**Chapitre** **3**

# State of Art

The section  starts with a description the Ambient Assisted Living domain from which social requirements have emerged, and presents some projects on this topic. It then introduces some home automation technologies, their use and goals in several projects. The overview of these two domains makes it possible to sense more precisely the underlying needs targeted by this thesis.

Section  gets the survey underway with mde and dsl. Some component models are studied in section  and section  describes some service-based tools. The two last sections are addressing component models for soa and resource oriented architectures.

## 3.1 Background on AAL and Home Automation

Ambient Assisted Living is a hot topic in Europe. Several projects have been led to address different aspects, and try to cover the needs inherent to a home keeping situation. This section introduces some of them.

### 3.1.1 Projects in AAL

The aalaal Joint Programme [[1]](#footnote--1) is a collaborative association of twenty European Union member states, plus three Associated states. They grouped in the AAL Association, which main objective is to enhance the quality of life of elderly people, by the use of ict. Their main activity is to found R&D projects in the aal domain, and to publish annual calls for project proposals.

The total budget planned for the aal Joint Programme is 700M€, funded by public resources(AAL Partners plus European Commission) for a half, and by private participating organizations for the rest. According to the 13th article of 743/2008/EC [**Erreur ! Source du renvoi introuvable.**] EC decision, funding from European commission are taken from the budget allocated to the ict theme of the ’Cooperation’ Specific Programme of FP7. Thus, the aal-JP is set set last from 2008 to 2013, to fit the FP7 dates.

The ASK-IT project [[2]](#footnote-0) ran from 2004 to 2008, and aimed at providing *Ambient Intelligence* to support and promote mobility of impaired people [**Erreur ! Source du renvoi introuvable.**]. They developed a software framework, and a set of tools for mobility. Among others, a *Domestic Module* has been created to support the provision of seamless home environment management, to the mobility-impaired traveller on the move.

In a slightly similar approach, the SOPRANO project [[3]](#footnote-1) intends to design an ict system to foster the comfort and safety of elderly people in their everyday life. In this project, a strong effort is led to measure the acceptability of the solutions. SOPRANO [**Erreur ! Source du renvoi introuvable.**] aims to maximise the acceptability of the services, especially in populations vulnerable to loss of independent living. This is achieved by ensuring the maintain of the control on the environment by users, where they wish to, and using extended user-centred design techniques.

In-Home IST Project [[4]](#footnote-2) is also an interesting project in this context. Ended in December 2008, its goal was to enhance the autonomy and the safety for elderly people at home [**Erreur ! Source du renvoi introuvable.**]. They developed a set of services like activity monitoring, home environment management, tasks scheduling, flexible AV streams handling, or hence, flexible access to household appliances.

On its side, the GUIDE project [[5]](#footnote-3) tends to provide a framework [**Erreur ! Source du renvoi introuvable.**], a user model toolbox, and a handbook, along with graphical user interface components, in order to ease the creation of graphical user interfaces answering the special needs of elderly people.

All these projects try to improve the comfort, safety and security of elderly people by means of technical help. Home Automation is a vast field that offers many solutions to ease or help in the realization of everyday tasks that can become difficult with ageing. Basic elements of this domain have been introduced by section **Erreur ! Source du renvoi introuvable.**.

### 3.1.2 European research

The European ’Cooperation’ programme of research of the FP7 is quite a good representative of European countries’ current preoccupations and research axes. According to the 7th Framework Programme factsheet[**Erreur ! Source du renvoi introuvable.**], 32.365M€ are allocated to different themes. The major theme, which is allocated 28,72% of the FP7 ’Cooperation’ funds, is ict. In second position with 19,07% comes the health theme. Third and fourth positions are respectively Transport and Nano-Productions with 13,18% and 11,03%, followed by Energy and Environment themes consuming 7,25% and 5,67% of the overall budget.

Thus, three of the top five preoccupations of Europe are ict, Health, and Energy and Environment(assuming Energy and Environment can be grouped as a unique theme).

Using ict to improve the quality of life of people is an idea that can be identified in watermark on several projects in the world. Section  presents some of them with a focus on those using home automation technologies.

### 3.1.3 Home automation in projects

Information reported in this section have been partly collected from a talk by Luc BALANGER, director of the Communication Networks department at FFIE (French Federation of Electricians).

**Asian** countries developed a strong market sensitivity to video games. Some TV channels are even specialized on live transmission of gaming parties, involving professional and sponsored gamers. Over the past few years, several studies and news reported a kind of addiction of a part of the population to video games. In particular, A. Faiola in [**Erreur ! Source du renvoi introuvable.**] states that about 2,4% of 9 to 39 years South-Koreans are believed to be suffering from game addiction, according to a government-funded survey. Another 10,2% of them were found to be obsessed with playing electronic games.

Home automation manufacturers are thus working on products making the game more real. The goal is to give gamers the sensation of being into the game, and play as a first person, using for instance 7.1 speaker systems or 3D visualization devices. This aspect of home automation is clearly entertainment oriented.

**The United States** are also concerned by video game addiction of youth. However, home automation does not target this domain, but a much more prominent one in the US. Safety and security of people and goods has been a huge market in the United States since 9-11 events, as explained by Terrell E. Arnold, a retired Senior Foreign Service Officer of the United States Department of State in [**Erreur ! Source du renvoi introuvable.**]. Also called the "fear" market, American people are investing a lot of money to feel safe. In this field, security video camera can be deployed in homes, for inhabitants to be able to remotely see what is happening. Presence simulation is a must in this kind of systems, figuring there is somebody at home by automatically switching on and off lights when inhabitants are away.

In **Japan**, safety and security are also two important requirements and necessities, but not for the same reasons. As recently demonstrated, the nature is not clement with Japan. Volcano, earthquakes and tsunamis are real dangers. The JEITA House Project [[6]](#footnote-4) made it possible to automatically secure the house when an earthquake is detected. For instance, the solenoid valve controlling the arrival of gas in the house is switched off when an alert is received. This security has been enabled by the use of home devices, able to communicate with each other, or with the centralized house manager. The JETI project also had an orientation toward improving the comfort of people. Asian people are fond of multifunction toilets or showers, and more generally, pay a strong attention to their well-being and health. A centralized house manager can improve the well being of inhabitants by making everyday devices smarter. Manufacturers in this domain design their products to be more and more connected, and full of high-technology features. Moreover, each device is given a specific address making it possible to remotely control almost everything in the house. For example, remotely run a bath using a smart phone for it to be ready when back home.

Home automation has a bad reputation, due to several advertisements that promoted useless functions. It thus has been considered as a costly, useless technology, for people fond of high technologies. Nevertheless, home automation technologies have quietly grown, and are today very rich in terms of communications, functionalities and uses.

