Fpt partie II

Thesis and achievements

*A Scout is never taken by surprise.*

*He knows exactly what to do when anything unexpected happens.*

Sir Robert Baden-Powell

The study of the state of the art in software engineering highlighted the lack of a software solution to address all requirements identified in the context of home automation for Ambient Assisted Living. According to this observation, the goal of this thesis is to fill this gap by providing sucha tool.

This part presents the achievements of this thesis. While the first chapter gives information about some leitmotivs that animated this work, the remainder of the part details the contribution.

**Chapitre 5**

# Contribution

## 5.1 Global ideas

All along this thesis work, some recurrent ideas drove both the research of solutions and the development of the proof of concept.

### 5.1.1 Get Inspired by electronics

Variability, interoperability, and adaptation, are qualifiers that appear regularly in electronics. Indeed, any integrated circuit has to be able to operate with any other. Signals exchanged between components may have to be adapted, in order for the signal to reach the shape required by the receiver. Various solutions using many different electronic components can be envisioned to fulfil a need. People in this domain had to find and deploy tools, such as data-sheets or simulators, to get rid of these constraints.

This thesis strongly inspired of this domain to come to a solution. The link between electronics’ solutions, and the contribution of this thesis, is stressed all along this part.

### 5.1.2 Make it possible

Scientific discoveries, and advances, are often due to hazardous reactions, unpredicted situations, and even due to errors. Software engineering tools of today, limit runtime failures by cutting down design elements to a set in which interactions are well known. This mode of protection makes it difficult to design new systems, by using existing components in an unexpected way. Research or engineering phases must not be limited by these concerns.

This discussion can be compared with debates about static or dynamic typing in programming languages[**Erreur ! Source du renvoi introuvable.**, **Erreur ! Source du renvoi introuvable.**] : where static typing brings safety, it looses the flexibility of dynamic type systems.

This thesis work paid a strong effort in making validations highly customizable. Researchers are thus allowed to loosen checkers, in order to experiment new behaviors. They can make mistakes, experiment failures and adapt checkers with specific rules for their concerns.

### 5.1.3 Keep end-users in mind

Products are too often released to be sold, without end-users’ tests. As a result, these products may be considered too expensive or useless. From the beginning to the end, solutions brought by this thesis have been designed for targeted users. Tools and methods were adapted to reduce the gap between, how people are intended to behave, and the way they actually use the solution.

Two populations of users have been particularly considered. The first population is the engineers and technicians community, which require some tools to ease their work. Secondly is the system users population, like carers and elderly people, who just want to be able to interact with the system. In both cases, it calls for the tools and the system to be highly intuitive. Intuitiveness has been improved by presenting the system to elderly people, and by using the tools to design solutions.

## 5.2 Overview of the contribution

The contribution of this thesis is threefold. (1) A new component model, (2) tools to handle models, and (3) a runtime environment. To go into details, these elements are presented as interacting layers. Each of them targets a particular concern, and their synergistic collaboration makes the solution. The different layers are visible on figure , which provides an overview of the contribution.

