

Thanks to the component model and modeling tools, the creation of applications is made very simple. Libraries of components can be imported in editors, and components are assembled and connected using drag-n-drop interactions. If checkers authorize, connections from port to port make it possible to connect any port to any other, whatever their roles or actions.

Designers of home automation device are already familiar with this way of connecting things, since the model is very close to the electronic components. The conception of a solution customized to meet a person’s specific needs is thus quite quick and simple for engineers and technicians.

### 9.7.3 Sustainability and precision

The adaptation and evolution abilities of EnTiMid, improve the sustainability of deployed solutions. Present by default in the system, they offer the support for evolutions of both technologies, and elderly people’s pathologies, with no need to change the entire system. This way, a software system created to meet specific needs at a time, can be changed with a very limited cost, which makes the solution always precise and sustainable.

### 9.7.4 Seamless integration of IoT and IoS

The new component model enables the connection of heterogeneous components not designed to be working together. The heterogeneity can be due to the difference of manufacturer, protocol used, or media used, but can also be due to the thing they represent. Several services available through the Internet have been wrapped into components, to enable the application to access to a service such as a on-line calendar, a weather service, a picture sharing service, and even a famous social networking service. The component model enables to seamlessly connect a Google Calendar from internet to a light component for instance. The effect of such a connection could be to switch off the light where no appointments are specified.

## 9.8 Limitations identified

### 9.8.1 Behavioral description

The component model eases the structural description of a software system, where people are much familiar with the description of its behavior. In addition to the component model (thus the structural description), it would be helpful to have a second tool to check and describe the behavior of the system. In this condition, an end-user could be able to change the behavior of the system, without dealing with the structural description.

If the answer has not been provided yet, it may be because the problem is not simple as soon the behavior of the system can be described in several pieces. Indeed, thinking in a functional way makes people define how the system must behave when the door opens, or when an alarm is triggered, but with no consideration about the consequences on the global behavior. Moreover, the structural and the behavioral descriptions have interactions with each other.

Lastly, as non-experts of the domain, the description people can make of how the system must behave never takes errors or failures into account. From a linear description, erroneous paths have to be guessed and tested [**Erreur ! Source du renvoi introuvable.**].

### 9.8.2 Port parameters

Classical component models have been excluded in this thesis, because of the too strict specification of ports, making it impossible to connect two ports if their APIs are not aligned. But the problem of alignment has not completely disappeared in our component model. It has been moved from the implementation to the virtual representative of each component. Thus, the alignment of parameters passed through the ports has to be solved at the model level, before the deployment.

If this problem has not been addressed in this thesis, it has already been identified and scientists already proposed some solutions under terms like component connectors, which can have complex behaviors (cf Beugnard et al. in [**Erreur ! Source du renvoi introuvable.**]). Mechanisms such as renaming or mappings of parameters could be implemented to cope with this issue.

### 9.8.3 Too light checkers

Our experimentations did not require complex checks on models. Only simple structural checks, like cycles detections, have been implemented and used. Many checkers had not been completed since they were related to the business targeted. In case of real deployments, they have to be completed to verify that no configuration marked as failing is asked for deployment.

Realized at different steps, the checkers are different, and they address various aspects of the application. Thus, the information, required to be able to perform each check properly, depend on what has to been checked. Since none had been completely implemented, the information currently available in the model can be too poor for checkers to work.

### 9.8.4 Variability management

The variability issues have not been completely addressed, since a small set of components is sufficient for testing. Also, the number of components in the ida experimentation was small enough to be handled by hand. In the perspective of real deployments, the variations of configuration will impose the creation of tools to help in facing this huge variability.

### 9.8.5 Not tested on embedded platforms

In the context of the ida project, the choice has been made to deploy EnTiMid on a touch-screen PC. This PC has a high computational power, and lots of memory compared to some more embedded platforms. But the deployment of a touch-screen PC may not be necessary in all cases, and some more embedded devices may be sufficient to automate some tasks. Anyway, the runtime of EnTiMid requires a Java virtual machine to be deployed on the platform first, and thus enough power for the JVM to run.

## 9.9 Contribution to the S-Cube NoE

The contribution of this thesis is aligned with the Work Package 1.2  : *Adaptation and Monitoring Principles, Techniques and Methodologies for Service-based Systems* of the Joint Research Activity(JRA) 1  : *Engineering and Adaptation Methodologies for Service-based Systems*

The general objective of the JRA-1, is to "devise an integrated set of principles, techniques and methodologies for engineering, adapting and monitoring hybrid service-based applications, while guaranteeing end-to-end quality provision and SLA conformance", according to the S-Cube description of work[[1]](#footnote--1).

This thesis provides a new component model that : implies new engineering techniques and methodology, enables the adaptation of hybrid service-based application, and offers means to perform checks and verifications to ensure the quality of services.

More precisely, the contribution of this thesis takes part in the JRA-1.2 work package, which aims to define novel principles and techniques for cross-layer monitoring and adaptation of Service-based Applications. If EnTiMid does not address monitoring issues, it actually copes with adaptation requirements.

From the S-Cube perspective, EnTiMid can be considered for handling adaptations of the infrastructure, or of the composition and coordination layer(see figure **Erreur ! Source du renvoi introuvable.**). Coupled with other layers, it can take part in the cross-layer adaptation mechanism.

1. DoW Amendment 4, December 6th, 2010 [↑](#footnote-ref--1)