### 3.1.4 Home Automation details

Evolutions of concerns and needs in the home automation domain led manufacturers to adapt their products over the years. As a consequence, lots of products communicating through many different media are available today. According to their manufacturer, and to their domain of use, devices also come with their own transportation protocols making the diversity even greater. This section briefly presents a non exhaustive list of protocols and media used by home automation products to enlighten the complexity of this domain.

**Communication Media**

**Communication Bus** The bus is physically a wire, a link between devices responsible for transporting data packets from a source to a destination. When a data packet is sent, every device connected to the bus receives the packet. Most of the time, devices ignore the packets, unless the destination address of the packet is the device’s address. The probably most famous ambassador of this communication medium is Ethernet.

**plc** is a special kind of bus. The main idea of this communication medium is to use again existing cable infrastructure to avoid adding a new specific communication cable. As the most common cable present in all houses is the power line, this technology injects data on the power line by modulating in a carrier wave. All transceivers plugged on the line can then decode the data from the carrier wave modulations. This communication medium tends to be more and more used, as it offers great data transfer rates and does not impose any new wiring.

**Radio** The radio medium is widely used, thanks to its wireless property. Radio devices are quite complicated to conceive, because of the trade off between energy consumption and protocol reliability. However their deployment is easy, and does not require a heavy work. Many manufacturers are using the free ISM radio band. As a consequence, the ISM frequencies are noisy, and protocols have to secure their communications.

**Infrared** The infrared communication medium is employed for specific appliances. Indeed, there can be a lot of noise coming from the natural light disturbing the receivers. Since this medium is based on an optical link (using infra-red), transmitter and receiver have to face each other. This is a strong limitation that makes the use of this medium difficult in home automation. In fact, devices are rarely facing each other. Nevertheless, it still is a good medium for a human-computer interface like a remote control. Users can give orders to the system using a remote control, thanks to receivers disposed in many places around the house.

**Links between media** It is sometimes necessary to use a combination of different media to cover requirements from both the users, and the infrastructure of the solution. As a consequence, manufacturers have developed families of devices, which are using several communication media. Obviously, they also created products to transfer orders/communication frames from one media to another, for all their products to get the information.

**Transportation Protocols**

**IO-Homecontrol** is a two-way radio technology, and a proprietary protocol. Organized in an association, a dozen of industrial manufacturers are providing products compatible with Io-homecontrol. Partners of this association are Honeywell, Niko, Somfy, Velux, Groupe Atlantic, Assa Abloy, Ciat, Renson, WindowMaster, SecuYou and Overkiz. This technology is embedded in equipments of the house like roof window, vertical window, roller shutter, gate, garage door, front door, alarm system, lighting, ventilation, heating system, etc. A single control can monitor and pilot all the io-homecontrol compatible equipment in the house.

**IOBL** In One By Legrand is a proprietary protocol used by Legrand (a French electrical component manufacturer). Legrand offers many devices able to interact with each other. These devices aim to control lights, shutters, or heating systems. Being proprietary, no other device manufacturer offers compatible products. IOBL products are able to send and receive orders from infrared medium or plc.

**X2D** Delta Dore is historically a French heating system manufacturer. This company has extended its activities to home automation and alarm systems, for home and building control. For their products to communicate, they developed a communication protocol called X2D, using radio, bus and PLC media. Just as IOBL, the protocol is private, and no other manufacturer offer compatible products.

**Z-Wave** is a wireless technology to remotely control home appliances, entertainment products, and access systems, running in the 868MHz ISM band. Grouped into the Z-Wave Alliance[[7]](#footnote-5), 160 manufacturers are offering interoperable products in these domains, using this close protocol. Based on a mesh topology, Z-Wave data frames can transit through several devices around the house to reach their destination. This is an interesting faculty to overpass radio obstacles, and ensure delivery.

**European Eib/KNX consortium** The KNX Association[[8]](#footnote-6) has been founded in May 1999, by the members of the EIBA (European Installation Bus Association), EHSA (European Home Systems Association) and BCI (BatiBUS Club International) associations. Its mission is to develop and promote the KNX standard so that it is recognised as "The worldwide standard for home and building control". The KNX standard has been designed to enable the control of applications in industrial, commercial and residential buildings worldwide.

KNX has been approved as a European Standard (CENELEC EN 50090 and CEN EN 13321-1), an International Standard (ISO/IEC 14543-3), a Chinese Standard (GB/Z 20965) and a US Standard (ANSI/ASHRAE 135). It groups about 7000 KNX certified products from 200 member companies, installed by more than 30,000 KNX partner installers in 100 countries.

KNX is designed to automate and control lights, shutters and heating systems in homes and buildings.

**LonMark Intl.** Echelon Corporation[[9]](#footnote-7) is an international company that targets a worldwide transformation of the electricity grid into a smart grid. To achieve this task, Echelon developed LonWorks, a family of products able to interact with each other using the LonTalk protocol. The promotion of LonWorks products to end-users, manufacturers and integrators is ensured by the LonMark Intl.[[10]](#footnote-8) organization. This organization is also responsible for giving guidelines and help, to manufacturers, integrators and end-users to build or simply use LonMark certified products. Lastly, its role is to ensure the interoperability of all products, by verifying that each of them meets the LonMark guideline to operate on a LonWorks network. The LonTalk protocol designed by Echelon is currently recognized as an international standard (ISO/IEC 14908), a European standard (EN14908), and a Chinese standard (GB/Z 20177.1-2006). As for February 2009, over 700 organizations joined LonMark.

LonWorks products are mainly used for technical building management concerning lights and HVAC (Heating, Ventilation and Air Conditioning).

**BACnet** Developed under the auspices of the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE), BACnet[[11]](#footnote-9) is a Data Communication Protocol for Building Automation and Control Networks. It has been released as an American national standard, a European standard, a national standard in more than 30 countries, and an ISO global standard. The protocol is supported and maintained by ASHRAE Standing Standard Project Committee 135, divided into 13 working groups. These groups are working to integrate issues from various building aspects from "Objects and Services" collaborations to elevator or lighting management. As for February 25, 2011, 503 Vendor IDs have been issued from all over the world.

**X10** is a rather simple communication protocol for low cost home automation products. Sensors and actuators are sending data frames over plc or radio to a maximum of 256 addresses, with no acknowledgement. It is thus impossible to know if an order has reached its destination, and there can be only one order at a time on the network.

This protocol is quite limiting in term of number of devices, and guarantee of service, but it works and is pretty cheap compared to other product families. Moreover, technical specifications are available and design of compatible products is rather simple.

**ZigBee** is an open standard addressing low-cost, low-power M2M wireless networks. It has been released by the IEEE in 2003, and over 300 manufacturers have then join into the ZigBee Alliance. Working on 2.4GHz, 900MHz and 868MHz, ZigBee has been designed to be able to work in hostile radio environments. Network topology can be point-to-point, infrastructure or mesh, and accepts up to 65,000 nodes. Wireless products compatible with the ZibBee specifications are targeting remote control, light management or sensing domains.

