FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

QuiiQ Automation Foundation

Rui Manuel Andrade Barreira

Dissertation subtimed to partially fulfill the requirements to obtain the Master Degree in Informatics Engineering

Supervisor: Ana Paiva (PhD) Assistant Professor at the Faculdade de Engenharia da Universidade do Porto

October 26, 2009

"Insist on yourself; Never imitate.
Your own gift you can present every moment with
the cumulative force of a whole life's cultivation;
But of the adopted talent of another you have only an extemporaneous half possession...

Ralph Waldo Emerson

Abstract

The emergence of the concept of a digital home has led to the proliferation of gadgets and home automation protocols, separately developed by a number of different companies and entities. As a consequence, these protocols and devices each have their own protocols and control interfaces, with no thought given to interoperability - paving the way for a unifying application to enter the market.

At the moment, QUiiQ AUTOMATION is the only application on the market to implement higher level concepts which permit it to communicate with almost any device and many of the applications available for the purpose of home automation. As the need for more intelligence in controlling these devices arises, we propose the "QUiiQ Automation Framework" as a possible solution.

In the kernel of the framework, are modeled the base concepts (like divisions and devices) and higher level and extensible concepts (like routines, environments, etc.). The abstract representation of a device, is based on its basic information (name, image, etc.), its spatial location and its functional behavior. This level of abstraction allows the representation of almost all devices existing in home automation systems, bridging the intercommunication/control gap between the several home automation protocols.

The materialization of the base concepts of division (sets of components based on the real house organization) and device (spatial identification, physical identification and functional identification), allows the modeling of higher level abstract concepts (environments, routines, operation modes and alerts/alarms) which can be extended (all the concepts are modeled into the kernel of the "QuiiQ Automation Framework") and the abstraction provided by the device concept allows the appearance of new concepts and features.

Context-awareness is a recent concept, which allows a software application to be aware of its execution environment and to react accordingly. After analyzing different frameworks, the Context Toolkit was identified as being the most adequate for integration with the proposed system (this integration is enabled by the concept of device). It is presented a proposal which will overcome the difficulties existing in multiple protocols, wide variety of communication mechanisms and different control interfaces, while providing a superior level of "intelligence" to the system, allowing more flexibility in the development of home automation software applications.

The work presented in this dissertation consists of a new vision for home automation software, and it pursuits the minimization of time and cost for developing home automation

applications.

Resumo

O aparecimento do conceito de casa digital levou à proliferação de diversos *gadgets* e ao aparecimento de diversos protocolos de domótica no mercado, sendo estes o ponto de entrada da automação nas habitações. Estes protocolos são completamente independentes, apresentam diferentes formas de controlo e diferentes interfaces de comunicação. Devido a esta invasão de novas tecnologias nos mercados, existem diversas aplicações (são não só incompletas, como também por vezes inadequadas) de visualização e controlo de redes de domótica apresentando cada uma as suas vantagens e desvantagens.

Presentemente, existe apenas no mercado uma soluçao de software (QUiiQ AUTOMATION) que implementa conceitos de alto nível, permitindo assim, a abstracção e controlo de diversos protocolos de forma completamente transparente. Como resposta a este problema de proliferação de tecnologia e em busca de novas formas mais "inteligentes" de controlo, surge a proposta "QuiiQ Automation Foundation" (materializada nesta dissertação), que constitui um núcleo para uma futura *framework* de automação e controlo de habitações: "QuiiQ Automation Framework".

No núcleo da *framework*, são modelados conceitos base (como dispositivo e divisão) e conceitos de mais alto nível extensiveis (como ambientes, rotinas, etc). A representação abstracta de um dispositivo, passa pela representação da sua informação mais básica (nome, imagem, etc.), da sua localização espacial e do seu comportamento funcional. Este nível de abstracção permite representar quase a totalidade de dispositivos existentes nos sistemas de domótica, colmatando assim, a falha de intercomunicação/controlo dos diversos protocolos.

A materializaçao dos conceitos base de divisão (são conjuntos de componentes baseados nas divisões reais da casa) e dispositivo (identificação espacial, física e funcional), permite a modelação de conceitos abstractos de alto nível (ambientes, rotinas, modos de operação e alertas/alarmes), extensíveis (tanto os conceitos base como os de alto nível são modelados como sendo o núcleo da "QuiiQ Automation Framework"). A abstração dada pelo conceito de dispositivo, permite também o aparecimento de novos conceitos e funcionalidades.

Context-awareness é um conceito recente que permite às aplicações conhecerem o seu ambiente, e reagir em consistencia. Após a análise de diversas *frameworks*, o *Context Toolkit* é identificado como sendo o mais adequado para a integração (esta integração é realizada através do conceito de dispositivo) com o sistema proposto. Apresenta-se uma proposta que resolve as dificuldades inerentes ao uso de multiprotocolos, de diferentes formas de comunicação e interfaces, dando também um nível superior de "inteligencia" ao sistema, permitindo uma maior flexibilidade no desenvolvimento de aplicações de controlo de sistemas

de domótica.

O trabalho apresentado nesta dissertação constitui uma nova visão sobre a domótica que procura minimizar o tempo e os custos de desenvolvimento de aplicações neste dominio.

Resumé

L'émergence du concept de maison domotique a conduit à la prolifération de gadgets et de domotique procédes, spécialement mis au point par un certain nombre dé entreprises et entités. Ces procédés sont completement indépendants, ils presentment different forms de contrôle et different faces de communication. Suite á l'invasion de ces nouvelles technologies sur nos marches, il existe diverses applications (qui sont parfois incompletes et innadaptées) de visualization et de controle de réscaux de domotique qui présentent chacune leurs avantages et leur inconvénients.

Actuellement il existe sur le marche une soluion de software (QUiiQ AUTOMATION) qui permet de mettre en oeuvre des concepts de haut niveau, permettant ainsi l'abstraction et le contrôle de différents procédés de manière complétement transparante. Comme le besoin de plus d'intelligence dans le contrôle de ces dispositifs se fait sentir, nous vous proposons le "QUiiQ Automation Foundation" comme une solution possible.

Dans le noyau du *framework*, son modélisés les concepts de base (comme les division et dispositif), et le niveau et plus élevé concepts (comme les habitudes, les environnements,etc.). La représentation abstraite d'un dispositif est fondée sur sa base d'informations (nom, image, etc.), sa localisation spatiale et ses comportements. Ce niveau déabstraction permet la représentation de presque tous les dispositifs existant dans les systémes de domotique, de combler le manque d'intercommunication / de contrôle entre les différents procédés de la domotique.

La matérialisation des concepts de base division (sont des ensembles de composants à base de réels des division dans la maison) et de dispositif (identification spatiale, l'identification physique et fonctionnelle), permet la modélisation de concepts abstraits de haut niveau (environnement, habitudes, modes opératoires et alertes / alarmes) qui peuvent évoluer (tous ces concepts représentent le noyau de la "QuiiQ Automation Framework"). L'abstraction fournie par l'appareil concept permet l'apparition de nouveaux concepts et de fonctionnalités.

Le *Context-awareness* est un concept récent qui permet àun logiciel d'application d'être conscient de son environnement d'exécution, et de réagir en conséquence. Aprés analyse de différents *frameworks*, le *Context Toolkit* a été identifié comme étant le plus adéquat pour l'intégration avec les systêmes proposés (cette intégration est rendue possible par le concept de l'appareil).

Nous présentons une solution qui permettra de surmonter les difficultés de plusieurs procédés, la grande variété de mécanismes de communication et les différentes interfaces

de contrôle, tandis que fournir un niveau supérieur de l' "intelligence" du systéme, permettant plus de flexibilité dans le développement de la maison logiciel d'automatisation des applications.

Le travail présenté dans cette dissertation est une nouvelle vision pour la domotique logiciel, et il convient de minimiser le temps et le coût de développement d'applications dans ce domaine.

Preface

This dissertation concludes a "work in progress" that lasts for the past three years - my master degree; finally! I began my studies in the field of Computer Science in 2000, and I was fortunate enough to get a very nice job that introduced me and allowed me to start studying the field of home automation. Home automation is a relatively new field of study and, as such, it as several approaches to address the same problems. Two of the major problems found in the home automation world are the diversity of protocols (it is difficult to develop software applications to support multiple protocols), and the lack of user-friendly control points. In this research work, we address these problems by abstracting the home automation network into concepts which allows the scalability of the system and are well known to the final user, but we went a little bit further. The models defined on this research work were designed to be used as a framework which allows others to use it as a basis to develop new concepts and applications. The concept of context-awareness is introduced to give a little bit more intelligence to the system. Basically, we present the kernel for a future framework that, hopefully, will change the way people develop home automation software and, consequently, the way people interact with there own houses and the way houses interact with the human being.

The study and dedication implicit in the writing of this research work, combined with my general life (I have been studying and working as an engineer at the same time), naturally slowed things down, but never to a complete halt, often because of the friendly reminders from my family and friends. All of these friendly reminders my friends tireless support, allowed me to go through all the different stages of my life with success. To all of them, I only have one expression: a big smile and a sincere "Thank You!". To my parents (Teresa e Silvano) I say Thank You!, for all the love and support from my first steps into this world to this particular day. You made my life easier and you were there to support me all the way along. To my aunts (Isabel, Ni and Fernanda) I say Thank You!, with all your caring, support and sermons you made this degree possible. To my brother (Luis) I say Thank You!, for all the endless talks about engineering and for hearing me when I most needed. To my special one (Aurelie) I say Thank You!, for never letting me quit from my goals, never letting me quit from finishing this dissertation and for putting up with me in the most stressful times. To my good friend Pepe I say Thank You!, for its endless patience in reviewing this dissertation and rewriting some of its english. Finally to my friends (Sara, Raven, Quinhentos, Sé, Norte, G, and all the others) I say Thank You!, for being there in the lazy times and in the hard times.

This work will present the culmination of these three years of hard work. It is from the products developed in this company, QUiiQ, from which all of this work is based. So, this work would not be possible if it wasn't QUiiQ. They were the ones which presented me the field of home automation and gave me all the necessary tools to learn a lot about home au-

tomation. So, to my directors (Luis and Rui) I say Thank You!, beside all the work we have to do every day you always found the time to talk a little bit about my dissertation, you gave me very good ideas and the necessary motivation to take this work further. To my colleagues (Eduardo and Gonçalo), I say Thank You!, for all the endless conversations about software architecture, home automation, user experience and a lot of other subjects which revealed very, very useful to successfully finish this work.

All the knowledge I acquired during my studies in the master degree at FEUP and all the institutional support it gave me to successfully fulfill this work. So, I say a special Thank You! to my adviser Ana Paiva, for never letting me quit, for the endless meetings and all the technical and scientific support she gave me during this period. Whithout you, this would not be possible. To FEUP, I say Thank You!, for providing all the structural conditions to finish this work. To a lot of professors (Ademar Aguiar, Falcão e Cunha, Pascoal Faria, Correia Lopes and a others) I say Thank You!, for teaching and advising me in a variety of fields, which reveled very useful during this period.

Finally, I would like to say Thank You! to everyone I forgot to mention, but in some way helped me to grow professionally and as a person...

To all of you, a big Thank You!

Rui Barreira

Contents

1 Introduction **1**
1.1 Context and Motivation . . . 1
1.2 The Power of Frameworks . . . 3
1.3 Model Driven Architecture . . . 4
1.3.1 Unified Modeling Language . . . 5
1.4 Research Goals . . . 6
1.5 Research Strategy . . . 9
1.5.1 Domain analysis . . . 9
1.5.2 Design . . . 10
1.5.3 Model Validation . . . 11
1.6 Research Work Contributions . . . 13
1.7 How to Read this Dissertation . . . 14

2 Home Automation **17**
2.1 What is Home Automation? . . . 17
2.1.1 Definition . . . 17
2.1.2 Connections & Protocols . . . 18
2.1.3 Brief Literature Review . . . 21
2.2 Products in the Market . . . 23
2.2.1 QUiiQ AUTOMATION . . . 23
2.2.2 StarDraw . . . 24
2.2.3 MControl . . . 25
2.3 Home Automation Software Main Requirements . . . 26
2.4 The Infrastructure - QuiiQ Automation Framework . . . 27

3 Frameworks **29**
3.1 What is a Framework? . . . 29
3.2 Object Oriented Frameworks and Software Reuse . . . 29
3.3 Fundamental Concepts . . . 31
3.4 Frameworks Development Methodologies . . . 33
3.5 Frameworks Success Factors . . . 35

4 From the Physical World to Smart Living Environments **37**
4.1 Software Variability . . . 37
4.2 Empirical Concepts . . . 38
4.2.1 Physical Identification . . . 40
4.2.2 Functional Identification . . . 41

4.2.3 Space Identification 42

5 Context Awareness as a way to Ambient Intelligence 43
5.1 What is Context? 43
5.2 What is Context-Awareness? 45
5.3 Application domains for context-aware applications 46
5.4 Design principles for context-aware systems 47
5.4.1 Architecture 47
5.4.2 Sensors Infrastructure 49
5.4.3 Context Modeling 50
5.4.4 Reasoning Methodologies 51
5.4.5 Historical Data 52
5.5 Survey of Context-Aware Frameworks 52
5.5.1 Context-Awareness Sub-Structure (CASS) 53
5.5.2 Context Broker Architecture (CoBrA) 54
5.5.3 Context Management Framework 55
5.5.4 Context Toolkit 56
5.5.5 CORTEX 57
5.5.6 Gaia 58
5.5.7 Hydrogen 59
5.5.8 Service-Oriented Context-Aware Middleware (SOCAM) 60
5.5.9 Discussion 62
5.6 Context Awareness in Home Automation 64

6 QuiiQ Automation Foundation 67
6.1 Development Methodology 68
6.2 System Overview 68
6.3 Home Automation Network Representation 70
6.3.1 Division 70
6.3.2 Device 71
6.4 Ambient Intelligence 77
6.4.1 Environments 77
6.4.2 Routines 79
6.4.3 Alerts/Alarms 83
6.4.4 Adding Context Awareness - The Context Toolkit 84
6.4.5 QuiiQ Automation Foundation & Context Toolkit 92

7 Conclusions and Future Work 99
7.1 Research Summary 99
7.2 Future Research Directions 101

References 103

List of Figures

1.1 Model Driven Architecture Methodology 5
1.2 Activities of framework development process. 9

2.1 QuiiQ AUTOMATION Architecture. 24
2.2 QuiiQ Automation Framework 28

3.1 Template Method 32

4.1 Characterization and connections from the physical to the virtual world. . . . 39

5.1 Context-Awareness Sub-Structure (CASS) Architecture. 54
5.2 Context Broker Architecture (CoBrA) Architecture. 55
5.3 Context Management Framework Architecture. 55
5.4 Context Toolkit sample configuration. 57
5.5 Sentient Object Model. 58
5.6 Gaia Architecture. 59
5.7 Hydrogen Architecture. 60
5.8 Service-Oriented Context-Aware Middleware (SOCAM) Architecture. 61
5.9 Basic components of a context-aware system interacting with a user. 64

6.1 QuiiQ Automation Foundation Main System as a UML class diagram. 69
6.2 QuiiQ Automation Foundation division class model. 71
6.3 QuiiQ Automation Foundation devices Abstract Type class model. 73
6.4 QuiiQ Automation Foundation devices Behaviors class model. 75
6.5 QuiiQ Automation Foundation devices class model. 76
6.6 QuiiQ Automation Foundation environment class model. 78
6.7 QuiiQ Automation Foundation environment activity model. 79
6.8 QuiiQ Automation Foundation routines class model. 80
6.9 QuiiQ Automation Foundation routines activity model. 82
6.10 QuiiQ Automation Foundation Alarms and Alerts Class model. 85
6.11 QuiiQ Automation Foundation Alarms and Alerts Arm Sequence activity model. 86
6.12 Direct sensor access - Traditional design without a discover. 88
6.13 Middleware approach - Application design using discoverer. 88
6.14 Enactor approach - Application design using Enactor. 89
6.15 Sample Configuration of context components. Arrows indicate data flow. . . 89
6.16 Sample Architecture for a QuiiQ Automation Foundation with Context Awareness. 94
6.17 QuiiQ Automation Foundation devices class model. 95

6.18 QuiiQ Automation Foundation devices class model. 98

List of Tables

5.1 Context-Aware application categories depending on Schilit *et al.*. 45
5.2 Context-Awareness Frameworks Comparison. 63

6.1 Layered Architecture approach. 93

Abbreviations

ADT	Abstract Data Type
CASS	Context-Awareness Sub-Structure
CC/PP	Composite Capabilities/Preference Profile
CoBra	Context Broker Architecture
COBRA	Common Object Request Broker Architecture
HVAC	Heat, Ventilation and Air Conditioning
HTTP	Hypertext Transfer Protocol
MDA	Model-Driven Architectures
OMG	Object Management Group
ORM	Object Role Modeling
OWL	Web Ontology Language
RDF	Resource Description Framework
RMI	Remote Method Invocation
SOA	Service Oriented Architecture
SOAP	Simple Object Protocol
SQL	Structured Query Language
UAProf	User Agent Profile
UML	Unified Modelling Language
XML	eXtensible Markup Language

Chapter 1

Introduction

1.1 Context and Motivation

The digital era is here, everyone knows it and everyone wants to benefit from it. This is noticed in several technological areas, from industrial automation, to office work and home automation. Home automation is a recent technology that allows the integrated management of all habitational resources. Home automation is a concept that integrates the notion of house, informatics and telecommunications. The conjunction of the last two elements gives the system all its meaning, simplifying the user's daily life and satisfying its needs for communication, comfort and security. When the first buildings with automation started appearing in our society (80's), the system was intended to control the lighting, climatization, security and the interoperability of these three elements.

In our days, the basic idea is exactly the same, but the context in which the system is thought of a bit different: it is not the military and industrial context, but people homes. Though still little known and disseminated, but because of the comfort it promotes, home automation has already several enthusiasts. Home automation uses several elements, usually independent ones, in a systematic way, represented by a set of devices and subsystems (air conditioning, lighting security, etc), which are connected by a communication network of some kind. Several elements of the system share common resources to accomplish specific tasks according to the user's requirements [1].

This type of system connects the resources of electronics and computer systems, in a way that allows the system to obtain an integrated management of the several equipments in the house. Home automation allows the routines and boring tasks to be carried out automatically. There are several options for the installation of home automation systems proliferating in the international markets, but each one has its own characteristics, works with a limited number

of protocols and has no "intelligence" built into it.

This research work proposes several solutions for the home automation problem and the home automation software development problem. Considering several theoretical concepts, this research work presents an initial type of intelligence for the system based in concepts inherited from our daily life and the interaction with our home. These concepts give some value added to home automation products, because they can emulate, in a limited way, the way people think, giving them very simple ways of control, for example the concept of routine gives the system the ability to automatically perform an action, but this action should be configured by the user, as well as the trigger that starts the action.

With the introduction of context-aware frameworks, presented in Chapter 5, the integrated intelligence on a home automation system is significantly increased, being the system aware of the state of its surroundings, it is possible the infer new higher level context, and take intelligent actions based on this information.

The problem of software development productivity in home automation applications allied with this new point of view about home automation, creates the perfect scenario for implementing a framework. With the increase of the home automation user base, more and more companies will implement software products to visualize and control these systems, with lots of concepts and design rewritten over and over again for each particular product. The framework will increase the productivity, give more functionality to the system, and allow concepts to be instantiated as needed to each particular system.

So far, home automation as been approached predominantly, as a technological platform (automation and control) and not as new way of inter action, a new way of living, without thinking about the user. The house was conceived of pretty much, as using the cybernetic analogy, a machine whose goal was controlling it by way of various automatic devices.

The main focus of this research work is not on technological problems or options, but rather the technological convergence between control and automation technologies, on one side, and information and communication technologies, on the other. These two worlds, in conjunction, contribute both for the enrichment of the technological infrastructure of the home and to the benefits of its dwellers, which should be the main purpose of technological endeavor.

The state of the art, in present day technology, provides easy avenues for adequately putting together electricity, electronics, computer and telecommunications in a single technological mix [2]. With all this power and the analysis of how people behave and interact, we

can define theoretical concepts that will exchange the existing systems to all new meaning, giving people what they really need in a fashionable and easy way.

QuiiQ Automation Foundation aims to do this but in a different stage because it will promote a development platform to build a framework on top of it, that will allow the programmers to develop applications (very low time to market) using all the concepts inherited from the QuiiQ Automation Foundation platform.

The ultimate task to be done on top of QuiiQ Automation Foundation is the design of a full-featured framework, promoting graphical interface design, functionality implementation, ambient intelligence and several other concepts will emerge from several research projects.

1.2 The Power of Frameworks

Software productivity has increased significantly, but the relationship between supply and demand is still very uneven. The low productivity of the software development teams is due to the number of code lines that needs to be written to create certain functionality. So, instead of looking for ways of writing code faster, we should be looking for ways of writing less code [3].

In the recent years, several research projects in the area of software engineering have been carried out, but most of them have concluded that the only realistic approach to write less code would be the reuse of software. Another approach could be automatic programming, whereby a system would be capable of generating executable code based on informal, incomplete and incoherent user requirements, which is still very far from reality [4]. In a very simple way, software reuse is the activity of using code, design or knowledge from other systems in the development of new software products. In the process of software reuse, we could apply several techniques that use software artifacts in different scales and complexity, ranging from source code to highly abstract architectures. We can set object-oriented frameworks between these two realities, being a powerful technique of software reuse in large scale [4].

A framework can be defined as the skeleton of an application that can be customized by the application programmer. It is an abstract design that can be reused in a specific application domain. In a framework, the set of common aspects is represented by a set of abstract classes that collaborate inside the same structure, being the behaviors implemented by a set of template methods inside a base class. The variable aspects of the system, called hot spots, are represented by a set of hook methods that are implemented as concrete methods in the final system[5].

Through software reuse and all the information previously presented, a framework helps a programmer to be more productive, increase the product quality and decrease the time to market. When combined with software patterns and components, frameworks are considered the best technological solution, supporting large-scale reuse. However, there are some problems regarding the use of frameworks that can be divided in two groups: development problems and using problems. The *development problems* range from the specific application domain of the frameworks to the lack of appropriate development methods, framework testing, and documentation and the maintenance and evolution strategies. Regarding *frameworks use*, the capacity of framework evaluation, the lack of framework use processes in the applications development and the ease of learning and understanding should be referenced. Before using a framework, it is necessary to understand it and understand how it should be used in developing the system in question. Grady Booch stated that "the most profoundly elegant framework will never be reused unless the cost of understanding it and then using its abstractions is lower than the programmer's perceived cost of writing them from scratch" [6].

The mitigation of the learning curve required for the adoption of a framework is thus essential to its adoption. In the particular case of this research work, the application domain is very specific and strongly connected to the world of home automation. Given the complexity of such systems, their size, diversity of protocols and possible applicable concepts, makes the development of applications slow and specific to each protocol, so there is a strong need to reuse software, minimizing the effort of development. The creation of a framework that contains all that is needed for the development of an application in this field of knowledge is therefore essential and very useful for entities that can use it.

1.3 Model Driven Architecture

The Model Driven Architecture (MDA) approach to software development is a new way of writing software specifications, based on a platform independent model. The MDA approach focuses on functionality and behavior of distributed application or systems, not on the technology in which it will be implemented. Using MDA, it is not necessary to rewrite the model each time a new technology comes along, being the system modeled once and only once. In picture 1.1 it is represented graphically this methodology being the Unified Modeling Language (UML) one of the main technologies used in the design process, being the key technology for the MDA process. Every application or system developed using the MDA approach is based on a normative, platform-independent UML model. The methodology used in the development of this research work is MDA and the main technology used is UML.

