Requirements and Traces for Home Automated Systems with MDE Approach
### Resumen

En la actualidad los sistemas reactivos tienen herramientas de programación comerciales que se adhieren a una plataforma específica. El grupo de investigación DSIE (División de Sistemas en Ingeniería Electrónica) de la Universidad Politécnica de Cartagena ha propuesto desarrollar software para sistemas reactivos mediante el uso de modelos. Las últimas investigaciones se han centrado en el desarrollo de una herramienta basada en modelos con un lenguaje de dominio específico DSL para sistemas domóticos con el proyecto HAbitATION. Este documento aporta a dicho proyecto el procedimiento para la elaboración un catalogo confeccionado con los requisitos domóticos y sus representaciones DSL mediante la herramienta de software Eclipse. Asimismo incluye un catalogo de trazas para dichos requisitos con la finalidad de detectar los dispositivos compartidos entre estos. Finalmente se presenta una metodología de apoyo para el uso de dichos catálogos junto una serie de aplicaciones previstas para una sala de juntas. Una vez terminadas las aplicaciones con sus respectivas representaciones DSL, el objetivo final de HAbitATION es poder transformar de manera automática dichas aplicaciones a la plataforma final ETS, mediante la herramienta Eclipse para ser implantadas físicamente en la sala de juntas.

### Titulación

Máster en Tecnologías de la Información y las Comunicaciones

### Intensificación

Dpto. de Tecnologías de la Información y las Comunicaciones

### Fecha de Presentación

Septiembre - 2009
Acknowledgement

This Project has only been possible thanks to the support given by Francisca Rosique, Manuel Jiménez and Pedro Sánchez members of the System and Electronic Engineering Division (DSIE) research group. The research thesis from Dr. Manuel Jiménez and two articles published by the DSIE research group have been of great inspiration during the development of this work.

September 2009
Meta-Model Requirements And Traces For Home Automation with MDE Approach

Structure of Paper:

This paper is structured by a brief introduction, motivation and objectives; followed by 8 chapters and an appendix containing the requirement catalogue and trace reports.

Chapter 1: Introduction to Home Automation

This chapter consists of a brief introduction to home automation. Starting by its definition, its four objectives (comfort, energy saving, security and communications), different architectures, together with its components and finally a brief description of current technologies available with their outstanding features.

Chapter 2: Introduction to Model Driven Development

This chapter defines the basics of models, metamodels, and systems, their relationships and transformations, introduced then into a software engineering approach known as MDE (Model Driven Engineering) based on MDA (Model Driven Architecture) and OMG standards that support Model Driven Architecture. A generic view of MDE and MDA is given, as the project will be based on this software architecture approach.

Chapter 3: Introduction to Traceability

This chapter explains the importance of traceability within software development and how MDE can make traceability more efficient. It also gives an introduction to the traceability model created by the DSIE Research Group, which later on is used during the project development.

Chapter 4: The Domain Specific Language and MDE Tools

This chapter introduces the concept of Domain Specific Language and its elements (abstract and concrete syntax as well as semantics) followed by the brief description of a DSL independent platform with a MDE approach and a DSL for a specific platform (ETS tool). Then MDE tools available in the market are described as well as some applications that have been developed for reactive systems with the MDE approach.

Chapter 5: An Overview Of The HAbitATION Project

This chapter introduces the concept of Domain Specific Language and its elements (abstract and concrete syntax as well as semantics) followed by the brief description of a DSL independent platform with a MDE approach and a DSL for a specific platform (ETS tool). Then MDE tools available in the market are described as well as some applications that have been developed for reactive systems with the MDE approach.

Chapter 6: The Requirement Catalogue and Trace Reports

In this chapter two main concepts are highlighted requirements and traces. The requirement concept is introduced along with the description of the Requirement Catalogue which has been developed. For every requirement a DSL template has been created and some
brief examples are explained along with a simple home automation application, in order to demonstrate the usefulness of the catalogue. Further on the concept of traces is introduced along with a brief demonstration of trace reports for DSL templates and its benefits.

Chapter 7: Methodology To Use Catalogues For An Application

The objective of this chapter is to provide a methodology that can be used during the creation of a new application, taking into account the necessary steps which would be needed, such as how to use the available requirements and devices or if new categories, requirements or devices should be included within their respective catalogs.

Chapter 8: Meeting Room Applications

This chapter covers a set of home automation applications for the Universities meeting room. Applications have been developed with two different approaches. One focused on using all the available devices within the meeting room by the categories (Illumination, Motorized, Clima and Security). The other approach developed the simplest application possible and evolved it into a middle application using most of the devices, except for the ones of the security category. Finally an application was developed which uses all of the devices available within the meeting room and gathers all the DSL elements of the applications by category.

Appendix A: Requirement Catalog, DSL Templates and Traces.
Content
Introduction: .......................................................................................................................... 5
Motivation: ............................................................................................................................ 6
Objectives ............................................................................................................................... 7
CHAPTER 1: Introduction to Home Automation .................................................................... 9
Abstract ................................................................................................................................. 10
I. Introduction and Definitions .............................................................................................. 10
II. Home Automation Objectives: ......................................................................................... 11
   1. Energy Saving and Resources: ....................................................................................... 11
   2. Comfort: ......................................................................................................................... 11
   4. Communications: .......................................................................................................... 13
III. Home Automation Components, Transmission and Architectures: ............................ 13
   1. Components and Transmission ..................................................................................... 13
   2. Centralized System Architecture .................................................................................. 15
   3. Distributed System Architecture .................................................................................. 15
IV. Current Technologies: .................................................................................................... 16
   1. CEBus ............................................................................................................................ 16
   2. X10 ............................................................................................................................... 17
   3. Lonworks ....................................................................................................................... 18
   4. EHS .............................................................................................................................. 19
   5. KNX/EIB ....................................................................................................................... 19
Conclusions: ............................................................................................................................ 23
CHAPTER 2: Introduction to Model Driven Engineering ..................................................... 24
Abstract ................................................................................................................................. 25
Introduction .............................................................................................................................. 25
I. Models and Metamodel ....................................................................................................... 26
   1. Model Definition ........................................................................................................... 26
   2. Models and Systems ..................................................................................................... 27
   3. Metamodel Definition .................................................................................................. 28
II. Model Driven Engineering MDE .................................................................................... 29
III. Model Transformations ................................................................................................... 29
IV. Model Driven Architecture MDA .................................................................................... 31
   1. MOF ............................................................................................................................ 32
Introduction:

The demand of software development increases exponentially, but the quality for software development increases gradually, so it becomes difficult to develop good quality software within a reduced time period. Therefore developers for software tools have been searching for an adequate solution which can provide an effective development of high quality software.

On the other hand, reactive systems (robotic systems, artificial visions, sensor networks, home automation, etc.) have become important in the development of software, as these systems are highly complex for their interaction with the physical environment, making it difficult to develop quality software for such application.

Previously, in order to develop those systems, the experience of the designer, the specific functionality features, costs among others where crucial. Because of this, each new system needs a specific ad-hoc software development, even when the structure and logic might be almost identical to another system previously designed in another platform. So, the software’s structure or logic cannot be reused. For this reason, different proposals to find a way to reuse software in reactive systems have emerged.

One solution has been given through Model Driven Development MDD based on Model Driven Engineering MDE and Model Driven Architecture MDA [22 MDA], which belong to the Object Management Group OMG. This group has also encouraged the investigation of Domain Specific Languages DSLs in order to automatically generate code for different platforms, making it possible to reuse software in an efficient way.

The objective of MDE is to develop software based on models, instead of the traditional code. Having the possibility of using DSLs within the MDE gives great advantage as DSLs allow automated code generation and simple description of the system by its graphic representation. MDE is defined as an approximation of a System based on the creation of certain models which represent different aspects of that system. The use of models allows to increase the abstraction level at which the system can be developed and it helps to make the system reusable.
Motivation:

Home Automation Systems are reactive systems which interact with the surrounding environment, providing services such as security, communication, comfort and energy saving, by the complete integration of intelligent devices into our home, improving the quality of life. Such systems lack of a predefined methodology which could allow the recognition of the system's requirement, so that the design could become independent of the platform. Therefore their development requires specialization and the reuse of software becomes almost impossible.

Today, in the domain of home automation, there are no specific software languages which can capture the requirements with a high level of abstractions in such way that the requirements can be independent of the platform. Even so, there have been several proposals based on MDS and using UML [23 Voelter] [24 Muñoz J.], but not very practical towards concepts used by experts in home automation domain. Other interesting commercial proposals have been developed such as ETS (Engineering Tool Software) and LonMaker, which are widely used, but with a disadvantage; ETS is attached to the KNX/EIB Platform and LonMaker is attached to the LonWorks platform.
**Objectives**

As commercial solutions to home automation systems are attached to platforms, a new proposal was developed by Dr. Diego Alonso Cáceres [25 Alonso]. The main idea is to develop an application based on methods using MDE and DSLs, so that the system's design can be completely unattached from the platform, and then it can be attached to different platforms by simple transformations.

This paper is an additional collaboration for the HAbitATION project that has been developed by the DSIE (División de Sistemas e Ingeniería Electrónica) research group of the UPCT (Polytechnic University of Cartagena).

DSIE was founded on 1999 as a multidiscipline research group for the following areas:

- Robotic and Control Systems for Industrial Applications
- Service Robots
- Automation of Vision Systems for Inspections
- Electronic Technology for Robotics and Artificial Vision
- Wireless Sensor Networks
- Buildings and Home Automation

The research of this group was integrate within the MEDWSA project (“Marco conceptual y tecnológico para el desarrollo de software de sistemas reactivos”), directed by the “Comisión Interministerial de Ciencia y Tecnología” (CICYT TIN2006-15175-C05-02). MEDWSA is a sub-project of the META Project (Models, Environments, Transformations and Applications) which has been in development from 2007-2009. The main objective of the MEDWSA project is to define the basis for the development of reactive systems using Model Driven Development.

A new branch developed from MEDWSA, which would focus on the study of a particular reactive system type, home automation systems. The HAbitATION project emerged and with it, a tool with a methodology capable of developing software models for home automation systems. A Model Driven Architecture was used to develop the methodology and later integrated into the proposed tool along with DSLs, which need the support of a defined requirement catalog and a procedure to trace its modifications and relations.

The overall objective of the global HAbitATION project is to define complex quality software (from the requirements to the transformations into code for each platform) that can then be used to design home automation applications and where the user of the software must not necessarily be an expert of home automation systems. The transformations and equivalences of the software artifacts will be supported by (QVT+OCL), with associated quality metrics and bidirectional traceability, which will allow the definition of well founded processes for the software development. The project uses industrial standards with formal methods in order to validate and verify the properties and transformations for each artifact, being developed in the ECLIPSE platform.
There are two main objectives for this specific work. One is to create a catalog with the basic requirements for home automation systems taking into account the existing automated meeting room in the University. This will be achieved by using metamodels to create a catalogue of requirements which then will be represented by DSL templates, along with traces using trace models based on a previously defined trace metamodel. Second a definition of several procedures will be established, so the catalog can be used in order to support the designer in the development of home automation. Finally, three real examples will be given, from simple, medium until high complexity design, which will clearly use the requirements, based on the automated meeting room of the University.
CHAPTER 1: Introduction to Home Automation

Abstract ........................................................................................................... 10

I. Introduction and Definitions .............................................................................. 10

II. Home Automation Objectives: .......................................................................... 11
    1. Energy Saving and Resources: ..................................................................... 11
    2. Comfort: ................................................................................................ 11
    3. Security: ................................................................................................ 12
    4. Communications: ..................................................................................... 13

III. Home Automation Components, Transmission and Architectures: .................. 13
    1. Components and Transmission .................................................................... 13
    2. Centralized System Architecture .................................................................. 15
    3. Distributed System Architecture .................................................................. 15

IV. Current Technologies: .................................................................................... 16
    1. CEBus ....................................................................................................... 16
    2. X10 ......................................................................................................... 17
    3. Lonworks .................................................................................................. 18
    4. EHS ......................................................................................................... 19
    5. KNX/EIB ................................................................................................. 19

Conclusions: ....................................................................................................... 23
Abstract

The following chapter consists of a brief introduction to home automation. Starting by its definition, its four objectives (comfort, energy saving, security and communications), different architectures, together with its components and finally a brief description of current technologies available with their outstanding features.

I. Introduction and Definitions

In the last decade, technology and communications have been increasingly integrated into our way of life, including houses. This integration has brought a new concept known as home automation or “domotique systems".

From an etymologic perspective, the term “Domotique" was developed in France, as a union of “domus" for house and “automatique". There are several definitions and terms used to define home automation, besides “domotique", such as smart home, home systems, etc. The most complete definition of home automation states as follows:

“Automation systems, electricity saving and security for homes and building: Are those centralized or distributed systems, capable of gathering information from inputs such as sensors, process the information and finally emit commands to the outputs or actuator, with the objective of achieving comfort, energy saving or security for people and animals within their properties. These systems can access an external network, for information or services, such as phone line rings, internet services, among others.

There are three main branches between these types of automation systems:

1. Domotique or Home Automation specialized in houses, apartments, specific offices.

2. Inmotique, or Building Automation, specialized in entire buildings, e.g. Hotels, Corporative Buildings, Hospitals, etc.

3. Urbotique or Urban Automation specialized in installing automated systems for communities or urbanizations.
II. **Home Automation Objectives:**

There are four main objectives to cover with home automation systems, for which specific applications are developed:

1. **Energy Saving and Resources:** HVAC (Heat, Ventilation, Air Conditioning) regulation, management of electrical and power consumption of the entire electrical installation including electronic household equipment, as well as water, gas and oil resources.

2. **Security:** Detection, surveillance and warning of any intrusion or technical security break out such as fire, smoke, floods, etc.

3. **Communications:** Internal communication such as Interphones, home cinemas, acoustic signals, or any type of communication with and external environment e.g. SMS sending or access from/to the house through Internet.

4. **Comfort:** Automation of certain repetitive or pre-scheduled tasks, such as illumination scenes, shutters, blinds or awnings movement pre-scheduled or capable of responding to presence/absence of light, among many others.

These main objectives very often overlap within the same application. For example, by programming the lights to turn on automatically with a certain dim, when there is no presence of light, comfort is being achieved, as well as energy saving requirements, and even security with presence simulation. This capability of integrating the four objectives makes home automation systems outstanding.

**1. Energy Saving and Resources:**

This objective is achieved through the following controlled aspects:

a) Regulation of devices, e.g. light dim, with which the evolution of electrical consumption of the house or building can be obtained.

b) Programming of devices, based on parameters such as room temperature, luminosity, humidity, daily or weekly schedulers, among others, by using them individually or a combination of some or all of them.

c) Optimization in order to reduce energy consumption. By a correct use of energy and a reduction of its consumptions the home automation installation can be considered an interesting investment in the medium or long term. The capability of integrating all the devices within the system combined with other actions are crucial to achieve the reduction of energy consumption. Such actions could be for example, reduction of consumption for HVAC outside normal working hours, detection of water loses within the A/C system, followed by a turn off for the system, or suspension of the HVAC when windows are detected as opened, automatic control over shutters and blinds to benefit from daylight, among many others.

**2. Comfort:**

Comfort can be achieved by an adequate control of the HVAC systems, lighting, audio, video, home cinema, through automatic programming, response to sensors, push button or
even IR controllers, giving the final users commodity and well-being. All these possibilities adapted to the user’s specific demand, for example that each room, can have individual control of certain aspects within the installation. Some practical examples of this comfort can be IR control over several devices, automatic open/close doors, windows, entrances, centralized surveillance and status of the devices through touch screens, central multimedia control, among others.