**6lowPan** is the name of a working group at the Internet Engineering Task Force(IETF). This group aims at reducing the memory footprint of IP frames (and principally IPV6 frames) for the protocol to be used in devices and networks with low power availability. This protocol would make it possible to use the powerful IP routing mechanisms, in sensor networks or embedded distributes applications, making them operable from any classical IP network. Specifications of 6LowPan can be found as RFC 4944[[12]](#footnote-10) and RFC 4919[[13]](#footnote-11).

**Application Protocols**

**Universal Plug&Play(UPnP)**

upnp [[14]](#footnote-12) is an application level protocol which aims to ease the connection and use of electronic devices, based on a discovery-search mechanism. As an UPnP-Device joins the upnp network, it sends an XML description file to all UPnP-ControlPoints. This description provides other devices on the upnp network with information such as manufacturer, device type, device model or the unique identifier of the device(uuid). The services offered by a UPnP-Devices are also specified in its description. Each service conforms to a type, a set of UPnP-Actions the service renders. Providing such information to other devices on the network makes it possible to use functionalities of the device without having to install anything (thanks to the standardization of the descriptions and method call mechanisms).

**dpws**

dpws [**Erreur ! Source du renvoi introuvable.**] could be considered as a upnp next generation. If the goals are the same, DPWS is using WebServices as a transportation mechanism where upnp uses simple XML over IP. dpws still based on the concept of *service*, *device*, *operation* and *parameter*. It includes numerous extension points, allowing for seamless integration of device-provided services in enterprise-wide application scenarios.

**SIP**

sip is a protocol for initiating, modifying, and terminating an interactive user session that involves multimedia elements such as video, voice, instant messaging, online games, and virtual reality. Developed by the IETF MMUSIC Working Group, it was initially published in 1996 as RFC 2543 and actualized by the RFC 3261 in 2002. SIP aims to ease the establishment of communications between multimedia devices using two protocols : RTP/RTCP and SDP. When RTP is used to transport voice data in real time (the same as H.323 protocol), the SDP protocol is used to negotiate the participant capabilities, codification type, etc. SIP has been designed in conformance with the Internet model, and all the logic is stored in end devices (except routing of SIP messages).

SIP can establish sessions for features such as audio/videoconferencing, interactive gaming, or home appliances over IP networks. It is based on request and answer messages, and can use again many concepts of previous standards like HTTP and SMTP.

**XMPP**

The Extensible Messaging and Presence Protocol (XMPP)[[15]](#footnote-13) is a protocol for real-time communication such as instant messaging, voice and video calls, collaboration or lightweight middleware communications. The core technology of XMPP was invented by Jeremie Miller in 1998 and formalized by the IETF in 2002 and 2003 in RFCs. Last reviews of RFCs for XMPP are RFC 6120, 6121 and 6122 published in 2011. The XMPP community continues to define various XMPP extensions through an open standards process run by the XMPP Standards Foundation. An active community of open-source and commercial developers produce a wide variety of XMPP-based software.

The complexity of the home automation domain clearly appears from this presentation. Taken apart, each product, transportation protocol, or medium is quite easy to handle. The complexity is due to the huge number of possible solutions for a given problem, and to the difficulty of getting aware of all benefits and drawbacks brought by each solution.

Home automation can be used in various contexts, like assisted living or energy saving. Professionals in these domains are not aware of the possibilities offered by home automation technologies, and home automation engineers are not familiar with each domain’s problems.

Tools are required to simplify the variability management, and conciliate home automation technologies with domain specific problems to bring new solutions.

## 3.2 Model Driven Engineering & Domain Specific Languages

### 3.2.1 Description

mde is an approach that promotes the use of an abstract representation of a software, before its actual realization. From this abstract view, tools and methods make it possible to automate the final software generation, tests and validations across pre-defined requirements. Models are human understandable representations of the reality. They can handle information about the structure, the data exchange, the communication links, or some building constraints of a software.

dsl are another mean of abstraction and description of software systems. Dedicated to a specific domain, they can be graphical, textual or both. They are designed to restrict the concepts manipulated to the ones from the application domain. This approach makes it easier for domain specialists to express their requirements, by using their own terminology.

The goal of these tools is to provide a sufficient level of abstraction to make the software development easier, more flexible, with an enhanced level of reliability, and shorter time-to-market.

The following of this section presents several approaches built around the concepts of mde and/or dsl, that simplifies the creation of applications in the domain of home automation and/or pervasive computing.

### 3.2.2 Projects

#### Habitation

Habitation is a methodology, a set of tools to address specific requirements of home automation application development and design. In [**Erreur ! Source du renvoi introuvable.**], Jimenez et al. describe how the combination of a dsl, and a mde approach, eases the creation of solutions in this domain. Habitation proposes three main tools. A catalogue of functional units, centralizes elements that can be reused in various applications. Home automation devices are composed of several functional units. The second tool is a workspace in which elements of the catalogue can be placed to define a specific application. Called the application view, this tool aims to simplify the assembly work to make it accessible for non domain experts. The last tool is a kind of engine, which translates from the model and dsl description, to a technology specific configuration file.

The approach proposed by Habitation is very promising and sounds helpful in providing non expert users with tools having a sufficient level of abstraction to be easy to handle. However, Habitation does only address pre-deployment design issues, and does not treat issues such as evolutions, hot deployments, remote control means, in short, runtime considerations.

In all this section, each approach or tool comes with a table. This table presents, in a synthetic way, strengths and weaknesses of each tool. Individual tables are merged altogether in section **Erreur ! Source du renvoi introuvable.** as a synthesis.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Interoperability | Openness | Adaptation | Evolution | Variability Management | Remote Control |
| + | - | - | - | + | - |

#### DiaSuite

DiaSuite[[16]](#footnote-14) is a software tool-suite designed to ease the creation of pervasive and/or distributed applications. DiaSuite[**Erreur ! Source du renvoi introuvable.**] is composed of several elements. DiaSpec is the adl of the suite, used to describe the applications at a convenient level of abstraction. From this description, DiaGen automates the code generation of the application, and DiaSim provides the support for the test, simulation and validation of the generated application. As an example, Bertrand et al. present in [**Erreur ! Source du renvoi introuvable.**] how they used the SIP protocol as a generic communication bus for a pervasive application realized with DiaSuite tools.

This tool-suite has been augmented with Pantagruel[[17]](#footnote-15), a visual dsl created to simplify again, the development of pervasive applications. A first step when using Pantagruel aims at defining the entities involved in the future application domain. In a second step, entities of the application are orchestrated in order to define the logic of the application. A last step generated an application code, compatible with the DiaSuite tools. Details about this tool are available in [**Erreur ! Source du renvoi introuvable.**].