The process of designing a model represents highly creative work. The software engineer,

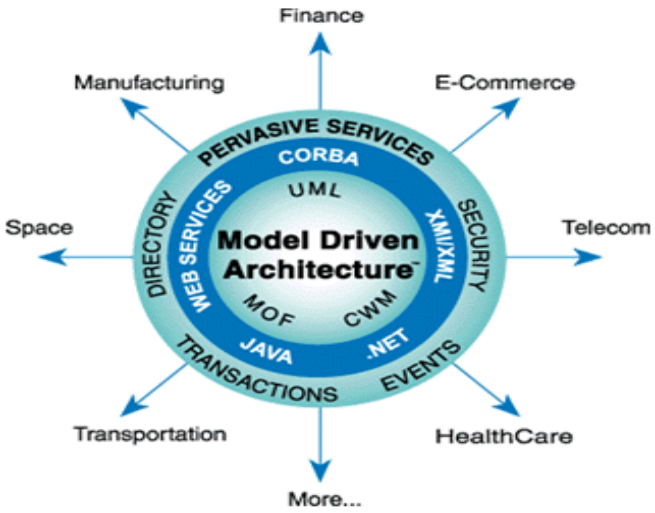

Figure 1.1: Model Driven Architecture Methodology

through a process of interactive design, ensures that the model achieves the roles and the requirements defined for the project in question. It evolves during the project integrating, the designer's experiences and the new insight identified for the project. Communicating a model in a clear way to the many people involved in a software project, represents the core feature of UML. The model represents, in an easy and understandable way, not only the detailed specifications of the software artifacts, but also high-level analysis, providing methods to connect the different levels of abstraction.

In a vary short definition, we can reduce usable models to [7]:

1. **Accurate.** They should precisely and correctly describe the system they represent;
2. **Understandable.** They must be as simple as possible, but not too simple to accurately fulfill their purpose, and must be easy to communicate;
3. **Consistent.** Different views must not express things that are in conflict with each other;
4. **Modifiable.** They must be easy to update.

1.3.1 Unified Modeling Language

In short, the Unified Modeling Language (UML) is the industry standard mechanism for visualizing, specifying, constructing and documenting software systems. In a more broad definition, UML provides a tool to describe system interactions, supported by a standard set of definitions managed by the Object Management Group (OMG). With the OMG support, UML now has a well-documented, formal specification of the language.

At a higher level, the details of UML are as follows [7]:

- All UML diagrams describe object-oriented information;
- Class and Object diagrams illustrate a system's static structure and the relationship between different objects;
- Interaction diagrams, State Machines and Activity diagrams, show the dynamic behavior of objects as well as messages between objects;
- Use-case and Activity diagrams show system requirements and process workflows;
- The Composite Structure diagram shows the collaborating features as they combine at run time to implement a specific UML element;
- Deployment diagrams help with the deployment of the software in a specific environment;

All of the above mentioned features of UML allow business annalists and software architects to collaborate in the product's design and specify, construct and document applications in standard way. The designers can ensure that all the requirements are captured, business problems addressed, and that the solutions are workable and practical.

1.4 Research Goals

Much of the work done in the area of home automation, aims to find solutions to a very restricted set of problems. It is possible to find solution proposals for each problem, but a solution for the global problem and development support cannot be found in the literature. Although there are some applications in the market and solutions published by the scientific community for the development of home automation applications, they present some flaws (they are protocol specific, they only address representation, do not present a general way of implementing home automation systems, etc. none of them addresses the problems existing in home automation, providing a way of abstractly representing the system, allowing the introduction of multi protocol, intelligent, and highly scalable systems) each one at their level ([8],[9],[10], [11], [12], [13]).

There are several theoretical concepts that can be implemented on a home automation system that can only be achieved by the introduction of computing in the house. These concepts emulate the way people think and interact with the house, giving the system an all-new way of interaction and meaning. The following points present a pre-research analysis of some of these concepts that are going to be object of research, presenting each one with a simple

description.

- **Devices.** This concept represents the abstract way of looking at real devices (i.e., a lamp or air conditioning equipment). It emulates all the complexity and differences of this type of hardware, grouping them by abstract type, and presenting the commands they execute, independently of the protocol they use (for example KNX or Domintell).
  - *On/Off*. On/Off devices are all the devices that only have two types of states, on and off. The actions related to this type of devices are exactly a Boolean behavior;
  - *Dimmer*. A Dimmer is slightly more complex than an On/Off because it allows several states between 0% and 100%. The behavior associated is a dimming behavior that receives as parameter the value of the state to be sent to the device;
  - *Blind*. Typically a Blind only allows an On/Off behavior and a Stop behavior. Actually, it starts to appear in the market blind controllers with dimming behaviors to, so this abstract type should have associated an On/Off, dimming, and stop behavior;
  - *Air Conditioning*. An Air Conditioning device allows to control the objective temperature, the actuation mode, the fan speed and the position of its lovers;
  - *Central Heating*. A Central Heating system only allows to control the required temperature;
  - *Sensors*. Sensors are special type of devices because they do not have any type of behavior, they simply read information and report it to the system, so they do not have a behavior, but they do have a state;
  - *Multi Room Audio*. A Multi Room Audio system allows playing different types of music in different house divisions. It is possible to control the inputs and the outputs of the system, for example, one wants to set the input to be the iPod and the output to be the radio in the bedroom;
  - *Cameras*. Cameras allows to visualize what is happening in its surveillance area in real time, but they may have some behaviors associated like movement behaviors, zooming and recording video;
  - *Video doorbell*. A video doorman, in concept, is the integration of a camera, with an On/Off device and a double channel audio system.
- **Actions.** An Action describes the behavior of a device in the system, for example, starting the air conditioning system. In the computational world, this concept can be used to group device actions;
- **Triggers.** A Trigger is a condition over the state of the system. The system reacts whenever that condition is true;

- **Environments.** An Environment is a pre-programmed sequence of actions, mapped to one button. When this button is pressed, the actions are sequentially executed.
- **Routines.** A routine is pretty much the same as an environment, but the actions are executed alongside and they start automatically, i.e., without the interaction of a human being. The concept of trigger is used here to provide the conditional formula that starts all actions when evaluated to true. Routines can be grouped in usual sets:
  - **Daily Routines.** A routine that is executed every day, for example, every day at eight all the blinds are opened and a music starts;
  - **Special Routines.** Special routines are more complex routines that only are executed in special dates, for example, the presence simulation routine;
  - **Alarms.** Alarms are system routines that are usual triggered by a sensor, if the fire sensor signals, a call is automatically made to the protection services.

After analyzing these base concepts, it is very interesting to verify the way they can be composed, creating a all-new group of functionalities, opening the system to the integration of functional/behavioral "intelligence".

Some of these concepts are already implemented in a product called QUiiQ AUTOMATION (the design of this product will serve as basis for the development of this research work), but this is only a product. The main objective of this research work is to study the way these concepts integrate and interact among them and design the foundations of a framework (basis for the development of a future framework that will integrate other areas of knowledge, like the way these concepts are presented to the user).This framework, will allow product developers to instantiate only the concepts that really interests them for a particular implementation, promoting a all new way of development for home automation applications in a way that it is possible to develop a system for a particular client requirements in a minimum time frame.

The goal of this research work is to answer to the following three simple research questions:

1. what are the main theoretical concepts directly present in home automation systems?
2. how can we create new concepts and compose the existing ones in order to give added value to the system?
3. how can we give some behavioral/functional "intelligence" to the system?

The answer to these questions is the main focus of this research work, being essential to define all the properties of this new way of looking at home automation and home interactions, allowing the design of a future system that will be the basis for the analysis and design

of the QuiiQ Automation Framework (introduced in section 2.4).

This research work will not focus on the complete framework, but on the model of an embrionary framework (the home automation kernel for the future QuiiQ Automation Framework) to serve as a basis for the continuity of the main framework design tasks.

1.5 Research Strategy

The development of a framework comprises the activities of domain analysis, framework design, model validation, framework documentation and framework evolution.

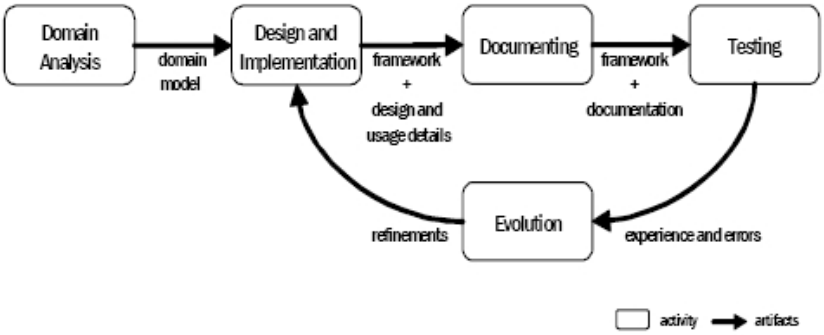

Figure 1.2: Activities of framework development process [4] .

This research work does not aim for the development of a complete framework, but the definition of conceptual models for the foundations of the future framework. So we only focused in the domain analysis, framework design and mode validation activities. The following sub sections, explain the typical steps on this process (figure 1.2).

1.5.1 Domain analysis

Domain analyses process all the information available about existing software in a particular domain, and extracts reusable assets. Systematic reuse requires an understanding of previous work, in particular the main problems of the domain, and the different way that each applications deals with it, identifying the components that can be used in a different context. In this view, domain analysis is a fundamental activity integrated in a software process, with a view on reuse.

During this phase all the information available about the target domain (user-visible aspects of a system in an application domain) is collected and organized; a domain terminology is created, and all the definitions, concepts and abbreviations are written and their meaning explained. This information will create limited boundaries to our domain characterization.

The functional requirements will be identified, and a structural description of the functionalities that formalize the requirements will be written [14].

The first stage of this domain analysis will be done through a an analyses of the source code of the QUiiQ AUTOMATION product, and the functionality analysis of several other products in the market. Beside this more technical analysis, a concept analysis will be made in order to identify theoretical concepts that should be present in the future framework. From this research work, a list of features and concepts will emerge, which will be the base for the next step of interaction, the design phase.

1.5.2 Design

The design phase will be the most time consuming activity of this project. The main goal of this phase is to define the design of a flexible framework, that explicitly divides the set of stable interfaces and behaviors from the variable parts intended to be customized by the final application developer's [4].

The first step is to design a preliminary version of the framework, by analyzing the commonalities and variants encountered in the domain analyses phase, presenting the key abstractions that will enable application programmers to implement valid solutions for their particular problems. Hot spots are the main concern on the framework design process, because they are the ones that will give the framework its flexibility and extendibility, which allied with the ease of use will promoted the software development, maximizing the performance and quality, and minimizing the time to market of the future home automation applications.

The first step to find these hot spots is to generalize by finding which objects with different names that are the same, parameterize and eliminating differences, breaking larger objects so that smaller components can be found and categorize things that are different (already started in the domain analyses phase).

After these analyses, the customization capability (hot spots) of a framework can be found by using abstract classes, polymorphism and dynamic binding. We can achieve this abstraction by creating super classes that give the operations common interfaces, moving operations within the same implementation to the super class and making the operations with different implementations abstract.

Each hot spot is implemented by a hot-spot subsystem that contains base classes, concrete derived classes and possibly additional classes and relationships. This subsystem will introduce the desired variability into the system either by inheritance (white-box framework) or by composition (black-box framework).

As the framework takes shape, it becomes more clear which domain aspects require variability and flexibility, from the others that represent fixed aspects of the domain. After introducing the variability required, the framework design should be continually refined to try to make it simpler, smaller and easier to use by clients [4]. A constant refinement will be made through several interactions optimizing the way hot spots are implemented, because it is crucial for the quality of the framework and its success.

Regarding the methodology selection, this will be made based on the following methodologies (see section 3.4 for a deeper description):

- Design through systematic generalization;
- Design Based on design patterns;
- Design based on application experiences;
- Design process based on viewpoints;

Regarding the way this research work will be done and presented, it will be based on MDA and UML. A model driven approach to software development using UML has several advantages because of the higher level of abstraction, completely language and platform independent and provides a convenient part of the documentation for the future system.

Models allow for easy communication of the system architecture and design, and promote the testing and validation in an initial phase of the projects, while it is still cheap to detect and correct software errors [15]. This model will allow us to get an abstract view of the system design, being represented by a set of UML models and, probably, represented by a more formal specification language (VDM++, Spec#,Z notation, etc.). Finally, after the model is designed, a validation of it is required to confirm if it is consistent and if it follows all the requirements defined in the domain analysis.

1.5.3 Model Validation

In software engineering, the validation of one's work can be approached in many different ways. It could be validated for being low cost, reliable, rapidly developed, safe or some other attribute that is important to a particular client. Claiming that a technique, product, analysis

or design is effective, contains no real information.

Software engineering research community do not have a very widely acceptance in what is the best way to validate a software engineering work, still being a matter of research and debate. Zelkowitz and Wallace references several research strategies that are quoted below [16]:

- **Scientific method.**. "Scientists develop a theory to explain a phenomenon; they propose a hypothesis and then test alternative variations of the hypothesis. As they do so, they collect data to verify or refute the claims of the hypothesis".
- **Engineering method.**. "Engineers develop and test a solution to a hypothesis. Based upon the results of the test, they improve the solution until it requires no further improvement".
- **Empirical method.**. "A statistical method is proposed as a means to validate a given hypothesis. Unlike the scientific methods, there may not be a formal model or theory describing the hypothesis. Data is collected to verify the hypothesis".
- **Analytical method.**. "A formal theory is developed, and results derived from a theory can be compared with empirical observations".

All of these approaches are applied to science in general, and are based on the collection of data. When we develop an experiment we are expecting to collect enough data from a sufficient number of subjects (team, programmer or source program module), all of them equally weighted in order to obtain a significant and valuable amount of data.

In their paper "Experimental Models for Validating Technology", Zelkowitz and Wallace propose twelve different types of experimental approaches to software engineering, grouped in three categories:

1. **Observational methods**. "An observational method generally collects relevant data as a project develops. There is relatively little control over the development process other than through using the new technology that is being studied." There is four types observational methods: *Project Monitoring, Case Study, Assertion and Field Study.*
2. **Historical methods**. "A historical method collects data from projects that have already been completed. The data already exists; it is only necessary to analyze what has already been collected." There are four types of historical methods: *Literature Search, Legacy, Lessons Learned and Static Analysis.*
3. **Controlled methods**. "A controlled method provides for multiple instances of an observation for statistical validity of the results. This method is the classical method of

experimental design in other scientific disciplines". There are four types of controlled methods: *Replicated, Synthetic, Dynamic Analysis and Simulation*

Having these twelve types as guides it is possible to apply four of them in the validation of this research work.

Assertion is a method that is going to be used to validate this research work. It uses ad hoc validating techniques but is useful to do a match between the requirements and research goals and the work done. It gives a basis for the continuity of this research work as well. Literature search allows one to check the state of the art, and to support the hypothesis in pre-existing and accepted technologies or methodologies. There are large available databases of knowledge giving full support to this type of experimental approach.

Due to our wide experience in the area of home automation software development, integration and validation, it is implicit that all the lessons learned will be a valuable tool not only for the development of this research work, but for its validation, matching my previous experiences with the output from this research work. The design of this tool is going to be presented to the development team of QUiiQ AUTOMATION, a team that as been working with home automation software for several years now, and a match between there experiences and the model proposed will be made, increasing this way the validity of the work done.

1.6 Research Work Contributions

The main contribution of this research work is the proposal for the foundations of a new home automation framework that aims to abstract physical house and final user day by day concepts and integrate it with a context-aware framework. This approach defines theoretical concepts of home automation from the physical network abstraction to higher level concepts. These concepts will directly interfere with the final user daily life and with the integration of one of the context-aware frameworks proposed in the research world, it will provides the foundations for a future complete (from communications, to home automation system to graphical user interfaces) home automation framework that derives directly from the QUiiQ AUTOMATION product model, hence it is coined as QuiiQ Automation Foundation.

The resulting system implements all the main abstract concepts for home automation, allowing the development of a multiprotocol system (very useful when a system that integrates sound, air conditioning, home automation, etc. is developed), with ambient intelligence and context-awareness.

In concrete the QuiiQ Automation Foundation, proposes a model for supporting abstract home automation networks, a model to abstract ambient intelligence concepts and a way of integrating these models with a context-aware framework called the Context Toolkit, making it a useful tool for companies that want to develop home automation software while allowing a software company to develop the middleware and the graphical user interfaces (in an abstract level) and a home automation integration company to develop the driver that better suites its protocol.

The definition of the following hierarchy of concepts was achieved and a framework defined using UML:

- Home Automation Network Representation
  - Divisions
  - Devices
  - Multi Device
- Ambient Intelligence
  - Environments
  - Routines
  - Operation Mode
  - Alerts/Alarms
  - Context Awareness

The components on this approach, its integration and the integration with the Context Toolkit are presented in Chapter 6.

1.7 How to Read this Dissertation

This research work involves several areas of knowledge, from home automation, to frameworks, to context-awareness in order to find the better way of creating the foundations for a all new way of interaction with the house and a all new way of developing home automation software. It analyses the world of home automation and abstracts concepts, it analyses the world of frameworks and integrate this concepts into a structure that can be used by others and it analyses the world of context awareness and finds the best option to give the system a little bit more intelligence.

So, this research work is divided in three parts. The first part reviews the most important concepts of home automation, frameworks and framework design (chapters 2 and 3) a

second part that starts addressing the proposed problems by analyzing a way of abstracting the structural concepts in a home automation systems, and analyzing the option of using a context-aware platform to give some "intelligence" to the system (chapters 4 and 5). Finally, the last part defines the models for the home automation theoretical concepts and its hot and frozen spots, presents a way of integrating the context-aware system with the QuiiQ Automation Foundation in the development of a home automation application and proposes some future research work subjects (chapters 6 and 7).

The following list summarizes the structure of this document:

- **Chapter 2.** This chapter gives an overview of the world of home automation and home automation software. It defines the concept of home automation presenting its definition and some existing wireless and wired protocols. It presents a summarized review of some of the most interesting literature about home automation in the research community, it presents some of the products in the market, it defines some of the home automation main requirements and finally it introduces the future QuiiQ Automation Framework, from which the QuiiQ Automation foundations will be the center part;

- **Chapter 3.** This chapter gives a wide view over the subject frameworks. It defines the concept of framework and presents the concept of object oriented frameworks and software reuse. The fundamental concepts for frameworks development are introduced in this chapter as well as the framework development methodologies and success factors;

- **Chapter 4.** This chapter starts addressing the main research subject of this research work by addressing the problem of software variability due to a wide choice of protocols, graphical user interfaces and functionalities. It starts introducing some empirical concepts defined by a *physical identification*, the *functional identification* and *space identification* that allow the abstraction of this variables so the system can be easily reused;

- **Chapter 5.** The previous chapter starts addressing only half of our research problem, it is a necessity to find a way to give some "intelligence" to the system. To achieve this objective the only option is to go through context (the only way for a system to be intelligent is through knowledge). We start by defining context and context-awareness, the usual domain of application of this kind of system is identified, and the context-aware systems main design principles are presented. Finally, a brief analysis of several context-aware systems is presented, and a final discussion about the choice of the system and the application of context-awareness in the home environment is written;

- **Chapter 6.** This is the main chapter of this research work. Based on the empirical concepts defined in chapter 4 the models that will implement these concepts (as division

and device) are presented as well as it hotspots. Then new concepts that are more automatic and in a higher level (environments, routines, alerts/alarms and operation modes) are defined and its models presents. Finally, a deeper analysis of the Context Toolkit is made and its integration with the previously defined models is made as well;

- **Chapter 7.** Finally, in this chapter, a summary of the research work is written as well as some features and guidelines are proposed for future research work.

For a comprehensive understanding of this research work, the chapters should be read by the order they are presented. Those already familiar with home automation and home automation software may jump chapter 2, those familiar with framework concepts and framework development methodologies may jump chapter 3 and finally those familiar with context-aware concepts and context aware frameworks may jump chapter 5.

Chapter 2

Home Automation

2.1 What is Home Automation?

2.1.1 Definition

In the last few years, ubiquitous computing has grown in popularity. The recent advances and integration of computing technologies in everyday life activities provides a fast, accessible and important means of interaction between people and resources. From mobile phones to vehicle GPS navigators, from digital television to domestic appliances, the information technologies are silently pervading our daily routines. Home Automation is an integrated part of the building automation field of study, being focused in the requirements for the automation of private houses, such as comfort, energy efficiency, security and entertainment of its residents [12].

The process of monitoring and controlling interconnected devices within the home environment can be performed either automatically or remotely. In recent years, home automation services have attracted the attention of major technology and service providers [17].

The main scopes of home automation are [18]:

- Heating Ventilation and Air conditioning Systems;
- Boilers/ Hot Water Systems;
- Plumbing systems including various pumps for treated water, sewage water, etc;
- Lighting;
- Intelligent Fire Detection & Alarm Systems;
- Smoke Control;

- Fire Suppression Systems;
- Access Control Systems;
- CCTV Systems;
- Intrusion Detection Systems;
- Perimeter Protection Systems;

2.1.2 Connections & Protocols

Home automation depends completely on communication protocols which were created to control the several electric and electronic components of the system. There are several players in the industry that can be divided in wireless and wired communication protocols, but they are not all compatible with each other. The following list presents the protocols with bigger presence in the market (in my opinion):

2.1.2.1 Wired Protocols

- **X10.** X10 is a home automation communication protocol that is flexible, powerful and the most inexpensive of them all. It is similar to TCP/IP, but it works on top of power lines and with a low bandwidth, sending about one command per second, being this commands as simple as "Device A1: turn on" [19]. This protocol has two huge advantages: it is inexpensive and works on power line, so it is not necessary to insert new cables in the house, but it is very slow and unreliable. X-10 is a proprietary communication protocol that allows devices to talk to each other using the existing electrical wiring in the home and it is developed by the company with the same name [20]. The products are inexpensive and easy to install by simply plugging the device into the electrical outlet. X-10 has existed for over 20 years, and many X-10 compatible products are available today. X-10 is able to address up to 256 unique devices (each device is known for its unique address, for example "A1"), however, if there is too much electrical noise on the power lines generated by some other electrical device, the X-10 devices might have problems communicating. X-10 is widely deployed and easy to install, but it also has some drawbacks. In certain contexts, the X-10 communication is unreliable because the technology is affected by the operation of other electrical appliances nearby, and X-10 signals can get lost. X-10 products are typically installed into electric outlets, which restricts the potential installation locations considerably, making it unsuitable for certain home automation scenarios. X-10 is a proprietary technology, although the company also sells the X-10 products through other companies, under other brand names [21];

- **KNX.** Is the only open standard in the world for applications in home control. Several manufacturers produce hardware on top of this standard, this being a big advantage because there are several modules of control, each with its unique properties. Regarding the standard, it has a single manufacturer, independent design and commissioning tool (ETS); it supports radio frequency, power line, twisted pair and Ethernet/Internet (over TCP/IP) as the communications media; several configuration modes (from system to easy mode) are available; can be coupled with other systems [22]. The big disadvantage of this protocol is that it is not plug and play and needs an expert to configure it. A software API (Falcon) is needed in order to communicate with the network through a computer, with the price being its biggest drawback. The Konnex Association [22] was established in 1999 by joining together three organizations: BatiBUS Club International (BCI); European Installation Bus Association (EIBA); European Home Systems Association (EHSA)... The goal of the organization is to promote a single standard for home and building automation, called KNX. Today, the Konnex Association has approximately 100 member companies. The next version of the standard (version 1.1) will also include support for RF and IR media. The protocol stack is based on the OSI model and specifies the link layer, the network layer, the transport layer and the application layer. Assessment Products based on the KNX protocol from the Konnex Association are not widely deployed in home automation. The KNX technology is mostly used in commercial building automation. However, the Konnex Association has expanded it business activities to the residential market. The Konnex Association has relatively strong industry support, especially in Europe [22];

- **Crestron.** This is a high-end home automation communication protocol, and the one with the widest presence in the world of home automation systems. This protocol is intended to provide easy and complete control of electronic and mechanical devices, being able to control everything through one touch-screen [23]. The communications between this home automation protocol and a computer are quite easy to setup, being only needed that a set of strings, corresponding to commands to be configured, uploaded to the system and the computer only needs to write these strings to the serial port to execute an action. This protocol is highly reliable but one of the most expensive and the configuration efforts are very time consuming;

- **LonWorks.** LonWorks is a networking platform for control systems in building, industry, utility and home automation, introduced by Echelon [24] in 1988. Today, LonWorks is a standard in commercial building automation and industry control. LonWorks has existed for more than 15 years, and it has become widely deployed. More than 1500 companies are developing LonWorks products. LonWorks devices communicate using the LonTalk protocol. Based on the Open Systems Interconnect (OSI) protocol stack

reference model, LonTalk defines 7 protocol layers. The protocol provides a set of services that allow the application program in a device to send and receive messages from other devices over the network without needing to know the topology of the network or the names, addresses, or functions of other devices. Networks can range in size from 2 to 32000 devices. LonTalk can be implemented upon many medium types including power lines, twisted pair, radio frequency (RF), infrared (IR), coaxial cable and fiber optics, although the most common choice is twisted pair cable. Assessment through the years, has proven LonWorks to be a reliable technology. However, the home control systems are typically smaller and simpler networks where important properties are low installation complexity, low battery consumption and low product cost. LonWorks was designed with none of these in mind. LonWorks was design to be reliable, and to cover a diversity of potential products and markets. It seems that the home automation is a secondary market for LonWorks. In order to integrate it in home automation, products need to be physically small, simple and inexpensive. They must be easily installed (plug-and-play). LonWorks products are relatively large (physical size) and quite complex to install and configure. The price of a typical LonWorks product is relatively high. Trained and approved LonWorks system integrators are required to install LonWorks products, which increases the total system complexity and cost. Although LonWorks theoretically might use a variety of physical mediums, the common choice is twisted pair cable. Wiring a house is expensive. Even if some wireless LonWorks products exist, they are relatively expensive and not optimized for battery-powered operation because of the characteristics of the LonTalk protocol.