3. **Security:**

Security is the most popular application, but before home automation development its entire integration was quite difficult. With today’s technology of home automation systems it is possible to have security features integrated within the entire system. There are two basic areas within security systems as shown by the following diagram:

![Diagram 1-1: Security Areas](image)

Personal security includes general features such as, turning lights on by presence detection in order to prevent accidents; automatic deactivation of plugs when they are not used; distance controllers for switches located at wet areas; SMS or telephone voice messages to pre-programmed phone numbers in case of different emergencies; detection of water or gas leak with automatic response by closing the corresponding valves and/or sending SMS to mobiles, etc. There are more specific features which emphasize on personal health issues, for example for people in need of special care, such as switches on the wall or carried on collars, to ask immediately for nursing, monitoring vital signs, or through wearable disposals and devices integrated within the house, sending constant information to an external or internal information center.

On the other hand property security has 3 main branches:

1. **Distance alarms:** When the user is not present, acoustic alarms, phone calls or SMS can be sent, to inform the situation.
2. Intrusion detection: Includes installation of sensors such as presence detection by volume, hyper-frequency, infrared, or sensors to supervise windows and doors (magnetic sensors), etc.

3. Technical alarms: Fire, smoke detectors, water, gas leak detection, as well as electricity supply malfunctions. Once these situations are detected, they are informed and some corrective actions can take place depending on the type of technical alarm. For example, if there is an electricity shut, an alternative UPS system can be automatically turned on; in case of leaks electronic valves can be automatically closed.

4. Communications:
The internal networks of the house or building are known as HAN (Home Area Networks) and there are basically three types:

1) Home Automation Control Network: is the one that controls the installed devices within the house or building, such as plugs, switches, lights, shutters, HVAC, etc. This type of network uses communication protocols such as KNX/EIB [1 KNX/EIB], Lonworks [2 Byoung], CEBus [3 EIA], X10 [4 X10], etc.

2) Data Network, typically Ethernet or wireless type.

3) Multimedia Network, such as TV cable.

As there are no fully integrated solutions for communication, the use of gateways becomes the best possible solution, with devices capable of interconnecting different internal networks within themselves and with external networks, such as internet, phone lines, etc.

Internal networks or internal communication can be achieved for example, with intercoms, megaphones, audio and video broadcast. General external communications, on the other hand, consist of basic phone lines, video conferences, Internet, digital TV, cable TV, fax, radio, external data transfer, etc. There is also a subdivision of external communications, belonging exclusively to the house, such voice or SMS messages for alarms, external control of the house devices, e.g. through a website or a phone number.

III. Home Automation Components, Transmission and Architectures:

1. Components and Transmission
There are three types of devices within a home automation installation:

- Sensors: Input devices, capable of perceiving changes in certain physical parameters of the surrounding, which then are sent as electronic signals through the media to the control system for interpretation and response decision. Push-buttons, keyboards, touch-screen and distance control buttons are considered sensor devices as well.

- Controllers: Input/Output devices which receive and process information to communicate with other controllers or send instructions actuators.

- Actuators: Output devices which receive specific orders from a controller, such as on/off for lights, plugs, up/down for motors or dimmers, etc.)
The communication between devices can take place through power line, parallel cables, twisted pair cables, coaxial cable, fiber optics, Infrared IR or radiofrequency RF.

Power lines are used as a communication infrastructure for houses which are already constructed, as the same power line already installed is reused, so that no reformation will be necessary. Nevertheless, there are several disadvantages. First the data signal has to be superimposed over the low voltage network (230V/50Hz in Spain) with expensive equipment. Second, several sources for interference can be found, causing to many harmonics and all the electrical lines installed have to be filtered, so that high voltage can not affect low voltage, adding up to the global costs.

The spectrum of low voltage is divided into various reserved areas for specific use as shown in figure 1-2:

![Figure 1-2: Frequency Band](image)

The spectrum of low voltage is divided into various reserved areas for specific use as shown by the following table:

<table>
<thead>
<tr>
<th>BAND</th>
<th>FREQUENCY</th>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>9-95KHz</td>
<td>Assigned to electrical providers. Does not require MAC</td>
</tr>
<tr>
<td>B</td>
<td>95-125KHz</td>
<td>Free to use. Designed for intercomm. applications. Requires MAC</td>
</tr>
<tr>
<td>C</td>
<td>125-140KHz</td>
<td>Free to use. Requires MAC</td>
</tr>
<tr>
<td>D</td>
<td>140-148.5KHz</td>
<td>Does not require MAC</td>
</tr>
</tbody>
</table>

Table 1-1: Band Frequency Use

Parallel cable traditionally used in phone lines is sometimes used for the interconnection of the controllers with the sensors. Whereas twisted pair cable is the most common used in home automation systems for data transmission because of its low cost and coaxial cable is mainly used for TV and radio signal.

Fiber optics has great advantages, because of its immunity to noise, low attenuation, long distance transition and high bandwidth. Still, in home automation systems for houses it is barely used because of its high costs, but for macro-installations with big distances it is use is quite common. There have been several interesting investigations for the use of this material in home automation [5 Arregui], [6 Kojima],[7 Muñiz].
Infrared transmission is used for internal communication within the installation, from mobile devices (controls) to audio and video equipment, in most cases, and sometimes, to other devices like shutters, blinds, HVAC, lights, etc. This type of transmission is immune to electromagnetic radiation, but visibility between transmission device and receptor is crucial. Another option for wireless transmission is radiofrequency. Though it had been previously popular, its demand has decreased for its high sensibility towards electromagnetic radiation, and low range.

Finally, an important impact is expected with the current developments on WSAN (Wireless Sensor Area Networks), and it’s quite recently defined IEEE 802.15.4 [8 Callaway], which considers home automation one of the main applications for WSAN.

2. Centralized System Architecture

In this architecture, all the information regarding to detection and response is sent to the central controller, by multiple sensors and actuators. The controller then decides which orders should be sent to which actuators. The following figure 1-3 is a schematic representation of this concept:

![Figure 1-3: Centralized Architecture System](image)

It uses a star topology for cabling, with a central unit and there sensors do not communicate among themselves.

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low cost installation</td>
<td>High cost for reconfiguration.</td>
</tr>
<tr>
<td>Simple installation</td>
<td>Fragile: Central unit failure unsupported.</td>
</tr>
<tr>
<td>Minimal requirements</td>
<td>Increased length of cabling</td>
</tr>
</tbody>
</table>

3. Distributed System Architecture

Opposite to centralized architecture, there is no centralized controller within a distributed system, as controllers, sensors and actuators are dispersed along the entire installation, giving possibility for multiple topologies. Even so, the most common topology is bus, as shown in figure 4. Each element is capable of treating the information it receives along with an autonomous response.
All devices have a corresponding coupler to the bus, with shared access to the media (CSMA Carrier Sense Multiple Access Protocol) through addressing for correct communication between devices.

---

**Advantages** | **Disadvantages**
--- | ---
High flexibility for reconfiguration. | Components costs increase.
Scalability for expanding installation. | Compatibility between devices (sensor, controller and actuator) is required.
Efficient use of cable. (Not so much cabling will be needed) | Products used have to belong to the same protocol for correct functioning.

---

**IV. Current Technologies:**

Home automation systems currently on the market, tend to change their transmission method and architecture depending on the square meters covered as shown on the following diagram:

In the United States market, the predominant platforms for home automation are CEBus, X10 and Lonworks, whereas in the European market we find KNX/EIB; Batibus and EHS.

1. **CEBus**

The Electric Industries Association from the United States, recognized the need to develop a Standard for home automation networks. In 1983 a committee was organized and a
standard was established in 1988 known as Home Automation Standard IS-60 known as Consumer Electronic Bus (CEBus) [11 Douligeris].

CEBus follows the Open Systems Interconnection model known as OSI and uses 4 of its 7 layers: Physical, Link, Network and Application. The interface among the nodes corresponding to different layers is defined as a set of service primitives, provided to the immediate service level above. There are 3 main areas in CEBus:

- Physical media and topology
- Communication Protocols
- Programming Language

The communication protocols and programming languages are common to all of the CEBus element, but the transmission media may vary (power line, twisted pair, Infrared, Radio frequency, coaxial cable and fiber optics), depending on which suite best to the application. The criteria for selections depends on a combination of requirements such as energy saving, comfort desired, security, costs, simplicity of system, scalability, etc.

CEBus has one control channel, with messages and orders for the devices and several other channels for data, such as multimedia, voice transmission, audio, music, video, TV, etc. The media streams of these data signal types are the following:

- Video: Coaxial cable
- Voice/Data: TP0 and TP3
- Other signals, e.g. for connected equipment to plugs, are sent through the BT network with a modulation technique of amplified spectrum form Intellon Corp.

As a final conclusion, CEBus is an ambitious standard currently developed by the European Union in collaboration with Japan, nevertheless there are only few systems installed, mainly because there are few commercial devices available and very expensive against other commercial solutions.

2. **X10**

X10 is a codification standard used by power line carrier. It was introduced in 1978 by Home Control System from Sears and Plug’n Power Systems from Radio Sack. Later on, some manufacture versions where extended to other corporations, such as Honeywell, Norweb, Busch Jaeger, among others. Some important features of this system are the following:

- It has a configurable decentralized architecture but not programmable.
- Simple plug and play installation for the user.
- Flexible and expandable.
- Perfect for small and medium simple installations.
There are three main device types, the ones that transmit orders (senders), the ones that receive them (receiver) and those capable of either transmitting or receiving orders (sender-receiver), by a given address id. The total capability of addressing is 2⁵⁶. Several receptors are allowed to have the same address within the same installation. All these receptors will do their pre-assigned functions once the sender ships data towards that corresponding address. A single receptor device is capable of receiving orders from different senders. Finally, there are other additional devices within the installation.

The coupler/repeater-X10, as a repeater ensures the signal quality for long distances between the sender and receiver, preventing signal attenuation. When it is programmed to function as coupler, it can be used to interconnect a triple phase electric installation.

The filter coupler/phases, prevents the internal signals to leave the house and produce interferences in other installations. It can also delete external interferences, which can come from a close by additional X10 installation and is capable of coupling the three phases within a three-phase electric installation.

The programmer/verifier transmits and receives each X10 command. It is an important tool for the installation of X10 devices as it allows knowing levels of noise, signal, etc.

3. Lonworks

In 1990, Elchelon Corporation had the objective of covering the aspects that X-10 standard could not embrace at that time because of the standard's simplicity. The communication protocol employed was LonTalk, based on the OSI model by implementing all of its layers.

Lonworks has two types of basic components. On one side, neurons, which are integrated circuits that contain input/output devices, 3 microprocessors and a memory with an operative system and on the other hand, transceivers, which are emitters-receivers devices capable of connecting the neurons to the transmission media.

The software used is known as LonBuilder, which emulates the neurons communication and can help to develop network oriented applications. The neurons are basic
nodes. Through the transceivers the communication protocol can be completely independent of the transmission media.

Lonworks allows five transmission media, twisted pair, fiber optics, power line for low voltage, radiofrequency and coaxial cable. It can use any type of topology as shown in figure 6.

![Figure 1-6: Topologies used by Lonworks and KNX/EIB](10 M. Jiménez)

4. **EHS**

At the end of the 80s, the European Union developed two projects know as SPRIT (Home System 2341 and Integrated Interactive home Project), out of which EHSA (European Home System Association) emerged with the following objectives. First, search for the possibility of integrating equipments of different manufacturers. Second, design a simple reconfiguration and installation to be used by the final user. Finally, make it possible to fully integrate all commercial devices and all possible transmission media, which could be used in a conventional house. The EHS bus emerged as an open system, with a distributed architecture and capable of functioning with different transmission media, such as power line, twisted pair (TP1,TP2), coaxial cable, radiofrequency and infrared. There are three classes of transmission media, one capable of transmitting control signals, a second one capable of transmitting data/voice at low speed and the last one, which transmits audio/video/data at high speed.

5. **KNX/EIB**

EIB (European Installation Bus) emerged from the concept of a unified system for building management, developed by the European Installation Bus Association or EIBA, with over 100 corporations as active members in 1990.

The main functions of the EIBA are to support and prepare a unified set of rules, define testing and requirements, in order to guarantee quality and more important compatibility among the products. Recently EIBA has been integrated into the Konnex Association (KNX) with the purpose of unifying the European system, which is an open system.

KNX/EIB is defined in a set of European and international rules, EN 50090 and ISO/IEC 14543-3X. Today it is the most popular open system installed in Spain and Europe [12 Mint]. Therefore the final implementation of our work will be based on a KNX/EIB system.

KNX/EIB has a distributed system architecture, where all the devices connected in to the communication bus for data transfer, have their own microprocessor and electronic circuit to access the media. The physical connection to the media can be through twisted pair (TP1
the most common one), power line (PL110) or radiofrequency or a mixture of them and the communication protocol will always be the same.

Sensors will send messages as telegrams to the actuators, as soon as they perceive a change in the environment. Actuators will execute the commands in response. Therefore sensors will function as inputs to the system and the actuators are outputs of the system which activate or regulate electrical charges.

In the interconnection for TP1, sensors will be connected to the red and black cables, as the twisted pair will be their voltage source. On the other hand, actuators will be connected into the twisted pairs, as well as to the conventional electricity line.

![Figure 1-7: Devices connection through TP1](image)

The data will be transmitted as alternate voltage overlapped over the current source from the bus, using the two cables of the twisted pair. Each line has its own source with filters in order to prevent filtering of data into high frequency.

5.1 **Topology**

Basically all types of topology are allowed to connect devices, which give a big advantage towards the endless possibilities of scalability in the design. Still there are 3 levels for interconnection within the topology as shown in Diagram 1-3.

![Diagram 1-3: Architecture for a KNX/EIB Network](image)
One line can have up to 64 devices, so if we want to connect more components into that line it is not possible. The solution is to put a main line with two line couplers (e.g. AL1 and AL2), for AL1 we can have up to 64 devices including the coupler and the same is for AL2. This expansion continues on the main line for up to 15 lines as shown in Diagram 3 and it becomes an Area, which the again can be connected to a Line Area. The Line Area can have up to 15 areas. It is very important to notice that each Line has to have a coupler and its own source, regardless of what type of line it is. The source and coupler are crucial as they physically separate the lines and areas, and are responsible for correct routing. Each Line regardless of its type can also support up to 64 devices, including the couplers. By adding up all the devices, a complete system can support up to 14,400 devices.

5.2 Addressing

The addressing method is very important in order to correctly identify devices within the system. There are two types of addressing:

- Physical Addresses
- Group Addresses

The physical address identifies the device with correspondence to the physical location within the topology of the system. The physical address is composed by 3 sections, the first one identifies the area where the device is located, followed by the line and ending by the number of the device (area-secondary line-device). For example the address given by 1.15.2 indicates that the device is located within area 1, Line 15 and is the 2nd device in the Line 15 as shown in Diagram 1-4.

![Diagram 1-4: Addressing Example for Devices](image-url)

The group address is used to define specific functions of the system. It is a format as shown in figure 8. Its assignment is of free design for the installer. For example, the main group can represent general functions such as lights, HVAC, Motors, etc. The intermediate group can
represent a physical location of the house, living room, kitchen, bathroom, etc. The last group of 8 bits can represent the number of device located within that room. Even if the design is free, it is strongly recommended to keep the same design for all the projects, as it becomes simplifies future control check-ups.

![Figure 8: Group Addressing Scheme in KNX/EIB](image)

**Figure 8: Group Addressing Scheme in KNX/EIB**

KNX/EIB has an application program ETS, where several parameters and communication objects can be pre-defined.

### 5.3 Data Transmission

As soon as an event occurs, it is detected by a sensor, which in response sends a telegram. This event can be random, e.g. detection of presence, or it can be cyclic, turning the lights every day at a given hour.

The telegram is sent following the CSMA/CA protocol. If the receiver gets the information and is correct, it responds with an acknowledgment, ACK, otherwise there is no response at all. When there is no response, retransmission takes place automatically for a given number of iteration.

The telegram is formed by seven fields, six for control requirements and one for the data, as shown in figure 1-10. At 9600bps, the transmission of a byte needs 1d.35ms, and the transmission of a complete telegram needs between 20 and 40ms.

Once the installation is finished and the conventional communication takes place, the group address will be included within the destination field, which allows sending control
orders to sensors and actuators. These orders are included in the data field and are defined by the EIB Interworking Standard EIS depending on their size and function.

