**Chapitre 10**

# Scientific perspectives

The contributions of this thesis leave some open questions, and opened some doors. So, the perspectives of this work aim at investigating and answering questions that have not been addressed yet, or go one step beyond into new uses of this contribution.

## 10.1 IDA, second phase

Conclusions from the experimentation led in the context of the ida project shaped a promising future for the use of such kind of tools, in the AAL domain. Even if the maturity of EnTiMid, and the objective of the first phase of the project did not permit to test EnTiMid in a real deployment, the protagonists (industrials, carers, social workers and elderly people) have shown an interest for the provision of customized solutions for each person.

Currently, the second phase if the IDA project is being set up. Hopefully, it will be the moment for EnTiMid to perform the last checks and to validate both the scalability and the variability management, on real deployment.

## 10.2 End User Programming

The End-User Programming [**Erreur ! Source du renvoi introuvable.**] relates to the ability for anybody to program something. For instance, when a user programs the hours of start and stop of the heating system, he is actually programming. In the context of the IDA project, and following the idea that inhabitant must be able to keep control on things in their homes, end-user programming sounds like very promising, but yet challenging perspective.

### 10.2.1 Which description language ?

Software developers like to be able to use the keyboard only. A graphical user interface, with drag-and-drop interactions to create assemblies, will probably not meet the requirements of this kind of population. For them, a textual language seems to be the simplest thing.

On the other hand, inhabitants do not all have skills in programming languages, and especially not elderly people. They probably would express their requirements for the behavior of the system in another way. The question is which one.

At our disposal, a range of description tool from a textual domain specific language, to a visual language composed of icons and boxes, linked with arrows. A solution for this problem probably stands somewhere in between these two extreme propositions, and is surely not unique. Indeed, for the same system, an elderly person will probably be lost in a textual language, and an engineer may be frustrated to be unable to express himself as usually.

The validity of the behavior described is also challenging. End-users may not have a global vision of the system and thus, may ask for a behavior that could lead the system to a failure. Secondly, people naturally express the nominal behavior, without taking care of possible deviations of this behavior. To address this problem, tools have to be proposed to check the validity of the nominal behavior, and track and check any possible variation in the scenario.

The unique and universal language for describing how things have to behave will probably never exist. Because each user has different kinship with technologies, systems should offer several edition tools, out of which the end-user selects the handiest for him.

### 10.2.2 Fuzy Logic and Learning algorithms

In the hypothesis people are able to describe a behavior of a function, how could they now about bound of values. In other words, if a user is defining a behavior for the light, how could he know the minimum and maximum values the light sensor can sense ?

The fuzzy logic paradigm [**Erreur ! Source du renvoi introuvable.**, **Erreur ! Source du renvoi introuvable.**] proposes to use terms and non-fixed values in decision algorithms. Indeed, a fixed value is never appropriate because a regulation value must be modifiable. This paradigm makes it possible to work with terms and rough values only, because thresholds are computed at runtime. During the execution, users can act on these thresholds by telling the system about good or bad situations. Quite close to ideas of artificial intelligence, this approach could be coupled with some learning mechanisms, to go a bit further and simplify again the description of an application behavior.

## 10.3 Distribution and Pervasiveness

The distribution of applications brings several interesting facilities, such as load balancing, or redundancy to cope with failures of system elements. This question has not been properly addressed in this thesis, but may rapidly become a limitation. Moreover, working with devices brings EnTiMid close to ideas of pervasive computing. In this domain, objects interactions are controlled by invisible nested software systems. Invisible for users, these systems have to self-reconfigure to take into account changes in their environments.

In the perspective of a large-scale deployment, distribution and pervasiveness can both come out as key requirements for some deployments. In [**Erreur ! Source du renvoi introuvable.**], Devescovi et al. propose algorithms for the self-organization of autonomic systems using the SelfLet approach. According to the presentation web page[[1]](#footnote--1), a SelfLet is a "self-sufficient piece of software which is situated in some kind of logical or physical network, where it can interact and communicate with other SelfLets". This definition is very close to the definition of a smart device, and SelfLets could be included in devices and device-controllers, as a firmware, to ease their integrations.

This approach could foster the distribution of EnTiMid on several nodes, help to prevent system failures, balance the load of resource consuming components, or ease the connection of smart devices.

## 10.4 Architecture Synthesis

The architecture synthesis goal is to assist the creation of an application. Feature diagrams and automatic derivations into products, templates and wizards guiding the developer through the steps of conception of a product, are two examples of tools enabling the synthesis of architectures.