2.1.2.2 Wireless Protocols

- **Z-Wave.** Z-Wave is a wireless networking communications technology that was developed by Zensys. It was designed for the residential market and it uses radio signals (no wiring needed) to send commands via a remote control to chips embedded in home automation devices [25]. A commercial company, Zensys [25], develops z-Wave. Z-Wave is a proprietary wireless RF-based communications technology designed for control and status reading applications. Z-Wave offers reliable duplex communication in a mesh network topology and operates at the rate of 9.6 Kbit/s. All Z-Wave communication happens on a single RF channel with a predefined frequency. Z-Wave implements a proprietary routing protocol allowing the devices to forward data packets from one device to another towards the correct destination. Although the bandwidth of the Z-Wave is relatively low, it should be a sufficient client to cover the majority of home automation scenarios. Z-Wave allows implementation of battery powered devices and supports wireless routing, making it well suited for home automation. Z-Wave is a closed, proprietary technology. The deployment of the technology is limited. A potential disadvantage in environments with a lot of interference is that Z-Wave has no

mechanisms to change the communication frequency in order to and a frequency minimizing the amount of noise, making Z-Wave less robust against interference.

- **ZigBee.** ZigBee has several areas of application, from home automation to industrial automation. Citing the ZigBee Alliance "ZigBee is the Global Wireless Language Connecting Dramatically Different Devices to Work Together and Enhance Everyday Life" [26] ZigBee defines the network, application and security layers suitable for use in building automation, industrial, medical and residential monitoring and control applications for wireless networks based on the IEEE 802.15.4 technology. Examples of ZigBee applications: Lighting control, Automatic Meter Reading, Wireless smoke detectors, Heating control, Home security, Environmental controls, Industrial and building automation .The ZigBee specification is still under development, and the first final version was released by the end of 2004. ZigBee is developed by the ZigBee Alliance [26], a non-profit organization with membership open to all. The Alliance targets to define a global specification for reliable, cost-effective, low power wireless applications. The ZigBee alliance has today more than 70 members, and the number is continuously growing. ZigBee devices are expected to have a low duty cycle and to be inactive most of their operating time. Combined with the low power consumption of the IEEE 802.15.4 technology, the users can expect the batteries to last for months and even years. ZigBee defines three types of devices: ZigBee coordinator, ZigBee end device and ZigBee router. The ZigBee coordinator is responsible for setting up and maintaining the ZigBee network. It stores information about the network and the network participants. The ZigBee coordinator is typically not battery powered and it is listening continuously. The ZigBee end device is designed and optimized for battery powered operation, and allows devices to be inactive in periods of time in order to minimize power consumption. The end device is able to search for available ZigBee networks and synchronize to one of these.

2.1.3 Brief Literature Review

In the currently available research literature, there are several solutions for the home automation control problem, but they all address particular technological problems, presenting nothing about theoretical concepts that would approach the system to the user. For example, the work presented in [10] describes a service-oriented solution framework for home environments. It presents a protocol independent approach, the architecture is service based which allows the system to control several communication protocols. This approach is a very interesting one, if the system needs to be expanded, the programmer only needs to create one more service, but it lacks the implementation of living and interaction concepts and lacks the software reuse features that we are going to approach in this research work, but it is a good

starting for the research work related to the future QuiiQ Communication Foundation.

[12] presents a different solution to the same problem, going a little bit further and applying an embedded rule based engine to automatically and dynamically anticipate user actions. This is a very interesting approach, as it increments the functionality of the system by predicting user behavior, but they do not give the user a means of interacting with the system, which is very important to let the user "feel the power" of control and the advantages of user configured actions. In this paper, there is no particular focus on software reuse to.

Finally, the work presented in [13] proposes a different approach to an adaptative control strategy. Based on the definition of the "fuzzy markup language", it proposes a system that, through XML style sheets and an agent based framework designed to provide proactive services in home automation environments, allows the creation of rules that are deployed for the native "fuzzy markup language" representation by performing an efficient distribution of pieces of the global control flow over the different computers. The agent-based framework is also used to capture user habits, to identify requests and to apply the artifact-mediated activity through an adaptive fuzzy control strategy.

There are several other studies published presenting solutions to home automation through local networks, internet, mobile devices, etc. but none of them presents the basic concepts of user interaction with the house (like actions, routines, environments). There are some key issues, presented in [2] that should be mapped in the software architecture:

- **Management of the house.** Things like lighting, heating, security, etc;
- **Daily Tasks.** Things like washing machine, oven, drier, etc;
- **Activities.** Things like PC services, photos, video, audio, etc.
- **Theoretical Concepts.** Things like actions, routines, video doorman, etc (presented in 1.4).

[27] introduces some empirical concepts that are in the same line as the main objectives of this research work, however, it as same aspects that are not completely fulfilled. This research work will identify that falls and propose some solutions, as well as the definition of higher level concepts that will increase system value. All these concepts represent the main social needs of their users demanding comfort, status, prestige, security, etc. which should be met by the adoption of technology.

2.2 Products in the Market

Beside the research papers published, there are several products in the market for home automation control and visualization software, being the next three the most efficient of them all. These products will be the bases for the domain analysis and requirements definition.

2.2.1 QUiiQ AUTOMATION

"QuiiQ is a home automation software bundle that gives you a simple, yet powerful control of home divisions, environments, routines and devices."

QUiiQ AUTOMATION is a distributed home control software that allows several points of control and supports several home automation protocols (but not at the same time). Is a user-centered software that promotes several of the theoretical concepts presented in 1.4. It presents a typical client server architecture based on a windows service as server and several clients (Vista Media Center, Desktop or Pocket) [28].

QUiiQ AUTOMATION creates an access layer to the home automation network, giving the user a new and easy way of interaction with the house. The Figure 2.1 represents a home automation installation, a computer network with several computers with QUiiQ AUTOMATION clients installed on them, being one of the computers connected to the home automation communication network through RS232/USB or TCP/IP from the computer that as QUiiQ Server installed [29].

Internally, QUiiQ AUTOMATION implements a unique communication language, being the translational to the specific communication protocol done by the QUiiQ Server. This product supports several protocols (KNX, Cardio, Domintell, X10, Crestron, Cytech, Velbus, CBUS, etc), but there is no modularity, only one driver can be used in each installation. QUiiQ AUTOMATION, implements several user interfaces, each one specially designed for each hardware item, and meeting high standards of usability, optimizing the user experience and control, giving the user all the tools to virtual use and configure all the intrinsic concepts like the routines, environments, etc. This product integrates with several security systems and IP cameras, implementing a video surveillance tool as well and an alarm control central.

QUiiQ AUTOMATION already implements the following concepts (based on an XML database):

- **Division.** It identifies the location of a device;

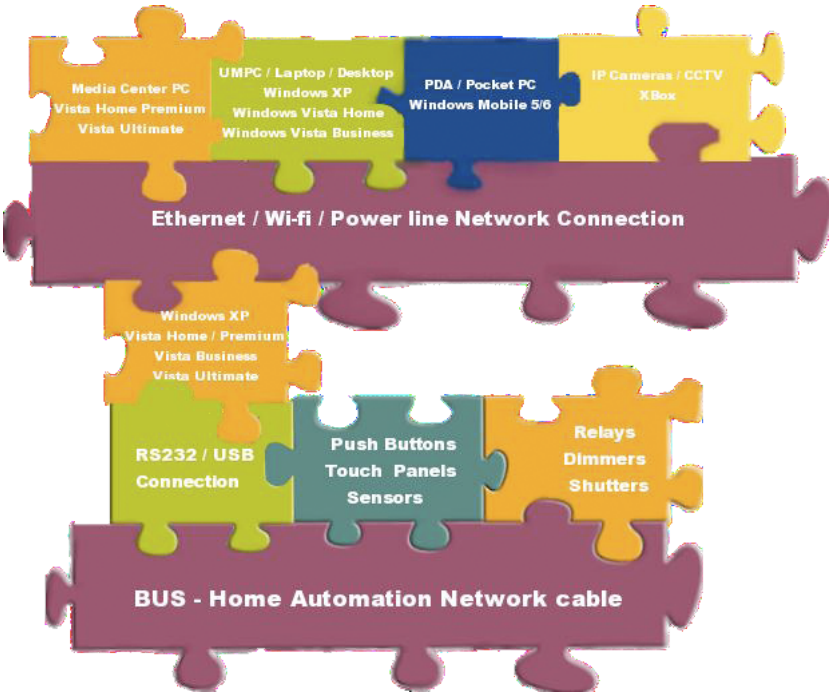

Figure 2.1: QUiiQ AUTOMATION Architecture.

- **Devices.** It supports devices with an abstract type and several behaviors but the state and the actions are related with the device itself and not with the behavior, so for each type of device, the saved state format can be different. The effort of parsing this new state is passed to the application level;

- **Environments.** They are defined ad a sequential step of action, but each action is on top of a device and not a behavior of a device, so this translation is made by the application level as well;

- **Routines.** A routine is a parallel set of actions that are triggered by a simple trigger (only AND operations is supported and no precedence's can be defined).

This product will be the main source of analysis, because we have access to the source code of the product and all its inherent studies and decisions.

2.2.2 StarDraw

StarDraw is software-based universal control platform designed to create custom user interfaces that can control several home automation and home entertainment systems, based on several communication protocols. It is an open-architecture approach, giving the system flexibility, minimizing the costs, and giving the system an extended functionality.

It generates a standalone or client-server application that can virtually control almost any device in a communication network. It uses an intuitive drag-and-drop interface, which can parameterize a control system without having to write a single line of code. We can divide its control properties in three different views, being two of them for the parameterization of the system and the other for the control itself.

The Topology view is used to present a graphical view of the system layout, allowing the parameterization and identification of the control inputs and outputs. It allows configuring the entire system including the communication protocols, supported functions and command/monitoring packet routing.

The Form view is used to create the control interface, being the controls dragged from a toolbox. This view creates a custom user graphical interface to control the exact home automation system that is installed in the house.

Finally, Control programs are the programs that derive from the use of the topology view and form view. They can be used in different hardware supports (laptops, desktops, pocket pc's), being compiled and self-contained execution [30].

This software approach is a very modular and interesting one, but it only allows the user to create multi-platform and customized "remote controls", not giving any option of other interaction concepts or automatisms to the system. The way of functionality of this software will be analyzed in a more detailed way in order to get more accurate information for the domain analysis phase.

2.2.3 MControl

The mControl Automation Service, as the name implies, manages the automation sub-system for mControl. The mControl Service has a built-in web server and "serve" the user interfaces to the mControl Clients.

It allows the interaction with several home automation, audio and security communication protocols, being the interface based on the windows media center and pocket pc. Implements all the functionalities of home automation (lighting, heating, security, etc) and allows the control through a user-friendly interface. The concept of macro is implemented in mControl, allowing the user to have a behavior similar to the routines referenced in 1.4 [31].

mControl is the least representative of these three products, because it is not a framework like stardraw and does not have the functionality of QuiiQ AUTOMATION, but its analysis will be important to check the common features and the base structure of home automation products. The embedded automation company (creator of mControl) already launched a press release announcing the implementation of the mControl SDK, a software development framework, the analysis o which will be very useful for this project, but there are no dates on when it is going to be available for the general public.

2.3 Home Automation Software Main Requirements

Home Automation suffers from a wide variety of protocols and consequently a difficult task of integrating them all in a unique solution. Because of this matter, flexibility is the main requirement of Home Automation systems. The target of the concepts presented in this paper should be able to receive new target protocols, and be flexible enough to allow a system developer to rapidly develop a new driver and integrate it in the system. Since the focus of this research work is in the foundations for a Home Automation framework, it should give enough flexibility for a developer to extend the proposed concepts and instantiate new ones [32].

Following this line of thinking, a few more technical requirements have been met:

1. **Uniform Commands** - The commands from each driver (each one implements the interface and the type of commands to each protocol) should be uniform independently of which driver they are speaking to. The protocols heterogeneity must be completely transparent to the application developer;

2. **Separation of Concepts** - The application logic must be completely independent of each protocol, making it possible to develop different systems using the same driver or develop a driver without changing the system;

3. **Grouping of Objects** - The concept of device must be defined in the system, and a definition of group of devices should be implemented, being the house division the better option for this;

4. **Ambient Intelligence** - A computer system that only allows the direct control of the home devices and appliances is not very useful, so some concepts should be implemented in order to give a higher value to the system;

5. **User Concepts** - Ease of interpretation for the user should be implemented in the system kernel, removing this way this responsibility from the graphical interface layer, and allowing the system to group the objects by is abstract type;

6. **System Instantiation** - Since this research work proposes the foundations for a future framework, a top object should be provided, allowing the developer to instantiate the system, and them introduce the devices and other concepts in it.

Regarding the context-awareness (presented in 5) a few more properties should be met by the set of technology that implements it:

- is aware of its own state and the state of related systems;
- is aware of users intentions, tasks and feelings;
- can autonomously adapt its behavior spontaneously to context changes.

2.4 The Infrastructure - QuiiQ Automation Framework

In the recent years, the term "smart living environments" have been an hot topic. There are many leading industry and academic researchers working on this topic. Many efforts have been made to create a smart home control software ([12], [10], [32]), but most of them adopt their own special devices or work over a specific home automation protocol or even based on very specific technologies, lacking the necessary abstraction to be implemented on top of any protocol and modular enough to allow the development of different components by different entities (for example, one entity develops the driver, other the user interface and other the system core) . This same period of time has been characterized by a huge development in computing industry (from computers and computers networks, to the integration of computer services and resources), being this development helped by the synergy of several factors like: the availability of cheap commercial off-the-shelf computing devices, the advances in network technologies and extensive adoption of standard network protocols, specially TCP/IP.

The QuiiQ Automation Framework (presented in figure 2.2) is the main objective to be fulfilled by several research works, being this particular one the research work that studies the QuiiQ Automation Foundation (the abstractions and the main core of a home automation software system). The basic assumption for the QuiiQ Automation Framework will address these issues, promoting a new distributed way of interconnecting several home automation protocols and graphical interfaces, by abstracting the main home automation concepts and integrating ambience intelligence into the system.

The QuiiQ Automation Framework will be divided in three main components:

- **QuiiQ Presentation Foundation.** This component will implement graphical interface that will allow the developer to easily develop clients for several supports that will interact with the home automation system;

- **QuiiQ Automation Foundation.** This component abstracts the home automation system to a language that all the components will understand. It will implement abstract execution concepts such as environments, routines and context awareness. This is the main object of study in this research work;

- **QuiiQ Communication Foundation.** This component could implement a typical SOA architecture that will allow a device (abstract concept present in the QuiiQ Automation Foundation) to execute an action using a certain service. Each of this services act over a home automation protocol or other kind of actuator, like send an email or start a phone call. These services will interact between them, allowing the service discovery and minimizing the configuration effort of the future system.

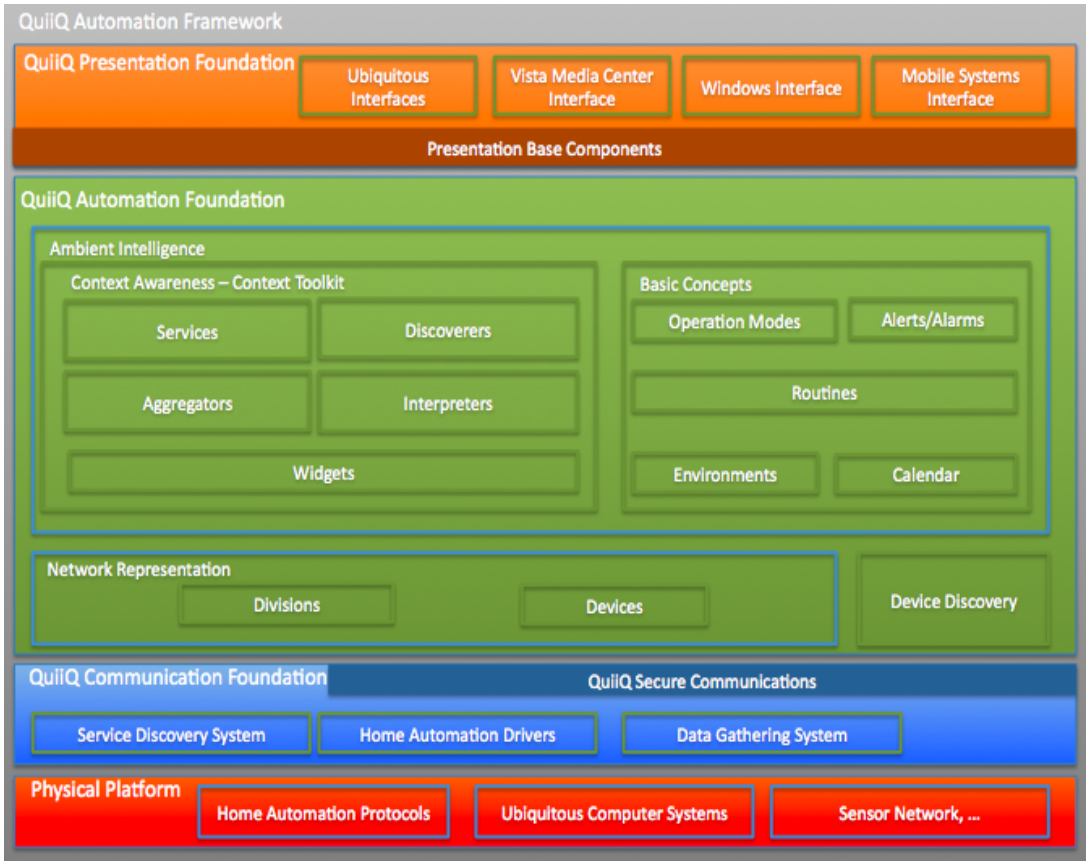

Figure 2.2: QuiiQ Automation Framework

In the future, the development of modular home automation applications will be possible. A development team can develop an application and another team from another company in some other part of the world will be able to develop its own driver to integrate it with all QuiiQ Automation Framework based applications.

Chapter 3

Frameworks

3.1 What is a Framework?

In its shorter definition, a framework is a set of reusable design together with a code implementation. It is a set of cooperating classes, both abstract and concrete, which represents a solution for problems in a certain problem domain [33]. But a framework is a lot more than a set of classes. It has layered structure indicating what kind of programs can be built and how they would interrelate, how they behave and which is the thread of control. Frameworks dictate the architecture of the future application by predefining design parameters and options from the beginning, allowing the developer to understand the key aspects right from the beginning, but leaving enough space for the developer to accommodate particular solution for each problem.

3.2 Object Oriented Frameworks and Software Reuse

In the past few years, it was possible to see lots of improvement in the area of computing and networking. Yet, the development of complex software continues expensive and error-prone. The growth of hardware architectures, operating systems and communications platforms demands the continuous reinvention and rediscovery of core concepts and components.

Object-oriented frameworks promise higher productivity and shorter time-to-market of application development through design and code reuse (not possible with non-framework based approaches). Object-oriented frameworks come as a solution for this problem, making it possible to using proven software design and architectures, reducing time costs and improving software quality. A framework is a reusable design and implementation that one can customize to produce custom applications [34]. It consists of a collection of abstract classes and connectors among them, that presents an abstract and customizable design (hot spots)

for a domain specific problem. Frameworks are also architectural, they define the skeleton of the applications, the implemented base classes, the way they interact and cooperate, their key responsibilities and the thread of control. Frameworks define the architecture of an application, but they leave enough space for an application programmer to focus on specifics of their applications [4].

The primary benefits of OO frameworks, are the modularity, reusability, extensibility, and inversion of control they provide to the application developer. Modularity allows the programmer to encapsulate the details behind stable interfaces, localizing the impact of the design and implementation changes. The stable interfaces provided by frameworks enhance reusability because they define abstract components that can me modified as needed by an application programmer, allowing for already tested solutions to be applied to a concrete problem [34].

So, the reuse of design, architectures and objects are the principal focus of frameworks. Software reuse is the process of creating software systems from existing software. Naur and Randal in the Software Engineering introduced this simple yet powerful vision in 1968: Report on a Conference Sponsored by the NATO Science Committee, Garmisch. There are several ways to accomplish software reuse, each of them using different artifacts to reach the objectives. Each artifact can be the source code, design, abstract specifications, documentation, etc [35].

Tools and techniques to be used in software reuse are usually divided in two big categories: compositional and generative approaches. The compositional approach is based on the reuse of software artifacts like design or abstract specifications; the generative approach is based on the idea of reusing software development processes and tools. A reuse technique must support, at least, one of the four software reuse activities: abstracting artifacts, selecting artifacts, specializing artifacts and integrating artifacts [36].

An ideal reuse technique should: allow the developer to find the components that exactly fit the needs; should be out-of-the-box and shouldn't force us to learn how to use them. The cognitive distance proposed by Krueger [35], basically defends that the cognitive distance between informal reasoning and the abstract concepts of the technology should be small. Citing Ademar Aguiar "Natural, succinct and high-level abstractions describing artifacts in terms of what they do, rather than how they do it, are thus very important for effective software reuse. From all the existing reuse techniques, the reuse of software architectures is probably the technique that comes closest to this ideal" [4].

The object-oriented paradigm is the perfect playground for software reuse, because they natively promote all the concepts of abstraction, inheritance, and polymorphism that encour-

age the reuse of existing code instead of writing new code from scratch. Taking advantage of these features, the code reuse is one of the most common reuse techniques based on class libraries. But the application domain limits code reuse and the underlying technology is very limited. Reusing the design of an application is more beneficial than reusing code, because the design is the main intellectual content of software, and much more difficult to develop than code [36].

Object oriented frameworks are a reuse technique that is a part of every reuse technique. They are more abstract and flexible (and harder to learn) than components, but more concrete and easier to reuse than a raw design (but less flexible and less likely to be applicable). Frameworks lead to the reuse of design, which is one of the most important kinds of reuse [4]. Comparing to other reuse techniques, frameworks have the advantage of being expressed in a programming language, so they are easier to learn, minimizing the learn curve of programmers. The object oriented frameworks main benefits are the reuse of code and design, that are higher than in any other technique, and the inversion of control, the modularity and the extensibility of that framework.

3.3 Fundamental Concepts

There are several definitions of object oriented frameworks, but the most referenced is the one found in [37] that says that a framework is a set of classes that embodies an abstract design for solutions to a family of related problems. This definition aggregates the essential aspects, which are: uses a set of classes, embodies a reusable design and addresses a set of problems in a particular domain.