The EIS contains the data for each function that has been assigned to the communication objects. According to the standard, there are seven different communication objects, each one assigned to a type of control action (communication, light regulation, sending an absolute value, floating point values, etc.). In this way it is possible to guarantee compatibility among devices from different manufacturers.

5.4 **ETS Program**

ETS is a commercial software tool which has been specifically developed for KNX/EIB to program home automation devices.

Programming the devices is the last part of the installation. A PC is connected through a gateway (RS232, USB or IP) into the installation. Then the following steps take place:

- Programming the physical address to the device according to the topology that has been used.
- Load the libraries, data bases, plugins and utilities created by the manufactures for the specific devices into the ETS.
- Configure the application, based on the parameters that each component will need.
- Assign group addressing between devices to unify sensors, actuators, controllers and communication objects.
- Load the programmed application of ETS into the installation.

A step by step example of an ETS program can be found at Appendix C.

**Conclusions:**

Home Automation has had an outstanding development during the last decade, but there is still much to keep developing. Now a days we have different technologies to work with available, but as mentioned before, KNX/EIB defined in a set of European and international rules, EN 50090 and ISO/IEC 14543-3X, is the most popular open system installed in Spain and Europe [12 Mint]. Therefore the development of the HAbitATION project of the UPCT University is considering KNX/EIB for its final implementation.
CHAPTER 2: Introduction to Model Driven Engineering

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>25</td>
</tr>
<tr>
<td>Introduction</td>
<td>25</td>
</tr>
<tr>
<td>I. Models and Metamodel</td>
<td>26</td>
</tr>
<tr>
<td>1. Model Definition</td>
<td>26</td>
</tr>
<tr>
<td>2. Models and Systems</td>
<td>27</td>
</tr>
<tr>
<td>3. Metamodel Definition</td>
<td>28</td>
</tr>
<tr>
<td>II. Model Driven Engineering MDE</td>
<td>29</td>
</tr>
<tr>
<td>III. Model Transformations</td>
<td>29</td>
</tr>
<tr>
<td>IV. Model Driven Architecture MDA</td>
<td>31</td>
</tr>
<tr>
<td>1. MOF</td>
<td>32</td>
</tr>
<tr>
<td>2. XMI</td>
<td>32</td>
</tr>
<tr>
<td>3. OCL</td>
<td>33</td>
</tr>
<tr>
<td>4. QVT</td>
<td>33</td>
</tr>
<tr>
<td>Conclusion</td>
<td>33</td>
</tr>
</tbody>
</table>
Abstract
This chapter defines the basics of models, metamodels, and systems, their relationships and transformations, introduced then into a software engineering approach known as MDE (Model Driven Engineering) based on MDA (Model Driven Architecture) and OMG standards that support Model Driven Architecture. A generic view of MDE and MDA is given, as the project will be based on this software architecture approach.

Introduction
During the evolution of software engineering several developments became transcendent. One of them was the transaction between structural programming to object oriented programming.

![Figure 2-1: Software Evolution Ref. [13 Bézivin]](image)

Nowadays, object oriented programming is not enough to face today’s requirements imposed by highly complex systems. Platforms evolve, increasing exponentially the amount of data, code and functional aspects of applications. According to a report given by the Stadish Group, referred to the European Union enterprises, only 16% of software projects finish as planned in timing and budget, 31% of projects are cancelled and 53% get finished but their initial quotation increased an average of 189%, which is disastrous for the companies.

Implementation of distributed systems keeps increasing and the applications become more complex, so the software automation has become a true necessity. Software automation implies besides automation of code generation, software reusability, standards, quality, and a remarkable decrease of production costs, from the economical point of view.

As previously mentioned, object oriented software development is becoming insufficient [14 Astrachan] in different aspects. Object oriented languages have lost their initial simplicity. Concepts as encapsulation, failed in describing global features, or when the software needs to implement crucial changes and the intention of reusing software has not
been as successful in practical terms. There are still concepts, which cannot be easily represented within object oriented software, such as, services, plugins, data bases, transactions, etc. Today, there seems to be a battle between object oriented middleware and components, but at the same time a new proposal is being considered: Transformation Models (initiative of MDA or Model Driven Architecture)

Model Driven Engineering MDE [13 Bézivin] emerged with the purpose of industrializing software in order to increase quality and productivity. For the moment, MDE is at its initial phase. It is based in the concept of using models as main element throughout the software developments. The use of models increases the level of abstraction, in such a way that implementation details can be ignored while working with concepts. Once the model is defined, with the use of software tools, code can be automatically obtained. The tendency is to create a transition from objects into models.

Certain parallelism has been found between model oriented development and object oriented [13 Bézivin], as shown in figure 2-2.

I. Models and Metamodel

The following section defines metamodels and models as well as their relationship. It is important to notice that a model can only exist if a metamodel exists, but a metamodel does not need a model in order to exist.

1. Model Definition

There are several definitions of a model. Seidewitz defines a model as a set of statements over the system that is being studied. Kleppe defines it as a description of a part of a system, described by a well defined language. Another definition states that a model is a simplification of a system constructed with an objective and it should behave exactly as the system it represents.

One of the best definitions was made by Rothenberg [15 Rothenberg]. Modeling takes place when a system that needs to be studied is too expensive to experiment with. A model
represents the reality of such purpose; a model is an abstraction of reality, as it does not represent all of its aspects. This allows treating the surrounding environment in a simplified way, avoiding complexity. So the most important features of modeling are the reference, the purpose or objective and the reduction of costs.

A model always refers to a system. The model is designed with a concrete purpose, which could be to understand, manipulate, predict or acquire experience on the system. The purpose is to define which aspects of the model will be modeled as well as the amount of accuracy implied. Finally, the reduction of costs is essential as it is cheaper to model a system for testing than creating several entire systems for tests.

A model has five basic features:

- Abstraction: as it offers a partial view in order to simplify reality. It should not have too much abstraction; otherwise it could become an unreliable representation of reality.

- Intelligibility: Besides a proper representation of reality, a model should be intelligible for the user. A model which is difficult to understand for the user lacks abstraction.

- Precision and predictability: The representation of reality must be precise enough to be able to predict certain features which are not obvious without the modeling procedure.

- Economy: The model must be cheaper than the system modeled.

2. Models and Systems

When it comes to models and systems, it is important to note that a model is the representation of a system (repOf), while a system is represented by a model (representedBy). Therefore a model cannot be considered a real system.

A model can have multiple view points, as it is an abstraction or a partial view of the whole system. Figure 3-3, shows that the human body can be represented by different models such as muscular model, nervous system, digestive system, skeleton, among others. Each partial view will have its own vocabulary, set of techniques and tools.

![Figure 2-3: Different models of the human body [10 M. Jiménez](image)](image)
Models are also often the basis for the construction of a system. Such is the case of house building, first different models are created; house distribution, electricity plans, piping, and then based on those models the real system is constructed.

For systems that already exist models can be constructed for their better understanding.

The nature of the system will then allow different model representations as shown in figure 2-4. From a dynamic system the model extracted could be a movie, but if the system is static, the model could be a static drawing. If the system is static and of 3 dimensions, a picture could be a model for representation of the system and so on as shown in the figure below.

![Figure 2-4: Extraction techniques dependent of system's nature [10 M. Jiménez]](image)

3. **Metamodel Definition**

A model is always based on its metamodel. There are several definitions. A metamodel is a model that defines a language to define a model [16 MOF]. A metamodel is a model specification for a system class, satisfying the following condition. Each model should be a valid model of itself for a certain modeling language [17 Seidewitz]. A metamodel is a model that defines the structure semantic and restrictions of a family of models [18 Mellor].

A metamodel is not necessarily unique, as there can be several metamodels which allow describing the model from different points of view and levels of abstraction. It is important to notice that a model that only represents another model is not a metamodel, as it doesn’t meet all the requirements to be a metamodel. So a model representing a model is only a model.
II. Model Driven Engineering MDE

Model Driven Engineering MDE is a technique that uses models as inputs and outputs throughout the process. By using models, the artifacts generated are independent from the programming language, as this one will be used only at the end stages. The ideas can then be expressed in terms of the problem’s domain, instead of having them widespread throughout the entire code.

According to Bézivin, MDE objectives are to separate the model from the platform on which it will be later implemented. It identifies specific aspects of the systems that will be treated as separate and as a combination of them. These identifications are done through the DSL domain specific language. Finally MDE will establish the relationships between different languages in a global architecture as well as the possibilities of transformations among them.

One of the outstanding features of MDA is the possibility of obtaining code automatically through model transformations. So at the end the written code for an application can be reduced, which dramatically decreases the costs of production. The generated code requires less verification and debugging. Still, the models not only generate code, but they can produce as outputs tools for verifications using OCL (Object Constrain Language). Quality and consistency are improved as the transformations of models to code use pattern design, generating a structural code which follows the architecture. Another feature is that MDE allows an agile development, as it gives a flexible response towards requirement changes, being able to obtain executable and reusable code, allowing a better understanding between the developer and the client.

There are three types of models as shown in figure 2-6. A terminal model, or end model is the one whose reference is a metamodel. A met model’s reference is a metamodel and a metametamodel is the one that refers to it-self.

III. Model Transformations

Transformations allow converting the original model into another destiny model, by different metamodels. One type of transformation, perhaps the most important is transforming models into models that are implemented in the final platform, in other words, transforming a model into executable code.
Transformation can be defined as using a transformation function to convert one model into another. A transformation function is a set of rules that defines each functional aspect of the transformation function [18 Mellor]. A transformation can also be seen as the automatic generation of a model, from a source model by following a set of rules or it can be defined as a set of transformation rules which together are capable of defining how to transform a described model from its original language into a destiny model that uses another language. A transformation rule is a description of how to transform one or more original language constructions into other different language constructions [19 Kleppe]. Transformations are also considered models; therefore the model of transformation must follow a metamodel. Figure 3-10 represents a scheme between different transformations that take place among models at different levels of abstractions.

Based on this figure we can observe a set of conclusions. A transformation generates a destiny model Mb from an origin model Ma. The transformation itself is also a model Mt. As models can have a visual representation, transformations can also have graphical representations. As all models are according to its metamodels MMMa and MMMb, the transformation model Mt: Ma \(\rightarrow\) Mb is also according to metamodel MMM which defines the common language from transformations between models. This opens the possibility to create transformations at a higher level, in other words transformations that have transformations as inputs and outputs. Transformations don't have to be one to one, as they can generate several models from an initial model, or unify several models into one model.

There are three main features for transformation according to Mens [20 Mens]:

- Automation: It is necessary to define mechanisms in order to program automatic transformations for a set of origin models; still there might be some transformations where the developer needs to take part manually.

- Complexity: The techniques used depend directly on the level of complexity that the transformation has.

- Preservation of meaning: Transformations must preserve certain features from the origin model so that the output model does not lose sense.
Czarnecki [21 Czarnecki] defines two different phases of transformations:

M2M: Model to Model, are those transformations that have as input a model and as an output another model. These are the type of transformations used to model processes throughout a development.

M2T: Model to Text, is a particular case of M2M, where the output is a textual model without a metamodel destiny. This type of transformation is used for code generation and documentation at the final phases of development.

IV. Model Driven Architecture MDA

Model Driven Architecture MDA is a proposal based in MDE directed by OMG (Object Management Group) whose main objective is to produce and maintain a set of specifications which allow interoperability between different software applications. MDA’s proposal states that software systems should be generated from models that represent those software systems.

![Figure 2-8: Model Driven Architecture and areas for applications. [10 M. Jiménez]](image)

MDA does not want to replace previous paradigms, tools or languages; instead it looks forward to their integration based on models. MDA organizes software development in three layers as shown in figure 2-9.

![Figure 2-9: Model Driven Architecture Layers [10 M. Jiménez]](image)
The layer M3 contains the reflexive metamodel known as MOF (Meta Object Facility). The second layer M2 is formed by metamodels such as UML or CWM, which then are used by the developer to create models capable of describing real systems. The last layer M1 contains all the models which represent real systems. These models are composed of three types, CIM, PIM and PSM.

CIM or Computer Independent Model is the one that focuses on the system’s domains and requirements. This perspective of the system generates a Computation Independent Model.

PIM or Platform Independent Model shows the specifications of a system considering internal specifications described by CIM and additional specifications to implement in a computational environment. PIM represents the aspects which will suffer no change from one platform to another, but it does not show any aspects of the implementation itself.

PSM or Platform Specific Model is a model completely dependent of the platform. This model shows all the specific details for implementation, taking into account operative systems, structural components, languages being used, etc.

OMG has developed a set of standards and metamodels to support the MDA methodology. Some of these standards are MOF, XMI, OCL and QVT.

1. MOF

Meta Object Facility of MOF is the standard which defines a common or abstract language in order to define modeling languages and how to access and exchange models expressed in those languages. Any language can be defined through MOF, including UML. MOF also allows the construction of tools to manipulate modeling languages.

The MOF Repository Interface allows obtaining models at level M1 based on MOF. Exchange of models is also possible as MOF defines an exchange format known as XMI (XML Metadata Exchange). As MOF is defined by itself, XMI can also be used to generate different exchange model formats.

2. XMI

XML Metadata exchange or XMI is a standard defined by OMG which describes the exchange format, as well as storage persistence of models and metamodels. XMI establishes correspondence among MOF and XML; it defines how the XML labels should be used to represent MOF models in XML. The MOF metamodels are transformed into document type definitions or DTD and models are converted into XML which have consistency with their corresponding DTD.

XMI defines important aspects for describing XML objects, such as the representation of objects in terms of XML elements and attributes and interconnection between elements. XMI has a mechanism to link objects within the same file or between different files. The object’s identity allows them to be referenced by other objects by using their corresponding
ids. XMI also manages the versions and definitions of objects. The validation for XMI documents is done through the corresponding DTDs or XMI schemes.

3. **OCL**

Object Constraint Language OCL specifies the semantic restrictions for UML class diagrams, along with other definition types. OCL can be used in any context where an associated class diagram is used.

4. **QVT**

Query, View and Transformation QVT is a language developed by OMG in order to define transformation between models. QVT can travel through a model searching for certain features or relations and defining transformations among models, the same model or metamodels.

With QTV it is possible to define unidirectional and bidirectional transformations; perform increasing updates between models; create and destroy objects that belong to the metamodel and it is capable of defining transformations in order to verify if certain models are related or not.

**Conclusion**

MDE can be applied for home automation systems. As the whole installation can be divided into models of components and requirements, such that they can be reused for other similar home automation systems.
CHAPTER 3: Introduction to Traceability

Abstract .......................................................................................................................... 35
I. Introduction to Traceability, Definitions and Objectives ........................................... 35
II. MDD as a solution towards Traceability, Limitations and Solutions ....................... 36
III. Use-Cases .................................................................................................................. 37
IV. Internal and External Traceability ........................................................................... 37
V. Meta-Model Traceability Package and System .......................................................... 38
VI. Traceability Services ................................................................................................ 39
VII. A Specific Meta-Model for Traceability developed by the DSIE Research Group of the University UPCT of Cartagena .............................................................................. 40
Conclusion ..................................................................................................................... 42
Abstract
This chapter explains the importance of traceability within software development and how MDE can make traceability more efficient. It also gives an introduction to the traceability model created by the DSIE Research Group, which later on is used during the project development.

I. Introduction to Traceability, Definitions and Objectives
Several definitions for traceability have been proposed, one of the by the IEE Standard Glossary for Software Engineering Terminology [26 Pinheiro], which defines traceability as the relationship degree assigned between two or more products of the developing process. Shaham Gafini [27 Gafini] defines it as a relationship between two set of identities, the source identities and the destiny identities. The exact meaning of the relationship depends directly on the context where the identities are being used.

A better definition given by Aizenbud-Rehesef [29 Aizenbund] defines it as a relationship between two artifacts involved in a software engineering life cycle, including explicit links and mappings which are generated as a result of transformations in both directions (forwards and backwards), statistically implied links which are calculated on previous existing information or in historical information regarding to changes on the system’s usage. Gotel and Finkenstein [30 Gotel] define it as the ability to describe and follow the life cycle of a requirement on both directions; from its origins, through its development, specifications, until its release, including all iterative improvements and phases as shown in figure 3-1.