These tools meet the demand of a tool chain to develop pervasive applications from a high-level description. The code generation and the simulation environment are very good tools to improve the development process and efficiency. Designed to ease the development of pervasive applications, these tools do not address issues about variability management, applications evolutions or adaptations.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Interoperability | Openness | Adaptation | Evolution | Variability Management | Remote Control |
| + | + |  |  |  | + |

#### Wired Application Description Language

Wired Application Description Language(WADL) is a language designed to ease the description of dynamic applications, and provide an explicit view of the relations between elements. In [**Erreur ! Source du renvoi introuvable.**] authors present how WADL has been used in the creation of a dynamic sensor-based application.

WADL has been implemented on OSGi(see section ) and relies on the WireAdmin service offered by the execution platform. In this implementation, wires between producers of information and consumers are dynamically created or deleted, according to the elements available in the system. Wires are specified by two filters. Each filter is used to make a selection among all available services, and capture producers’ (or consumers’) services required for the wire.

WADL provides a great tool to explicit the architecture of dynamic applications, which is often difficult to extract because of the runtime evolutions. By nature, this approach copes with the adaptation requirement. The interoperability is made simpler by the use of a Producer/Consumer pattern. Evolution is supported by the filters that can be flexible enough to admit future evolutions. Issues on Openness, Variability management and Remote control are not treated in this approach.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Interoperability | Openness | Adaptation | Evolution | Variability Management | Remote Control |
| + |  | + | + |  |  |

#### PervML

Muños et al. present PervML in [**Erreur ! Source du renvoi introuvable.**][**Erreur ! Source du renvoi introuvable.**] in the context of the management of a pervasive meeting room. PervML is a Model Driven Approach designed to ease the development of pervasive systems. This language separates the analyst view, describing the requirements of the system at a high level of abstraction, and the architect view, where devices and implementation details are specified.

This abstract model of the system is then used in a tool chain which ends up with an executable OSGi(see section ) code. This tool chain, detailed in [**Erreur ! Source du renvoi introuvable.**], firstly transforms the platform independent PervML model to a OSGi dependent model, then generates the executable Java code.

PervML and the associated generation tool chain are available as a plugin for Eclipse [**Erreur ! Source du renvoi introuvable.**].

In [**Erreur ! Source du renvoi introuvable.**], authors explain how they introduced system evolutions capabilities to adapt the generated systems to changes in the user behavior. Their solution uses a context model, to detect specific situations, and a task model describing the jobs to be executed for each detected context.

PervML offers a suitable solution. Developed to be executed on an OSGi platform, it naturally offers adaptation, evolution, openness and interoperability mechanisms. As presented in [**Erreur ! Source du renvoi introuvable.**], PervML also target the variability management issue. So far, a drawback of this approach is that people have to be familiar with UML to model a pervasive application. Also, the use of a pre-defined set of service interfaces described in the framework may become a barrier for the flexibility of the solution. Remote control is not addressed in this work.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Interoperability | Openness | Adaptation | Evolution | Variability Management | Remote Control |
| + | + | + | + | + | - |

mde and dsl are handy tools to ease the description and high level representation of software systems. Often based on generative approach, they can cover a large part of the software development chain. However, it is sometimes convenient to reuse existing software parts for several reasons. Reliability of a several time experienced piece of code, memory space reduction, or time saving argue in this sense. The promotion of reusable software pieces is notably pulled by software components.

## 3.3 Component models

### 3.3.1 Description

Douglas McIlroy first introduced the notion of software component in 1968 at the NATO conference. This new paradigm defends a mass reuse of existing components, and the creation of software as assemblies of components. Since then, several component models and their implementations have been proposed in the scientific community. The next section provides a brief list of component models.

### 3.3.2 Projects

#### Darwin

In [**Erreur ! Source du renvoi introuvable.**] Ioannis Georgiadis and al. present a component model to describe self-organizing software architectures of distributed systems. In this model, components are defined by component types, and can have multiple runtime instances. Instances can be statically specified at design time, or created on demand at runtime. Usually, components provide and require services. The provision or requirement is made through typed ports. These types are specified by the interface of the service they offer. Components are connected by their ports, and the semantic of bindings is a classical service call. Obviously, port types have to be the same. Components can be assembled into composite components. Their specifications describe the instances used, and how they are connected.

At runtime, component instances embed a view of the global configuration, and a manager handling the architecture constraints, in charge of maintaining the configuration view synchronized with the system state.

The clear separation of types and instances is a plus. Runtime creation of instances can help adapting the runtime to the environment. The configuration view synchronized with the runtime is a very interesting property. The use of Java class loading to change utility functions used by the policy manager is a good way to runtime evolution.

Typing of ports can act against the interoperability property. Bindings have a clear semantic, but can not use other communication link than the one they have been designed to use (Java RMI in this case). It is a limitation. The adaptation policies (in case of binding loss or component arrival for instance) are hard coded, and distributed in each instance. Thus it can not be easily changed at runtime.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Interoperability | Openness | Adaptation | Evolution | Variability Management | Remote Control |
| - | - | + | + | - | - |

#### Koala

The Koala component model has been designed to handle the increasing diversity and complexity of embedded software, and decrease development costs. Rob van Ommering and al. explain in [**Erreur ! Source du renvoi introuvable.**] that a way to achieve this is to model the software architecture, and reuse existing software components rather than re-implementing the wheel. In their approach, a clear separation is made between component development and the system configuration. It means that component developers can not make any assumption about the context of use of each component, and designers can not change component behaviours.

Components do require services, provided by other component’s typed ports.

A configuration describes an assembly of components. It handles the model of the application. To help in describing the system assembly, they propose compound components in which are described instances to be deployed, and their interactions(bindings). In this case, an action on a port of the compound component may have to be forwarded to internal components (eg : for initialization). To get rid of the ordering problem, they introduced *Modules* to handle one-to-many, many-to-one or many-to-many bindings. They are in charge of the propagation and have a pre-defined treatment.

The clear separation between the components development and the assembly creation (for a particular application) is a key of success. Composition mechanisms are also welcome to cope with diversity management and promote the reuse of existing components to create value-added compound components.

Here again, connection ports are typed and should conform to a specific interface. This may cause problems in future evolutions, in terms of interoperability. They introduced modules to handle specific communications between components and act like a proxy. They could have gone a bit further, and systematically use modules to specify how each connection. This information could help in solving issues, or specifying the system mor precisely.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Interoperability | Openness | Adaptation | Evolution | Variability Management | Remote Control |
| - | - | - | + | + | - |

#### Fractal

Fractal[**Erreur ! Source du renvoi introuvable.**] is a modular and extensible component model to design, implement, deploy and reconfigure various systems and applications. Famous implementations of Fractal are Julia and AOKell (Java), Cecilia (C), FractNet (.NET) and FracTalk (SmallTalk).