This definition encompasses several concepts regarding object oriented programming as:

- **Data Abstraction.** Abstraction refers to the act of representing essential features without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract attributes. Storing data and functions in a single unit (class) is encapsulation. Data cannot be accessible to the outside world and only those functions, which are stored in the class, can access it;

- **Inheritance.** Inheritance is the process by which objects can acquire the properties of objects of other class. In OOP, inheritance provides reusability, like, adding additional features to an existing class without modifying it. This is achieved by deriving a new class from the existing one. The new class will have combined features of both classes;

- **Polymorphism.** Polymorphism means the ability to take more than one form. An operation may exhibit different behaviors in different instances. The behavior depends on

the data types used in the operation. Polymorphism is extensively used in implementing Inheritance.

After the fundamental concept definitions, we have the theoretical background to define some mechanisms that are widely used in framework development, the template and hook methods. The template method defines the skeleton of an algorithm, allowing subclasses to redefine several steps of an algorithm, without changing its structure. The template method implies the existence of two types of classes:

- **Abstract Class.** Defines abstract primitive operations that concrete subclasses define to implement steps of an algorithm and implements a template method defining the skeleton of an algorithm. The template method calls primitive operations as well as operations defined in Abstract Class or those of other objects;

- **Concrete Class.** Implements the primitive operations to carry out subclass-specific steps of the algorithm;

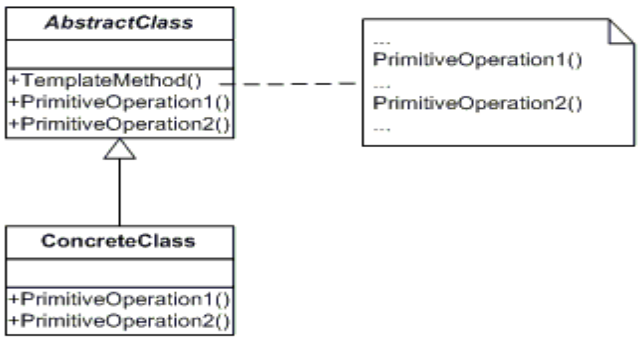

Figure 3.1: Template Method [4].

Figure 3.1 is the UML representation of a Template Method structure.

A hook method is a method that does nothing more than returning to its caller. Hook methods are typically defined in what are called "adaptor classes" which are classes that implement interfaces by overriding a number of abstract methods inherited from an interface with hook methods. Generally, template methods are used to implement the frozen spots of a framework, and hook methods for the hot spots. The frozen spot are characteristics that do not change from application to application in a certain domain. The hot spots are aspects of a domain that vary from application to application and must be kept flexible and customizable.

Finding the balance between hot spots and frozen spots is essential for the success of a framework, because hot spots offer more flexibility, but result on framework, which are more difficult to design and use.

Finally, regarding framework classification, they are usually classified regarding their extension and scope of work and are divided in white-box frameworks and black-box frameworks. A white-box framework relies heavily on inheritance and dynamic binding in order to achieve extensibility. Although white-box reuse is far more powerful, it is much harder to use. The component and application developers need to know the internal architecture of the framework in order to apply it to a concrete application, and the system hot spots are usually limited to inheritance. When the hot spots are well defined, it is relatively easy to design a white-box framework, but the big disadvantage is that the learning curve for an application programmer is bigger and the risk is very high.

Black-box frameworks are the easiest to use, because they use object composition and delegation, rather than inheritance. This type of frameworks are the more difficult to develop, because they need to identify the right interfaces and hooks, in order to anticipate the wider range of application requirements. A black-box framework hides its internal structure, and the user only knows a general description of the framework and its hot spots. These frameworks are easier to use, but more difficult to design and implement, being the hot spot mechanism, typically, composition.

In the real world, we usually use a mix of these two types called gray-box frameworks. They use inheritance and dynamic binding, as well as defining interfaces. They are designed to avoid the disadvantages of the white-box and black box frameworks. Frameworks can be defined by their scope of work. Fayad and Schmidt proposed in their work [34] three types of frameworks:

- **System Infrastructure Framework.** This type of frameworks aims to simplify the development of portable and efficient system infrastructure such as operating systems, communication frameworks and user interfaces and language processing frameworks. Typically these frameworks are for internal use only;
- **Middleware Integration Framework.** Typically these frameworks are used to integrate several applications, allowing one to reuse and extend a software infrastructure to work seamlessly in a distributed environment;
- **Enterprise Application Framework.** These frameworks try to solve typical problems in a certain domain of application.

3.4 Frameworks Development Methodologies

There are several research papers that propose lots of different framework design processes. However, there is no common consensus on a standard process to design frameworks. The

following list presents in a summarized way, several framework design processes that can be found in literature [38]:

- **Design through stepwise generalization.** This method proposes a two-phase framework design process. The first phase starts from the generalization of a representative application, and generalizes it in a sequence of steps into the most general form. In a second phase, the generalization levels defined in the previous phase are considered backwards, leading to an implementation in each level;

- **Design through systematic generalization.** This method decomposes the design process in three simple steps :

  1. Design a class model for an application in the domain;
  2. Identification of hot spots and identification of domain variability and flexibility;
  3. Generalizing the initial model, by integrating the domain variability through a sequence of transformations.

  In each transformation step, one hot spot variability is considered and a design pattern is selected [5];

- **Design Based on design patterns.** This process recommends that a domain application be developed as the first step. As a second phase, the development team should be trained in design patterns, and then they should systematically apply these design patterns to obtain a framework;

- **Design based on application experiences.** In this process, two or more domain applications are developed in order to gain experience in the field. Then, common aspects in all applications should be identified and included in the framework design. To validate the work, a derivation of the initial applications from the framework should be built. The last two procedures should be repeatedly applied, until the first applications are completely derived from the framework;

- **Design process based on viewpoints.** In this process, each developed application represents a different perspective of the framework domain. A set of unification rules should be defined, in order to describe how the viewpoints could be combined to compose a framework. The result of this procedure is a template hook model that is based on the template and hook methods previously presented. Then the hot spots of the framework should be identified;

3.5 Frameworks Success Factors

Through the years, the accumulated experience of several developers working with frameworks has defined several critical technical and organizational factors for the success of a framework. A brief explanation of this factors is as follows [39], [4]:

- **Well written and complete documentation and training.** A well-documented framework is easier to understand and apply. If training is provided, the learning curve will reduce radically;
- **No obligation of use.** A framework should be used freely. It should be viewed as a way to improve a programmer work, and should be his choice to use it;
- **Focused in relevant problems of the application domain.** The framework should represent and solve the main problems of specific domain, and not implement everything in that domain. It should be flexible and extensible enough to allow an application programmer to customize;
- **Enable simple applications to be implemented easily.** Simple applications should be constructed simply because they are not too general, too complex nor too flexible, but are easy too understand and extend;
- **Have an well defined architecture and vision.** A successful framework should have explicit architectural vision, which is defined and maintained through all of the development and maintenance processes;
- **Simple Architecture.** Only few key concepts, patterns and techniques should be used to design a framework, keeping it simple and focused from the beginning;
- **Managed separately from the application.** Distributed, promoted, supported and maintained separately;

Some popular examples of successful frameworks are Unidraw/HotDraw, ET++, Java Foundation Classes, JUnit and Eclipse.

Chapter 4

From the Physical World to Smart Living Environments

4.1 Software Variability

By definition, variability means "ability to adapt or likely to change or vary; changeable, inconstant, fickle, fluctuating, etc". The kind of variability present in software does not occur by chance but is brought about on purpose.

Variability modeling supports the development and the reuse of several software artifacts. Variability allows choosing between different options which we call *variants*. In the context of home automation, each system can differ from another in a wide variety of options. It is possible to find this variability in software components used in a daily basis:

- **Available Protocols.** The huge variety of home automation protocols available in the market widely differs in the way they communicate and interact. A simple example is the *domintell* and *KNX* protocols, while the first one communicate through the use of strings and is completely proprietary, the second one is an open standard and the communications is done by writing using the bit arrays directly to the BUS. Software variability allows the provision of different control drivers and algorithms according to the available protocol;

- **User Interface Variability.** One of the aspects in smart living environments where variability is easy to identify, is the user interface. Various realizations of local and remote user interfaces are presented by several systems and discussed in the technological community. Some of them constitute alternatives, while others tend to coexist in the same environment. Typical interfaces consist of a TV based interface, a PDA interface, a web based interface and touch-screen interfaces. Software Variability is necessary to support all these modes of control on top of the exact same home automation network;

- **Functional Variability.** Differing from installation to installation, it is needed in order to integrate different components in the same home automation installation. On top of protocols variability, there can be lots of different devices aiming to integrate into the smart living environment. Video doorbell systems or video surveillance systems are widely used in home automation as a complement to the typical home automation installation. Software variability is necessary to allow the integration of this kind of systems in the home environment and in exact same user interfaces provided by the system.

The essence of the above-presented examples is the support for different options, which leads to variability in the software for smart living environments.

One of the main causes of this variability are the *final customer wishes and technical constraints*. In order for a system to work, each configuration of hardware devices must be supplied with appropriate software. In order to support this hardware variability, the software must be sufficiently agile to adapt himself to the enormous amount of different configurations that can be present in a home automation system. This conflict between final client wishes, hardware and software is one of the big problems of home automation software, as the client demands quality and functionality, the software and the hardware must be able to work with each other transparently [40].

Different customers have different demands regarding their home automation installations and the quality of the installed system. During the development work, engineers, architects, programmers and test engineers have to deal with commonality as well as variability while developing a software platform, if it is to be reusable in different scenarios.

In this research work, the concept of software variability is extremely important because of the wide gamma of options in home automation protocols, the vast choices in user interfaces and the many options for higher-level functions to be introduced by the software into the house environment. This research work presents an abstract way of representing all of this variability (by introducing theoretical abstract concepts that represent the basic concepts in a higher level), presenting the kernel of a home automation framework, where variability is achieved by the implementation of several drivers and several user interfaces (completely independent from each other), using a middleware capable of adaptation to every scenario.

4.2 Empirical Concepts

In order to achieve this capability of system abstraction and adaption, discovering a way to abstract and generalize all the physical concepts into a software system is a priority.

Based on the idea presented in *"Hiding complexity and heterogeneity of the physical world in smart living environments"* by Thierry Bodhuin et al. in [27], it is possible to find a correspondence between the home automation physical world and the software world.

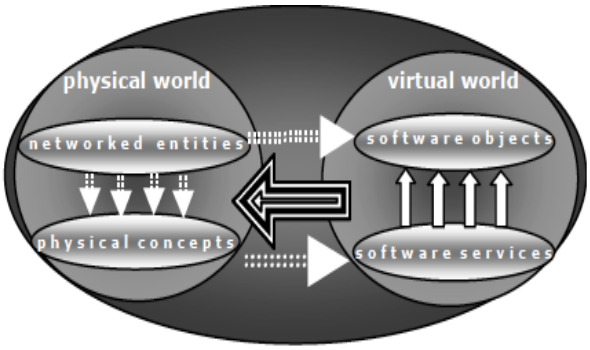

Figure 4.1: Characterization and connections from the physical to the virtual world [27].

As shown in figure 4.1, the connection between the physical world and the virtual/abstract world, used to represent the home automation network in the computational system, is very linear an straightforward. Regarding each world characterization, the physical world can be defined as follow:

- **Network Entities.** Represents each device (mobile or not) that is connected to the network. The use of interconnecting technologies (wired or wireless) guarantees that the devices are connected and can communicate between them;

- **Physical Concepts.** The description of each network entity determining their internal states. Lights, Air conditioning, temperature sensor, alarms, power switches, etc. All of these entities have their internal state that can be represented as On, Off, 25%, Armed, etc.

In order to create a software system that represents all of these concepts in such a way that they are reusable through different software systems, the software entities must be defined as generic as possible. The virtual/abstract world can be defined as follows:

- **Software Objects.** Made of software interfaces and artifacts. Represents the network entities and their implementation;

- **Software Services.** Sensitive to the measures of presence of the physical concepts.

Based on the analyses of figure 4.1, the definition of several behaviors and connections between these two worlds is trivial. The following list presents this information in a summarized way:

- The vertical rows with white hatches connecting networked entities and physical concepts. This occurs when a network entity has a direct effect over a physical concept, for example, a networked switch that turns a light ON;
- The arrow connecting a networked entity and a software object represents the abstract connection between these two entities. The parameters of a networked entity can affect a software object, but not the other way around;
- The arrow connecting the physical concepts and the software services represents the connection of the actual state of the physical device and the actual state of the abstract device. When the light is turned on, it sends a message that is captured by the driver (implemented based in the QuiiQ Communications Foundation).
- The vertical arrows connecting software services and the software objects guarantee that software services are based on software objects and that there exists a transmission of commands. When the driver receives the message it changes the internal state of the software object;

Based on these connections, the identification of abstract concepts to represent the network is highly important. The presented structure is used to describe the base theoretical concepts, facilitating the representation of the physical world in the virtual/abstract world.

A networked entity is a physical device on which it is possible to trigger actions that change the state of the device. Mainly, each entity is composed by its main information and a set of behaviors that can act on top of different types of actions.

A device to be represented in a home automation software system must have three main requirements:

- Physical Identification
- Functional Identification
- Space Identification

The following section presents these requirements, defining the tuples of information needed to achieve this representation.

4.2.1 Physical Identification

A device can be defined by its physical aspect (PhA), its behavior (B) and its abstract type ($AbsT$):

$$PhI =< PhA, B, AbsT >$$

The physical aspect of a device can be represented by its name (N) (for example "Kitchen Light"), its decryption (D) (for example "The Light on the kitchen near to the table"), its location in the house (L), its location in the house blueprint (<X,Y>) (for example "50,20" where x=50 and y=20) and a representative image (I):

$$PhA =< N, D, L, < X, Y >, I >$$

For example, a device can have the following information associated with it: PhI = «"Living room light", "The light in the living room near the sofa", L,(50,10), IMAGE>, B, AbsT>. The behavior and abstract type attributes are complex attributes that will be presented in the next section, *Functional Identification*, being the location presented in the *Space Identification section*.

Using this definition of physical entity we are able to represent all the components of a device, from the representation that is easily understood by the final user, to an abstract type that will allow the application of filters on groups of devices (for example only the security devices are showed) and, finally, the physical behavior of the device, which will allow the communication and the synchronism of states between the physical network and the software object.

4.2.2 Functional Identification

From the point of view of functional analyses over a device, two base concepts can be identified: the Behavior and the Abstract Type. A behavior is a complex type that can be described as a tuple of three attributes:

$$B =< Name, Cod, Ty >$$

The Name attribute is an abstract name given by the software programmer to better identify what kind of behavior it represents (for example "On"). The Cod attribute represents the physical address corresponding to the physical behavior in the physical device (for example, in KNX a same device may have different addresses to different behaviors and they are represented by a string like "1/2/2"), and, finally, the Ty attribute, which is the Type of behavior, being described by an enumeration.

Regarding the Abstract Type, it is an abstract representation of the type of device we are working with. It can be described as follows:

$$AbsT = < Name, Desc >$$

Because this concept is purely abstract and it is only used to logically group the devices in different sets, represented by a usually used name. For example, it is used to group all the lights (they can have different behaviors like on/off, dimming, etc.) in the same well-known group.

4.2.3 Space Identification

After the representation of the functional identification of the devices, they must be spatialy located to be useful. The space identification presents the concept of division, so its abstract type or its location can group a set of devices. The following line presents this concept in a more formal way.

$$L = < Name, < X, Y >, < DimX, DimY >, Desc >, I$$

A division is represented by its name (given by the system installer, for example Kitchen), its X,Y reference in the house blueprint (for graphical positioning of the device in blueprint views), its dimensions, a brief description of the division and a representative image.

Chapter 5

Context Awareness as a way to Ambient Intelligence

People are very successful in the discussion and conveying of ideas among them and reacting accordingly. In this kind of interaction, the participants are able to use implicit situations, or context, to better understand each other. One of the big problems on using technology is that this ability is not present when humans interact with technology. In traditional interactive systems, the humans have some kind of mechanism of providing input to the computer, like the mouse or the keyboard. By improving the computer access to context information, we can achieve an all new way of interaction and integration of the system in user's every day life, by increasing the richness of the communication between user and computer, we are able to produce more useful computational services [41].

As a consequence, it is imperative to find a way of giving the computer this information. But there are some difficult hurdles to achieving this objective: the input, the way of dealing with this input, and the output; Computers are only able to deal with pre-defined tasks that their programmers have defined, and this limits their ability to understand our language and activities.

The following sections present all the inherent concepts to context-awareness, some base design principles of context-aware applications, a brief description and discussion of several context-aware frameworks present in the research world, and, finally, its integration with home automation.

5.1 What is Context?

With the appearance of a wide variety of mobile hardware and ubiquitous / pervasive computing, developers increasingly focused on developing applications for PDA's, mobile phones, touch screen pc's, media center pc's, etc. and introduce this technology in daily tasks. This

resulted in the proliferation of applications that are completely integrated in a persons daily tasks in such a way that these applications became central for service development, suppressing all the technological complexity inherent to this type of systems [42].

The concept of context-awareness was first introduced by Schilit and Theimer in 1994 referring to a new class of applications which are aware of the context where they are executed, and defining context as the location, identities of nearby people/objects and changes in those objects. Using as an example an active map service that provides information about the location of objects and how they change in time, the term of "context" is used to refer to location, and context-awareness as the ability to discover and react to changes in the environment where they are being executed [43].

In a similar definition, Brown *et al.*[44] in 1997 defines context as location, identities of people around the user, the time of day, season, temperature, etc. while Ryan *et al.*[45] in 1998, defined it as the user's location, environment, identity and time. There are other definitions for context in the literature, but they only provide synonyms, for example, they defined context as being the environment or situation. Some consider context to be the user's environment, while others consider it to be the applications environment.

The definition of Schilit and Theimer is the closest one to the operational definition that is required. They claim that the important aspects of context are: where you are, whom you are with, and what resources are nearby. They define context to be constantly changing execution environment, including the following pieces of the environment:

- **Computing Environment.** available processors, devices accessible for user input and display, network capacity, connectivity, and costs of computing
- **User Environment.** location, collection of nearby people, and social situation;
- **Physical Environment.** lighting and noise level.

Beside all of these definitions of context they all are too specific, there is not a generic option to define context. Context is about all the events relevant to the application and its user, so we cannot use any of the previous definitions of context. The definition of context that betters represents this concept is presented by Anind K. Dey in is Phd thesis in 2000. He defines context as: *"any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves"*[46].

A context-aware application looks at the *who's, where's, when's* and *what's* of entities and uses this information to determined the *why* a situation is occurring. It is not actually the

application that determines the why a situation is occurring, but the designer of the application does using the incoming context information, encoding the consequent actions. For example, in a tour guide system when the user is approaching an interesting place, the application uses this context to present the information related to the given place. However, a common way to classify context, is differentiating between physical and logical context.

- **Physical Context** is the one that can be measured by hardware components;
- **Logical Context** is the one that is inferred by monitoring the user's behavior, his tasks and sis physical and emotional state.

5.2 What is Context-Awareness?

When humans talk with humans, they are able to use implicit contextual information to increase the conversational bandwidth. Unfortunately in human-machine interaction, this ability is not easily applied. In interactive systems, the user provides all the input for the computer, so this implicit context information is not present. By giving a computer the ability to access contextual information, we increase the richness of communication in human-computer interaction, making possible the design of increasingly useful computing systems [41].

In the same article published by Schilit and Theimer in 1994 they defined a context-aware application as *"an application that adapts according to its location of use, the collection early people and objects, as well as changes to those objects over time"*. Another definition was given by Pascoe *et al.* [47]: context-awareness is the ability to detect, gather, interpret and react to context changes in the user's surrounding and changes in its devices.

Shilit *et al.* defined four categories of context (shown in 5.1). These categories are divided in two properties, the kind of context and the *modus operandi*: whether the system gathers the information or performs an action and whether this is done manually or automatically.

Table 5.1: Context-Aware application categories depending on Schilit *et al.* [43].

	Manual	**Automatic**
Information	Proximate selection and context information	Automatic contextual reconfiguration
Command	Contextual commands	Context-triggered actions

Proximate selection is when applications request information manually based on its current disposable context, while the automatic contextual reconfiguration category encompasses applications that automatically retrieve information based on the current context. Contextual command applications are the ones that execute commands for the user when they are manually instructed to do so and context triggered applications refer to applications that provide

automatic execution of commands depending on available context [42].

There have been several definitions of context-awareness. The definition presented by Brown, Bovey *et al.* in 1997, Dey and Abowd in 1997 and other authors refer to context-awareness as the applications that dynamically change and adapt their behavior based on the context of themselves and the user. Brown in 1998 defines context-aware applications as applications that automatically provide information and/or take actions according to the user's present context as detected by sensors. Finally, Fickas *et al.* [48] defines environment-directed applications to be applications that monitor changes in the environment and adapt their operation modes according to predefined or user-guided guidelines.

The more widely accepted definition of context-aware is presented by Anind K. Dey in is Phd thesis in 2000 :

"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's tasks."

This definition differentiates from the previous ones because it only requires a system to be responsive to context changes. It allows the gathering and the interpretation of context to be performed by other components. The given definition is given in a broad sense so that it encompasses already developed context-aware applications. This author defined three categories of context-aware applications which deal with representation of the information, automatic execution and tagging of context information, and a combination of the categories previously mentioned.

5.3 Application domains for context-aware applications

Computer systems are being increasingly integrated in our day-to-day lives, existing in almost everything we use (our laptop, our cell phone, the metro ticket machines, our GPS, our car, our house, etc). Following this line of thinking, it is possible to describe the most important domains where context-aware applications will be/are important [42]:

- **Airports.** Emergency response solutions are based in context-aware systems, allocating specific personnel (security, maintenance, fire department, police, etc.) to a specific situation. Services directly aimed for passengers are in use too, giving the passenger information about the shopping zones nearby to his location;

- **Smart Homes.** When context-awareness is integrated with the technology present in the home, it can maximize comfort, increase the quality of life of elderly/disabled people, providing the following features:
  - *Security functions.* Generation of alerts and take security actions in case of a fire, gas leakage, tap open, intrusion detection, etc. These alerts can be sent to the user or to the appropriate institution, while at the same time taken some actions like for example closing electro-valves, starting the fire-sprinklers, etc.
  - *Appliances functions.* Home appliances can entirely be controlled by home automation system, allowing the definition of context-aware applications that maximize comfort, improve energy management, etc;
  - *Environmental information.* It can report and adapt through information not only available from the home sensors but also using information derived from web services (weather forecast for example);
  - *Healthcare functions.* In elderly/disabled people, the monitorization of its health, reminders, emergency care, etc. can all be achieved using context-awareness.
- **Entertainment.** This application is closely associated with the user location. A variety of services have been developed that given the user location, it provides information regarding cinemas, restaurants, bars, petrol stations, etc. nearby its location;
- **Offices.** Context-aware applications can be very valuable in an office environment. These can give valuable information and send alerts about equipment status, managing calendar entries, locating employees, energy management, etc;
- **Museums.** In museums, context-awareness is usually applied by locating the user's position in the building and guiding him through it, presenting relevant information about the room where the user is in.

5.4 Design principles for context-aware systems

In this section we provide an overview about the design principles when building context-aware applications, together with the units found in most context-aware frameworks: sensors infrastructure, context modeling, reasoning methodologies and historical data.

5.4.1 Architecture

The design of an architecture for context-aware systems can take different approaches depending on the location of sensors, the amount of possible users, the available resources on the user's device, reusability or the facility to further extensions to the system. The context

gathering system is very important when designing context-aware systems. Chen in 2004 [49] presents three architecture designs for context acquisition:

- **Direct Sensor Access.** This approach was often used in devices with sensors built in. There is no division between the sensor data acquisition and its logic processing; everything is tightly coupled in the application. Drivers for the sensors are hardwired into the application, so this tightly coupled method is only aplieble in rare cases. It is not suited for distributed systems, because it lacks a component able to manage multiple concurrent sensors access;

- **Middleware Infrastructure.** In the modern software design modularity is imposed in order to separate the business logic from the data acquisition and user interface. This approach introduces a layered architecture to context-aware systems, hiding the low-level sensing details. This approach (when compared to the direct sensor access), eases extensibility and reusability of the code, since the logic application layer is separated from the data acquisition layer which is specific to each sensor type;

- **Context Server.** This approach allows the access to the sensor data concurrently by multiple clients at a time, through collecting all sensor data by the server. Gathering sensor data is moved to this so-called context server to facilitate concurrent multiple access, reusing sensors and relieving the clients of resource intensive operations. In return, one has to consider about appropriate protocols, network performance, quality of services, etc. when designing context-aware systems based on client-server architecture.