Therefore traceability is achieved by defining and maintaining relationships between the software artifacts involved in the system’s cycle of development.

![Fig. 3-1: Goetel and Finkenstein trace definition from the life cycle of a requirement.](image)

Traceability has different objectives depending on the perspective of its user. From the manager’s project point of view, the purpose of traceability is to prove that each requirement has been satisfied and that each system’s component has satisfied a requirement. On the other hand, the manager of requirements needs traceability to allow the creation of links between requirements sources and destinies, capturing the necessary information to understand the evolution of those requirements and verify that the requirements have been
achieved. Finally, the designer uses traceability for supervising, when a change within the system is implemented before redesigning the entire system.

With a complete traceability, precise changes can take place without needing to relay on the software programmer, otherwise the software programmer would have to have knowledge of all the areas affected by those changes, and when it comes to large software projects, it becomes almost impossible to know all the areas.

Until now, traceability had not been widely spread as it had been manually developed, becoming very expensive but that will change soon as MDD allows automating the creation and findings of trace relationships through specific analysis methods [29 Aizenbund].

II. MDD as a solution towards Traceability, Limitations and Solutions

The most basic solution of traceability allows the possibility of unifying artifacts through links, but it does not have a specific semantic for those relationships. Through semantics, one can provide two different types of relationships for the traces:

a) Allow the user to add attributes to the relationships, in such way that a search by attributes can be made.

b) Provide a predefined standard with a set of relationship types that can be supported by a tool.

One of the most important limitations is that relationships are static, while the organizational demands vary along time. The solution would be a pre-defined meta-model link which allows defining, personalizing and extending new links. Another limitation is that in most cases the traceability solutions are given by commercial tools, which save the information link within the artifact, limiting the relationships only to the tool’s artifacts, making it impossible to relate with artifacts of other manufacturers. MDD would allow traceability tools capable of working with different models in different platforms, regardless of the manufacturer. The best solution is to automate the creation and maintenance of traces, this can be done by:

- Data mining and recovering information techniques in order to infer the relationships between artifacts.

- Analyze existing relationships to obtain new relationships, where tracking dependencies between requirements and code can be automated, by verifying which classes where first activated as a result of the scene examination for a specific requirement. Once this information is available, different method analysis will be employed to find additional tracing dependencies between other requirements and the model’s elements.

Relationships between artifacts can be classified as the following:

- Imposed relationship: Is the one that exists because it has been imposed by its creator and it exists until its creator decides to eliminate it.
Inferred relationship: Is the one that exists because two artifacts satisfy the rule which describes the relationship; e.g. there are 2 methods m1 and m2. If m1 calls m2, it can be inferred that a relationship between m1 and m2 exists.

- Manual relationship: Is the one created by a person.
- Computed relationship: Is a relation created automatically by the program.
- Derivation relationship: Is the one that can be obtained by the content of the artifact, as we can calculate content for another artifact.
- Analysis relationship: Is the type of relationship created by an analysis program which analyses the code or model against a set of rules.

III. Use-Cases

Use-Cases are defined a sequence of actions that return a value to the actuator [29 Aizenbund]. The collections of use-cases represent all the significant interactions between the system and the external environment and are the functional level of the system's requirements [29 Aizenbund]. By using this collection traceability can be better applied with MDD. When gathering the use-cases, a context for non-functional requirements can be provided, which facilitates traceability. As a use case returns a value, this value can be studied and organized by use case and the test for case type can be associated with the use-cases, so a chain is formed:

Use-Cases ➔ Elaboration of Use-Case ➔ Participant ➔ Operation

This chain provides the basis for traceability through different levels of decomposition as shown in Figure 3-2:

Fig. 3-2: Up-Down trace of a use case through a logic tree.

A significant part of the systems traces can be obtained by tracing the tree, which describes the collaboration among the subsystems and the requirements derived from each of the systems. Recursion can be applied to each subsystem as it can also be considered a system.

IV. Internal and External Traceability

MDD considers the models as primary artifacts. Different types of models are required to represent the system, from its initial requirements until its final implementation. Therefore models can represent several situations of the system such as:
- A system going through different phases of time as it evolves.
- Different aspects of the system such as its structure or behavior.
- A system through different levels of abstraction

Traceability must maintain consistency through the models and transformations which can be unidirectional or bidirectional. There are two ways traceability can be treated within MDD according to Czarnecki and Helsen [31 Czarnecki]:

- Models that provide dedicated support to traceability, in other words automated traceability.
- Models that depend on the user for the generation of traceability.

In the first case, automated traceability can control how many links are created, if they are source or destiny type and where to store them.

Traceability can be defined as internal or external. The internal one is maintained by a transformation machined during the transformation procedure in order to assist the transformation algorithm. External traceability persists during mapping and is maintained aside of the transformation.

V. Meta-Model Traceability Package and System
The authors of [32 Mirka] have proposed a solution for traceability with MDD. They use a Meta-Model package for traceability which defines trace models and consists of four classes.

- The class Trace Model specifies which model artifacts can be traced as well as which model artifacts traces or relationships between model artifacts can be created.
- Traceability Artifact Type: defines the specific model artifact mapping to its corresponding traceable artifact type.
- Artifact Trace Type: defines a type of specific trace for each type of traceability artifact.
- Relation Trace Type: defines a specific relationship type between the source and the destiny artifact.

![Fig. 3-3: Traceability Meta-Model [32 Mirka]](image-url)
Then a traceability system is defined which provides the basis to create, store a use traces. It is formed by 5 classes:

- Trace Repository: manages trace information represented by the Traceable Artifact, Artifact Trace and Relation Trace.
- Traceable Artifact: defines the type to be instantiated and when the model artifact should be traced.
- Artifact Trace: defines the type to be instantiated when a specific trace is related to the trace table artifact.
- Relation Trace: defines the type to be instantiated when a specific trace is related with a relationship among two related traceable artifacts.
- Trace Session: is the administrator's session when traceability is provided to the users.

VI. Traceability Services

There are four services defined for traceability:

1. Trace Model Management: defines how and which type of artifact can be traced.
2. Trace Creation: defines the creation of traceable artifacts by the user, traces and relationships among them and maintains them all in a repository.
3. Trace Use: Any functionality supported by the repository regarding to search and analysis of a set of traceable artifacts and relationships managed by the repository.
4. Trace Monitoring: is used to notify the users which events occur as a result of tracing situations of events within the repository.
VII. A Specific Meta-Model for Traceability developed by the DSIE Research Group of the University UPCT of Cartagena.

For this project and research directed by the DSIE, a specific meta-model for traceability [33 Rosique] has been developed as shown in figure 3-6. This meta-model contains a “Link” pointing to any “ModelElement” via two references: a “source” element and multiple “target” elements. The “Composite Link” allows defining different levels to group Links into more complex levels. "LinkType" allows to classifying the existing relations and distinguish at what level of the process development the trace is located.

![Traceability meta-model](image)

Figure 3-5: Traceability meta-model [33 Rosique]

For the particular case of traceability within home automation requirements and modeling correspondence with the DSL, the Link element of the meta-model is defined by:

- A ModelElement (source) which references an element of the home automation requirements meta-model.

- A ModelElement (target) which references an element of the DSL.

![Traces representation](image)

Figure 3-6: Traces can be represented by the traceability meta-model [33 Rosique].

Figure 3-6 shows an example of traced elements between two consecutive levels of abstraction (requirements to design, design to elements of language programming, etc.). It can be seen that it is possible to integrate one-to-multiple relations within the same link.
At the same time, a target can be the same destiny for several other traces with different origins.

Traceability models are created during the process transformation of models. Trace links between source and target artifacts of a transformation may be created implicit or explicit. In the first model of traceability (between requirements of home automation systems and the corresponding DSL model) traces are manually created by the user. Then from the DSL to the final code generation, all the traces are automatically obtained as part of the model transformation process form origin to destiny.

![Graph Transformation Rule](image)

**Figure 3-7:** Example of graph transformation rule: from DSL to component model level. **Black:** catalogue view (DSL) instance; **Red:** target (component model) instance; **Blue:** application view (DSL) instance; **Orange:** transformation instance. [33 Rosique]

![Extension Example](image)

**Figure 3-8:** Extension example of a graph transformation rule: from DSL to component model level. **Black:** catalogue view (DSL) instance; **Red:** target (component model) instance; **Blue:** application view (DSL) instance; **Orange:** transformation instance; **Green:** Traceability link instance. [33 Rosique]
In other words, while the transformations that obtain models close to the final platform are executed, traces among the elements involved and their corresponding links are recorded into the traceability repository. In order to achieve these traces, rules of transformation must have an extension for recording traces within the defined meta-model. Figure 3-8 includes an extension added over the graph of Figure 3-7 where traceability is included by modifying the RSHS part of the rule. It can be observed how a new element appears which corresponds to the traceability model. Each “TraceLink” element of the transformation rule stores the information (name, description) for the instances of “Link type” from the traceability meta-model represented by Figure 3-5. Finally, all the existing traces between the DSL elements and the components of the model are gathered at the “Traceability Model#2” [33 Rosique].

**Conclusion**

Traceability gives the advantage of controlling software artifacts and their relations during the development process and it can be greatly exploited if developed through MDE. A traceability meta-model has been proposed by the DSIE Research Group, which later on will be used in the development of this particular project. Still this model can be further enriched in feature work by allowing part of the traceability which by know is done manually to be automated. This will be further discussed later on in Chapter 6.
CHAPTER 4: The Domain Specific Language DSL and MDE Tools

Abstract ......................................................................................................................... 43

I. Introduction to Domain Specific Languages DSL ...................................................... 44

II. DSL Elements ........................................................................................................... 45

III. Independent Platform DSL based on MDA for Home Automation Systems .... 47

IV. DSL for Specific Platforms (ETS for KNX/EIB) for Home Automation Systems 48

V. MDE Tools ................................................................................................................. 49

VI. MDE Applications for Reactive Systems .............................................................. 50

Conclusion ...................................................................................................................... 52
Abstract

This chapter introduces the concept of Domain Specific Language and its elements (abstract and concrete syntax as well as semantics) followed by the brief description of a DSL independent platform with a MDE approach and a DSL for a specific platform (ETS tool). Then MDE tools available in the market are described as well as some applications that have been developed for reactive systems with the MDE approach.

I. Introduction to Domain Specific Languages DSL

Domain Specific Languages DSL is made by a set of specifications, sometimes executable, with notations and abstractions, which offer specific expressions towards a specific domain. Table 3-1 shows DSLs for different domain applications. MDE has strongly impulse the development of DSLs as it reduces the development costs.

<table>
<thead>
<tr>
<th>DSL</th>
<th>DOMAIN APPLICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNF</td>
<td>Syntax Specification</td>
</tr>
<tr>
<td>HTML</td>
<td>Hipertext pages</td>
</tr>
<tr>
<td>LATEX</td>
<td>Text Edition</td>
</tr>
<tr>
<td>Make</td>
<td>Executable Code Generation</td>
</tr>
<tr>
<td>SQL</td>
<td>Data Base Search</td>
</tr>
<tr>
<td>VHDL</td>
<td>Hardware Design</td>
</tr>
</tbody>
</table>

Table 4-1: common DSLs [34 Mernick]

According to experts [35 Deursen][36 Estublier] and [31 Czarnecki] some advantages of DSLs are:

- DSLs allow expressing solutions using rules and concepts of the domain's problem, obtaining a good level of abstraction and the experts of the domain can then understand, validate, modify or further develop DSLs on their own.

- The programs are concise, self explainable and reusable.

- They provide a graphic notation for more clarity.

- They improve maintenance, portability and reliability.

- Allow total or partial automation of the process development, improving productivity.

- They contain the domain concepts and allow their conservation and their reutilization as well.

- They significantly improve verification as they allow the construction of static analyzers.

- They allow generating optimized code based on the domain knowledge.

Even so, there are some disadvantages as the increment cost of the design, implementation and language maintenance. The cost also increases proportional to the users increase. Therefore DSL development is only recommended if a great amount of
applications within the domain will be designed and developed or if tools which reduce the implementation costs are available, such as MetaEdit+, Microsoft DSL Tools or Eclipse Modeling Project, among others.

Eclipse Modeling allows a set of plugins such as EMF, GMF, ATL and QVT as support for the entire development cycle based on models and it is a free tool very popular among the scientific community.

II. DSL Elements

All languages regardless if they are of specific or general purpose, for conventional programming or for models, share certain features; they should have an abstract and a concrete syntax followed by semantics.

- Concrete Syntax: is the notation which allows the construction and presentation of models and languages. There are two different types of concrete syntax:
  - Textual syntax: Allows describing models or programs in a text mode e.g. java code.
- Graphical syntax: Allows representing the model or program through diagrams based on icons which represent specific perspective views of the model.

- Abstract Syntax: describes the vocabulary for concepts provided by the language and how they can be combined to create models.

- Semantics: As abstract syntax contains little information about the language meaning it is necessary additional information in order to obtain the semantic so that the language representation and significance can be understood.

The use of metamodels in the definition is a crucial feature as it allows making unified definitions for a rich semantic. Therefore MDA is applied to create DSL using metamodels to describe the semantics, concrete and abstract syntax of the language.

Constructing the model to represent an abstract syntax is definitely not an easy task and it is the first step to design a modeling language. This model describes the concepts of the language, the relationships among them and defines the rules to verify if the models written on that language are correct.

The abstract syntax model is written in a metamodel language. For this particular project EMF (Eclipse Modeling Framework) has been used. It provides several interesting features such as:

- Classes to describe the language concepts
- Packages to divide the model
- Attributes and associations to describe relationships among concepts.
- Restrictions to express rules (using OCL)

The steps to create an abstract syntax model are the following:

1. Defining the concepts either by a list of candidate concepts or by constructing model examples by using the language. For the last case generic concepts and relationships can be extracted.

2. Use-Cases are a good technique in order to identify concepts by associating use-cases with the UML Language; it is like writing an interface for the model.

3. Modeling concepts: Once the concepts have been clearly identified, their own features can be used for modeling.

4. Rules: Once the basic model has been constructed, examples of correct and incorrect models must be identified, which then can be written in order to define the rules.

5. Validation and/or Verifying: it is very important to validate the model’s abstract syntax once it has been corrected. The best way to verify a model is by construction a tool which implements it in such a way that the final users can verify it.
In order to develop a concrete syntax it should be interpreted and then verified so that it can finally be used to construct the model of the abstract syntax. It is the same procedure for graphical or text syntax, only differing in the way the syntax is constructed. For the graphical syntax, diagrams are constructed by iteration and verified simultaneously, whereas textual is a procedure by sets of code.

Textual syntax is done with tools such as BNF (Backus-Naur formalism), while graphical diagrams are based on models such as XMI from OMG. These diagrams allow capturing features of the syntax concepts and their relations, and EMF form Eclipse is a tool that allows translating those concepts.

III. Independent Platform DSL based on MDA for Home Automation Systems

Within home automation industry there has been a lack of non commercial DSL for home automation systems. J. Muñoz [24 Muñoz J.] suggests a DSL based on MDA for Home Automation Systems. The requirements are captured by the system analyst through three different views:

- Modeling of Services
- Iteration Modeling
- Structural Modeling

From these views a software specialist generates the architectural views with the modeling language PervML. Figure 3-6 represents an example of those views. It can be seen that the requirements use UML diagram classes for modeling services, iteration diagrams for iteration models and diagrams for components and services to represent the structural models. An inconvenience of using UML diagrams instead of DSL is that it is difficult to understand them by the home automation domain specialists. Therefore further on the projects with UML diagrams evolved into DSL diagrams, being more intuitive for home automation specialists.
IV. DSL for Specific Platforms (ETS for KNX/EIB) for Home Automation Systems

KNX/EIB developed a software tool known as ETS (Engineering Tool Software) shown in figure 3-7. It allows to configuration, parameterization, programming and applying a diagnostic to the devices of this platform based in 3 views:

1. Topology View (Bottom–Right): it specifies the topology of the network installed and the devices included in each segment of the topology.