The Fractal component model supports the definition of primitive and composite components. Each Fractal component consists of two parts : a controller, which exposes the component’s interfaces, and a content, which can be either a user class or other components in composite components. The model makes explicit the bindings between the interfaces provided or required by these components, and hierarchic composition (including sharing).

Primitive components contain the actual code, and composite components are only used as a mechanism to deal with a group of components as a whole, while potentially hiding some of the features of the subcomponents. Primitives are actually simple, standard Java classes (in the Java distributions of Fractal) conforming to some coding conventions. Fractal does not impose any limit on the levels of composition, hence its name.

All interactions between components pass through their controller. The model thus provides two mechanisms to define the architecture of an application : bindings between interfaces of components, and encapsulation of a group of components into a composite. By default, Fractal proposes 6 controllers that may be present in components : Attribute, Name, Binding, Content, Lifecycle and Super controller.

DigiHome[**Erreur ! Source du renvoi introuvable.**] is a communication middleware built with Fractal. Its main objective is to offer a support for REST communications, and complex event processing, in a context of home automation.

Reflective execution platforms like Fractal or OpenCOM [**Erreur ! Source du renvoi introuvable.**], do not provide a clear distinction between the reflection model and the reality. Modifying the reflection model implies a modification in the runtime. There is no means to preview the effect of a reconfiguration, before actually executing it. No means to execute what-if scenarios to evaluate different possible configurations, etc. This lack of an explicit and independent reflection model, imposes most of the verifications to be realized while reconfiguring. Pre-condition on reconfiguration actions, as proposed by Léger [**Erreur ! Source du renvoi introuvable.**], are checked and roll-backs are performed if something goes wrong. In addition, component models as Fractal are a bit opaque with respect to the outside world, making the openness and reuse by third party applications complicated if not foreseen in advance. Lastly, the dynamicity of an application running over Fractal is compromised, because the deployment of new components can not be done at runtime without restarting.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Interoperability | Openness | Adaptation | Evolution | Variability Management | Remote Control |
| - | - | + | - | - | + |

#### uMiddle

Nakazawa et al are proposing a framework that bridges remote smart spaces called D-uMiddle in[**Erreur ! Source du renvoi introuvable.**]. It makes it possible for a device to interact with another, over the Internet. This is made available by four distinct features of D-uMiddle. First, a local mapper mechanism abstracts sensor nodes into common representations. Second, a mechanism translates data transmission protocols from a node-specific one to a D-uMiddle common one. Third, a remote mapper mechanism creates proxies of sensor nodes from remote smart spaces in the local space. Fourth, a transport module enabling devices to receive data over IPv4 NATs network. The consumer devices, as a result, can use sensor nodes in remote smart spaces without depending on their own protocols and semantics, and without burdening by the complicated IPv4 NATs.

D-uMiddle brings a solution for connecting remote smart spaces, with no need for a developer to care about transportation. Remote control of equipments is thus made possible for free. Nevertheless, it does not supply tools to handle variability, adaptation or evolution of a deployed system.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Interoperability | Openness | Adaptation | Evolution | Variability Management | Remote Control |
| - | - | - | - | - | + |

In the home automation domain, questions about space repartition of elements, remote communications or use are recurrent. It is sometimes complicated to spread a component based application over several execution platforms, because the communication links between components must be adapted.

## 3.4 (Web)Service Oriented Architectures

### 3.4.1 Description

Service oriented architecture is a paradigm, an idea driving the way to develop software. Software services are rendering services to other services or individuals. If an application, or a software, uses services to achieve its goals, it is a service-based software.

**SOA vs. WSOA**

In the Software Services community, service-oriented architecture is often used in place of web-services oriented architecture. However, there is a clear separation to respect between service-oriented architectures and web-service.

Then, if a service is accessible through the Internet, this service is called web-service, and is a particular case of a service. Then a service-oriented software can use this service as any other. In conclusion, I would say that a service-based application is not necessarily using web-services, and may even use no web-service at all. There are other means of creating software based on services.

**Web Services**

A Web-Service renders a service, using Internet as a support. Web services are composed of methods that can be called by clients. Customers do not have to care about how the service is given and can just use it.