Similarly, Winograd in 2001 [50] presents three different approaches for managing context and context-aware system components:

- **Widgets.** Widgets are software components that give a uniform way of handling device specific drivers that are coordinated by a widget manager. They are derived from the homonymous GUI elements, providing a public interface for a hardware sensor. The concept of widget was presented by Anind Dey in his Phd thesis in 2000 [46], being defined as a software component with a public interface that hide low level details about the specific sensor type by encapsulating the driver and physical connection ports, and furthermore providing contextual data directly to the applications. Instead of implementing notifications for each type of driver, it offers a consistent way of sending messages to and from the widgets. The drawback in this approach is that is susceptible to component failures, that may impact the overall system functioning;

- **Network Services.** Unlike the previously presented model, where a centralized component that keeps track of existing widgets is introduced, this model approaches context

acquisition sources as independent sources. This service-based approach is not as efficient as a widget architecture due to complex network based components but provides robustness;

- **Blackboard Model.** In opposite to the process centric view to the previously presented to models, this model uses a data centric approach. Instead of sending requests and process callbacks, this architecture processes post messages to a shared media (blackboard), and subscribes to it to be notified when some specified event occurs. The most notorious advantage of this model is easy configuration that allows the addition of new context sensors easily. One drawback is the need of centralized server to host the blackboard and the necessity of two hops in order to reach the application, which decreases the communication efficiency.

In this research work, we will present several layered context/aware systems and frameworks, most of them differing in functional range, location and naming of layers, the use of optional agents or other architectural concerns. Besides this adaptations and modifications that all models present, there is an indefinable common architecture transversal to all models.

5.4.2 Sensors Infrastructure

In order to improve reusability and improve the process of developing applications, it is usual to separate the sensing layer from the rest of the application, since a common basic module for all architectures is the sensing layer. It is notable that the 'sensor' word means not only the hardware sensors but also every object that may provide useful information to the context information. The sensing technology has significantly improved in the past few years powered by new solutions that increase quality, reliability, increased number of parameters that can be measured (temperature, humidity, luminosity, wind, pressure, motion, acceleration, etc.) while minimizing energy consumption, size and cost.

In a wide definition, sensors can be grouped in three sets (depending on the way data is acquired)[51]:

- **Physical Sensors.** These type of sensors are the more widely used in context-aware systems. They are capable of gathering information about the environment, people, objects and body functions, and are spread from temperature sensors, microphones, touch sensors, light sensors, location sensors, motion sensors, biosensors for measuring the human body functions, etc;

- **Virtual Sensors.** This kind of sensor provides information from software applications or services. They observe application events, operating system events or network events. For example a person location can be determined not only by a tracking system (physical sensor), but by consulting a virtual calendar;

- **Logic Sensors.** Logic sensors include data inferred from both, physical and virtual, sensors. They can use these information combined with additional information from databases or various other sources in order to solve higher tasks. One classical example of the application of this type of sensors is detecting an employee's current position by analyzing logins at desktop PCs and a database mapping of devices to location information.

All of these three types of sensors together provide the application with explicit information given by the user, implicit information about the abilities of the devices; existing information obtained by other services and captured data by physical sensors. Integrated, they create the sensing module of the architectural model, which provides information to the rest of the system.

5.4.3 Context Modeling

After the data acquisition, in order to efficiently use the retrieved data, it needs to be stored / represented in an appropriate form, suitable for further processing. The context model gives this power to the application. Strang and Linnhoff-Popien in 2004 [52] summarized the most relevant context modeling approaches:

- **Key-Value Models.** This is the simplest technique for modeling context, which is widely used. It represents context information with key-value pairs, which are later used by some matching techniques to perform sensor data discovery (e.g., LIGHT=75);
- **Markup Scheme Models.** All markup-based modules use a hierarchical data structure consisting of markup tags with attributes and content. User Agent Profile (UAProf) and Composites Capabilities/Preference Profile (CC/PP) are some of the specifications that describe the capabilities of mobile devices and different user agents, enabling the content providers to produce and deliver content suitable for each request;
- **Graphical Models.** The Unified Modeling Language (UML) is also suitable for modeling context;
- **Object Oriented Models.** Modeling context using object-oriented techniques promises to use the full power of object orientation (e.g., encapsulation, reusability, inheritance, polymorphism). An architecture already exists which uses a class ContextObject which is inherited by other context-specific classes which implement the common abstract methods, convert data streams to context objects and vice versa, and provide well known interfaces to access the contexts logic;
- **Logic Based Models.** A logic-based model has a high degree of formality and is based in facts, expressions and rules. A logic-based system manipulates with the elemental

items of this model and infers higher-level logic by utilizing the already defined rules to deduce new facts;

- **Ontology Based Models.** Ontology is a description of concepts and their relationship. However, it is not only a classification of concepts; it also includes higher relationship between them and enables interaction between systems that have computable ontology's. One way of implementing these ontology's is by using the Web Ontology Language (OWL), which consists of a set of classes, class hierarchies, set of property assertions, constraints on these elements, and types of permitted relationships between them. While another alternative is using a knowledge representation language - the Resource Description Framework (RDF).

Research has shown that the most complete modeling technique is the ontology model. It is the most expressive and meets the requirements of most of the systems. However, because each architecture tries to meet different goals, this model is not always chosen for a given architecture.

5.4.4 Reasoning Methodologies

A challenge in context-aware computing is to manage context properly and use it in an intelligent way, by using captured data from different sources in order to deduce new information. Interpreting low-level context data into higher-level ones can be done by using ontology's, logic or by using one of the following techniques to infer context and situations in context-aware systems[53]:

- **Artificial Neural Networks.** Artificial Neural Networks is used as a solid clustering algorithm for context-awareness in sensor networks. They perform well regardless the noise level produced when capturing sensor data and they support unsupervised learning of input data. New context data can be easily included, and the algorithm will adapt to this new information by recalculating its internal representation of context [53]. In this way, unpredictable context can be easily captured, complex relationships modeled and patterns in behavior detected;

- **Bayesian Networks.** Bayesian Networks are probabilistic graphical model that represents a set of variables and their probabilistic relationships. Given all the possible context types and values acquired by the sensors, a Bayesian network can predict (with a probabilistic value associated) the occurrence of a situation or the state of a variable;

- **Hidden Markov Languages.** Hidden Markov Languages are probabilistic models used to represent nondeterministic processes and consist states, actions and observations. In this statistical model, the system represented is assumed to be a Markov process with unknown context parameters and the aim is to find out the unknown parameters from

the context data on disposal. Just like the Bayesian Networks, the Hidden Markov Languages requires a training phase in order to classify activities, building a statistical memory of sequences of events. It is mostly used for modeling human behavior, because it is able to recognize sequences of activities [53].

None of the previously described techniques can be pointed as the most efficient or beneficial for deducing context-awareness and each of these methods addresses different issues that arise from context-aware sensing. For example, the Artificial Neural Networks, work with low-level sensor, but they produce some level of noise, but they can deal .very well with this noise. Because the data is able to approximate itself, the neural network can adapt to new contexts without the intervention of the user. Bayesian Networks are theoretical ideal, such they obtain the highest accuracy only when any node in the graph is strictly independent from any other variable, except its descendants. The Hidden Markov Languages, recognizes sequences of activities and they require a training phase in order to classify activities. Therefore, the choice of one over the others is guided by the framework requirements [53].

5.4.5 Historical Data

Keeping track of historical data can help the system to detect patterns, establish tendencies and predict future of a context parameter. This historical data will allow the implementation of intelligent learning algorithms, that would provide flexible and adjustable context-aware systems.

As most data sources constantly provide data, the maintenance of context history is mainly a memory concern, so centralized high-resource storage component is needed. To achieve this functionality lots of technologies can be used (plain text files, XML, relation databases, etc), being the database the most useful because it allows the use o Structured Query Language (SQL) which enables the construction of queries that will reveal more complex patterns and higher level relationships. Context-aware frameworks may also provide interfaces for accessing these databases and the possibility of storing other entities from context data such as domain knowledge and inference rules, allowing the system to infer higher-level context.

5.5 Survey of Context-Aware Frameworks

In this section, a brief introduction to different context-aware framework is presented, discussed and compared on a common criteria. All of these models present different approaches to context-awareness, describing different architectural styles mainly driven by context acquisition, different methods for context representation, processing logics and reasoning engines,

and some of them different communication patterns and storage approaches.

The following list represents the frameworks that will be shortly described:

- Context-Awareness Sub-Structure (CASS);
- Context Broker Architecture (CoBrA);
- Context Management Framework;
- Context Toolkit;
- CORTEX;
- Gaia;
- Hydrogen;
- Service-Oriented Context-Aware Middleware (SOCAM).

5.5.1 Context-Awareness Sub-Structure (CASS)

CASS provides the mobile context-aware application developer with high-level context derivations. This framework allows the separation of context-aware application code from high-level context reasoning and behaviors. Sensor nodes are computers with sensors attached and can be mobile or static. It is possible to use storage and artificial intelligent components (because the middleware is server-based, so it does not suffers from the processor and memory constraints that would apply to mobile computing), as well as the facility to store large amounts of data. To reduce the effect of intermittent connections (in the communications between server and client), CASS uses a local cache option [54].

The previously described architecture is presented in 5.1. This framework contains 'Interpreter', a 'ContextRetriever', a 'RuleEngine' and a 'SensorListener'. The 'SensorListener' listens for updates from sensors, which are located on distributed computers called sensor node. Then the gathered information is stored in a database by the 'SensorListener'. The 'ContextRetriever' is responsible for retrieving stored context data. The 'ChangeListener' is a component with communication capabilities that allows a mobile computer to listen for notification of context change events. Sensor and 'LocationFinder' classes also have a built-in communications capabilities. Finally the 'RuleEngine' is able to discover new context based on a set of rules[51].

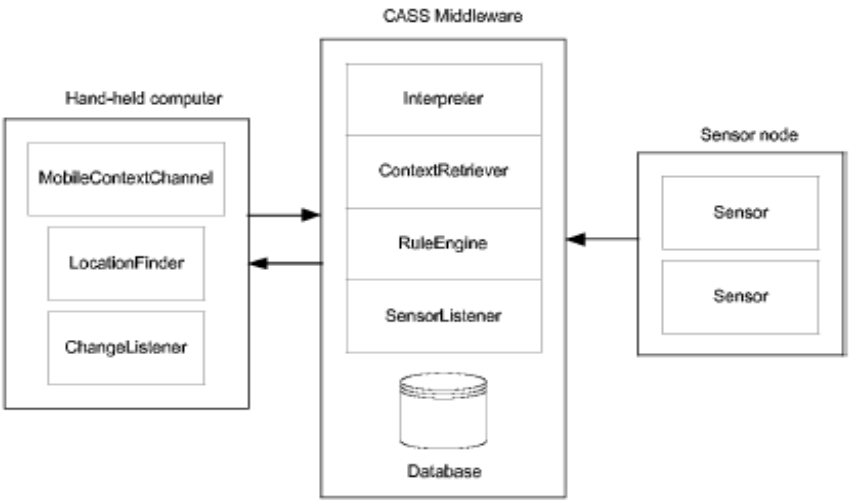

Figure 5.1: Context-Awareness Sub-Structure (CASS) Architecture [54].

5.5.2 Context Broker Architecture (CoBrA)

The Context Broker Architecture, is an agent based architecture for supporting context-awareness in so called living spaces (e.g. Houses, Cars, offices, yachts, etc) that are populated with intelligent systems that provide pervasive computing services to user [51]. Like presented in figure 5.2, the central component of this architecture is the Context Broker which maintains and manages a shared contextual model on behalf of the collection of agents and is consisted of four main components [49]:

- **Context Knowledge Base.** A persistent storage of context knowledge;
- **Context Reasoning Engine.** A reactive inference engine that reasons over the stored context knowledge;
- **Context Acquisition Module.** A library of procedures that form a middleware abstraction for context acquisition;
- **Privacy Management Module.** A set of inference rules that deduce instructions for deciding the right permissions for different computing entities to share a particular piece of contextual information and for selecting the recipients to receive notifications of context changes.

An ontology approach is used to model context (COBRA-ONT), which is integrated with a rule based inference engine, and also, the architecture includes its own policy language, Rei [55] , which controls access and enables security and privacy protection [51].

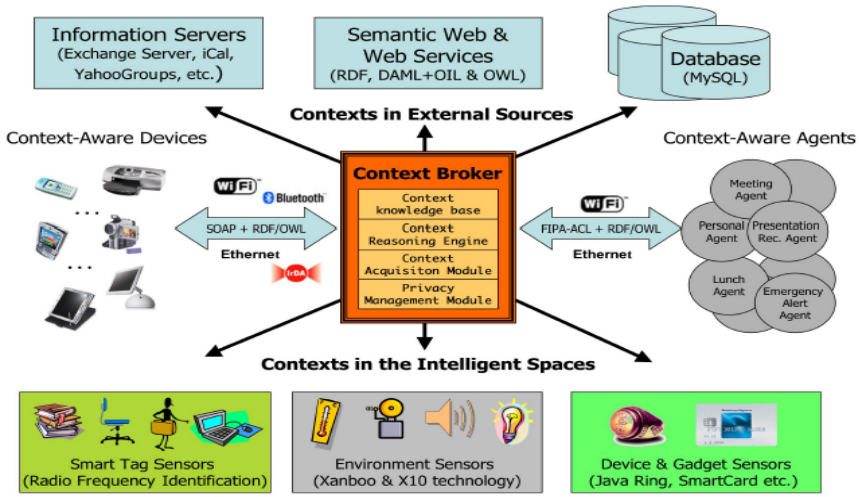

Figure 5.2: Context Broker Architecture (CoBrA) Architecture [49].

5.5.3 Context Management Framework

The Context Management Framework, was presented by Korpipaa *et al.* in 2003 [56] and permits the recognition of semantic contexts even in the presence of uncertain, noisy, and rapidly changing information and delivering contexts for the terminal applications in an event-based manner. It uses a blackboard approach as the underlying communication paradigm between framework entities. This framework is presented in figure 5.3 and is divided in four main functional entities: the context manager, the resources servers, the context recognition services and the application.

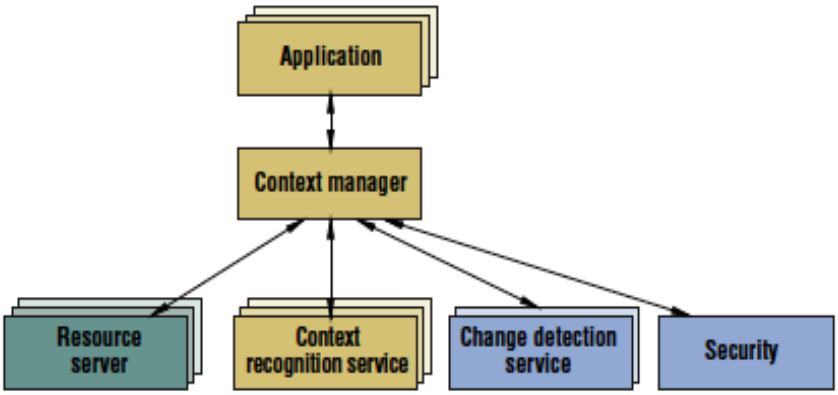

Figure 5.3: Context Management Framework Architecture [56].

When entities communicate, the context manager functions as a central server (manages the blackboard, stores context data, processes the context information, infers higher level

information and delivers this information to the clients), while the other entities (resource servers, the context recognition services and applications) act as clients and use services that the server provides [56]. Regarding the context model, the Context Management Framework uses an ontology defined in RDF.

The main difference between this framework and the others, apart from the architectural design, is the advanced way of handling context data represented through ontology's and the use of fuzzy logic to build higher-level data. However, a drawback of this architecture is that the context manager represents a single point of failure, since the applications normal functioning depends directly from it.

5.5.4 Context Toolkit

The Context Toolkit adopts the widget architecture style and its main purpose is to alleviate the development of context-aware applications by providing a common framework that will be taken as a basis for further uniform development of services. It consists of widgets, which comply to a distributed allocation managed by a central component called a discoverer that keeps record of all running widgets in the network.

The toolkit enables application developers to build context-aware applications and it consists in three main abstractions[57]:

- **Widgets.** Context widgets encapsulate information about a single piece of context, such as location or activity, for example. They provide an uniform interface to components or applications that use the context, hiding the details of the underlying context-sensing mechanisms;
- **Aggregators.** Context aggregators can be thought of as meta-widgets, taking on all capabilities of widgets, plus providing the ability to aggregate context information of real world entities such as users or places. By acting as a gateway between applications and elementary widgets, aggregators hide even more complexity about the context-sensing mechanisms;
- **Interpreters.** A context interpreter is used to abstract or interpret low-level context information into higher-level information. For example, identity, locations, and sound level could be used to interpret that a meeting is taking place.

The toolkit makes the distributed nature of context to context-aware applications. The communication between two components is peer-to-peer, without any direct intermediaries and most of the components inherit the communication capabilities from the super class called

BaseObject , which provides the ability for interconnection between the context-aware system components. The framework adopts the key-value pairs (encoded using XML for transmissions) for context modeling, has no reasoning engine is present.

Figure 5.4, represents a sample configuration of context components (arrows indicate the flow).

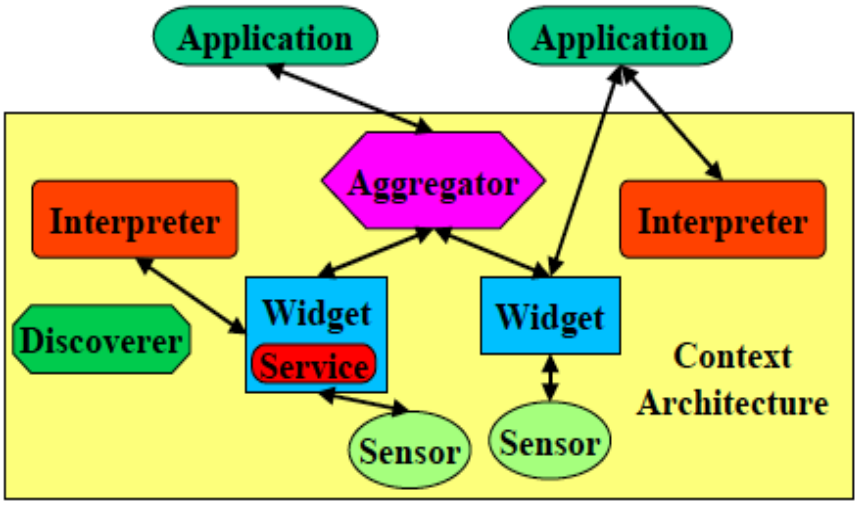

Figure 5.4: Context Toolkit sample configuration [51].

The Context Toolkit uses the already mentioned discover in order to provide the ability to search and find new sensors at runtime. The discovers works as a registry component which interpreters, aggregators and widgets have to notify about their presence and their contact possibilities. After this registration, the component is pinged in order to validate its state; if a component does not respond to a consecutive ping commands, the discoverer determines that the component is not available and removes it from its registries list.

5.5.5 CORTEX

The CORTEX system is an example for a context-aware middleware approach, and it is based on the Sentient Object Model [58] that was designed for the development of context-aware applications in an ad-hoc mobile environment. A sentient object is an encapsulated entity, with its interfaces being sensors and actuators. Action is controlled based on sensor input according to internal control logic, consisting event filtering , sensor fusion and intelligent inference. In this object model, sensors are defined as entities that produce software events in reaction to a real world stimulus, while actuators are defined as entities, which consume software events and react by attempting to change the state of the real world in some way via

some hardware device [58].

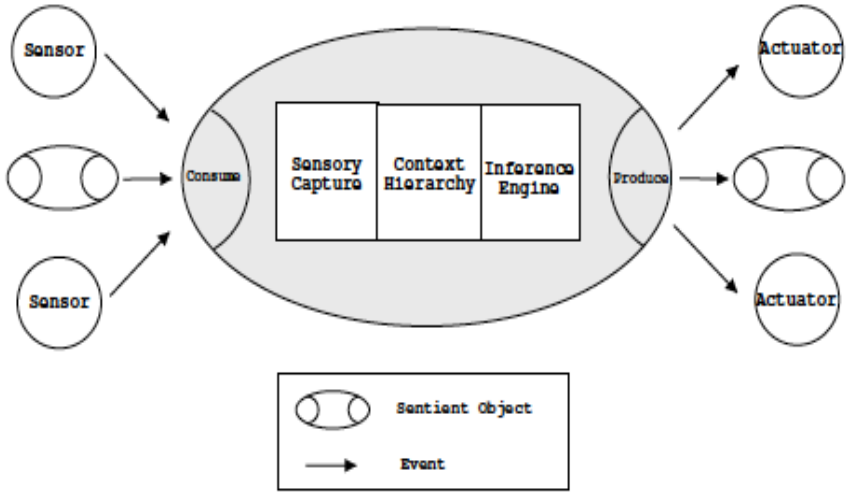

Figure 5.5: Sentient Object Model [58].

Figure 5.5, represents the previously described model, and it is divided in three major internal components, which provide these functions:

- **Sensory Capture and Context Fusion.** The sensory captures component of a sentient object performs sensor fusion in order to manage uncertainty of sensor data and derive higher-level context information (using Bayesian Networks) from multi-modal data sources;
- **Context Hierarchy.** The context hierarchy encapsulates knowledge about actions to be taken and possible future situations into contexts. By defining a hierarchy of contexts in which a sentient object may exist and by associating specific actions to be undertaken in each context, a sentient objects behavior in influenced by its context;
- **Inference Engine.** The inference engine component is responsible for changing application behavior according to context and by observing a certain facts, and pre-defined rules, is able to decide which rule to fire.

5.5.6 Gaia

The Gaia project is a distributed middleware infrastructure, which extends the typical operating system concept. It is intended to coordinate the development and execution of mobile applications for active spaces, typically a single room and provides the following functionalities: program execution, I/O operations, file-system access, communications, error detection and resource allocation [59].

It is structured as a traditional file system with a kernel composing the core of the system (as shown in figure 5.6) and applications built on top of it, which provide specific services. Here, context is acquired by context providers, classified by the type of information they gather, therefore the architecture distinguishes between three types of context:location, context and events which enables reasoning on a higher level and inferring activities. The frameworks processing is implemented into the Context Service Module which performs first order logic operations such as "and" and "or", and applications query this module for obtaining specific context information.

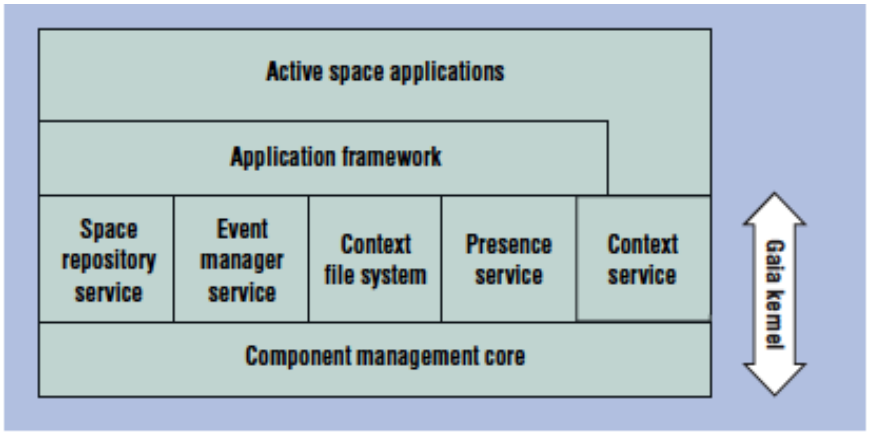

Figure 5.6: Gaia Architecture [59].

Context is modeled in a special manner (this representation is used for representing context and defining rules), by using a 4-ary predicates consisting of:

$$< ContextType >, < Subject >, < Relater >, < Object >$$

In addition, context is represented as directory, where the path represents the context type and the value (e.g. location=kitchen is represented as: /location:/kitchen/).