2. Building View (Top): It shows the distribution of the devices through the building, by floors, rooms, etc.

3. Addressing View (Bottom-Left): It shows in detail the address groups created to associate devices within the installation.

![ETS views](image)

Fig. 4-3: ETS views.

With these views the developer is capable of designing the installation, but for every new and similar installation the whole procedure must be repeated as it does not allow an efficient reuse of programs.
V. MDE Tools

Model Driven Development can only be possible through its developing tools, which allow modeling tasks, transformations among models, verifying models, DSLs and code generation. Some of the most recognized software tools are Eclipse Modeling Framework (EMF).

NetBeans MetaData Repository supports standard MOF and can be modeled in UML. AndroMDA and ArcStyler are commercial tools that are based on a software artifact which includes software modeling and transformations for certain platforms; one of this artifact is available for Java and another one for web services. OptimalJ has three layers architecture with Struts for the support of web applications. A similar concept is followed by Model Integrated Computing [Sprinkle 04] and some tools for support are Generic Modeling Environment and MetaEdit+.

All these tools are based on a metamodel through an abstract language which allows them to define syntax, semantics and DSL visualizations. When it comes to choosing a tool it has got to be done carefully as not all the tools are compatible with all platforms. Eclipse Modeling Framework is today the most popular tool as it is compatible with almost any platform. The following tools are considered best in today’s market.

1. Eclipse: It is used not only to develop software products, but also to develop software tools which allow constructing software products. It allows modeling and data integration by storing metamodels and metadata. One of its main advantages is that its architectures allow locating most of the functionality through independent plugins or even a set of related plugins. Therefore Eclipse can be considered as an extensible tool.

2. Eclipse Modeling Framework (EMF): It is a modeling environment which allows code generation. EMF can be used as a metamodel frame where the common language is a MOF sub-set. EMF is integrated into the set of plugins belonging to Eclipse and it allows using a large variety of software artifacts, with XML schemes, UML models among others. EMF is one of the main tools for IBM.

3. Graphical Modeling Languages (GMF): It is capable of automatically generating models into graphic editors as plugins for Eclipse by using its own infrastructure and EMF.

4. Atlas Transformation Language (ATL): It is a tool based on OMG, MOF, QVT and OCL 2.0 standards. ATL is part of the framework management for models AMMA, which is integrated within Eclipse and EMF. It uses ATLAS software language to define transformations and it is executed over JAVA for a clearer environment.

5. MOFScript: It is a tool for transforming models into text presented by OMG. It is compatible with QVT and MOF. MOFScript defines a source metamodel over which transformation rules operate. The target is generated as a text file output.
6. Java Emitter Template (JET): It is an EMF subproject with the purpose of simplifying the process for generating code automatically. Incoming templates are translated into Java classes, which then are executed by the generator class created by the user.

VI. MDE Applications for Reactive Systems

Reactive systems (robotic systems, artificial visions, sensor networks, home automation, etc.) have become important in the development of software, as these systems are highly complex for their interaction with the physical environment, making it difficult to develop quality software for such application.

Previously, in order to develop those systems, the experience of the designer, the specific functionality features, costs among other features where crucial. Because of this, each new system needs a specific ad-hoc software development, even when the structure and logic might be almost identical to another system previously designed in a different platform. So, the software's structure or logic cannot be reused. So, several proposals have emerged to find a way to reuse software in reactive systems.

One interesting solution is to develop reactive systems applications based on MDE. Still as MDA is quite recent, there are only few studies of MDE applied to reactive systems. Some advantages brought by MDA would be the following:

<table>
<thead>
<tr>
<th>Reactive Systems Disadvantages</th>
<th>Development</th>
<th>MDA Advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ad-hoc solutions by the platform used for each application.</td>
<td>Code non reusable and non flexible applications</td>
<td>Code can be reused and flexible applications. Generic solutions independent of platforms.</td>
</tr>
<tr>
<td>High degree of experience in the infrastructure through the whole process required.</td>
<td>Inefficient and slow design process.</td>
<td>Process design more efficient. Initial phases use DSL concepts, independent of platform.</td>
</tr>
<tr>
<td>Development tools are extremely dependent on the platform being used; therefore knowledge of the platform is crucial.</td>
<td>The knowledge of infrastructure can barely be reused for other solutions.</td>
<td>Generic tools are independent of platforms. Solutions can be reused in different platforms. Code generation can be automated in the final phases of design PSM.</td>
</tr>
</tbody>
</table>

For Home Automation Systems Markus Voelter [23 Voelter] presents a case of study shown in figure 4-5. A metamodel is defined at PIM level, related with the problem and at the PSM level the solution of the domain is given. The models evolve though transformations M2M (model to model) and M2T (model to text). It is an outstanding project, still it has some disadvantages. First, the metamodel for the home automation domain has to be constructed.
for each application. Second, the code generation uses Java services, but there are no implementations considered for other extended systems such as KNX or Lonworks.

Fig. 4-5: MDA proposal for Home Automation Systems by Voelter [23 Voelter]

Another approach is presented by J. Muñoz [24 Muñoz J.] which states the need to use MDE for home automation systems in order to increase the level of abstraction, productivity and software quality. Figure 4-6 shows the proposed methodology in four steps:

1. First requirements need to be specified by the analyst using a modeling language in this case PervML, which consists of UML specifications for the service model (class diagram), the interaction model and the structural model (diagram of components and services).

2. The architect of the system will select the devices and software systems to be used in order to generate an architectural view.

3. Then the developer will implement the OSGi drivers to manage the devices or software systems.
4. Finally a transformation motor will be applied and Java files will be automatically generated as well as other auxiliary resources. These files will be configured to use the OSGi drivers and the generated files will be compiled and packaged within JAR files to be installed in the OSGi server together with the drivers.

![Diagram of MDA approach for Home Automation Systems by J. Muñoz.](image)

This approach is quite enriching but still it has some disadvantages that could be improved. PervML uses UML views of the analyst and architect, but this language is not intuitive for those implicated within the home automation domain. Another disadvantage is that it uses OSGi drivers to control the automated devices, so it is not possible to use all the processing capacity of devices from specific platforms such as KNX/EIB for example. Finally the control is made from a central processor instead of using the advantages of a distributed architecture like the one used by KNX/EIB or Lonworks.

**Conclusion**

The concept of DSL has been introduced, as well as available MDE tools in the market along with some approaches for reactive systems with MDE. Still they could be improved by developing a specific DSL tool which could provide a more realistic description of the model. Next chapter presents a DSL tool for home automation systems, which has an intuitive description of automated systems quite similar to reality.
CHAPTER 5: An Overview Of The HAbitATION Project

CHAPTER 5: An Overview Of The HAbitATION Project .......................................................... 53

ABSTRACT .............................................................................................................................. 54

I. Previous Work DSIE Research Group .................................................................................. 54
II. A DSL for Home Automation Applications ........................................................................ 55
III. Case of Study: Meeting Room .......................................................................................... 59

CONCLUSION ........................................................................................................................ 60
**ABSTRACT**
This chapter gives a brief introduction of how the **HAbitATION** project emerged in the University of Cartagena, which applies MDE to home automation systems, followed by a description of the project itself and its main elements: the catalogue device and the DSL templates. The catalogue device is a set of home automation functional units that are to be used in the creation of DSL models.

I. **Previous Work DSIE Research Group**
As commercial solutions to home automation systems are attached to platforms, a new proposal was developed by Dr. Diego Alonso Cáceres [25 Alonso]. The main idea is to develop an application based on methods using MDE and DSLs, so that the system’s design can be completely unattached from the platform, and then it can be attached to different platforms by simple transformations.

This paper is an additional extension of the work that has been developed by the DSIE (Systems and Electronic Engineering Division) research group of the UPCT (Technical University of Cartagena). DSIE was founded on 1999 as a multidiscipline research group for the following areas:

- Robotic and Control Systems for Industrial Applications
- Service Robots
- Automation of Vision Systems for Inspections
- Electronic Technology for Robotics and Artificial Vision
- Wireless Sensor Networks
- Buildings and Home Automation

The research of this group was integrated within the MEDWSA project (“Marco conceptual y tecnológico para el desarrollo de software de sistemas reactivos”), directed by the “Comisión Interministerial de Ciencia y Tecnología” (CICYT TIN2006-15175-C05-02). MEDWSA is a sub-project of the META Project (Models, Environments, Transformations and Applications) which has been in development from 2007-2009. The main objective of the MEDWSA project is to define the basis for the development of reactive systems using Model Driven Development.

A new branch developed from MEDWSA, which would focus on the study of a particular reactive system type, home automation systems. The HAbitATION project emerged and with it, a tool and methodology capable of developing software models for home automation systems. A Model Driven Architecture was used to develop the methodology and later integrated into the proposed tool along with DSLs, which need the support of a defined requirement catalog and a procedure to trace its modifications and relations.
The overall objective of the HAbitATION project is to define complex quality software (from the requirements until the transformations into code for each platform) that can then be used to design home automation applications and where the user of the software must not necessarily be an expert of home automation systems. The transformations and equivalences of the software artifacts will be supported by (QVT+OCL), with associated quality metrics and bidirectional traceability, which will allow the definition of well founded processes for the software development. The project uses industrial standards with formal methods in order to validate and verify the properties and transformations for each artifact, being developed in the ECLIPSE platform.

II. A DSL for Home Automation Applications

Developing a DSL for Home Automation Systems allows the user to have a visual environment with a catalogue of reusable functional units and a set of home automation interconnections primitives and through the MDE approach generating code automatically for independent platforms can be achieved. The HAbitATION project combines a MDA approach with DSLs to support the definition of these applications, as mentioned previously.

A Home Automation System has functional units, which differ in their architecture and protocols but are identical when it comes to capability. The HAbitATION project proposes a Catalog of Functional Units, so they can be reused in new applications instead of defining them again. Each functional unit has a set of services through which it can interact with other units. These services are often repeated among the functional units, so a Catalog for Services has also been created with Definitions of Services that can be reused in any functional unit.

The DSL graphical view allows modeling the catalogue of functional units and services shown in figure 5-1, which will later be used within applications.
Fig. 5-1: Functional Unit Catalog for Passive Devices and Controllers [10 M. Jiménez]
The Final Passive Functional Units represent those elements that cannot be programmed, such as lights, push buttons, motors, some sensors, etc. On the other hand, the controller functional units are those elements which can be programmed to respond with certain behavior, for example a timer, logic gates, SMS sending, etc. The DSL Application view is used by the developer to design new applications, and in this case the developer does not have to be an expert of home automation. Applications can be specified thorough the catalogue by means of Functional Units, links between the functional units indicating services that are being used and the way in which functional units will interact. As shown in Fig 5-2:

![Fig. 5-2: DSL example: source model at CIM level. [10 M. Jiménez]](image)

A pushbutton PB-1 switches a light LO-1. Elements SWI-1 and SWO-1 are the controller's which provide the switching functionality. So PB-1 interacts through the service PBactivated with SWI-1, which receives an input signal from SWIactivated service. Then SWI-1 interacts with SWO-1 through the services SWIswitchOut and SWOsswitchIn respectively. Finally SWO-1 interacts with LO through the services SWOswitchOut and LOSwitch. So, once the PB-1 gets pushed, the signal travels through the services, the controllers decide what should be done and finally the light LO turns on.

![Fig. 5-3 to 5-6: Proposed method for development of home automation applications. [39 M. Jiménez]](image)
Transformations through the MDA approach occur as shown in figures 5-3 to 5-6: Models in the CIM (Computing Independent Model) are automatically transformed into architectural components in the PIM (Platform Independent Model) layer. By new transformations, components are transformed into executable models for each platform PSM (Platform Specific Model). A meta-model is defined for each MDA level. The CIM meta-model represents the syntax and part of the semantics of that defines DSL. The PIM meta-model is a simplification of the UML meta-model for reactive systems and considers components, activities and state-chart diagrams. For the PSM layer, a meta-model has been defined for the KNX/EIB technology which considers the DOM (Domain Object Model) used by the ETS tool. The tool that has been developed to support the methodology uses the Eclipse development Environment.

The DSL tool has a drawing area to build graphic models for the catalogue and the applications; a graphic palette containing elements that can be dragged to the drawing area and an area where the available properties (attributes, parameters, etc.) are displayed and can be modified for de selected element. This DSL tool can be downloaded from http://hdl.handle.net/10317/854.

![Fig. 5-7: DSL Application View, with options for functional units and links at the right. Inside the window an application diagram example.](image)

The transformation between CIM and PIM layers are completely defined using graph grammar-based approach. Transformation is expressed with rules. Each rule has a left hand side LHS and a right hand side RHS, both of which are graphs. A rule may also have a Boolean application condition, which must not be satisfied to apply it NAC. To apply a rule to the graph which has to be transformed, there must exist a sub-graph isomorphism form the LHS to the host graph. After the application, there must be a sub-graph isomorphism from the RHSL to
the result graph as shown in figure 5-8 b). The result of applying the rules is a component model (see Fig. 5-8 c).

III. Case of Study: Meeting Room

The case of study proposed for the project is based on a meeting room which is used for activities such as meetings, seminars and presentations, shown in figure 5-9.

The case of study groups the control of devices in the meeting room through four main categories: energy, security, comfort and communications.

The Application View of the DSL is used to display an example of a set of requirements. The elements managed by the DSL are 6 lighting points, controlled by 2 pushbuttons.

A pushbutton PB-1 is used to switch lights LDM-1 to LDM-4. Pushbutton PB-2 is used to control dimming lights LDM-5 and LDM-6. PB-2 is connected to dimming controllers DMI-1 and DMI-2 through services PBactivated and DMIactivated. DMI-1, DMI-2, SWI-1 and SWI-2 are then connected to DMO-1 to DMO-6, which are responsible for switching and dimming the lights.

By considering energy saving requirements applied to the lights, some presence detectors are added PIR3-1 and PIR-2, interconnected to SWI-1 and SWI-2 as shown in fig. 5-10. These controllers call the TMPtempIn service from the Timer TM-1 every time presence is detected. So the timer switches on the lights when presence is detected and switches them off after no presence is detected in a period of 300 seconds.
CONCLUSION

HAbitATION project developed a catalogue device for home automation systems which then is used within the DSL models. A brief description of the model transformations is given and followed by a small example on how DSL models can be applied in a practical case of study: a meeting room, which later will be referenced and further detailed in Chapter 6 and 8.
## ABSTRACT

I. A Framework for Home Automation Model Driven Development .................................. 62  
II. Requirement Management .............................................................................................. 62  
III. The Requirement Catalogue .......................................................................................... 65  
IV. DSL Representations ....................................................................................................... 68  
V. Using the Requirement Catalogue with a small practical example. ............................... 69  
VI. Traceability for Home Automation Systems with MDE ................................................ 71  
VII. A Meta-mode for Traceability ......................................................................................... 72  
VIII. Requirements-Trace Tools ............................................................................................. 74  
IX. Example on how to create a Trace ............................................................................... 76  
X. Traceability Benefits ......................................................................................................... 77  
CONCLUSION ......................................................................................................................... 79
ABSTRACT

In this chapter, two main concepts are highlighted requirements and traces. The requirement concept is introduced along with the description of the Requirement Catalogue which has been developed. For every requirement a DSL template has been created and some brief examples are explained along with a simple home automation application, in order to demonstrate the usefulness of the catalogue. Further on the concept of traces is introduced along with a brief demonstration of trace reports for DSL templates and its benefits.

I. A Framework for Home Automation Model Driven Development

Figure 6-1 is a generic representation of a Home Automation System by a MDA approach at different layers: (1) home automation requirements, (2) DSL representations, (3) Integrated Model for devices and (4) code generation for a specific platform. Each model is constructed as to its corresponding meta-model. The correspondences at the requirement layer and the DSL are manually established, in order to have all the partial solutions for each requirement catalogued. This is crucial, because when a new application is developed, the user inspects the catalogue in order to identify if the requirements that will be used already exist for reutilization.

