**Chapitre** **4**

# Synthesis

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 bring essential ideas to properly handle arrival and departures of elements, since a service can be started and stopped at any time.

Component-based architectures provide an ideal abstraction level that meets the requirements for a virtual representative of home automation devices. However, the specialized interfaces used as descriptions for ports, may prevent the realization of unpredicted connections.

Components for soa is certainly the best approach for our concerns. Bridging components and services makes the benefits balance the drawbacks of each other.

Transversally, 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.

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

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Habitation | + | - | - | - | + | - |
|  | Dia Suite | + | + | - | - | - | + |
|  | PervML | + | + | + | + | + | - |
| 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 | - | - | + | + | - | - |
|  | Gaïa | + | - | + | + | - | + |
|  | Niagara | + | + | - | - | - | - |

Tab. 4.1: Summary of existing approaches

Table  summarize the good points and the lacks of the approaches described in the state of the art.

## 4.1 Good properties identified

Along the way through existing approaches, some good properties of design have been collected. These properties are not sufficient to address all our issues, but are still necessary to properly cope with challenges. Some of these properties had been suggested in [**Erreur ! Source du renvoi introuvable.**] to cope with the requirements.

**Reflexive Model**

Coming from the mde domain, the goal is to get, and keep synchronized, an explicit and independent model reflecting the architecture living at runtime. This model makes it possible to reason about the application state, and perform any required operation with no risk for the running system because of the decoupling. An adaptation engine, for instance, is thus able to select, test and validate an adaptation scenario on the model, before actually performing the adaptation on the running system[**Erreur ! Source du renvoi introuvable.**]. Component-based execution systems often offer introspection capabilities making it possible to build this kind of models.

**Externalized coupling**

For a system to be handled in the right way, interactions between its composing elements have to be explicit. Component Models and dsl offer means of description for these interactions. A clear and explicit description of the relations between components, gives a better understanding, and makes the analyse of the system much more accurate. It leads to better adaptation decisions, taking into account concurrency problems or dependency cycles for instance.

Moreover, this externalization and description of interactions and dependencies are enforcing the independence of elements composing the system. It also improves the flexibility of the system, thanks to the possibility of modification of the resolution and connection policies with no need to deal with business components.

**Hot deployment**

The possibility for a service to be dynamically deployed or removed during the runtime of a system is an essential principle to be considered while dealing with flexibility, adaptations and evolutions. The execution platform must thus, support dynamic deployments and adaptations of the application during runtime with no restart. This is a basic facility offered by soa execution environments.

**Close Isolation enforcement**

Component Models promote the close isolation principle, meaning that all components must have independent life cycles, and no execution dependencies with each other(in term of libraries). This is necessary to enable and ease the replacement of elements in a system. Indeed, inter-component dependencies may imply a huge alteration of the system to replace a single component, just because it depends on other components. It may also result in a more complex computation process of impacts for a change, or worse, an impossibility for the system to evolve or be adapted.

**Openness**

Interoperability and openness to third party applications/contributions are the reasons why service-oriented architectures have been designed. Their goal is to offer services in a standardized way, to allow them to be used by any other system : any third party application must be able to use the services offered. The Internet of Services makes use of interfaces descriptions and registries to expose the services to the world.

## 4.2 Points of contribution

In the electronic domain, the number of components and their always-possible connectivity offered technicians, and engineers, means to create various solutions. Even many years after their assembly, electronic devices can still be repaired or completed with new features. The proposition made in this thesis is to take advantage of the electronic way of doing to improve the flexibility of software systems while keeping a high level of safety and security.

To this end, the contribution of this thesis can be described by three aspects  :

- A new component model that improves flexibility of software systems, by offering means for connecting any component to any other. This aspect addresses issues from interoperability and evolution requirements.

- Modeling tools to create, modify and simulate component assemblies, check their consistency and validity before their (re-)deployment at runtime. Safety and security, as long as variability management are requirements covered by this aspect of the contribution.

- An execution environment built over a Service-Oriented runtime, to support the proposed component model, cope with requirements of adaptation and evolutions at runtime, and validate the proposition