In the Gaia framework, different context providers are stored in a registry component. This registry maintained a list of available context providers and applications use this registry to find providers of the desired context [59].

5.5.7 Hydrogen

While the availability of a central component is essential to the majority of existing distributed context-aware systems, the Hydrogen system tries to avoid this dependency, by applying the differentiation between remote and local context. The remote context is information another

device knows about, the local context is knowledge our own device knows about.

The Hydrogen architecture comprises three layers to separate the concerns of interacting with physical sensors, storing and maintaining the context from the application itself. Figure 5.7 represents this architecture that is organized in three different layers [60]:

- **Adaptor Layer.** This is the layer responsible for retrieving data from the physical sensors and possibly enriches this information with logic context information;
- **Management Layer.** The Context Server, embedded in the Management Layer, provides simple methods for the application for retrieving or subscribing to a context;
- **Application Layer.** This layer uses the context provided by the layers below.

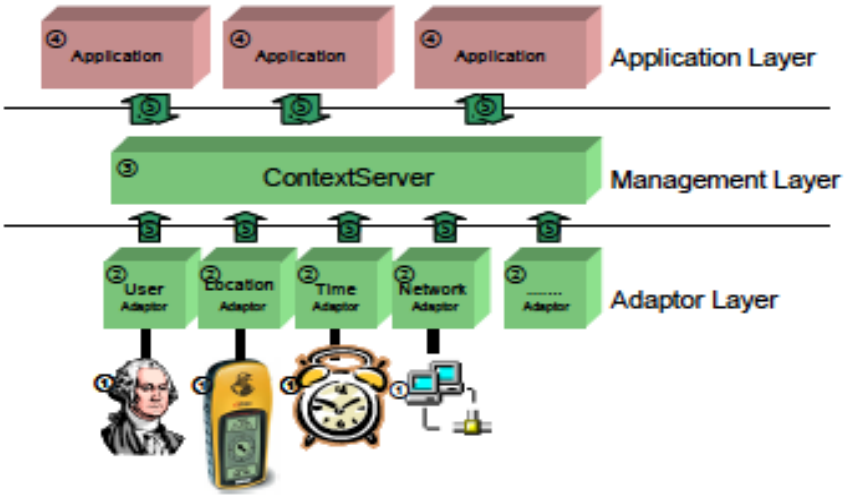

Figure 5.7: Hydrogen Architecture [60].

Due to platform and language independency, all inter-level communications is based on XML. regarding resource discovery, hydrogen only uses local built-in sensors and does not connect to distributed sensors, therefore, no discovery mechanism is involved. Hydrogen uses a object-oriented context model approach with a superclass called 'ContextObject' which offers abstract methods to convert data streams from XML representation to context objects and vice-versa.

5.5.8 Service-Oriented Context-Aware Middleware (SOCAM)

SOCAM is a ontology based context-aware framework that aims to provide all the architectural support to develop context-aware applications in the pervasive computing domain. It

is a distributed middleware that uses a central server (called context interpreter) which gains context data through distributed context providers and offers it in mostly processed form to the clients [51].

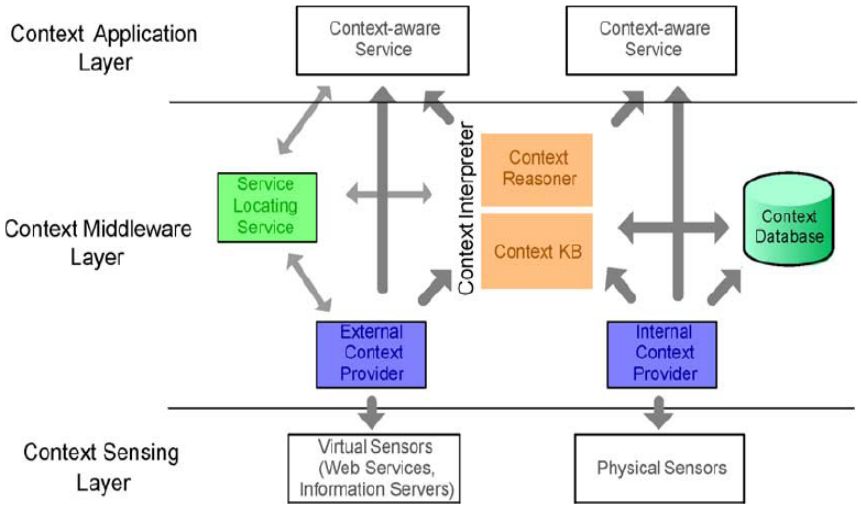

Figure 5.8: Service-Oriented Context-Aware Middleware (SOCAM) Architecture [61].

As represented in figure 5.8, it consists of the following components which act as independent service components [61]:

- **Context Providers.** These components abstract context from heterogeneous sources and convert it to an OWL representation so this information can be shared and reused by other service components;
- **Context Interpreter.** Through the use of logic reasoning, these components process the context information gathered by the Context Providers;
- **Context Database.** It stores the ontology's and past context for a sub-domain (i.e. home automation domain);
- **Context-aware Services.** This components use different levels of context and use this information to adapt their behavior;
- **Service Locating Service.** It implements a mechanism in which context providers and context interpreter can advertise their presence; it also allows users and applications to locate these services.

SOCAM offer a discovery system called *Service Locating Service*. Different context providers, which reside in both the internal and external network advertise their services

through this component [61]. Regarding the context model, authors divide a pervasive computing domain into several sub-domains, and define individual low-level ontology's in each sub-domain to reduce the complexity of context processing. Each of these ontology's implemented in OWL provides a special vocabulary used for representing and sharing context knowledge [51].

5.5.9 Discussion

Table 5.2 summarizes the main aspects of the discussed approaches to context-aware frameworks. The main criterion for an architectural approach to a context-aware system is the separation between the context acquisition and the user components. The sensing technology is implemented differently in every architectural approach, and it is important to encapsulate the sensing mechanisms in separate components to support the compartimentation of responsibilities and allow the access to the contextual data through different interfaces.

There is no standard description language or ontology to represent contextual information that can be reused across different middleware applications, so different and proprietary solutions have emerged. For example, SOCAM uses external virtual sensors to acquire information from web services, and internal providers for querying sensors are processed by using events represented in OWL, this being the most sophisticated approach.

Regarding the context model and the context processing logic, every model implements its own being the ones that use ontology's the ones that can derive new concepts (using reasoning engines and highly sophisticated ontology's) to adapt the service behavior accordingly. The major drawback of the Context Toolkit its the context model, a set of attribute-value topples. The development of intelligent context processing and aggregation services is limited due the fact that such attributes do not have a meaning.

Managing the historical context data provides a knowledge base to the development of intelligent learning algorithms which allows the system to provide highly adaptable context-aware services, proactively providing a way to predict context information.

In the domain of home automation another important aspect to be taken care of in a context-aware system is security and privacy The CoBra system uses Rei policy language to express security policies about contextual information. Gaia uses several mechanisms to define restrictions and secure communications for tracking locations and people. The Context Toolkit uses the concept of ownership to allow the access to the information to the owner only. These facets are among the most important, and are often disregarded by the system's

designers, components of a context-aware system as the protection of sensitive context data must be guaranteed [51].

Mainly, the common aspect between all the analyzed solutions is the decoupling between the sensing infrastructure and the rest of the system, which increases reusability of context sources within the system. Each of the analyzed frameworks have a different context model and uses different communication principles, which makes the communication between frameworks difficult to accomplish and prevents developers from reusing services based on another framework. The standardization of formats and protocols is an important step for the enhancement of context-aware systems, to make the development of context-aware services the focus rather than the communication between context sources and users.

Table 5.2: Context-Awareness Frameworks Comparation [51].

Architecture	*Sensing*	*Context Model*	*Context Processing*	*Resource Discovery*	*Historical Context Data*	*Security and Privacy*	
CASS	Centralized middleware	Sensor nodes Relational data model	Inference engine and knowledge base	n.a.	Available	n.a.	
CoBra	Agent based	Context acquisition module	Ontology's (OWL)	Inference engine and knowledge base	n.a.	Available	Rei policy language
Context Management Framework	Blackboard based	Resource servers	Ontology's (RDF)	Context recognition service	Resource servers + subscription mechanism	n.a.	n.a.
Context Toolkit	Widget based Context widgets	Attribute-value topples	Context interpretation and aggregation	Discoverer component	Available	Context ownership	
CORTEX	Sentient object model	Context component framework	Relational data model	Service discovery framework	Resource management component framework	Available	n.a.
Gaia	MVC (extended)	Context providers	4-ary predicates (DAML + OIL)	Context-service module (first-order logic)	Discovery service	Available	Supported (e.g., secure tracking, location privacy, access control)
Hydrogen	Three layered architecture	Adapters for various context types	Object-oriented	Interpretation and aggregation of raw data only	n.a.	n.a.	n.a.
SOCAM	Distributed with centralized server	Context providers	Ontology's (OWL)	Context reasoning engine	Service locating service	Available	n.a.

5.6 Context Awareness in Home Automation

Several context-aware office-style research environments have been developed so far to support people while at work. In an office-style environment there usually are system administrators to keep the system up and running, higher levels of investment can be taken if the system brings real cost savings to the company. Developing context-aware environments for people at home creates additional challenges. In contrast with an office environment, at home people freely decide how they organize space and time, what activities they undertake, when, where, how often and who is involved. Despite giving them a real personal, additional value, special care has to be taken to fulfill their requirements for (the first seven items are shared in a office and home environments) [62]: safe, supportive, convenient, pleasant, enjoyable, entertaining, relaxing, usability, usefulness, social acceptance, privacy protection, low cost and zero system administration.

In order for the system to have natural and meaningful interactions with the home users, the system must be aware of its occupants context, whereabouts, activities, needs, emotions and situations, allowing the system to adapt its interaction with the user. There are several research dimensions (these dimensions and its relations to the user are present in 5.9) that need to be addressed when designing context-aware systems: Instrumentation, middleware, user experience, privacy/security and applications [62].

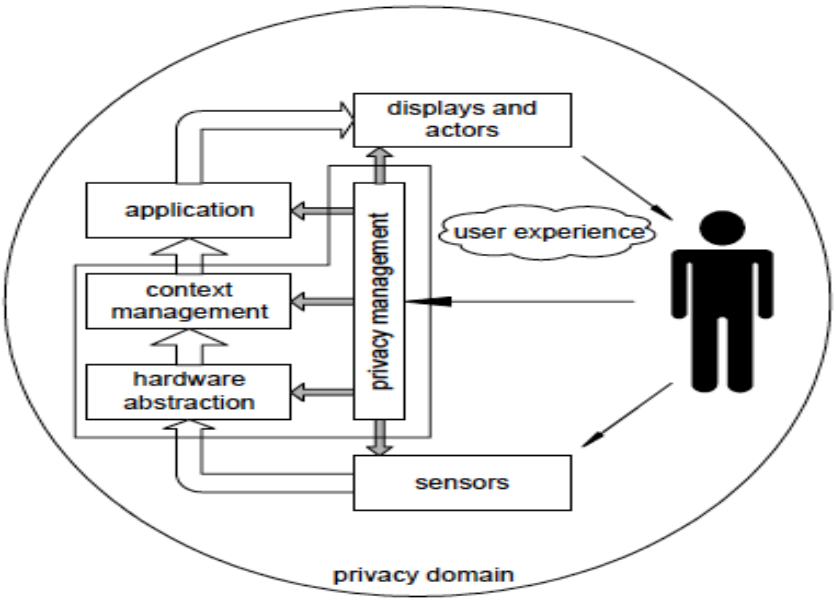

Figure 5.9: Basic components of a context-aware system interacting with a user [62].

This research work is about the design of the foundations for the middleware to a future home automation framework so, we will only focus on the necessary requirements for the

middleware component and privacy manager of the system.

Regarding to the middleware, a context-aware system for home automation should respect the following requirements:

- Have an hardware abstraction layer;
- Have a context manager that derives basic context information from raw data and maps into the context model;
- Have a privacy manager that controls the access to the context information.

Chapter 6

QuiiQ Automation Foundation

Throughout the day, people execute many actions within their home environment, such as communicate with each other, relax, seek entertainment, sleep, cook, etc. As a response to this, in the past few years many appliances have been introduced in the home environment to provide comfort, entertainment, security, in short, making life easier and more enjoyable.

Telephones, for example, have been revolutionary in allowing people to communicate without being in the same physical environment, internet has allowed people easy access to information and services, radio, television, DVD player and other multimedia solutions have brought new means of entertainment for a wide variety of tastes. As a consequence of this search for comfort, entertainment, security, etc., the wide variety of hardware introduced in the home environment means that the user is overwhelmed by a multitude of devices with different capabilities, different ways of operation, different access and control interfaces. There is an urgent need for these devices to interact with the user and communicate between them in an easy and comfortable way, which can be achieved through intelligent systems and interfaces.

Home automation and home entertainment systems may have revolutionary effects in the way people live and interact between themselves and their homes, and in the way they spend their days, handling routine jobs, increasing spare time and find more enjoyable experiences with their friends and family at home. The user must be the main focus of the *Ambient Intelligence Systems*, these being adaptable to each home and to the personal tastes of its users.[63]

QuiiQ Automation Foundation emerges as a response to this problem, being the kernel of an all-new framework (presented in section 2.4) for home automation. In order to represent all of the home automation network, effective ways of interacting with it, reach higher levels of usability and user experience, an analyses of the problem is performed and a means of representing all the main concepts in a way that is easy to understand by the software architects,

the final user (because they can be directly represented in the graphical user interface) and a way of reaching high flexibilility and extendibility is achieved.

6.1 Development Methodology

The QuiiQ Automation Foundation was designed based on the analyses of three home automation products (QUiiQ Automation, MControl and StarDraw), on the analyses of the current research literature and on my own experience as a professional of home automation hardware, software and systems integration. The methodology applied in the development of this research work was based on framework development methodologies and it was a mix of the following two:

- **Design based on application experiences.** This approach followed my experience on the field of home automation software. It is mainly based on a product called QUiiQ AUTOMATION and some analyses of the MControl software. It is a derivation of the QUiiQ AUTOMATION kernel with some improvements extracted from field experiences and the contact with the system final users;
- **Design through systematic generalization.** This approach was also applied because a first class model of QUiiQ AUTOMATION was defined, hot spots and domain variability were identified and finally a generalization of the initial model with some improvements was designed.

For the design of system models, the use of UML was the best available option because it is an industry standard, gives the necessary flexibility, allows the representation of static and dynamic features and allows the definition of formal restrictions. The following research uses UML to represent class and activity diagrams and English to explain the concepts and present some restrictions to the model.

6.2 System Overview

QuiiQ Automation Foundation and the relations between its components are represented by figure 6.1. The class division represents the theoretical concept for space identification (4.2.3), so the entire system is organized around this class since it aggregates all the other components. From the final user point of view, the concept of division encompasses the physical location for all the physical components (e.g., the blind is in the living room).

A device is a n-tuple which is composed by its abstract type (grouping information based on every day use classes, which represent the physical component that the user accesses (lamp,

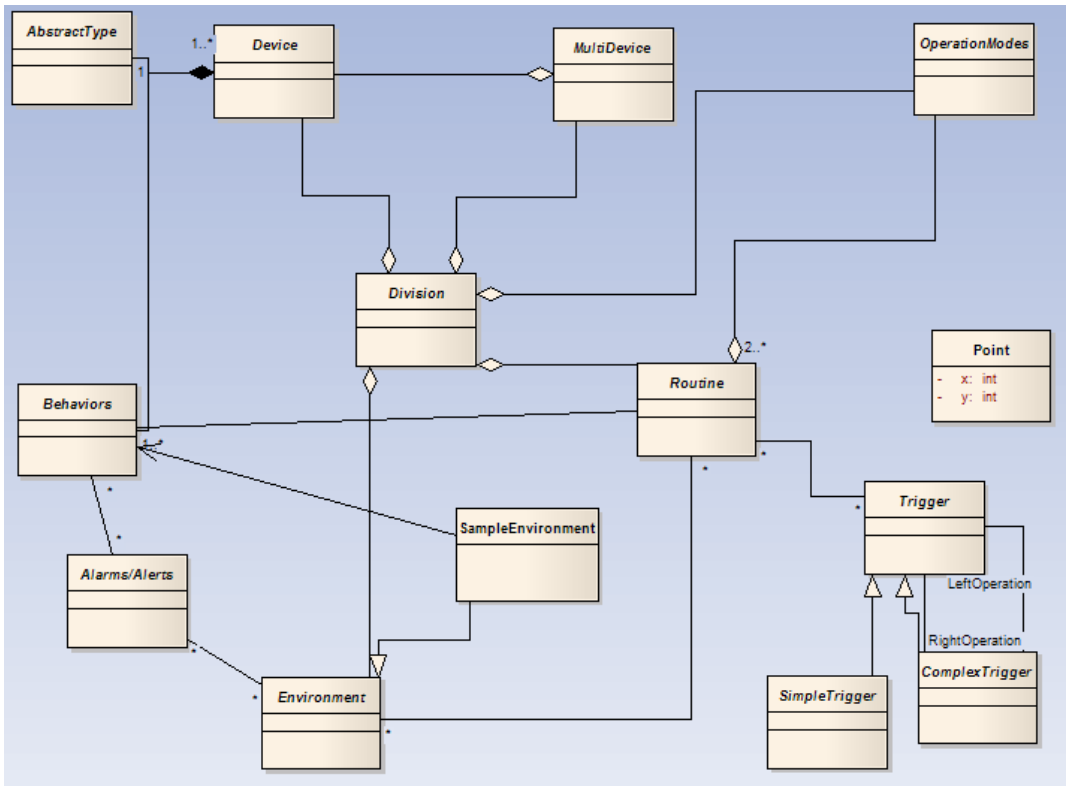

Figure 6.1: QuiiQ Automation Foundation Main System as a UML class diagram.

blind, HVAC, etc.)), and one or more behaviors (based on physical device behaviors, which represent the module output where the action is initialized). A multidevice is a theoretical device which in reality is a set of devices sharing the same behavior. This will allow the final user to control a set of devices by using only one button (for example, control all the lights in the bathroom with only one action).

Alarms are a special kind of device which is more complex than the ones allowed by the previously defined device concept. An alarm consists mainly of a set of zones, modes and alarm types. When an alarm or alert is raised, the system should react in accordance. There are two different kinds of alarms in the home automation market: the ones where you set the mode for each zone independently and the alarm is almost always partially armed, and the ones where you set a mode for all the zones, and the alarm is always armed or disarmed.

Environments and routines are completely theoretical concepts without any physical correspondence. An Environment allows a new way of interacting with the system, by creating a set of devices exhibiting different behaviors, allowing its activation with only one click. A routine allows the definition of automatic triggers to execute a set of actions (with properties different from the ones found in the environments, and beside this an environment can be an

action of a routine too).

The following sections present and analyze these concepts in depth.

6.3 Home Automation Network Representation

Before thinking of functional increments to home automation systems, a way of representing and organizing the entire system must be found. In order to achieve the previous objective, this research proposes the definition of two UML models (they are static, i.e., they only represent and organize the information), allowing the future dynamic concepts to have a base to work upon.

By observing a house, it is easily noticeable that it is partitioned by divisions and each division has several devices in it. Keeping an analogy with the real world, the best way to organize an home automation system is by reusing the same concepts. These concepts are based on the analyses made in chapter 4, where a set of abstract definitions were found to overcome the variability of physical devices and its presence in the house environment.

6.3.1 Division

By definition a division is *"something that divides or separates; a partition"*. In the context of architecture a division is something that divides the house logically over several partitions such as bedroom, kitchen, etc. In this research, a division is exactly the same thing, it divides the concept of house/system in several parts, providing a way of grouping all the other objects in a logical way.

Divisions can be nested: a division can contain other divisions, so it can represent the existence of several nested levels in the house blueprints (floor, room, etc). Figure 6.2, represents this concept in the form of a class diagram and is based on the *Composite Pattern* (a composite is an object designed as a composition of one-or-more similar objects, all exhibiting the same functionality). This concept is particularly interesting to organize the future graphical user interface (for the final user) and to spatially locate the sensors network for the context-aware system.

It represents the fundamental information "about a division" based on a UML model, and a way of creating multidimensional divisions, allowing the organization of the house into several sections (i.e., a division can have subdivisions contained in it). The presented interface

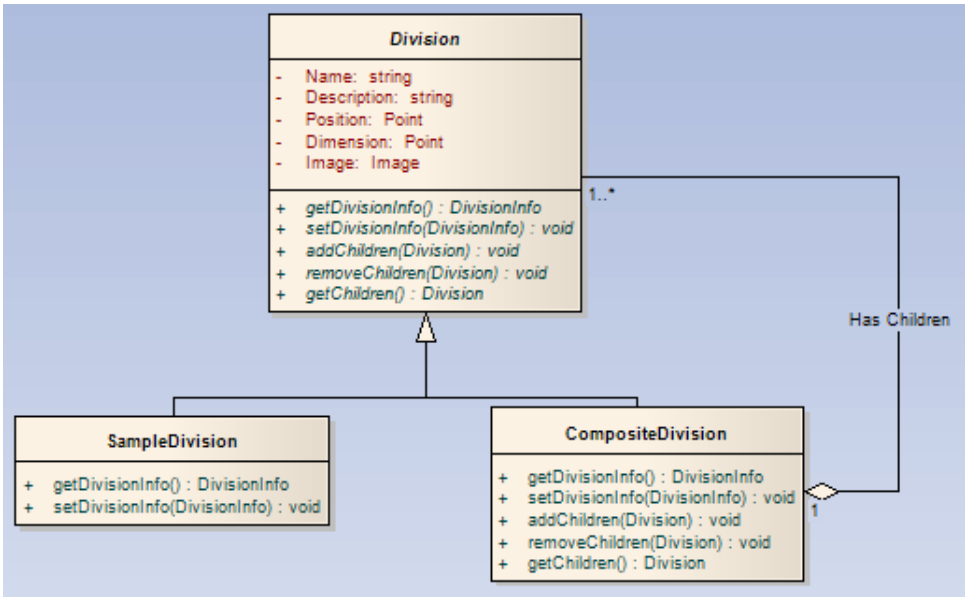

Figure 6.2: QuiiQ Automation Foundation division class model.

provides a way of expanding the concept, which can be adapted to several particular situations in an easy and elegant way. Divisions are a framework hotspot, so this concept can be extended by the application developers to better suit their needs, as long as they maintain its basic structure.

6.3.2 Device

The concept of device integrates different perspectives. From the point of view of the system integrator (its behavior and data types) and from the point of view of the final user (the abstract type). These two views are important because they give the system the necessary flexibility to represent almost all different kinds of devices existing in the home automation market, and allowing their representation and aggregation in a way that the final user understands, giving it an easy way of interacting in an non intrusive way.

A device consists of a name, one or more behaviors, and one and only one abstract type. The abstract type is important to group and represent the devices in the same way people do, allowing the existence of filters in the graphical user interface, which provide a better and more usable way of interacting with the future system that can be implemented based on these foundations. It hides all the complexity and variation within this type of hardware, grouping them by abstract type (e.g., lighting, blinds, sprinklers, etc.), and presenting the actions these perform independently of protocol specifics (e.g., KNX or Domintell).

The following list presents the abstract types already defined in the model:

- Air Conditioning;
- Lights;
- Power Outlets;
- Blinds;
- Video Doorman;
- Central Heating;
- Security Alarm;
- Electric Door;
- Multi Room Audio;
- Sprinklers;

All these abstract types are represented by words and concepts used by people in their daily activities, which is the best way of representing the home automation devices on the final user interface.

In figure 6.3, one can see the way the abstract class implements the concept of abstract device, and by specialization, all the pre-presented abstract types are already integrated into the system. This allows the next system developer to create new abstract types (based on the abstract class) to better suit the integration needs for a particular situation.

The concept of behavior is used internally to allow the execution of commands. A device can implement several behaviors and have a different code number (the object identifier in the physical network) for each one and each behavior can have a different way of interacting with the physical system, being completely independent of the protocol that is being used (i.e., an On behavior in KNX sends a one bit command to the physical network, and a setValue behavior sends a byte command to the network). This abstraction is very useful if the system developer wants to develop a multi protocol system, in which the house can contain several different protocols, allowing the integration of various physical subsystems with all of them being controlled by the same user interface. This is possible due to the abstraction level achieved by the QuiiQ Automation Foundation, and by the use of a future SOA (Service Oriented Architecture) approach in the QuiiQ Communication Foundation.