II. Requirement Management

The first step in any software design is to understand the client’s needs and translate them into requirements. For home automation systems, requirements are crucial as they represent the basis of all the projects.

Requirements are defined as the necessity of certain devices in order to accomplish tasks, e.g. turning lights on/off through a switch.
It is common that in home automation systems most of the basic requirements are repeated or quite similar among projects. So by strategically defining them a catalogue can be created which can be applied to any home automation system design.

![Figure 6-5: Requirement Meta-Model [33 Rosique]]

Currently it is possible to find different requirements meta-models in the literature [33 Rosique], each one including different concepts and relationships. Such meta-models improve requirements reuse and serve as a structured requirement reference model. Figure 6-2 shows a meta-model definition for home automation requirements representations.

This meta-model is integrated within the home automation framework shown in Figure 6-1 and has sufficient versatility with a simple structure that could be also applied to other domains. The root element is "Catalogue" where the set of requirements are included. Each of this requirements include a name, description and a validation procedure for the user, once the final system is presented for a final verification, in order to validate whether the system accomplishes the expected functionality.

![Figure 6-3: Requirement Meta-Model and Model within the Eclipse Tool.]

For example, suppose we have a requirement that states "Smoke detection must be included within the installation". The verification procedure for the user would be "Create a situation with sufficient smoke and verify that the corresponding alarms are turned on". Requirements are rarely isolated, instead they are strongly related with other requirements, so this condition gives birth to a new element "RelationShip" which allows the connection of a
“source relationship” to a “target relationship”. The correspondence of type towards the requirement’s relationship is identified with the element “TypeOfRelationship”. Table 6-1 shows an abstraction of the Requirement Catalog based on the structure of this meta-model. The complete Requirement Catalog can be seen at Appendix A.

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>DESCRIPTION</th>
<th>VALIDATION PROCEDURE</th>
<th>TARGETS</th>
<th>TYPE OF RELATIONSHIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Req.1</td>
<td>ON/OFF</td>
<td>Control of light through a push-button or any other domestic mechanism.</td>
<td>Apply push button P8 to verify the correct functioning of the light.</td>
<td>None</td>
<td>--------</td>
</tr>
<tr>
<td>Req.2</td>
<td>Dimmer</td>
<td>Light intensity regulation through the mechanism, short push for ON/OFF, longer push for Dimmer Up or Down.</td>
<td>Push the mechanism (up: push-button) short for ON/OFF and longer for regulation in order to verify correct functioning.</td>
<td>None</td>
<td>--------</td>
</tr>
<tr>
<td>Req.17</td>
<td>Automatic Day Light Sensor</td>
<td>Automatic light switching through a daylight sensor.</td>
<td>Wait for daylight sensor to detect the diminishing light outside, and see if the light gets turned on.</td>
<td>Req1, Req2</td>
<td>Requires at least one of them.</td>
</tr>
<tr>
<td>Req.18</td>
<td>Automatic-Timer</td>
<td>Automatic light switching through time programming (clock).</td>
<td>Program the timer and verify if the light turns on at the programmed time.</td>
<td>Req1, Req2</td>
<td>Requires at least one of them.</td>
</tr>
<tr>
<td>Req.3</td>
<td>Automatic-</td>
<td>Automatic light switching through a presence detector.</td>
<td>Walk near the presence detector to verify the correct functioning of the light.</td>
<td>Req1, Req2</td>
<td>Requires at least one of them.</td>
</tr>
<tr>
<td>Req. Comfort -Illumination</td>
<td>Automatic-</td>
<td>Automatic light switching through a presence detector when sensor has been programmed to function.</td>
<td>Walk near the presence detector when sensor has been programmed for functioning to verify the correct functioning of the light.</td>
<td>Req1, Req2, Req3, Req18</td>
<td>Requires all of them.</td>
</tr>
</tbody>
</table>

Table 6-1: Abstract of Requirement Catalogue Descriptive Table (Note: Full Table can be found in Appendix A)

By searching within the Requirement Catalogue, the user can extract the corresponding fragments of models specified using the DSL. It can be observed on Table 6-1, the defined relationships for each requirement. For example Req#3, requires Req#1 and Req#2, while Req#1 which does not need a relationship as it only requires itself. There can even be inclusion type requirements, a requirement of a higher conceptual level, which would need all previous requirements as shown in figure 6-4 underneath. The type of relationship is not closed to change as it completely depends on the user’s needs.

For this particular project, 3 types or Relationships have been defined:
- Single Requirement: When a Requirement is only needs a relationship with a single specific requirement e.g. Req#23 requires Req#8 (see Appendix A).
- One or More Requirements: When a requirement needs at least one requirement, but can also have more requirements e.g. Req#18 requires Req#1 or Req#2 (see Appendix A).

- Multiple Requirements: When a requirement necessarily requires at least 2 requirements e.g. Req#33 requires Req#14 and Req#8 (see Appendix A).

III. The Requirement Catalogue

A basic set of requirements has been proposed for Home Automation Systems. These requirements were divided into four basic categories:

- Comfort-Lighting or Illumination
- Comfort-Motorized (For devices moved by motors such as Shutters, Blinds, Projectors, etc.)
- Comfort-Climate or HVAC (Heater, Fan, Air Conditioning, etc.)
- Security

Communication requirements have only been included within security for information regarding to alarm types sent to mobiles by SMS. Home Cinema, Internet access and control of the installation have not been included within this project, as it has been developed at an experimental level for a meeting room.

Energy saving as a category is not specifically considered within the catalogue as for the purpose of this project, those requirements types are achieved once specific applications develop including the four basic requirement categories mentioned before. For example, an application might include putting on Standby the A/C system when no presence is detected and turning lights on at a specific dimmer level when presence is detected. The first requirement belongs to Comfort-Climate category, while the second belongs to Comfort-Illumination. Once the application uses both requirements, it achieves to save energy when no presence is detected.

Comfort-Lighting category considers the control of illumination internal or external where the actuators are defined as simple light lamps (LO) or dimmer types (LDM), which respond to different sensors such as presence sensors (PIR), push buttons (PB), sunset sensors (SS), or even week timer (WT). The sensors send the signal to corresponding Input-Output switches (SWI-SWO; DMI-DMO) to turn the lights On/Off or regulate them.

Comfort-Motorized category includes actuators such as Blinds (SHM) or venetian blinds (VBlinds) and Projectors (SC) which are moved by motors, with an input signal (MTI) and an output signal (MTO). They can either be moved by simple push buttons (PB-Up or PB-Down). The blinds can also be moved by a programmable week timer (WT), as well as sunset sensors (SS) or wind sensors (WS).

Comfort-Climate category controls basically to types of devices: Fan-Coil and Air Conditioning. The Fan-Coil has 3 different fan speeds and an electronic valve to let water pass through the fan. This water can be either cold to cool up the room or hot for heat mode use.
The Air Condition on the other hand has two basic modes, Cool-Mode and Heat-Mode. Both devices are controlled by a temperature controller TC, which constantly receives information for the temperature sensor TS. To know the current temperature and the desired temperature it can be set through simple push buttons (PB+ or PB-). The temperature controller has two additional features, TCnight mode and TCcomfortStdby. The first mode is programmed through a week timer (WT). For example, set the temperature level from December to February during the night at 5°C in order to save energy and from June to August to 28°C. The TCcomfortStdby, sets the temperature at a given pre-defined degree, when no presence is detected within the room.

Security category includes a surveillance alarm as well as technical alarms. For surveillance presence sensors (PIR) and a push button are used (PB). If the PB is turned on, then when presence is detected within the room a signal is sent to the bell (BL) for the alarm to sound and at the same time an SMS is sent to a mobile in order to inform the type of alarm being activated. Regarding to technical alarms, three types of sensors are considered for this particular project, fire sensor FD, water sensor WD and smoke sensor SD. In case a sensor is activated, a signal is again sent to the bell BL and at the same time an SMS is sent to the mobile.

The following Table 6-2 is an abstract of the requirement catalogue that has been developed within the four categories.
function for that specific requirement is given. The Catalogue Device, previously introduced in Chapter 5 defines the representation for each of the symbols used in the table above:

At the right side bottom there is an additional table, which shows a subset of requirements that are later defined within the project as:

- Req_Comfort_Light: Is a requirement which uses the set of Req1, Req2, Req3 and Req18 belonging to the Comfort-Lighting category.

- Req_Comfort_Motorized: Is a defined requirement which uses the set of Req4, Req10 and Req11 belonging to Comfort-Motorized category.

- Req_Comfort_Clima: Is a defined requirement which uses the set of Req14, Req15 and Req26 belonging to Comfort-Clima category.

- Req_Security: Is a requirement which uses the set of Req6, Req16 and Req36 belonging to the Security category.

Figure 6-6: Catalogue Device for Home Automation Devices
IV. DSL Representations

Once a requirement has been defined it needs to have its corresponding DSL representation. Figure 6-6 shows the modeling used for a DSL representation corresponding to Req4, which defines the control of Blinds or Shutters through push buttons, PB-Up and PB-Down. The push buttons can be defined as the sensor devices and depending on how long they are pushed the behavior of the motors vary.

When PB-Up or PB-Down are pushed for a small period of time e.g. 0.5 seconds, the shutter goes slightly up or down and stops immediately when the button is no longer pushed. If they are pushed for 3 seconds, the shutter goes up or down even if the buttons are no longer being pushed. In order to stop the shutter or blind when it is moving, a slight push to the PB-Up when the shutter is moving Down or a slight push to PB-Down when the shutter is moving Up should stop the shutter from moving. Venetian Blinds have a similar function and can be represented by the same devices, the only thing that changes is that with short push the layers rotate to close up or open up depending on the button being pushed.

This DSL representation is formed by five functional units: PB-Up_Ext_Blinds, PB-Down_Ext_Blinds (which represent push button up and down respectively), MTI_Ext_Blinds and MTO_Ext_Blinds (which represent the input signal and output signal of the motor) and finally SHM_Ext_Blinds, which represent the Shutter or the Blind itself.

PB-Up_Ext_Blinds is connected to the input signal of the motor MTI_Ext_Blinds. PB-Up sends a signal "PBUpActivated" which is received by the MTI device as "MTIUpActivated" input signal. MTI has a parameter which stores for how long the push button is being activated in order to know how long the motor should rotate. A similar signal and connection is sent by PB_Down_Ext_Blind to the MTI device, but this time the "MTIDownActivated" signal is received. Then the MTI device sends the order to the Motor to move and for how long it should move. The MTO_Ext_Blinds sends a signal to the shutter "MTOup" or "MTOdown" depending on which button was pushed and it is received by the shutter as an "SHMup" or "SHMdown" signal.

![Figure 6-6: DSL representation for Req4 (Blinds Up/Down PB)](image)

This simple DSL diagram clearly demonstrates how the behavior of home automation devices can be modeled.

The next figure 6-7, represents the DSL modeling to control a Fan-Coil and an Air-Condition. It has a total of 23 functional units. On the left side, the sensors are represented in the middle TC is the temperature controller ("intelligent device") and on the right side the actuators, the fan, the water valve VO and the air condition A/C.
The TS or temperature sensor is crucial for the TC in order to know the current room temperature. This sensor is connected into AI, an analog converter device which interprets data and converts it so that the temperature controller is capable of reading the data through the “TCCurrentTemp” input signal. The push buttons PB+ and PB- allow the user to increase or decrease the desired temperature. This push buttons are then connected to their corresponding switches, which then send a signal to the controller as “TCsetPoint+” or “TCsetPoint-". The week timer WT, allows the user to program a given temperature during certain days and hours e.g. on the whole Sunday and during the week from 8:00pm to 7:00am leave the temperature at 7°C in the month of February. WT sends "WToutput" signal and is received as “TCnight” input signal by TC. Finally we have a set of presence detectors which send “1” when presence is detected and “0” when no presence is detected. If any presence detector detects presence it sends a “1” signal which is then transformed to a “0”, so that the TC input signal “TCcomfortStdby” cannot be activated. Then when no presence is detected a “0” is sent and transformed with the NOT gate into “1” so that the input signal of the TC gets activated and the controller gets into standby mode. The TC controller has several output signals, TCFanMin, TCFanMid and TCFanMax to specifically control the speed of the fan through switches SWO_Fan1 ... SWO_Fan3. The TCcoolSwitch is sent to the valve VO through the SWO_Vo switch and to the Air Condition cool-mode through its corresponding switch. The same is for TCheatSwitcht, which is sent to turn on the Heat-Mode of the Air-Condition.

**Figure 6-7**: DSL representation for Req32 (A/C and Fan-Coil Ctrl. Through TS, PIR, WT and PBs)

V. **Using the Requirement Catalogue with a small practical example.**

The following is a supposition of clients needs to automate his or her home. When interviewing a client the following information was acquired:

“I want to have lights that I can regulate in my living room and dining room. In the patio and the balcony I’m concerned about security as it is easy to break in. During the night we enjoy staying outside so if lights switch on automatically that would be neat. My wife often forgets to turn off the stove in the kitchen and it gets fed up with smoke, if you could do anything about it that would be great. I want the shutters of the bedrooms to go up automatically on mornings and down on night. Last year we had a broken pipe at the bathroom and we got both levels of..."
the house flooded, I want to have something that tells me if there is a flood but just in the two bathrooms from the upper floor as I don’t want the installation to be expensive and the toilet is broke so we only use it to keep things in there. Finally I would only accept the installations if you could do it without reinstalling cable in the whole house, with a wireless thing or so. Do you want to join us on Friday? Every Friday at 9:00pm we play poker at the living room”.

The house would need the following home automation devices as represented in figure 6-8

![House Plan with devices included in it and a descriptive table of the devices.](image)

The client’s needs can be represented by the following requirements from the catalogue:

**Req_1**: Lights to turn On/Off (Hall, Kitchen, Patio, Terrace, 2 Bathrooms)

**Req_2**: Dimmer Lights to turn On/Off and to be able to regulate their intensity (Living room, dining room, 3 bedrooms).

**Req_3**: Lights to turn On/Off with Presence (Living room, patio and terrace)

**Req_5**: Shutters to get Up/Down with Sunset Sensor (3 Bedrooms)

**Req_17**: Light On/Off with Sunset Sensor (Patio Light)

**Req_18**: Light On/Off at the Living Room at 9:00pm on Fridays with a Weekly Timer

**Req_36**: Smoke Detector at the kitchen

**Req_16**: Water Detector at both bathrooms

**Req_6**: Presence detector at three different points, patio, terrace and living-room.
As DSLs for each requirement are created it can be noticed that the same device can be used for several requirements. So an overlapping effect occurs. For example lights that turn On/Off by a push-button, can also be turned On/Off by a presence sensor or by a weekly timer. The same presence sensor that turns On/Off the lights, also switches an alarm and sends an SMS to a mobile when it is activated. It turns out to be the same alarm and mobile that is used by the smoke detection and flood detection as shown in figure 6-9. There can be even more devices overlapping and as the installation increases; making it even harder to keep a trace of the correspondence between device and requirement. Therefore, classifying which devices belong to which requirements becomes a complex and important task.

Figure 6-9: DSL representations for various requirements needed within the previous application.

A proper solution is presented by Traceability, where traces for each requirement to its DSL devices or Functional Units and their links or connections as well, are created. Further on any application can be traced into several requirements with sub-requirements, having sub-traces from each requirement to its DSL devices and links.

VI. Traceability for Home Automation Systems with MDE

According to Goetel [40 Goetel], Requirements Traceability refers to “the ability to describe and follow the life of a requirement, in both a forward and backward direction. Forward traceability traces software artifacts which are obtained during the software development, whereas backward traceability traces each software element with its
corresponding artifact involved in their acquisition. Tracing software artifacts allow knowing in detail the systems development and response towards change. Thomas Behrens defines several key goals for traceability in software development:

1. To be able to validate if different requirements have been considered.
2. Validate if the implementation obtained accomplishes all the initial requirements and how this accomplishment has been possible.
3. Know the impact that can be caused if a requirement is modified.

In a MDE process traceability is crucial. Due to the extensive use of transformations used throughout an MDE process it becomes essential in order to understand how and why artifacts are created.