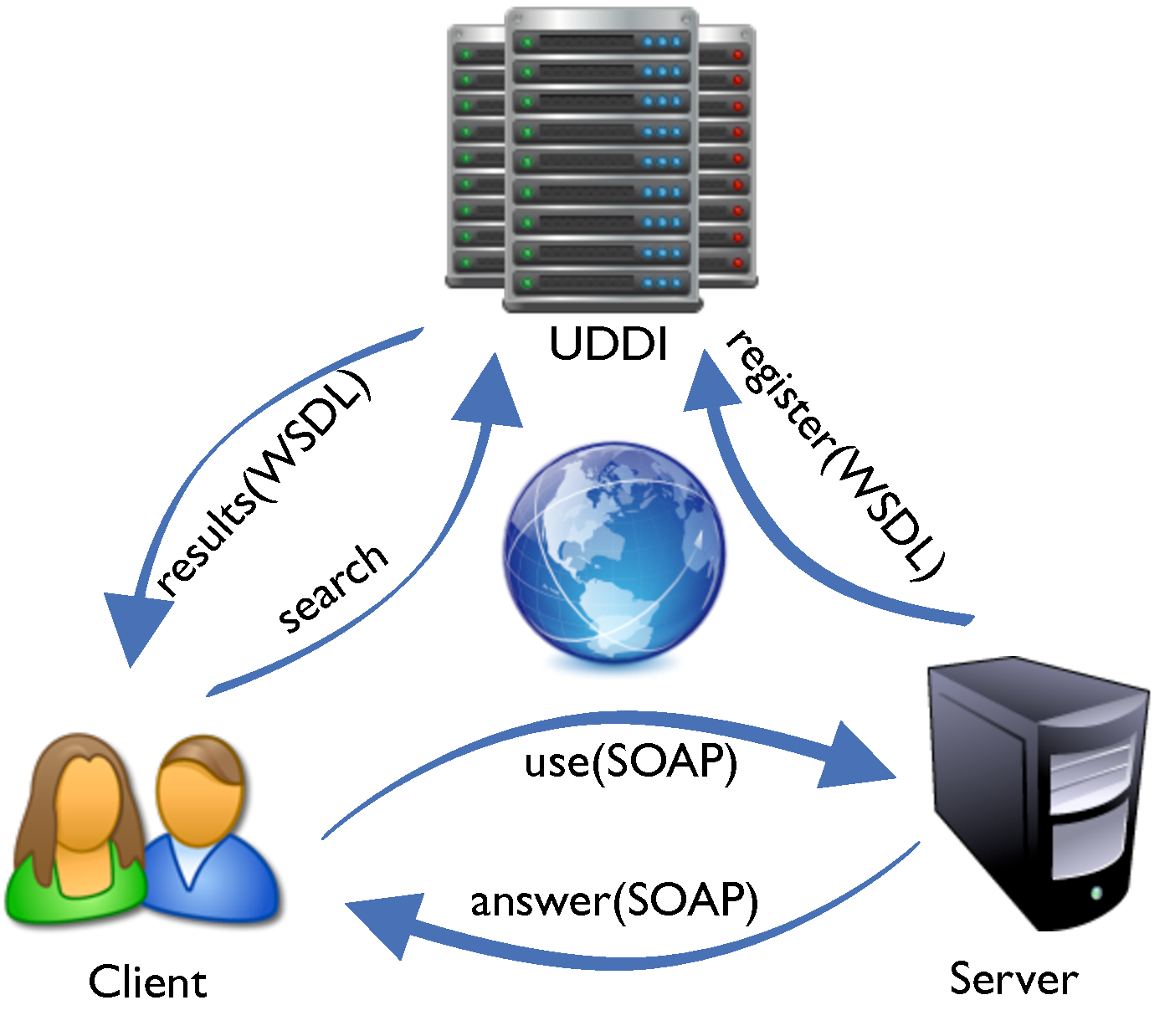


Fig. : WebService Architecture

The use of Web-Services is based on a "search and use" mechanism. Service providers are responsible for the registration of their services into a uddi directory. When a client wants to use a service, he first searches in a service directory. Each registered service comes with a description, which helps clients in their service selection process. The real call to the service is made directly from the client to the server. Figure  illustrates this mechanism.

Registrations and descriptions of services in the uddi are based on a description language for web services called wsdl. The description of a service informs about the list of operations offered, parameters, and types of objects manipulated. Dynamic discovery and use of services are enabled this way.

Communications between clients, servers and uddi instances use a unique carriage protocol called SOAP. SOAP has been based on XML descriptions in order to make use of HTTP, SMTP, and other application protocols as carriers.

If this approach provides mechanisms for dynamic search and use with precise descriptions, the amount data exchanged and the complexity of the SOAP messages structure can become an issue. Resource-limited platforms may not be able to embed a SOAP parser, or have a power supply designed to send a large amount of communication data. To cope with this problem, Resource Oriented Architectures have been proposed.

**Resources Oriented Architectures**

In his doctoral dissertation[**Erreur ! Source du renvoi introuvable.**], Roy Fielding introduced in 2000 the term and idea of Representational State Transfer(REST).

REST has been designed to lighten transfers of information through web-based communications. In place of heavy serializations of concrete program objects, REST architectures are sharing representations of these objects using XML for instance. These representations are handling all coherent and meaningful information for the request(resp. answer) to be processed by the server(or client). REST defines only four standard crud[**Erreur ! Source du renvoi introuvable.**] operations to manage resources.

**GET** is to retrieve the resource pointed by the called url. **POST** asks the server to add the information contained in the request (hence a resource representation) to the resources pointed at the requested url. **PUT** operation is used to create or entirely replace a resource, based on the representation contained in the request. Finally, **DELETE** requests the server to delete the specified resource.

URLs of REST servers are handling information about the resource concerned by a request. For instance, a GET request on an URL like *http ://myMedia.org/* would be answered with an XML representation of the entire media library ; a POST containing information about a new book, called on the *http ://myMedia.org/books* would result in the addition of this book in the book library. Last example, a DELETE request on *http ://myMedia.org/books/2517* should remove the book which unique identifier is 2517 from the book collection.

For its transportation, REST was initially described in the context of HTTP but is not limited to that protocol. Supported by an application-level transfer protocol, REST is thus development technology agnostic.

Thanks to the apparition of these two paradigms, ideas emerged about connecting software systems and everyday life objects. The presence of registries, or auto-discovery mechanism also lead to an abstraction of the physical location details. These interconnections finally brought new paradigms known as Internet of \*.

#### Internet Of \* and the Cloud

Researchers and industrial companies are done with Internet. Now, they want to investigate other kinds of Internets. More than sharing information, and facilitating interactions between people, researchers and solutions are today focussing on services through Internet. From this new breath was born the Internet Of Things (in which devices are offering services to the world) and Internet Of Services (where services are rendering services to other services, and sometimes, to human people).

**Internet Of Things**

Issued from the evolution of electronic devices, the iot relates to an approach in which objects of the everyday life have reached a sufficient level of maturity to interact one with each other. This interaction gives them the ability to act differently according to the situation sensed, whith a stronger added value. Still in its infancy, the iot is looking for software tools to develop and describe these interactions, as long as the services rendered.

Software components are quite convenient to virtualize everyday life objects, and service-oriented computing is convenient to describe and implement interactions. Software solutions developed in the future would probably merge both of these approaches. As long as it can be considered that a component offers services to other components, the collaboration of services and components produce promising results.

**Internet Of Services**

Synergy is probably the best word to describe the ios goal. Many services are now available on Internet, such as hotel booking, flight reservation or car rental. ios aims at orchestrating interactions of existing services to create a more integrated service, dedicated to a task or a user. The goal is to create, for instance, a "Travel booking service" which aggregates booking of flight, hotel and car in a unique process.

Surfing on the service wave, hardware providers, software providers and others are offering more and more "things" as a service. Among others, saas and paas are the two main paradigms showing that everything can now be provided as a service.

**The Cloud**

The Cloud could be defined as the Internet of tomorrow. As everything is being served as a service, there is no more need to precisely locate services. Do you need a printer ? Ask the cloud for the closest printer from you, and just use it. May you need a lawyer ? Ask the cloud for the best of them, and have a video-conference with him.

This approach makes it possible to get the hardware configuration you need, just-in-time, to run your software as a service. You will never know where your software is really executed, but it will run in the cloud.

In this global context, the desire of a part of European research community is to lead the deployment and democratization of these ideas. To reach this objective, several computer science research laboratories join together in a Network of Excellence for Software, Services and Systems.

#### The S-Cube Network of Excellence

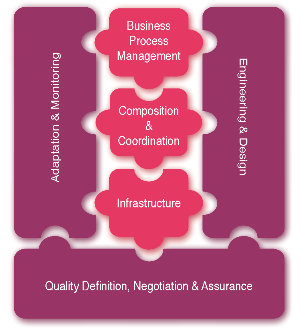


Fig. 3.2: S-Cube Research Framework

S-Cube [[18]](#footnote-16) is a European Network of Excellence(NoE) in Software, Services and Systems(S3). This NoE aims at making European research the leader in software-services revolution. By connecting research to industry, and unifying multidisciplinary researches, S-Cube tends to develop agile and holistic service engineering methods, and specify principles and techniques of service adaptation.

This European NoE has been funded by the European FP7 ’Coordination’ programme research under the ICT theme. Along with strong collaboration and mobility opportunities beyond European research centres, S-Cube funded several PhD thesis in different layers of the S-Cube "BigPicture" (fig. ).