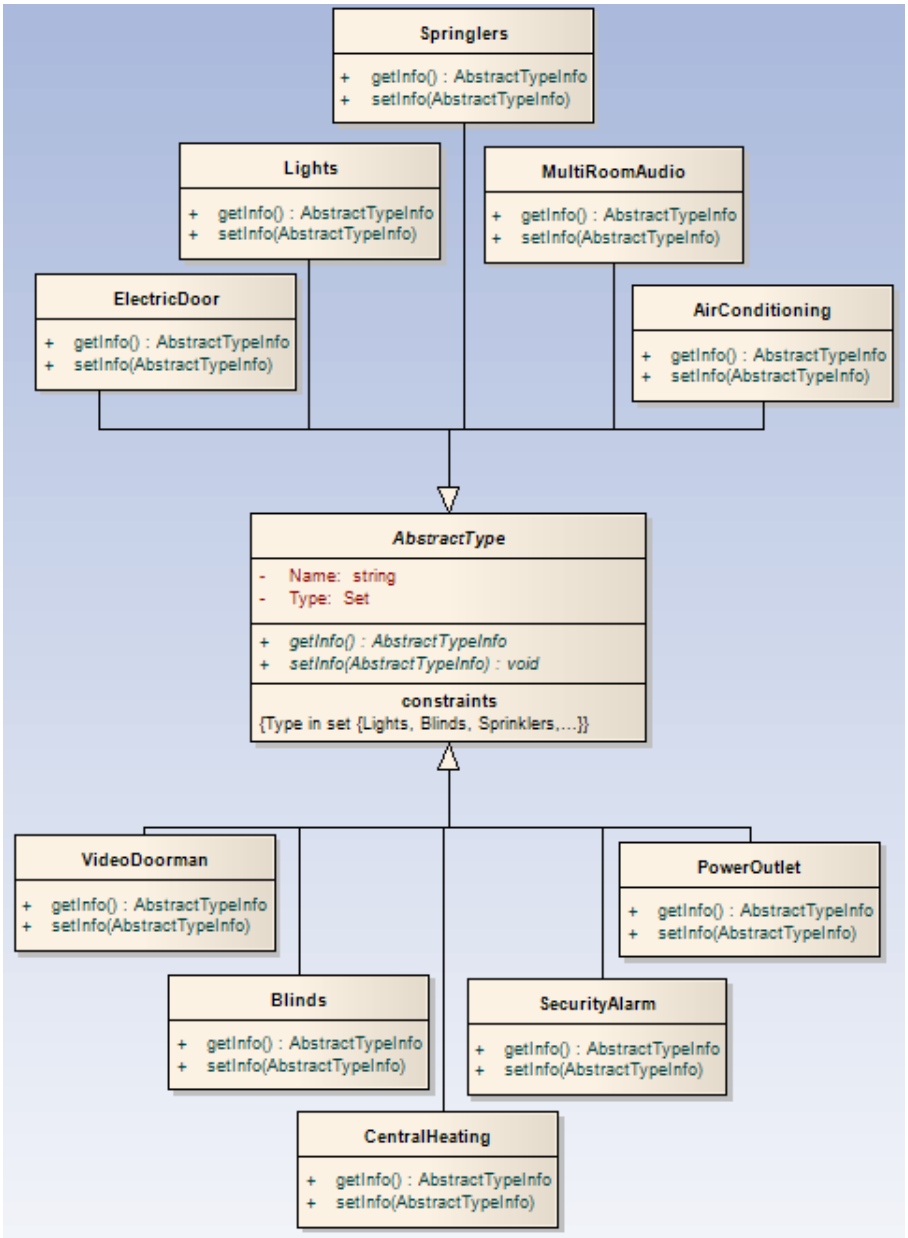

Figure 6.3: QuiiQ Automation Foundation devices Abstract Type class model.

A device can adopt several behaviors. This model (figure 6.4) implements several of them and also allows the design of additional suitable behaviors for each particular problem. The following list presents the behaviors already implemented in the system:

- **On** - This behavior is used in all the devices that have a binary behavior;
- **Off** - Exactly the same as the above, but it is necessary to discriminate this behavior because there are several devices that only have an On behavior, i.e. a macro device that only runs the macro;
- **setValue** - Some devices have more than a binary behavior, the status being represented as a range of values. This type is useful to represent devices like dimmers (0...100%), dimming blinds, etc;
- **Stop** - Internally, this as exactly the same behavior as the On, but it needs to be represented as a separate behavior because all the devices that have a stop behavior have another on to start the execution, i.e., blind;
- **setBlades** - In devices like an air conditioning systems there is the need to control the blades of the device. This behavior allows the system to control them;
- **setMode** - This behavior allows the interaction with the devices that have several modes, like an air conditioning system or a security alarm;
- **setFan** - All air conditioning or central heating systems have fans, being this behavior used to control this particular component.
- **setTemperature** - Temperature control systems usually allow the user to set the maximum and the minimum temperatures, being this behavior the one that adds such functionality;
- **setInput** - Typically multi room audio systems have several outputs and several inputs. In the device model, several devices, each one the output of a physical device, represent a multi room audio device. This behavior allows the user to redirect a certain input to an output;
- **setVolume** - This behavior is used to change the volume in radio or multi-room audio devices;
- **setStation** - A multi room audio device or a radio device, usually allows the user to change the station being played in a certain output.

This model is based on the *decorator* design pattern (attaching additional responsibilities to an object dynamically, providing a flexible alternative to sub classing for extending functionality), so beside the concrete components it is possible to use the decorator to increase its

functionality by, for example, grouping several concrete decorators.

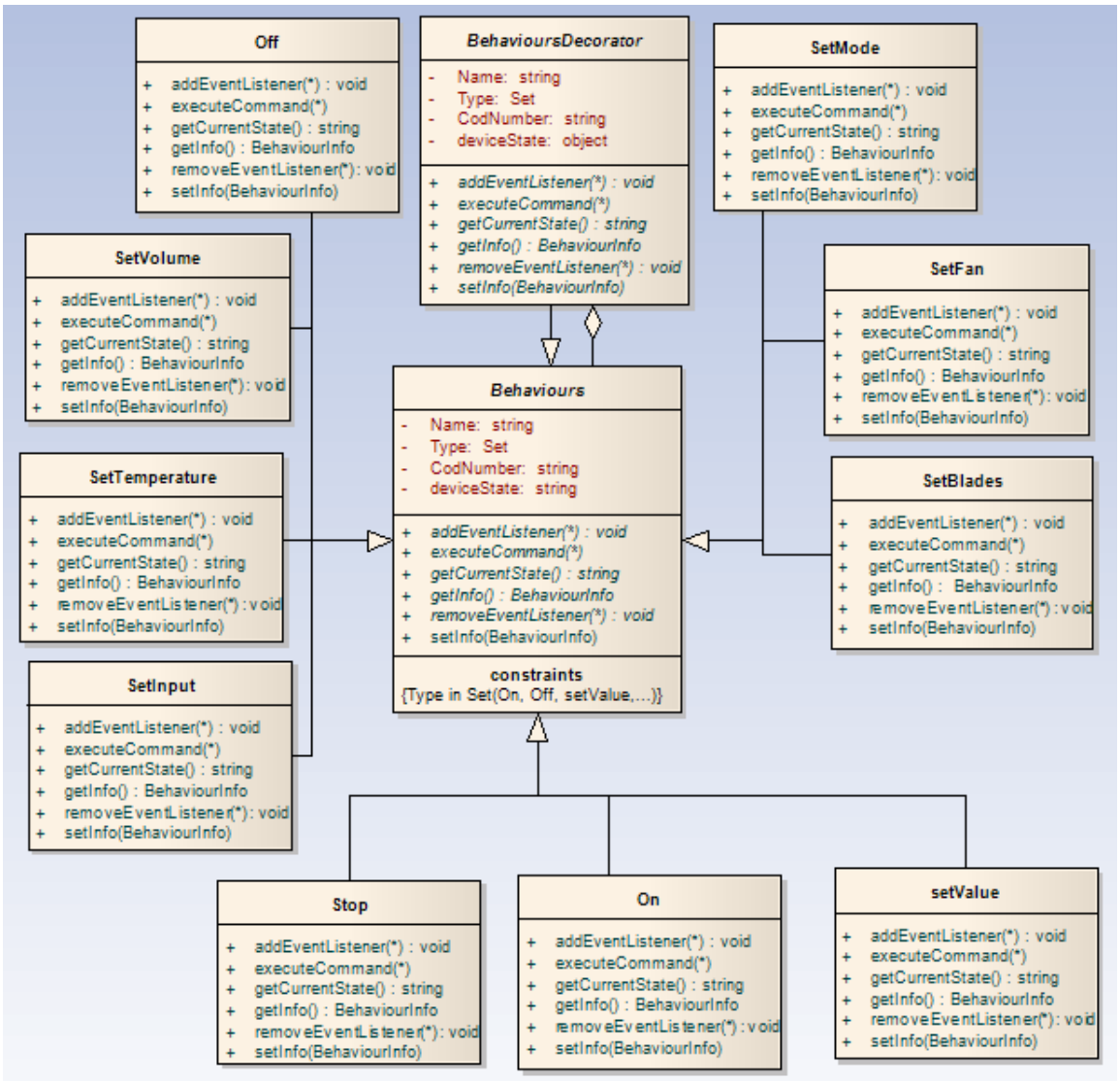

Figure 6.4: QuiiQ Automation Foundation devices Behaviors class model.

Figure 6.5 represents the previously described system. It represents the concept of device, representing the behaviors, abstract types, its cardinality and association with the device concept. Through the use of interfaces and abstract classes, we can easily expand all of these concepts, allowing the software architect and programmer to adjust these concepts and create new simple types to suit his particular needs.

There is one more concept represented in the diagram, that of multiple device, which allows the existence of a group of devices behaving as one. There is the additional restriction that the devices belonging to the same group (multiple device) must have exactly the same set

of behaviors.

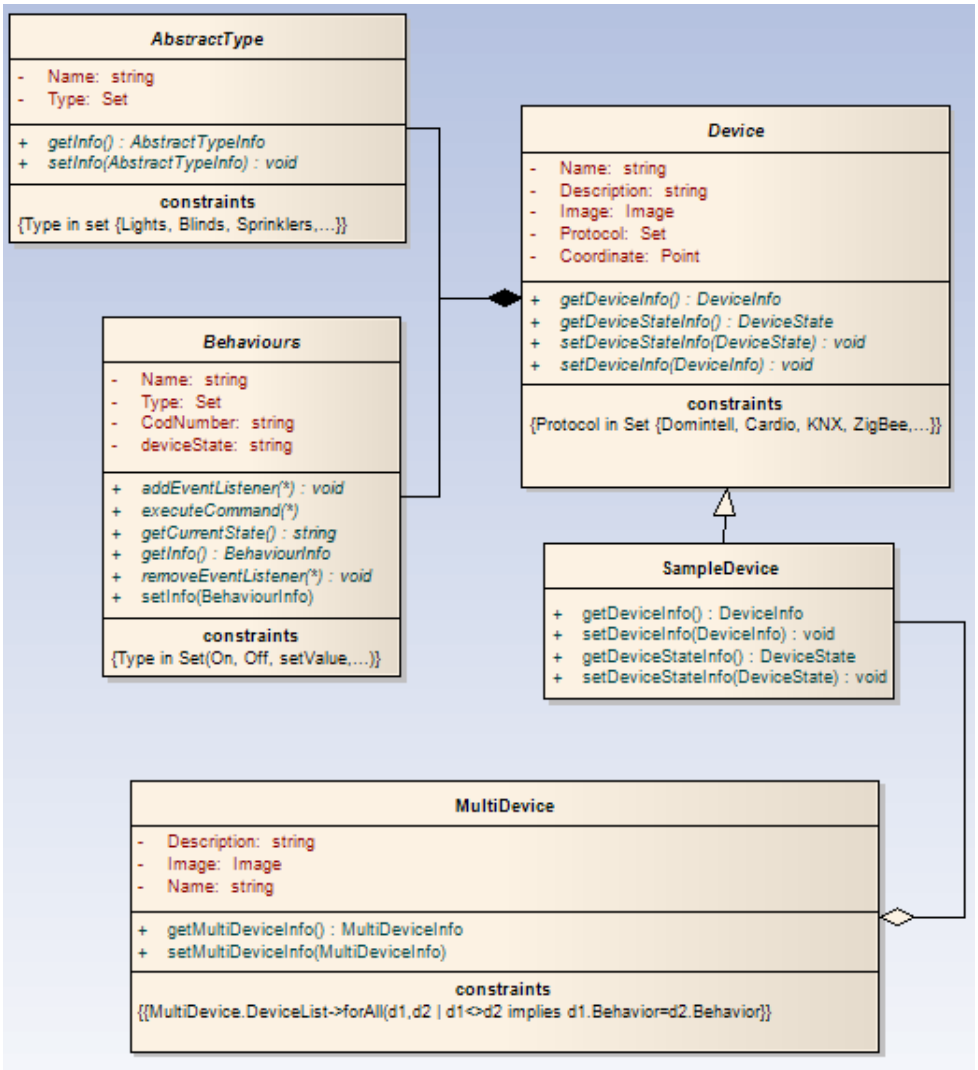

Figure 6.5: QuiiQ Automation Foundation devices class model.

With the concept of multiple device the physical network abstraction and representation is complete. All other concepts presented in this research use these concepts to organize themselves in the system and interact with the physical network. The following sections will present the ambient intelligence approach, giving the system an increasing degree of functionality.

6.4 Ambient Intelligence

The quality of life and enjoyment that a person feels heavily depends on efficiency, comfort and confines of the place this person calls home. Ambient intelligence is an idea that aims to provide an easy way of living, while the actual technology stays in the background, integrated in a non-intrusive way. Ambient Intelligence depends not only on the evolution of the different 'components' domain, but in the mechanisms that can be harnessed to achieve a seamless integration of all this components, and their convergence into one system [63].

In order to better achieve the Ambient Intelligence vision, there is a need to integrate all the technology in the house (this is only possible through the abstraction achieved and presented in section 6.3), but one needs to go one step further in order to give the final user convenience, security, enjoyability and for them to feel comfortable in their own house. The following sub sections will present some concepts which realize part of these objectives in several scenarios.

All of this ambient intelligence can be divided in two main groups, the basic concepts and the context-awareness. The first one, basic concepts, presents automated actions that are created by the final user to better suit their needs. There is no learning curve and the only interaction that exists are actions that have been pre-configured by the user and triggers which are pre-configured as well. The second topic, context awareness, takes the system a step further by integrating a context-awareness framework into the QuiiQ Automation Foundation. It will allow automatic routines to be dynamically created (for example an energy efficient routine can never be created by the user due to the huge number of variables implicated) and allow the presentation to the user of timely and useful information.

6.4.1 Environments

In their daily lives, sometimes consciously other times unconsciously, people set the lights to a certain intensity, the music to a certain volume, the blinds to a certain position... They create the environment where they feel good and comfortable. These actions are usually repeated on a daily basis, and it is possible to find a pattern in them, leading us to search for a concept in which all of these defined actions are already created and the user only needs the "push of a button" to execute all of them. This way of looking at automation allows definition/creation of environments, which can be repeatedly used with single click.

So, an environment is a pre-defined set of actions which are executed in a certain order (e.g., set a device to a certain state, play a music or a video, create a delay between actions,

etc.) creating the perfect environment for the user without any effort.

In figure 6.6, it is possible to observe this concept materialized in a class diagram. In its essence, an environment is a group of sequential actions characterized by a behavior and a state value. The behavior expected from an environment is simple and straightforward, if a user executes an environment ("pushes the button") all of its actions will be sent to the correspondent network communication service (QuiiQ Communication Foundation), where they will be executed in a sequential way (as presented in the UML activity diagram of figure 6.7).

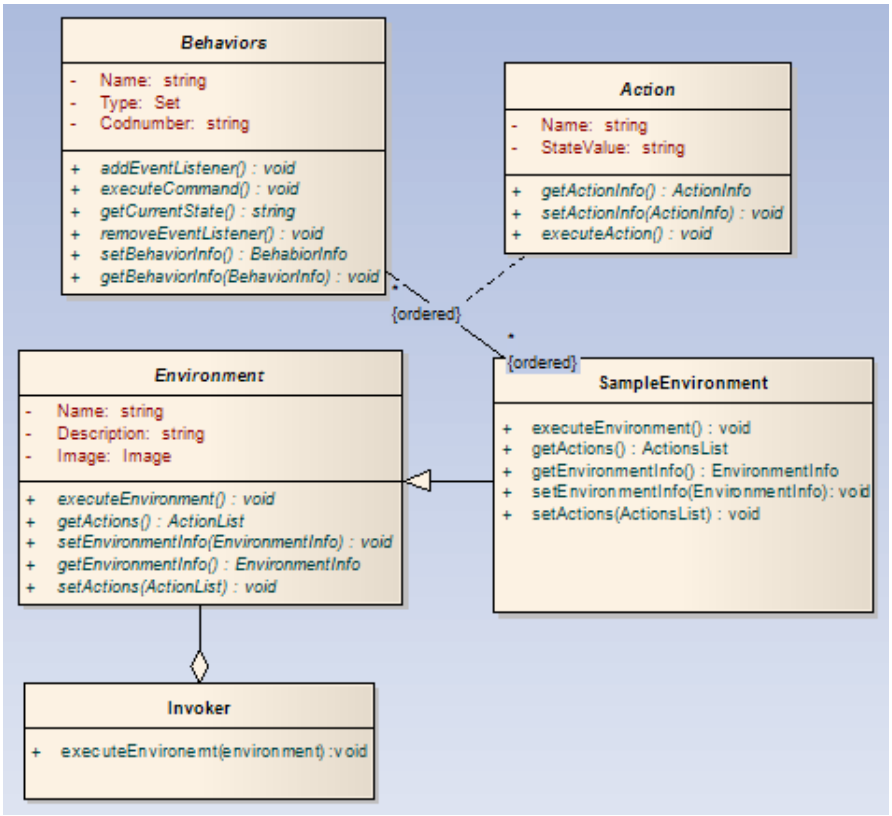

Figure 6.6: QuiiQ Automation Foundation environment class model.

The model for this concept is heavily based on the *command* design pattern (the command pattern is a design pattern in which an object is used to represent and encapsulate all the information needed to call a method at a later time). By using this design pattern, it makes it easier to construct general components which need to delegate, sequence or execute method calls at a time of their choosing without the need to know the owner of the method or the method parameters.

As referenced before, the execution of an environment should be sequential, where one action only executes when the previous command is successfully sent to the physical network.

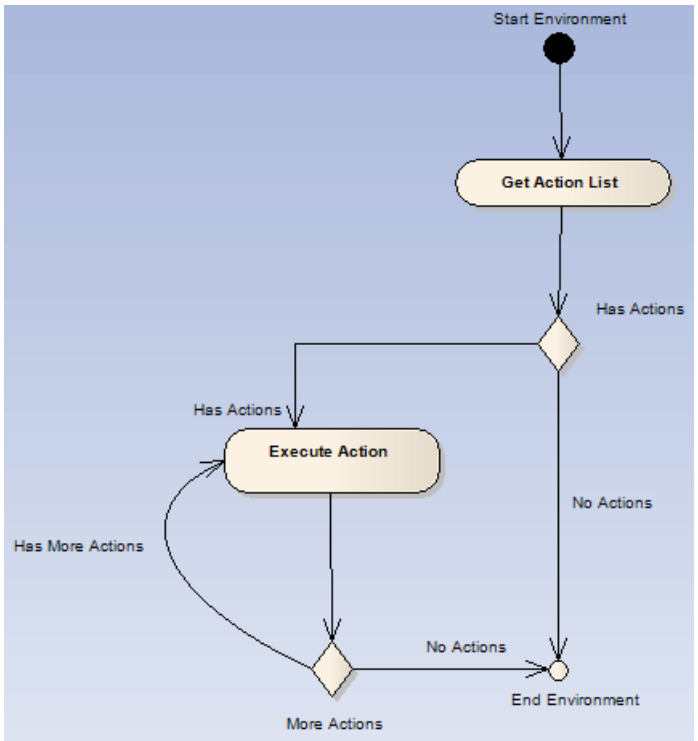

Figure 6.7: QuiiQ Automation Foundation environment activity model.

This allows the user to introduce delays between actions and really create an environment that suits its needs.

6.4.2 Routines

In the home automation context, a routine is *"commonplace tasks, chores, or duties which must be done automatically, regularly or at specified intervals; typically an everyday activity"*. It is a particular set of actions a user performs every day, under certain weather conditions or in some other particular situation (when one goes to bed, for example), and wants the system to go through it automatically. This concept was exported to the software world through the diagram presented in 6.8, a routine being the conjunction of a set of triggers with a set of actions. These triggers are logical expressions (propositions) which can be composed using the AND, OR and NOT operators over a certain type of environment information (i.e., day/hour, device state, temperature, luminosity, email received, IPod connected, etc.).

A routine action is the execution of an environment or a direct action over some behavior of a specific device. On verifying that the trigger conditions are met, each command will be sent to the physical home automation system (each command associated with an action), subject to the following:

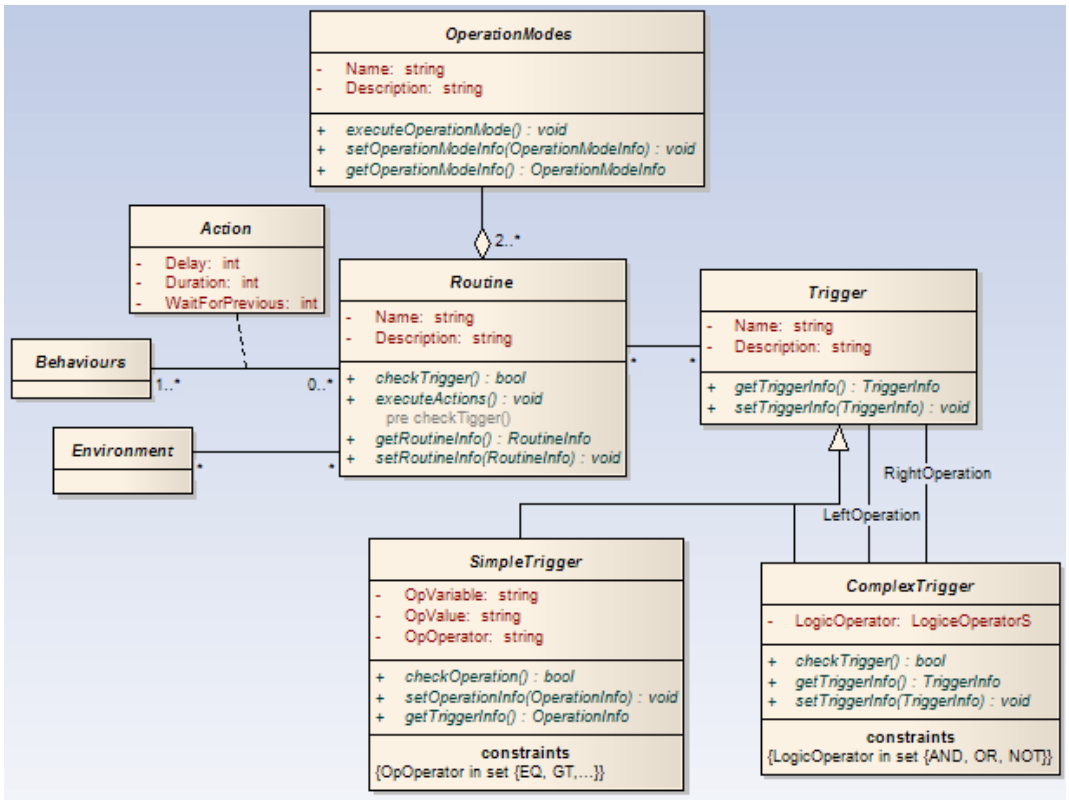

Figure 6.8: QuiiQ Automation Foundation routines class model.