### 10.4.1 Dynamic Software Product Lines for the management of variability

Not really addressed in the contribution, nor experimented in the validation, the management of the variability in the domain of AAL and Home Automation still is a real problem. Luckily, the omnipresence of the model in all steps of the application making process enables the use of well-known modeling tools to help in handling the variability.

As proposed in [**Erreur ! Source du renvoi introuvable.**], Aspect Oriented Programming, coupled with Software Product Lines can be used to address this problem. Product Lines, well-known variability management tool for supply chains, has been transposed in software domain under the name of spl. Large scale productions like cars handle the variability of customers’ requests using these product lines.

A product line consists of a base product that can be augmented with options selected by the customer. Software systems with huge number of variable elements, such as component-based applications, can be defined the in the same way. Base functionalities of the software are described in the base product, and specific options are plugged-in according to the customer selection. The problem is that these tools have been set up to ease the one-shot creation of a products.

Dynamically adaptive software systems are able to dynamically evolve after their creation, and spl are no longer sufficient to help in handling the description of things that can be changed at runtime. To cope with this issue, dspl have been proposed. They enable the description of variation points during the execution of an application, and make it possible to identify the exchangeable elements. The work realized by Carlos Cetina et al. in this domain, presented in [**Erreur ! Source du renvoi introuvable.**], is very close to what we want to achieve and reflects our future.

### 10.4.2 How to describe the behavior ?

In EnTiMid, mapping components to leaf features in the dspl makes it very simple to describe the desired configuration of the software at a high level of abstraction. Nevertheless, components in EnTiMid are developed with a strong effort to respect the close entity principle, and they do not know, nor depend on each other. As a consequence, the dspl can only support the description of the number of components, their types and the different options in case of reconfiguration. In a word, the structure of the assembly. Nowhere the interactions between components can be specified.

While still in its infancy, we proposed in [**Erreur ! Source du renvoi introuvable.**] an approach combining dspl and Business Process Modeling. It enables the description of both architecture, and behavior, by a combination of two modeling tools. Once coupled, these two models describe the structure of the application, and its required behavior, which makes it possible to generate the entire system with connected components. As a work in progress, this approach still has to be experimented in more depth.

Cassou et al. recently presented another approach to this problem of description of interactions. In [**Erreur ! Source du renvoi introuvable.**], they introduce a "behavioral contract". These contracts aim at offering means to express the set of allowed interactions between components, and describe both data and control-flow constraints. The integration of this idea with EnTiMid may be studied in a future work.

## 10.5 Kevoree

EnTiMid, as an achievement of this thesis, has been marked as an interesting approach to address some identical issues in other domains.

For the principles of this contribution to be used in other contexts, EnTiMid has recently been re-designed to become a customization for home automation and AAL, of a more generic tool. Its name is Kevoree[[2]](#footnote-0). The core mechanisms of adaptation, evolution, etc. have been moved in Kevoree in order to make them available for other use cases than home automation.

Now, if EnTiMid is responsible for providing a set of services, and components for Home Automation and AAL, Kevoree offers a set of tools for the component model. Among them, a framework to ease the implementation of components, a graphical editor to create component assemblies, and a specialized runtime.

The development of Kevoree is actually a work in progress in the context of another thesis, which explore different improvements. For instance, questions about distribution, and meanings of links between components.

**Kermeta** is a dsl optimized for metamodeling engineering. Developed in the TRISKELL team, it provides an integrated environment for Model Driven engineering activities. Initially developed as a set of plugins for Eclipse, a work in progress tries to make it run using the Kevoree tools.

**Arduino** is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It’s intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments, according to the manufacturer web page. Recent developments shaped the idea of using Kevoree and the component model to ease sketches, and deployment of applications on Arduino platforms.

## 10.6 Open Control/Command operating system

The component model of this contribution has been designed to allow for connection of heterogeneous components. Inspired by electronics, the elements to be modeled and connected, just need to be expressed in terms of components with inputs and outputs. Since almost all automated systems can be expressed this way, almost all systems can be modeled using the component model proposed in this contribution.

The independence of the model with relation to real devices specificities, enable the description of any application/system. Thus, EnTiMid could take on the role of a universal control/command operating system, since the only difficult point is to develop the driver in charge of the interface between the real world and the component level. If a driver for an application can be created, EnTiMid can help in controlling it.

1. http ://selflet.sourceforge.net/ [↑](#footnote-ref--1)
2. http ://kevoree.org [↑](#footnote-ref-0)