Triskell skills of excellence are dedicated to ease, and improve, software development methods, by the use of components, services, models and validations. Thanks to this team orientation, Triskell is involved in the S-Cube NoE, which brought funding for two PhD thesis. The research leading to the results presented in this thesis has received credits from the European Community’s Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

Several tools have emerged to support these new ideas and paradigms for software developments. The following section details some existing projects and frameworks in this domain.

### 3.4.2 Projects

#### JBI

jbi or JSR 208 is an industrial Java standard developed to ease the integration of software systems over Service-Oriented Architectures. It uses an esb as a basis to define a component model.

esb refers to a business middleware family for service-oriented applications. These middlewares act as the only mediator of services in the enterprise, by providing a runtime environment for deploying business services. Legacy software can be integrated as services into the business service orchestration. They are declared as any other service, within the scope of the ESB runtime.

The jbi component model has been designed to reuse Java technologies like WebServices, BPEL or JMS, and thus, avoid new specific developments. The specifications have been successfully implemented in several frameworks such as OpenESB by SunMicrosystem, ServiceMix by Apache Foundation, or PeTaLs by OW2. In jbi, components have independent life-cycle, and communicate through their services over a normalized message middleware. Actually, this middleware acts as an abstraction layer for communications, and eases the integration. jbi components are split in two categories :

**Service Engine Components** are directly hosted by the jbi runtime environment, and are in charge of message processing, routing or orchestration of services. They can not communicate outside of this scope.

**Binding Components** expose or consume standard jbi services, and perform the bindings with external non-standard software.

The packaging as components is described by the framework like this : service descriptions are encapsulated into Service Units, which are then encapsulated into deployable business component called Service Assemblies.

The message middleware makes jbi a serious candidate in terms of interoperability of components. Openness is offered by the Binding Components, and evolutions are natively supported thanks to its service-oriented nature. Besides the good properties there is no clear separation between types and instances of services/components. Moreover, no introspection of services is offered, and interconnections between components, are not explicitly expressed, and sometimes, even hard-coded inside the components. This lack of clarity in the component inter-dependencies, makes it impossible to dynamically replace components and/or reason about the system state.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Interoperability | Openness | Adaptation | Evolution | Variability Management | Remote Control |
| + | + | - | + | - | - |

#### OSGi

OSGi is the Open Service Gateway Initiative. This association created in 1999 aimed at providing facilities for software integration and development. To achieve this task, the association released a set of specifications defining what should do any runtime implementation to reach a given service level. These levels specify which minimum set of service a runtime must offer to be compliant with the level.

In OSGi, services are given and contained in units of deployments called *bundles*. Each bundle contains a *Manifest* file giving information about the runtime dependencies, the classes offered and some other information about what the bundle needs to run. Bundles can be installed or removed at any time, and their services can be started and stopped, with no need to restart the runtime platform.

Services are defined by Java interfaces (for Java runtime implementations), and are stored in the runtime context. Thus, any client on the platform who needs a service, can search in the context registry for the service they need. Service method calls are locally handled inside a JVM, which makes this service-oriented runtime much faster than web-service based applications.

In OSGi, relations between bundles are never made explicit. Even worse, relations between bundles can be due to service dependencies that are just hard-coded, and can only be changed by rebuilding and redeploying the bundle. In a static application, with few updates in time, this is not a real problem. However, in our context, high dynamicity and adaptation skills are required. Moreover, there are very few reflection primitives and thus, interactions between bundles can hardly be traced or even made explicit.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Interoperability | Openness | Adaptation | Evolution | Variability Management | Remote Control |
| - | + | - | + | - | + |

#### SOPRANO

SOPRANO (Service-oriented programmable smart environments for Older Europeans) was an Integrated European Project, which successfully ended in April 2010. Their main achievement was the release of openAAL[**Erreur ! Source du renvoi introuvable.**], a framework built on top of an OSGi execution platform. OpenAAL helps in getting information from devices, and acting on them from a higher level of abstraction. They integrated in their framework a context manager, able to give a virtual view of all devices, a process manager in charge of taking decisions for any change in the context, and a composer, dealing with the actual services for interaction with the real environment.

OpenAAL proposes a solution to efficiently built applications ready to evolve with the needs, and able to adapt to changes in the context. However, no attention is paid to the variability management, remote control or interoperability of devices.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Interoperability | Openness | Adaptation | Evolution | Variability Management | Remote Control |
| - | - | + | + | - | - |

By nature, Service-Oriented software are highly dynamic and their architecture is not always easy to figure out. Indeed, the services used and the connection between software elements are never known prior to the execution because of the search and use principle. Component models for soa have been invented to make the description of this kind of application more explicit. They also merge well known software component techniques with new services ideas, and conciliate the best of each approach.

## 3.5 Component Models for SOA

### 3.5.1 Description

Components, as defined by component models, are providing services to other components through their ports. Components’ ports are defined by an API. Services, from service-oriented architectures, are intended to be used in orchestrations to create value-added applications.

Since both of these concepts are offering services, new component models have been designed to merge both paradigms. This section presents some famous implementations of these component models.

### 3.5.2 Projects

#### SCA

sca[[19]](#footnote-17) provides a component model for both the composition of services, and for the creation of service components. sca is a model that aims to encompass different programming languages, frameworks and environments commonly used to build components and services, as Web Services, Messaging systems and rpc for communication purposes. Its goal is to setup a single and common way to access and assemble service-based applications.

sca can be presented through four major parts of the specification.

**Specifications**

**Assembly** defines how components are packaged as services, and how they can be combined into composites that perform a particular task. Composite components can be used as classical service components, which simplifies the reuse. Assembly in sca also defines how components and composites are connected. Functional service properties, such as data encryption, or authentication, are described outside the service business code, which saves developers’ valuable time. Indeed, it enables the modification of the connections or the properties, without changing the business code.

**Client and Implementation Model** defines how services are packaged and accessed in various languages. API implementations in Java, BPEL, C++ or the Spring Framework, are offering simple means to package a service or access any sca service. For development concerns, it means that there is only one interface and packaging method to learn to provide and use any sca service. This interface makes it possible to accede to the component using Web services, JMS, JCA and EJBs natively. Here again, properties of services are described outside the code, to make changes much easier.

**Policy Framework** aims at offering means for the definition of security, authentication, quality of service, and other important policies of a service. Actually, the sca Policy Framework makes use of the WS-Policy and WS-PolicyFramework open standards as a support to describe policies. This way, descriptions of policies such as "any data sent to this service must be encrypted" or "the user of this service must be authenticated" are made available. Here again, policies can be defined outside the business code of the service

**Bindings** specify the mechanisms that can be used to access or connect a component. Bindings can be implemented using Web services, JMS, JCA, EJBs or any other communication way. Keeping the consistency of the approach, bindings are defined outside the component business code.