VII. A Meta-mode for Traceability

For this project directed by the DSIE research group, a specific meta-model for traceability has been developed as shown in figure 6-10. This meta-model contains a “Link” pointing to any “ModelElement” via two references: a “source” element and multiple “target” elements. The “Composite Link” allows defining different levels to group Links into more complex levels. “LinkType” allows to classifying the existing relations and distinguish at what level of the process development the trace is located.

The process considers a model using traceability among two consecutive abstraction levels as shown in figure 6-1.

For example, for the particular case of traceability within home automation requirements and modeling correspondence with the DSL, the Link element of the meta-model is defined by:

- A ModelElement (source) which references an element of the home automation requirements meta-model.
- A ModelElement (target) which references an element of the DSL.

Figure 6-11 shows an example of traced elements between two consecutive levels of abstraction (requirements to design, design to elements of language programming, etc.).
It can be seen that it is possible to integrate one-to-multiple relations within the same link. At the same time, a target can be the same destiny for several other traces with different origins.

![Figure 11: Traces can be represented by the traceability meta-model [33 Rosique]](image)

Traceability models are created during the process transformation of models. Trace links between source and target artifacts of a transformation may be created implicit or explicit.

In the first model of traceability (between requirements of home automation systems and the corresponding DSL model) traces are manually created by the user. Then from the DSL to the final code generation, all the traces are automatically obtained as part of the model transformation process from origin to destiny.

In other words, while the transformations that obtain models close to the final platform are executed, traces among the elements involved and their corresponding links are recorded into the traceability repository.

In order to achieve these traces, rules of transformation must have an extension for recording traces within the defined meta-model. Figure 6-13 includes an extension added over the graph of figure 6-12, where traceability is included by modifying the RSHS part of the rule.

It can be observed how a new element appears which corresponds to the traceability model. Each “TraceLink” element of the transformation rule stores the information (name, description) for the instances of “Link type” from the traceability meta-model represented by figure 6-10.

Finally, all the existing traces between the DSL elements and the components of the model are gathered at the “Traceability Model#2”.
VIII. Requirements-Trace Tools

Several authors consider of extreme importance taking into account certain features when developing a tool from traceability support:

1. It must be possible to manage models at different levels of abstraction.
2. A unique metamodel must exist along with its respective models at different levels or layers.
3. The stored information in the traceability models must be persistent.
4. It must be possible to identify where a traced element is located and when was the trace created.
5. It is crucial for the tool to be capable of full integration with other external applications.

6. The tool must allow automatic creation of traces as well as manually.

Considering these features [33 Rosique] a RTT (Requirement-Trace-Tool) has been developed. It allows generating detailed reports of the traced elements from the trace models. Trace inspection could be used to retrieve all artifacts that were generated from a certain artifact, and find the transformations responsible for creating them. This report allows the system designer to analyze how a requirement has been considered in the automated code generation process and information on the given solution.

Figure 6-14 shows a screen capture of the RTT tool. It shows the elements that are stored on the “Traceability Model#1”. Each trace is detailed by its name, description and information regarding to its source and target elements which have been traced.

The first row of the report shows a CompositeLink which groups all the information of the involved Links in the solution using the DSL for the illumination requirement.

In this particular case, the represented trace links are Req#1 and Req#2, Req#3 and Req#18 which target their corresponding DSL elements as shown in Figure 6-14.

<table>
<thead>
<tr>
<th>Trace Link</th>
<th>Source Link</th>
<th>Target Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Description</td>
<td>Source Model</td>
</tr>
<tr>
<td>Req#1</td>
<td>CompositeLink</td>
<td>Req#1, Req#2</td>
</tr>
</tbody>
</table>

Figure 6-14: Abstraction of a Trace Report
Figure 6-15 represents the corresponding report to “Traceability Model#2”, which is automatically generated as the transformation process of the specification by using the corresponding DSL for the illumination requirement.

The “Traceability Model#3” is generated during the execution process of the M2M transformations from the model components to the corresponding platform in this case KNX/EIB.

IX. Example on how to create a Trace

First the traceability meta-model is created as “Traceability.ecore”. From this metamodel all traces are created as corresponding models towards it.

Traceability.ecore → Root → Create Dynamic Instance: TReq1_I.xmi

Figure 6-17: A trace model in Eclipse Tool
A trace model which represents the trace for Req1 illumination requirement is created. TReq1_I.xmi is formed by a Composite Link TCompReq1(Lights On/Off), which is further on composed in this particular case of a single Link TReq1. This link has as source the “Requirement LIGHT On/Off” and as target several functional units such as PB-1 and functional unit links as well, not all of them can be seen, that is one of the reasons why a trace report can be very useful. The trace model TReq1_I.xmi needs to load the traceability meta-model, the requirement meta-model, the requirement catalogue model and the model for the particular requirement being used in this case Req1.

As requirements become more complex, a single trace can be formed by several Composite Links as shown in Figure 6-17. In this particular case a specific application is represented by a model MeetingRoomV3.xmi. It uses a set of comfort-clime, comfort-motorized, comfort-illumination and security requirements.

When the trace model TMeetingRoomV3 is created, four composite links are created, each one corresponding to a category. For example, for comfort-clime a composite link CompTReqComfortMotorized is created. This composite link includes 3 sub-links which correspond to trace of Requirement for TReq14, TReq15 and TReq26. A similar situation occurs with the other categories.

![Figure 6-18: A two levels composed trace model in Eclipse Tool.](image)

For each trace requirement, no matter how simple or complex it can be a corresponding trace report can be created. The trace catalogue can be studied with further detail in appendix A along with the full requirement catalogue and DSL representation.

X. Traceability Benefits

The use of traceability brings up the following benefits:

It allows validating whether all the automation requirements have been supported. This validation can take place through the traceability reports shown previously on figure 6-18. First, it is possible to prove that all the considered requirements are present at the current DSL
Second, each specific requirement can be verified independently, e.g. Req#37 which includes smoke detection, fire detection and flood detection alarms (a subset of 3 requirements Req#6, Req#16 and Req#36) can be inspected when verifying the Functional Units and Links that it should have as shown in its DSL model figure 6-19.

Another important benefit is that it can be verified whether the DSL model is compliant with the home automation requirements.

By using the traceability report, it can be proved if a given element of the DSL model is compliant towards its corresponding requirement. In order to do so, DSL elements must have the semantic of the related requirements for home automation validated. The traceability report allows the analyst to know which home automation requirements have given place to the DSL model being analyzed.

This is possible if the analyst has correctly cataloged all the templates that were used during the rules implementations for translations. Thus the verification of requirements also needs the traceability report. Still, when the verification shows a correction is needed; it is not automated as the correction needs to be done manually, considering all the devices, links or connections and parameters involved within the requirement.
Traceability also allows establishing the impact of changing a home automation requirement. It is quite common that the client suggests continuous changes among certain requirements during the development process. The traceability report allows evaluating the impact of those changes before applying them.

The first useful data to take into account by the analyst is how many DSL elements, architectural components and how many elements of the specific platform layer need to be modified. This study of impact is crucial as it allows having an integrated vision of the changes that need to be taken into account when modifying a requirement.

Finally, the analyst can verify the impact of changing a single requirement over the rest of the requirements that have already been implemented. For example consider Req#1 which states “Turning Lights On/Off through a push-button”, then the analyst decides that this requirement should also include the possibility of turning the lights automatically when presence is detected.

This modification would imply to add a presence sensor functional unit, a switch and the corresponding links interconnections. Then again this new requirement as it has a presence sensor device which is used in security requirements overlaps with other requirements such as Req#6. The report allows knowing which requirements are associated to each device; so that when new devices are included, the analyst is capable of verifying that the new device will not alter the behavior of other overlapped requirements, as they use the same device.

CONCLUSION

Requirements are the essence of the client’s needs, therefore it is important to gather them into a catalogue so they can be reused. Perhaps it is one of the most outstanding features and advantages for using a modeling tool, as the reuse of requirements is what makes the application unique, for ad-hoc applications don’t have to be done from zero, thanks to the possibility of reusing requirements. But when a great amount of requirements is being reused, it becomes extremely important to keep trace on how does requirements affect other requirements when shared devices are used. Therefore a trace tool has been developed and some examples of trace reports have been presented.

There is still a road ahead within this project. At the moment the creation of traces for requirements within applications is done manually. A feature work could emphasize on the automation of those traces. For the requirement part, now that a catalogue has been developed; the next step would be to create a tool which would allow to use those requirements at the click of a button, as well as to load new requirements designed into the tool.
# CHAPTER 7: Methodology To Use Catalogues For An Application

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>81</td>
</tr>
<tr>
<td>I. Introduction</td>
<td>81</td>
</tr>
<tr>
<td>II. Overall procedure for Using Requirement Catalog in a given Application</td>
<td>81</td>
</tr>
<tr>
<td>III. Procedures for Inspecting the Requirement Catalogue and Reusing Requirements or Adding New Requirements for a Given Application</td>
<td>82</td>
</tr>
<tr>
<td>III. Procedure for adding new devices into the Catalogue Device:</td>
<td>86</td>
</tr>
<tr>
<td>CONCLUSION</td>
<td>88</td>
</tr>
</tbody>
</table>
ABSTRACT

The objective of this chapter is to provide a methodology that can be used during the creation of a new application, taking into account the necessary steps which would be needed, such as how to use the available requirements and devices or if new categories, requirements or devices should be included within their respective catalogs.

I. Introduction

At this point a Home Automation Devices Catalogue has been presented in Chapter 5, followed by a Requirement Catalogue, DSL representations for requirements and traces from requirements to their DSL representations in Chapter 6. The purpose of this chapter is to explain step by step how to use and modify the Requirement Catalog and the Device Catalog according to the developers needs.

II. Overall procedure for Using Requirement Catalog in a given Application.

The overall procedure towards the development of a new application is as follows. The problem must be first analyzed. Then the client’s needs should be interpreted as a set of requirements.

Once the requirements have been specified, the developer must inspect the Requirement Catalogue. If the requirements are available, then the user can use the current DSL templates, otherwise the user needs to add the new requirement into the Catalogue and create its new DSL template.

Parallel to the creation of the template, the developer will have to manually create the new traces from the new requirement to its DSL devices along with a trace report.

In case a specific device is needed and it does not currently exist in the Device Catalogue, the user will have to create the new device, with its output and input signals as well as the needed parameters. Only then can the user proceed to create the requirement, the template and the traces.

Once all the requirements have been implemented within the application along with the corresponding DSLs, the automated code generation into the specific platform can take place.
III. Procedures for Inspecting the Requirement Catalogue and Reusing Requirements or Adding New Requirements for a Given Application.

The first step is to analyze each single requirement obtained from the client's needs, by making sure that it cannot be further divided into more simple requirements as shown in figure 7-2. Once all requirements have been simplified as much as possible, their existence can be verified by procedure shown in figure 7-4.

While verifying a requirement three different things can happen. The best scenario would be if the requirement is found within the catalogue, as it can then be used along with its DSL template.

Another scenario could be that a requirement which should be in one of the predefined categories is not found. In that case the procedure for defining a new requirement should take place.

A worse scenario would be that the category does not exist, and then the procedure for defining a new category should take place, followed by the procedure for defining a new requirement. The entire procedure would be as shown in figure 7-5 and 7-6.
Figures 7-3 and 7-4: Decomposing and Verifying Requirements existence within the Catalogue.

Figures 7-5 and 7-6: Creation of a new category and insertion of a requirement into its category.
Once the Requirement has been defined two things have to be taken into account. If the requirement is related to other requirements, it needs to follow the “Assign Relationship procedure”, and only then its corresponding DSL template can be created.

The relationship type assignment proceeds as indicated in figure 7-7.

![Diagram](image)

Figure 7-7: Assigning Relationship type and relations to a requirement.

The creation procedure of a DSL template is shown in Figure 7-8.

Once the DSL has been created, the trace from the Requirement to its DSL components needs to be created as shown in figure 7-9.

All requirements with their respective DSL representations are gathered within a single application, in such way that previously created requirements can be reused by passing parameters, links and functional units of the original DSL template into the new specific DSL template for a given application.
Figure 7-8: Procedure to Create DSL Templates

CREATE NEW METADATA:
CREATE NEW FUNCTIONAL UNIT (LOGICAL DEVICE)
ADD DESCRIPTION TO FUNCTIONAL UNIT
ADD NAME TO FUNCTIONAL UNIT
SELECT FUNCTIONAL UNIT
ARE ALL FUNCTIONAL UNITS DEFINED?
NO
YES
SELECT INSTANCE TYPE FROM CATALOG
DO YOU CREATE NEW CATALOG?
NO
YES
SELECT INSTANCE TYPE DEFINED IN CATALOG
END

Figure 7-9: Procedure to create a new trace for a requirement

CREATE NEW TRACE FOR THE NEW REQUIREMENT:
START:
CREATE NEW TRACE
NAME TRACE
SELECT SOURCE (REQUIREMENT NAME)
SELECT DSL TARGET (ALL FUNCTIONAL UNITS AND LINKS OF THE REQUIREMENT'S DSL)
END
III. Procedure for adding new devices into the Catalogue Device:

When a new device needs to be added within the catalogue, first it has to be specified whether it is a passive device or a controller device. Passive devices are usually sensors (input devices) or actuators (output device) which respond or send a signal. The controller devices are usually microprocessor, which decide what to do with a given input signal and where to send it, e.g. temperature controller. Once it has been decided whether it is a controller device or a passive device it needs to be classified. The classification for controller devices is rather simple and is shown within figure 10. After it has been classified a module which represents it is created with a given name, input /output signals and parameters that this device might use. The classification for passive input and output devices is shown in figures 7-11 and 7-12.
Figure 7-111: Classification into Final Passive Output Devices

START

CLASSIFY FINAL PASSIVE OUTPUT DEVICES

Is it a Light type?

YES

CREATE NEW MOBILE DEVICE

NAME DEVICE

DEFINE INPUT AND/OR OUTPUT SIGNALS

ADD NEW PARAMETER TYPES FOR THE DEVICE

END

NO

Is it a Motor type?

NO

Is it a Misc type?

NO

ADD NEW FINAL PASSIVE INPUT CLASSIFICATION

Figure 7-12: Classification into Final Passive Input Devices

START

CLASSIFY FINAL PASSIVE INPUT DEVICES

Is it a Sensor?

NO

CREATE NEW MOBILE DEVICE

NAME DEVICE

DEFINE INPUT AND/OR OUTPUT SIGNALS

ADD NEW PARAMETER TYPES FOR THE DEVICE

END

YES

Is it a Detector?

NO

Is it a Button type?

NO

ADD NEW FINAL PASSIVE INPUT CLASSIFICATION
CONCLUSION

A generic methodology for developing a home automation system from its model until the transformation to its platform is presented. Then steps 2, 3 and 4 (Catalogue Inspection; reuse of Requirements and DSLs; Creation of New Requirements and DSL Templates) of this methodology are furthered detailed, by simplifying requirements and verifying their existence within a category. In case the category does not exist, procedure to create a category is explained, the same as for the creation of new requirements, followed by the steps to creating DSL templates and finally the methodology for creating traces is further explained. In case a device does not exist within the Catalogue, the necessary steps for its inclusion are also briefly explained. In Appendix B the methodology is implemented within the Eclipse tool, step by step on how to create requirements, DSL templates and traces.
CHAPTER 8: Meeting Room Applications

ABSTRACT .................................................................................................................. 90
Introduction .................................................................................................................. 90

I. Security Application .................................................................................................. 91
II. Comfort – Clima ....................................................................................................... 92
III. Comfort – Motorized .............................................................................................. 94
IV. Comfort- Illumination ............................................................................................. 95
V. Use-Cases: Basic application of Meeting Room ...................................................... 97
VI. Use-Cases: Middle application of Meeting Room .................................................. 98
VII. Use-Cases: Complete application of Meeting Room ............................................. 100
CONCLUSION .............................................................................................................. 102
ABSTRACT
This chapter covers a set of home automation applications for the Universities meeting room. Applications have been developed with two different approaches. One focused on using all the available devices within the meeting room by the categories (Illumination, Motorized, Clime and Security). The other approach developed the simplest application possible and evolved it into a middle application using most of the devices, except for the ones of the security category. Finally an application was developed which uses all of the devices available within the meeting room and gathers all the DSL elements of the applications by category.