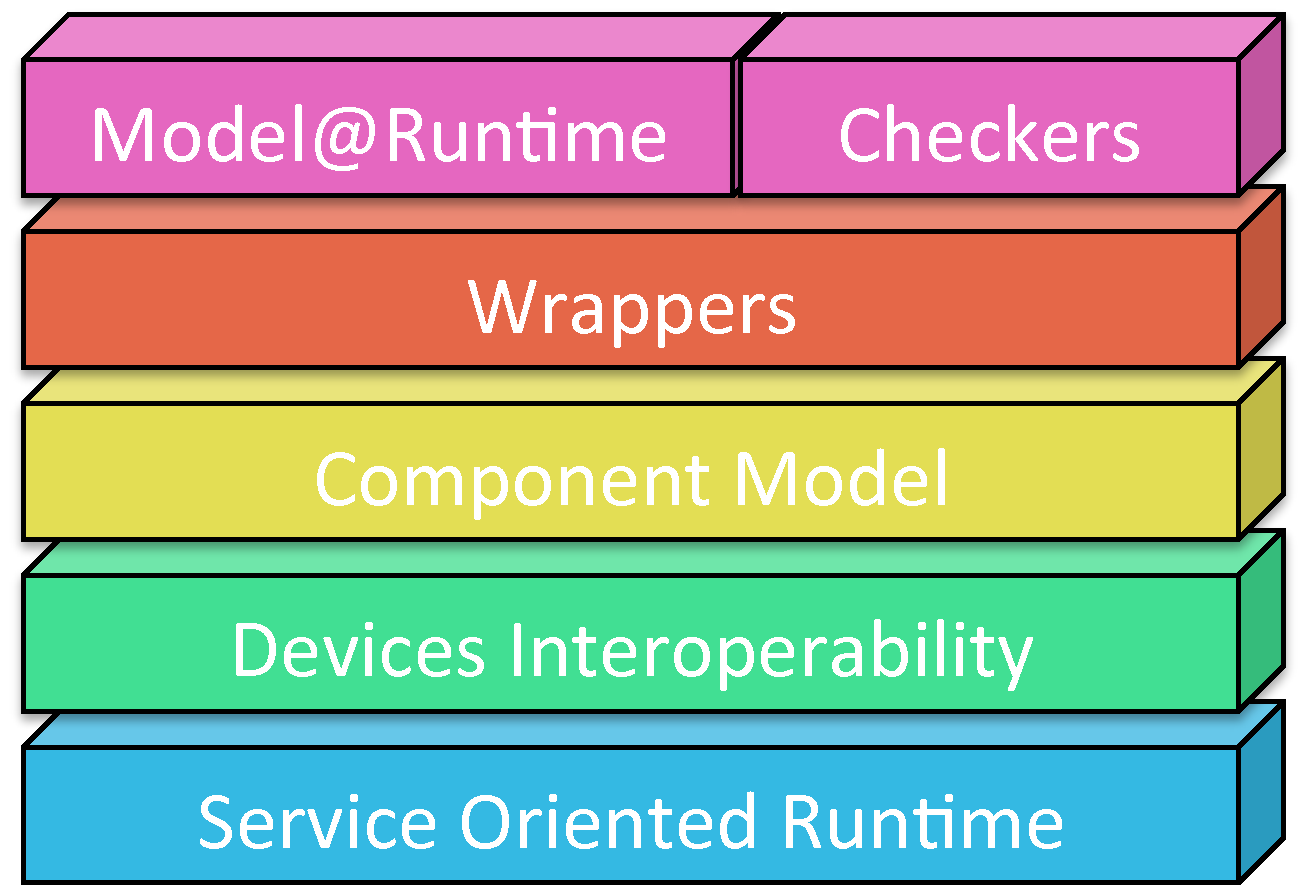


Fig. 5.1: Overview of the EnTiMid layers

**Device Interoperability** addresses the mandatory need of *interoperability*. It is responsible for the communication with real devices, and their representatives in the *Component Model*.

**Component Model** brings up structures and methods, to handle abstract representations of real devices. It provides a unified description of available ports, parameters, and any other useful information for the *Model@Runtime* layer to work in good conditions. It enables the creation of tools to cope with variability, interoperability and safety concerns.

**Model@Runtime & Checkers** layer entails necessary tools to ease the management of the system. Implementation specificities of components 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. Model@Runtime enables the management of the system while running, and helps in dealing with the variability management. Checkers offers tools for validation, and improvements of the safety of the solution.

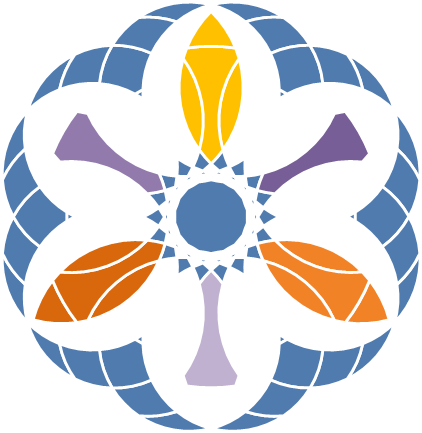
**Wrappers** layer takes responsibility for publishing the devices present in the system, on application level networks. This ability opens our solution to existing and future, protocols and evolutions. Often too heavy to be embedded, this layer offers the devices, for free, an access through application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

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

Each level participates in the answer to the requirements identified in chapter **Erreur ! Source du renvoi introuvable.**. Table  shows what concern is addressed by each layer. Separately, each layer does not satisfy all needs, but their collaboration does.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Interoperability | Openness | Adaptation | Evolution | Variability Management | Safety & Security |
| Model@Runtime |  |  | + | + | + | + |
| Wrappers |  | + |  | + |  |  |
| Component Model | + |  | + | + |  |  |
| Device Interop. | + |  |  |  |  |  |
| Service-Oriented Runtime |  |  | + | + |  |  |

Tab. 5.1: Mapping layers to requirements



This contribution has been implemented. The runtime, called EnTiMid, has been developed on top of an OSGi platform. By the way, EnTiMid is a compound word from the Britany "En Ti", which means "In house", and "Mid", for Middleware. It is thus, the middleware in the house. The component model has been realized using classical modeling techniques. Tools have been created to enable all functionalities.

Chapter **Erreur ! Source du renvoi introuvable.** details each layer of this contribution.