SCA is a standard. A set of definitions describing how such kind of system should behave. Thus, it imposes implementations to propose mechanisms for openness, evolution or remote control.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Interoperability | Openness | Adaptation | Evolution | Variability Management | Remote Control |
| - | + | - | + | - | + |

**FraSCAti**

FraSCAti is an implementation of the SCA specifications. It is certainly the closest approach to what is required. Components can be composed into composite component. Communications between components are made using services, and can use several communication media. These elements make FraSCAti a serious candidate to address openness and remote control concerns.

In the last months, efforts have been spent on integrating FraSCAti and OSGi, which improved its faculties of evolution. In terms of interoperability, FraSCAti offers mechanisms for the connection of services, but do not address directly the integration of smart devices. Thus, interoperability of components in our context is still compromised by the use of APIs, for the definitions of services rendered, and required, by ports. If two components have not been designed to be connected, a ad-hoc connector has to be created.

The FraSCAti script tool enables reconfigurations, adaptations of component assemblies. But the adaptations are limited to the manipulation of binding and component instances, which types are available on the platform. New instances can be created, new bindings can be set, but no new types can be installed using FraSCAti Script.

The variability of FraSCAti itself is managed using a Software Product Line(SPL). Each runtime instance of FraSCAti is a product issued from the SPL. Thus, features of FraSCAti can be deployed on-demand. However, this variability management concerns only internal features of FraSCAti, and an external tool still required to handle the variability of applications made with FraSCAti.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Interoperability | Openness | Adaptation | Evolution | Variability Management | Remote Control |
| + | + | + | + | - | + |

#### AutoHome & iPOJO

iPOJO[**Erreur ! Source du renvoi introuvable.**] is the Apache service-oriented component runtime built on top of OSGi soa platforms. The iPOJO framework merges the advantages of component- and service-oriented paradigms. Specifically, application functionalities are implemented following the component paradigm. Each component is fully encapsulated, self-sufficient, and provides server and client interfaces as services. An iPOJO component is actually managed by a reusable container, which provides common middleware functionalities. Each component container can be configured with a different set of middleware services.

In[**Erreur ! Source du renvoi introuvable.**], Bourcier et al. present AutoHome as an autonomic management framework for pervasive home applications. AutoHome is described as a middleware that extends the iPOJO component model, to create a framework to host autonomic home applications. Using this approach, authors aim to separate the design and development of the application itself, from autonomic management components. They tend to enable the development of autonomic management functions, ease their integration with the applications, and finally deploy the resulting autonomic application on execution environments shared with other applications. As a consequence, an application on top of AutoHome has the following architectural elements : a middleware offering autonomic service-oriented component and a context facility, a runtime that includes monitoring and reconfiguration abilities, a set of service-oriented applications which represent pervasive components to be autonomously managed, and a set of managers organized in a hierarchy.

AutoHome, and the underlying iPojo component model, focus in giving management facilities for pervasive applications based on component models, and a service oriented runtime. This approach makes it possible to include in the application, specialized components that monitor and react on component or platform events. However, this solution does not seem to offer means for variability management, or interoperability between components.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Interoperability | Openness | Adaptation | Evolution | Variability Management | Remote Control |
| - | - | + | + | - | - |

#### Gaïa Framework

Gaïa[**Erreur ! Source du renvoi introuvable.**] is presented as a meta-operating system for ubiquitous computing, built on top of classical operating system. Its goal is to abstract from the heterogeneity and complexity associated to ubiquitous environments. Gaïa is composed of a Kernel, responsible for the runtime management of applications, and a Framework to build these applications. An application runs in an Active Space, a physically limited space where services and devices are available for ubiquitous computing.

Each Gaïa instance is specifically configured for the active space it manages. To allow for describing Gaïa applications for several active spaces, Olympus[**Erreur ! Source du renvoi introuvable.**] proposes a high-level DSL working with virtual entities. From a Olympus application model, the underlying Gaïa OS takes responsibility for mapping each virtual entity to a service, or device, available in the active space.

It has been implemented in CORBA, and can be ported to other communication middleware architectures such as SOAP or RMI.

The interoperability of services and devices is ensured by the common set of basic services. Adaptations and evolutions are made possible by the ComponentManagementCore of Gaïa which can dynamically load, transfer, create or destroy components or applications. Remote control is made available by the underlying CORBA platform, in the implementation described. Variability management of components or applications, and Openness of the solution are not targeted by this work.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Interoperability | Openness | Adaptation | Evolution | Variability Management | Remote Control |
| + | - | + | + | - | + |

#### Niagara

NiagaraAX is a software framework, and a development environment, that leverage the accessibility of a device toward an Internet access. The normalization proposed by NiagaraAX, of the behaviour and data gathered from several devices, enables the implementation of seamless, Internet-connected, web-based systems. And this, whatever their manufacturer or communication protocol. This normalization has been enabled by the Niagara’s unique, patented component model that transforms the data from diverse external systems into uniform software components. These components shame the foundation for building applications to manage and control the devices.

Elevating devices access and control, at the level of services available through Internet, promotes interoperability since services are standardized. It also natively allows for the remote control of physical devices. According to adaptation, evolution or variability management of such kind of application, Niagara does not seem to bring interesting support.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Interoperability | Openness | Adaptation | Evolution | Variability Management | Remote Control |
| + | + | - | - | - | + |

1. http ://www.aal-europe.eu [↑](#footnote-ref--1)
2. http ://www.ask-it.org/ (march 2011) [↑](#footnote-ref-0)
3. http ://www.soprano-ip.org/ (march 2011) [↑](#footnote-ref-1)
4. http ://cordis.europa.eu/fetch? ACTION=D&CALLER=PROJ\_IST&RCN=80489 (march 2011) [↑](#footnote-ref-2)
5. http ://www.guide-project.eu/ (march 2011) [↑](#footnote-ref-3)
6. http ://www.eclipse-jp.com/jeita [↑](#footnote-ref-4)
7. http ://www.z-wavealliance.org/ [↑](#footnote-ref-5)
8. http ://www.knx.org [↑](#footnote-ref-6)
9. http ://www.echelon.com [↑](#footnote-ref-7)
10. http ://www.lonmark.org [↑](#footnote-ref-8)
11. http ://www.bacnet.org [↑](#footnote-ref-9)
12. http ://tools.ietf.org/html/rfc4944 [↑](#footnote-ref-10)
13. http ://tools.ietf.org/html/rfc4919 [↑](#footnote-ref-11)
14. http ://www.upnp.org [↑](#footnote-ref-12)
15. http ://xmpp.org/ (May 2011) [↑](#footnote-ref-13)
16. http ://phoenix.inria.fr/projects/diasuite [↑](#footnote-ref-14)
17. https ://pantagruel.bordeaux.inria.fr/ [↑](#footnote-ref-15)
18. http ://www.s-cube-network.eu/ [↑](#footnote-ref-16)
19. http ://www.osoa.org/ [↑](#footnote-ref-17)