Introduction
Several applications have been developed considering the automated meeting room of the UPCT University in Cartagena. Applications have been developed from two different perspectives. One perspective is to develop specific applications for a given category with all the devices within the installation. In that case four applications were developed:

- Req_Comfort_Illumination
- Req_Comfort_Motorized
- Req_Comfort_Clima
- Req_Security

The second perspective is to develop a basic application for the meeting room “MeetingRoomV1” which only includes light control, shutter and projector. Then a second application “MeetingRoomV2” was developed which includes all the features of the first application plus clime control, always considering only the devices available in the meeting room of the university. Finally a third application was developed “MeetingRoomV3” which includes the features of the previous applications, plus security requirements and it tries to reach all possible requirements that can be achieved with the meeting room devices. This final application is also formed by Req_Comfort_Illumination, Req_Comfort_Motorized, Req_Comfort_Clima and Req_Security. The following sections offer a brief description of each application.

Figure 8-1: Meeting Room and its devices
The architectural plan is shown in figure 8-1. The meeting room has a set of home automated devices, six dimmer lights represented by LDM-1 to LDM-6, two presence sensors PIR3-1 and PIR3-2, a temperature sensor and a temperature controller TS, and TC, two simple push buttons PB-1 and PB-2 for light control and alarm activation, then four Push Buttons Up/Down, PB-Up-1, PB-Down-1, PB-Up-2, PB-Down-2 for the shutter and the projector, two more Push buttons PB+ and PB-, to indicate the desired room temperature, a Fan coil, with a temperature controller TC and a temperature sensor TS and finally security devices, two water detectors WD-1 and WD-2 and a smoke detector SD-1.

I. Security Application

With the purpose of controlling security devices within the Universities meeting room, the following application was developed: Req_Security.xmi.

The devices controlled are marked on figure 8-2. Figure 8-3 shows its DSL model representation for the application. This security requirement is formed by three previous requirements which are reused within this application. For this particular case Req6 (Intrusion Detection), Req16 (Flood Alarm) and Req36 (Smoke Detection) are used.

The DSL representation clearly demonstrates how certain devices are shared among some requirements, e.g. the bell device is shared by the three requirements. A corresponding traceability model was developed for this particular application in order to keep track of the functional units and links that are shared among requirements.

The water and smoke sensors are connected to internal devices called switches, which turn on whenever something is sensed. The signal is then sent to a timer TM2. The timer switches on the bell as long as it receives the “active” signal from the sensor.

As soon as the sensor stops detecting water or smoke, the signal is switched off, so the timer switches off the bell in the pre-established time. The timer also sends an SMS message when either smoke or water has been detected.

The presence sensors are connected to switches which send the signal when presence is detected. If the PB for alarm is turned ON, this signal is also sent to the timer TM2, which will then turn on the alarm and send a message to the mobile. But if the PB of the alarm is turned off, then the presence detection will not arrive to TM2. This is achieved by the AND logic gate.

Finally, regardless of whether the PB alarm is turned On/Off, the presence detectors will always send a signal to timer TM, which will then send signals to the switches DMO-6 and DMO-5 to turn on the lights LDM-6 and LDM-5.
II. Comfort – Clime

To control the clime devices (fan-coil) in this case a specific application model was also developed Req_Comfort_Clima.xmi. Figure 8-4 shows the meeting room and the devices being controlled within this application. Figure 8-5 shows the DSL model representation for this application.
Figure 8-4: Meeting Room and devices used for the comfort-clime application.

Figure 8-5: DSL template for the comfort-clime application.

Again in this representation it can be seen how devices are shared by several requirements. The requirements being used are Req14 (Fan-Coil Temp. Ctrl.), Req26 (Fan-Coil Timer Ctrl.), Req15 (Fan-Coil Presence Ctrl.).

The temperature sensor is connected to an analog converter, which reports the current temperature to the controller TC.

The push buttons PB+ and PB- regulate the desired temperature and when they are turned on their corresponding switches SWI_PB+ and SWI_PB- are activated.

These switches indicate the controller to increase or reduce the temperature by one degree. The week timer WT is programmed by week and hour to turn on the controller TC into the TCnight mode to reduce energy consumption.

The presence sensors have a similar connection as in the previous security requirement but with a small difference. When presence is detected a signal “1” is sent through the switches, but the controller needs a “1” when no presence is detected, so a logic NOT gate is added before the TC controller.
Once the controller receives any input signal, it will decide whether to speed up/down the fan by the signals TCFanmax, TCFanmid, TCFanmin and to turn on/off the water valve VO with cool water.

Between the TC controller outputs and the final devices FAN and VO, switches are set to turn On/Off the signals to those devices. At the appendix A the traceability model for all the applications in this chapter can be seen.

III. Comfort – Motorized

Req_Comfort_Motorized.xmi DSL model was designed to control motorized devices within the meeting room, in this case a Blind near the window and a Projector at the entrance of the meeting room.

The particular needs for this meeting room require the projector to be controlled only by PBUp-1 and PBDown-1 and not automatically because of security reasons. The shutter or blind on the other hand, can be controlled by PBUp-2 and PBDown-2, and by a week timer WT to bring up the blinds during the day, down during night and during pre-scheduled presentations. Figure 8-6 shows which devices within the meeting room are being controlled and their physical location.

![Figure 8-6: Meeting Room and devices used for the comfort-motorized application.](image)

Figure 8-7 is the DSL representation for this particular application. The control of the Projector occurs as following. As soon as PB-Up or PB-Down is activated a signal is sent to MTI, which is the Motor input signal, it indicates whether the Projector should go up or down and for how long.

If the push has been for a large period of time e.g. 5 seconds the projector should go completely Up/Down.

For short pushes the projector goes slightly Up/Down if it wasn’t previously moving, but if it was moving short pushes stop the movement. MTI sends the movement signal “move” to MTO which is the output device of the Motor. MTO sends the signal that finally moves the projector Up or Down.

PB-Up_Ext_Blinds is connected to the input signal of the motor MTI_Ext_Blinds. PB-Up sends a signal “PBUpActivated” which is received by the MTI_Ext_Blinds device as
“MTIUpActivated” input signal. MTI_Ext_Blinds has a parameter which stores for how long the push button is being activated in order to know how long the motor should rotate.

A similar signal and connection is sent by PB_Down_Ext_Blind to the MTI_Ext_Blinds device, but this time the “MTIDownActivated” signal is received. Then the MTI device sends the order to the Motor to move and for how long it should move. The MTO_Ext_Blinds sends a signal to the shutter “MTUp” or “MTDown” depending on which button was pushed and it is received by the shutter as a “SHMUp” or “SHMdown” signal.

MTI_Ext_Blinds receives an additional incoming signal from WT_Blinds (week timer) and a NOT logic gate. The signal that comes direct from WT_Blinds to MTI_Ext_Blinds is used to move upwards the blinds as it has been previously programmed by the WT_Blinds.

As the WT_Blinds output is always “1”, it can be programmed to give a “0” output for the Blinds to go down, therefore it would need a Logic Not gate to indicate to MTI_Ext_Blinds that the Blinds should come down.

![Figure 8-7: DSL template for the Comfort-Motorized Application](image)

### IV. Comfort- Illumination

The DSL Model Req_Comfort_Illumination.xmi controls the entire illumination of the room by presence sensors, a single push button PB and a week timer. Presence sensors PIR3-1 and PIR3-2 turn on lights LDM-5 and LDM-6.

The week timer when programmed turns on the lights LDM-2 and LDM-3. The push button PB-1 turns on all the lights from LDM-1 to LDM-6. Devices are again shared among different requirements as shown in Figure 8-9
The presence sensors shown in Figure 8-9 have a similar connection as the security requirements, where a logic gate OR groups them, so that when presence is detected by any of the sensors a signal is sent to TM2, which activates the switches as soon as presence is detected and leaves them activated for a longer period of time e.g. 30 seconds after presence is no longer detected.

So when presence is no longer detected the lights will not turn off immediately. The activation of TM2 is sent through switches DMO-6 and DMO-5 to the lights LDM-5 and LDM-6 as a switch on signal. The week timer is sent to Switch_WT only for clarity in the diagram, because in practice this switch can be avoided by connecting WT_Lights directly to DMO-2 and DMO-3 output switches, which then send the dimmer signal to lights LDM-2 and LDM-3. Finally the push button PB_LDM_ON sends an activation signal to the input dimmer switch DMI-all, which then activates all the lights at the indicated dim level, through output dimmer switches DMO-1 to DMO-6, who then turn on lights LDM-1 to LDM-6 respectively by a dimmer signal.
V. Use-Cases: Basic application of Meeting Room

A basic application has been developed for the Meeting Room which uses some requirements of the motorized and illumination categories. The devices being used are shown on figure 8-10. It uses all the lights of the room controlled by a PB-1, the shutter and the projector controlled by PB-Up-1, PB-Up-2, PB-Down-1, PB-Down-2.

![Figure 8-10: Meeting Room and devices used for the basic application.](image)

In order to accomplish this application the following requirements of two different categories where re-used in figure 8-11:

![Figure 8-112: DSL template for the Basic Application](image)
Comfort-Motorized: Req4 (Blinds Up/Down PB); Req10 (Blinds Up/Down WT); Req11 (Projector Up/Down PB). It can be seen that Req4 and Req10 share functional units MTI\_Ext\_Blinds, MTO\_Ext\_Blinds and SHM\_Ext\_Blinds. On the other hand Req11 is completely independent as it does not share devices with any other requirement.

Comfort-Illumination: Req1 (Lights On/Off by switch); Req2 (Lights On/Off by Dimmer). This case is particular, as both requirements share all devices. With Req2 lights can be turned On/Off the same way as if it would be done by Req1, so Req2 will always include all the features of Req1 and that is why they share the same devices.

VI. **Use-Cases: Middle application of Meeting Room**

To enrich further the basic application, simple fan-coil control was included as sown in figure 8-12 where the devices marked by red are the new ones corresponding to the middle application and the black ones correspond to the basic application.

![Figure 8-13: Meeting Room and devices used for the medium application.](image)

Figure 8-13 shows the DSL model for this application. The new feature included is Fan-Coil Temperature control through the temperature sensor TS and the push buttons Temp+ and Temp- to indicate the desired temperature within the room.

An additional requirement can be found within the control of the lights, this time they can also be programmed by the week timer WT\_Lights. This time requirements belonging to three categories are being used:

Comfort-Motorized: Exactly the same requirements as in the basic application for motorized category where used.

Comfort-Illumination: Req1 (Lights On/Off by switch); Req2 (Lights On/Off by Dimmer); Req18 (Lights On/Off WT). The same as before Req1 and Req2 share all devices, but Req18 shares with them devices DMO-2, DMO-3, LDM-2 and LDM-3.

Comfort-Climme: Req14 (Fan-Coil Temp. Ctrl.)
Figure 8-13: DSL template for the medium application.
VII. Use-Cases: Complete application of Meeting Room

![Meeting Room and devices used for the complete application.](image)

This final application uses all the devices within the meeting room except for BL-2. In black the devices used in the basic application are marked, while the blue marks belong to the devices added within the middle application and the red once belong to the new devices added for this particular application. This application uses devices belonging to all the defined categories. So by gathering all the components of previous applications Req_Comfort_Illumination, Req_Comfort_Motorized, Req_Comfort_Clima and Req_Security this new application is created.

Comfort-Motorized: Req4 (Blinds Up/Down PB); Req10 (Blinds Up/Down WT); Req11 (Projector Up/Down PB). It can be seen that Req4 and Req10 share functional units MTI_Ext_Blinds, MTO_Ext_Blinds and SHM_Ext_Blinds. On the other hand Req11 is completely independent as it does not share devices with any other requirement.

Comfort-Illumination: Req1 (Lights On/Off by switch); Req2 (Lights On/Off by Dimmer); Req18 (Lights On/Off WT); Req3 (Lights On/Off by presence sensors). The same as before Req1 and Req2 share all devices, but Req18 shares with them devices DMO-2, DMO-3, LDM-2 and LDM-3 and Req3 shares with Req1, Req2 devices DMO-5, DMO-6, LDM-5 and LDM-6.

Comfort-Clima: Req14 (Fan-Coil Temp. Ctrl.), Req26 (Fan-Coil WT Ctrl.), Req15 (Fan-Coil Presence Ctrl.). Req26 and Req15 need all the devices included within Req14, so they share all devices belonging to Req14 and their links, TS, AS, PB+, SWL_PB+, PB-, SWL_PB-, TC, SWO_Fmax, SWO_Fmid, SWO_Fmin, SWO_VO, FAN and VO. Req26 and Req15 accomplish energy saving goals, as they put the TC controller into TCnight mode as programmed by the week timer WT and TCStdBy mode when no presence is detected in the room. For example on Sundays when the meeting room is not used the fan-coil could be set to TCnight mode during the whole day, whereas on week days, when the meeting room is frequently used, when no presence is detected it can be set to TCStdBy mode.
Security: Req6 (Intrusion Alarm), Req16 (Flood Alarm), Req36 (Smoke Alarm), they all share devices BL, SMS, TM, SWO_BL and are an exact copy of the security application previously presented, therefore all of its requirements can be reused.

Figure 8-15: DSL template for the complete application.
CONCLUSION

Throughout this chapter it has been clearly demonstrated the impact of modeling a home automation system by the applications presented. It is important to notice that for all DSL diagrams shown trace reports are available at appendix A for proper control over requirements being used with shared devices. Some improvements can be done e.g. in the security category the device BL-2 was not included, so if the user in the future would like to separate a bell for intrusion alarms and the other bell for technical alarms it would be possible, with little effort, modifying just two applications, as there would be no need to change any requirement. Once again, this demonstrates the flexibility in design that can be achieved by using MDE.
CONCLUSIONS AND FUTURE WORK

Home Automation has had an outstanding development during the last decade, but there is still much to keep developing. Now a days we have different technologies to work with available, but as mentioned before, KNX/EIB defined in a set of European and international rules, EN 50090 and ISO/IEC 14543-3X, is the most popular open system installed in Spain and Europe [12 Mint]. Therefore the development of the HAbitATION project of the UPCT University considered KNX/EIB for its final implementation.

The advantages of MDE have been presented and it has been proposed to apply this approach for home automation systems. As the whole installation can be divided into models of components and requirements, such that they can be reused for other similar home automation systems. Together with MDE the use of traceability becomes very helpful, because traceability gives the advantage of controlling software artifacts and their relations during the development process obtaining efficient results if developed through MDE.

Domain Specific Languages have been presented and are considered an essential tool throughout the development of the project. The DSL tool for home automation systems developed by the DSIE research group proves out to be efficient and intuitive in the design of home automation systems. A catalog of devices with incoming and outgoing services that can be used with the DSL tool was presented, along with some examples of DSL templates for small applications within the meeting room of the University.

The importance of requirements has been highlighted and abstractions of the developed requirement have been presented in order to use them with applications. But requirements themselves together with their DSL model representation were not enough, because as devices get shared by different requirements the task of tracing requirements becomes complex. Therefore a trace model was presented which would be capable of generating reports of the relationships between a given requirement and its DSL functional units and links.

Further on a procedure for using both catalogues (Devices and Requirements) was presented so that the user can have an easy going use of them. Finally a set of applications based on several requirements together with the DSL templates had been presented, for which the trace reports can be found on Appendix A.

Still there is a long way to go for improvement, for example up to now traces for requirements are being done manually. It would be an interesting project to automate the creation of those traces. Another improvement that is necessary is regarding to the requirement DSL templates. By now in order to reuse requirements a set of parameters have to be manually put into each requirement. It would be quite useful to develop a tool which would allow the use of requirements at a click of a button and to import new requirements as the catalogue keeps getting enriched by the user.
REFERENCES