- **Delay.** This attribute sets the amount of time the execution should wait before it interacts with the device;
- **Duration.** The amount of time a particular device is in the defined state. When the time period is over, the systems automatically sets the device to its previous state;
- **WaitForPrevious.** This attribute says that the current action should only be executed when the previous one as ended (not only the command has been sent, but the action executed and a reply received - for example a sprinkler only ends its action when the timer ends, which is particularly useful for watering the grass when we want a zone sprinkler to start only when the previously activated zone ends).

The most interesting part of a routine is the implementation of a trigger. A trigger is a logic expression which when validated as true, triggers a set of actions. Pursuing the sprinkler example, a trigger is: "(Temperature1>=20 && humidity1<10) || (DayOfWeek==Saturday && Hour == 13)", which means that when the Temperature registered by the device "Temperature1" is higher or equal to 20 degree and the humidity registered by the device "humidity1" is lower that 10, or its 13 hours and it is Saturday a set of actions will be executed, firing the sprinklers in turn minimizing water flow (due to capacity issues). To implement the triggers parser three classes were used:

- **Trigger.** The base of the trigger class, it initiates the parse and maintains the current logical value of the expression;
- **SimpleTrigger.** It parses a simple trigger (Temperature1>=20);
- **ComplexTrigger.** It splits the Logical expression, validating the left part of the expression (as another complex expression or a simple expression) and the simple part of the expression.

In its behavior, a routine is pretty much the same as an environment, except that all actions are executed in parallel (with delay periods and response acknowledgement or "wait for previous" eventually included) and they start automatically, i.e., without the user intervention. So, when the evaluation of the routine trigger (logic expression) is true, all the actions are executed (each action can go from a device action, playing music, execute an environment, etc). Figure 6.9 is a UML activity diagram representing the routine execution sequence previously described.

Furthermore, the concept of *operation mode* is introduced and presented in figure 6.8. It represents a wider concept than routines in that it is considered to be a group of routines. Following this interpretation, a routine is a particular situation that the user usually does and wants the system to do for him, but an operation mode is the definition of the house behavior

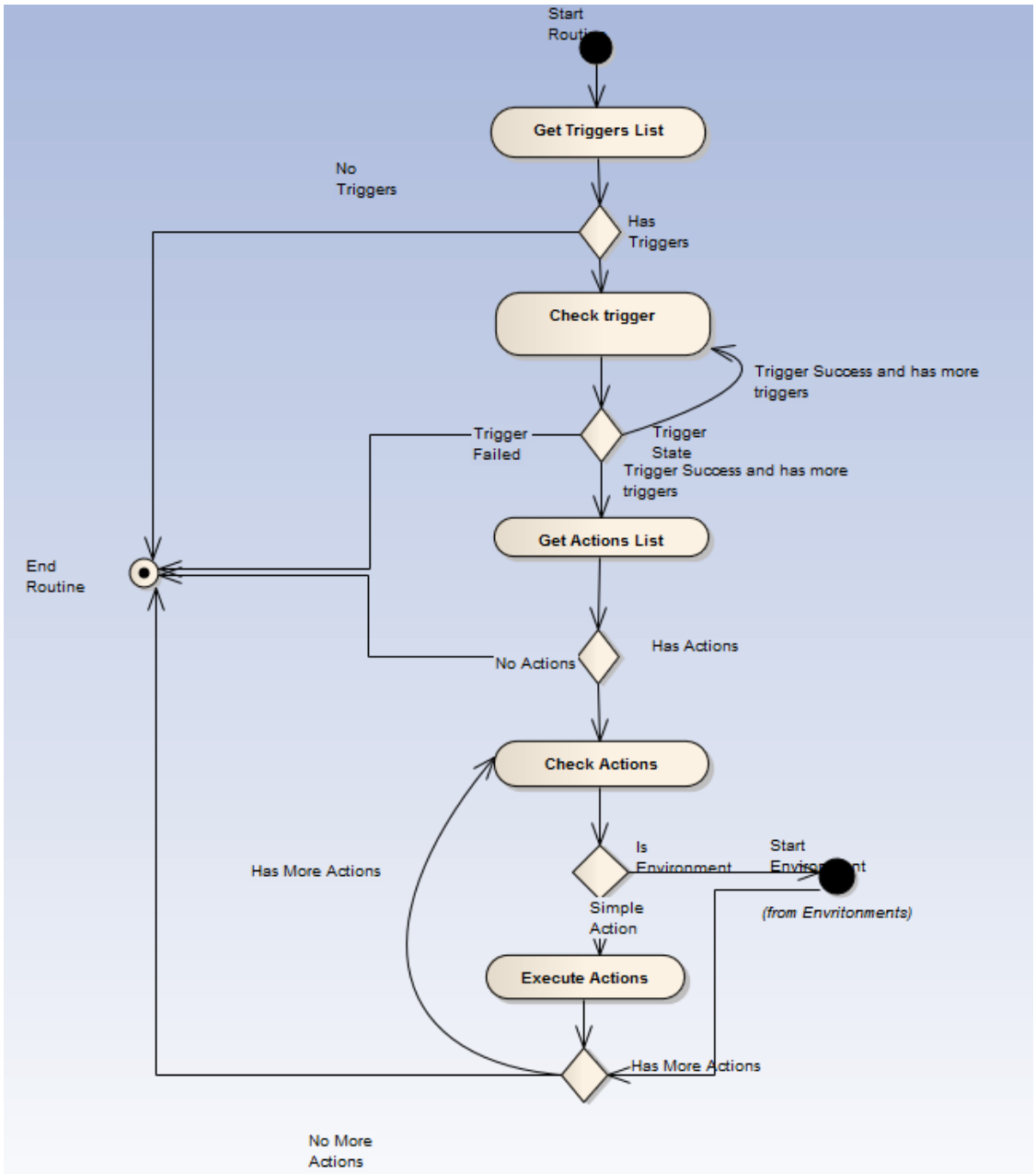

Figure 6.9: QuiiQ Automation Foundation routines activity model.

with one common objective, like for example, minimize energy consumptions, presence simulator, winter or summer mode, etc. without any user intervention. Just like the routines, the operation mode parameterization is fully performed by the user and is a set of routines that only react to predefined ambient states.

In section 6.4.4, a new approach to this matter is presented. It significantly increases the concept of operation mode by the introduction of context awareness.

6.4.3 Alerts/Alarms

Alerts and alarms implement the security subsystem of a home automation system. This system complements home security with respect to intrusions, fire, gas leak or flood, and technical problems within the home automation network. An alarm can receive messages of the following types:

- **Idle.** When a state is reported and no alarm or alert as been fired (arming or disarming the security;
- **Alert.** When an alert is fired by the system. An alert is an alarm type where no interaction by the user is possible and it is critical to the house security (e.g., fire alarm, carbon monoxide alarm, etc);
- **Trouble.** When there is something wrong with the system, and a technical intervention is needed (e.g., Phone trouble, communications problems, etc);
- **Alarm.** When an intrusion alarm is fired and the user can disarm the alarm.

These messages are received from the physical device when one or more zones are in alarm mode, which is, when there is some problem with the physical network, when there is an intrusion or even when there are technical alarms. All of the analyzed alarms receive messages of the previously defined types, but they can produce very different results.

Usually, an alarm will function in one of two ways:

1. **Arm each zone individually.** In these kind of alarms, each zone is individually armed and set to a certain mode; f
2. **Arm the entire house.** In this kind of alarms, all the zones are armed with the same mode.

In this context, a mode represents a set of physical sensors which will be active for each zone.

The class diagram presented in 6.10, allows the representation of these two types of alarms, but takes the first type a little bit further. The definition of Security Profiles, allows the user to have several security profiles to be armed in his house, for example the "Maid" security profile arms each zone in one of the modes, the "Full Protect" in another mode and the "My Profile" in yet another mode (imagine restricting the maid's access to certain sensitive areas such as the office, or disallowing visitors from opening the windows - this of course includes the possibility of a full lockdown or simply turning off all the alarms!). This allows the system to have different arming profiles for the alarm in an easy, comfortable way. The second type of alarm only allows the user to arm all the zones in a particular mode, so it does not support the detailed Security Profiles.

The zone defined as the 24H cannot have its mode changed by the final user, therefore being equal in all profiles. In addition, each zone has awareness of its own mode, allowing a user to, at the moment of activating a new profile, ignore the action to be performed under any specific zone, thus bypassing it.

When an alarm is raised (and a message is received by the system) the system should react accordingly guaranteeing in this way the house's security in the most efficient manner. For example, when a flood is detected, the water pump is turned off.

Figure 6.11 shows how the alarm should be armed when a security profile is being used (it must be supported by the physical alarm central), if no security profile is being used, it only sends one command to the alarm central.

6.4.4 Adding Context Awareness - The Context Toolkit

Due to the wide variety in context-aware frameworks under current research (see chapter 5 for more details) there seems to be no need to develop a new context-aware framework, the best option being the integration of an existing one. Utilizing the concept of device (see section 6.3.2), the component of context gathering and action execution is already present in the QuiiQ Automation Foundation. This way, the only change that needs to be made on the model is its extension for supporting the integration of a context-aware framework. There is no need to further develop the model for routine (see section 6.4.2), alerts/alarms (see section 6.4.3) and environment (see section 6.4.1) because they communicate with the network through the device component, and the context information is guaranteed by this component.

After examining several context-aware frameworks, a decision was made to select the Context Toolkit because the concept of device can be easily integrated into this framework,

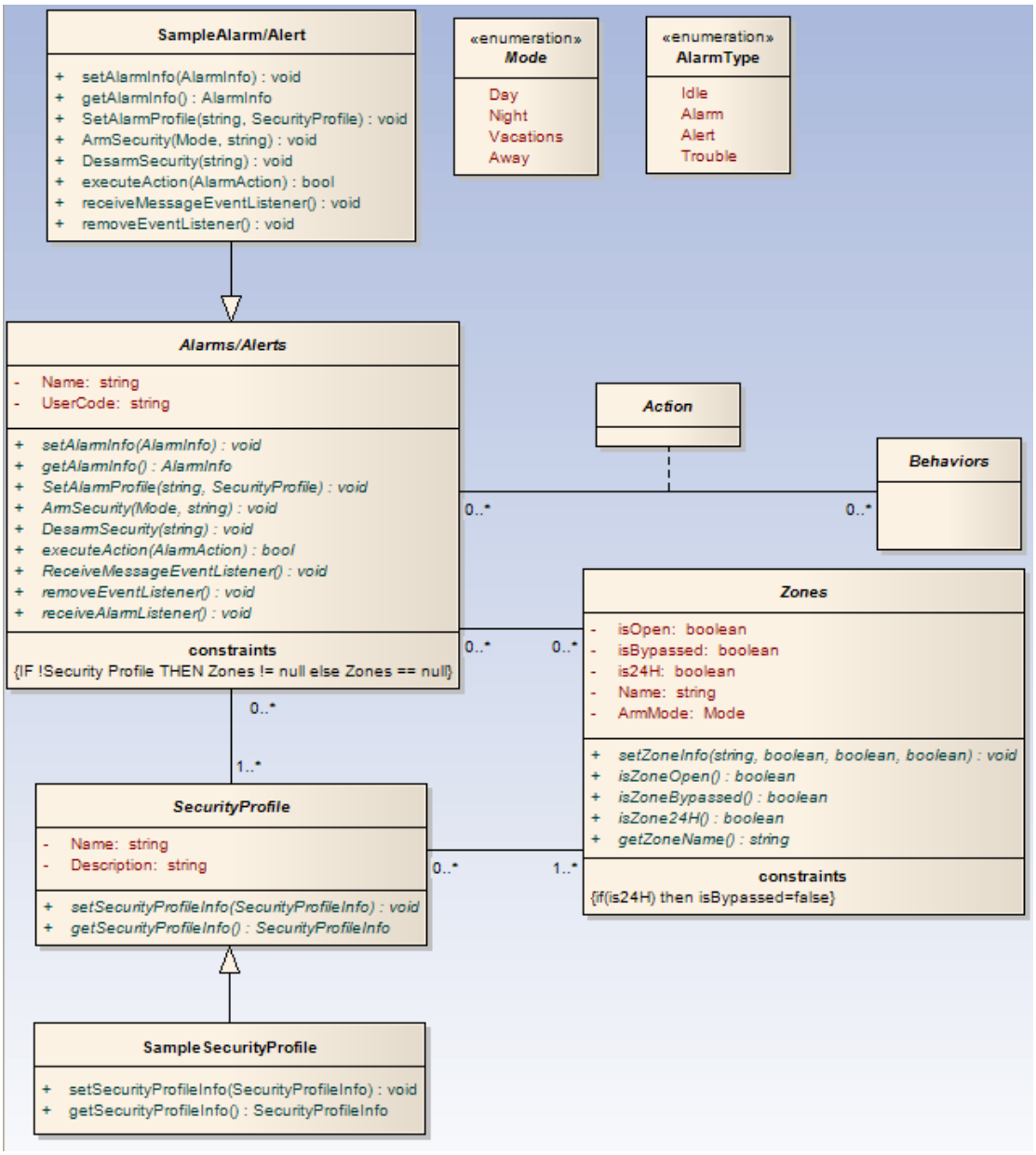

Figure 6.10: QuiiQ Automation Foundation Alarms and Alerts Class model.

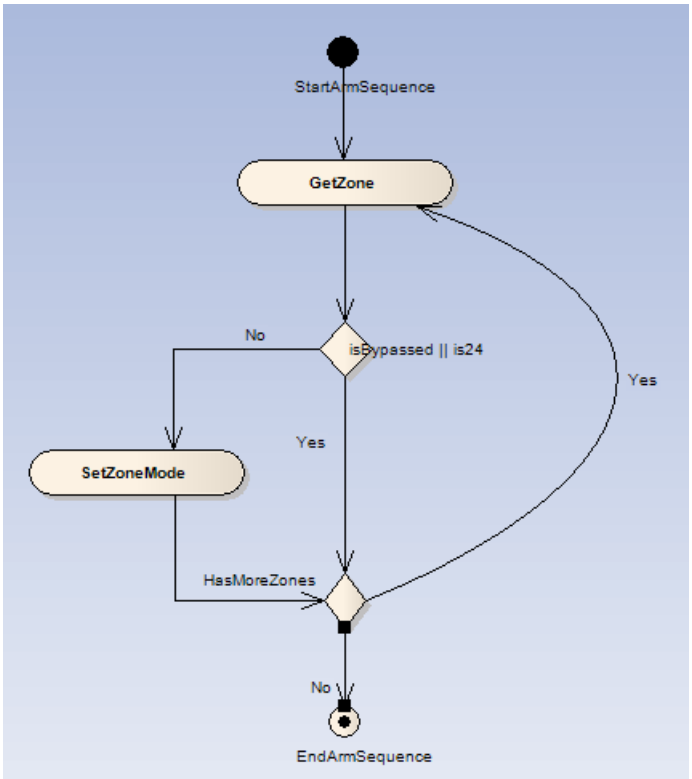

Figure 6.11: QuiiQ Automation Foundation Alarms and Alerts Arm Sequence activity model.

and some extensions have already been proposed for when it falls short of required specifications. In the following pages we will describe the Context Toolkit on a deeper level, and the integration between this architecture and the theoretical concepts defined in the QuiiQ Automation Foundation.

6.4.4.1 Introduction

The Context Toolkit was developed by three researchers: Danel Saber, Anind Dey and Gregory Abowd from the Georgia Institute of Technology in the USA for 5 years and it was firsts presented in [46]. The toolkit represents the description of a formalism to describe context and its abstractions. The context is represented by data acquired from context widgets (6.4.4.3), which encapsulate the communication with sensors and transform raw data into a format that the system can understand. The context can then be combined from several widgets in an aggregator or be translated by the context interpreters. The toolkit allows the system to maintain historical data in its widgets and enables subscription and discovery mechanisms so that this information can be easily used, and appropriated components can easily locate it.

The Context Toolkit is currently implemented in JAVA, but there are several examples and components implemented in different programming languages such as C#, VB.NET, Python, C++, etc. This framework provides several features to support the development of context-aware systems such as [46]:

- Context Specification.
- Separation of Concerns and Context Handling.
- Context Interpretation.
- Transparent Distributed Communications.
- Constant Availability of Context Acquisition.
- Context Storage.
- Resource Discovery.
- Privacy and Security.

The toolkit employs the key-value context model required for defining and storing context data. Attribute-value pairs are used to describe the context information and they are utilized for service discovery by matching the required data according to these pairs. This section presents the Context Toolkit describing several modes of operation and introducing its architecture, its main components and its way of facilitating communication between components. Finally, it will identify some strengths and weaknesses of this model and its integration with the home automation fundamental concepts presented and discussed in this research.

6.4.4.2 Application Design

There are several ways of designing a context-aware system depending of several parameters like number of users, location of sensors, device capabilities, etc. The Context Toolkit encapsulates three different design principles for handling context and building applications.

The first approach is direct sensor access as shown in figure 6.12. This approach allows a direct access to the context widget without the use of any additional layer for accessing and processing data. Because of the way in which the subsidiary applications and sensors are coupled and hardwired into the application, this approach does not result in good software practices and imposes on the developer the problem of dealing with complex context acquisition, resulting in poor reusability that leads to rare utilization.

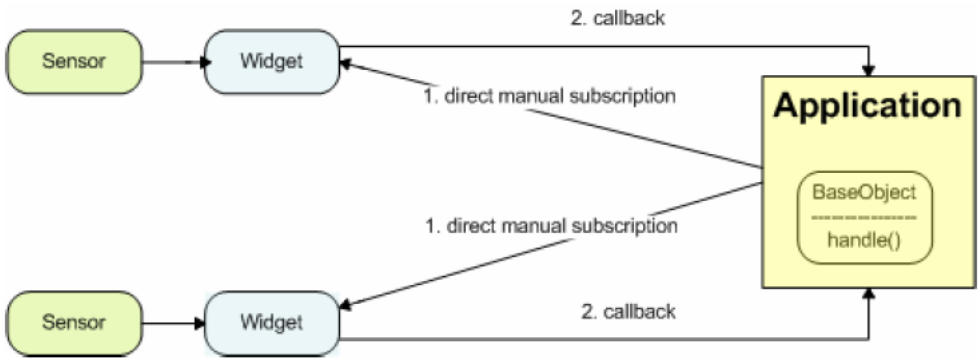

Figure 6.12: Direct sensor access - Traditional design without a discover [42].

The second approach uses the middleware infrastructure principles and is presented in6.13. In this approach the user interface, the business logic and the sensing infrastructure are separated. This layered architecture, which enables hiding the low level details, is accomplished by the use of the discoverer, which acts as registry where all the system components are registered. Applications subscribe to the discoverer to dynamically locate the widgets they communicate with. The automatic discovery of widgets is provided, the sensing and application layers are divided, and access to the remote data sources by multiple clients is enabled.

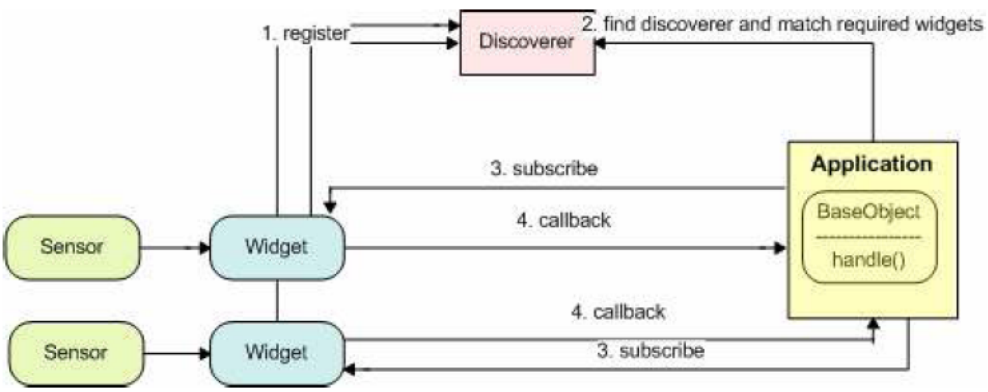

Figure 6.13: Middleware approach - Application design using discoverer[42].

Later in [64] Newberger and Dey discuss the issue of monitoring and controlling context-aware applications by introducing the concept of enactor. The Enactor architecture, presented in figure 6.14, reduces the designer's access to the application state and behavior, and allows developers to easily encapsulate application logic in this component. Enactor Subscription Manager manages the Context Toolkit Subscriptions by notifying all enactors and subscribing the widgets based on the enactors description.

Although a context-aware application can be placed under only one enactor which manipulates all the context data, the use of several enactors, all encapsulating and processing different kinds of information, enables sustainable modularity and increases its reusability.

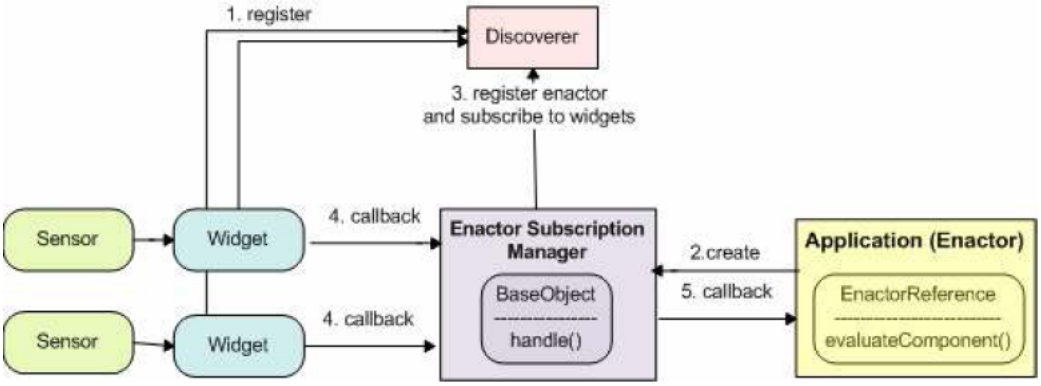

Figure 6.14: Enactor approach - Application design using Enactor [42].

6.4.4.3 Architecture

The context Toolkit framework uses an object-oriented approach and it consists of several abstractions that are presented in figure 6.15.

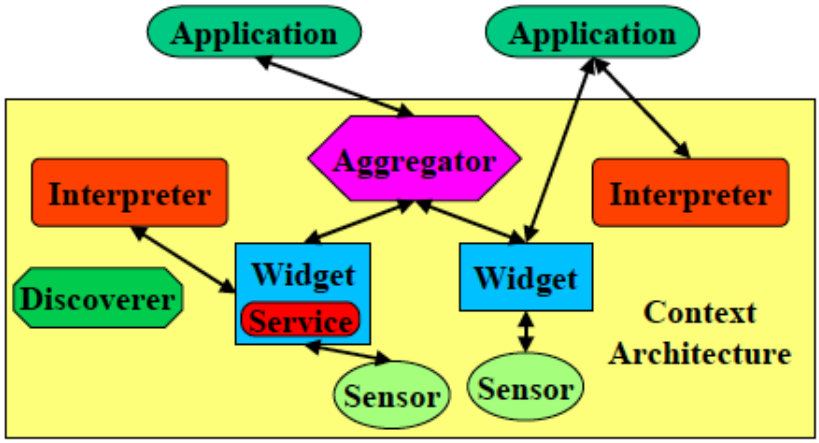

Figure 6.15: Sample Configuration of context components. Arrows indicate data flow [46].

The relationship between the different elements of the system is presented in the following example (in figure 6.15): two widgets, an aggregator, two interpreters and two applications. The sensors acquire the data and provide it to the widgets, being this data further translated by an interpreter to comply with a form that is intelligible to the application. The acquired data

from several widgets can be aggregated into a single component (Aggregator), which extracts and combines the logic for a certain entity. Finally, this context is delivered to the application, which further manipulates it and develops the application logic.

These elements are presented in depth in the following sections.

Widgets A context widget is a software component that provides applications with access to context information from their operating environment. The widgets provide several benefits to context aware applications with the following being the most important [46]:

- The separation of concerns by hiding the complexity of the physical sensors used by the system;
- They abstract the context information (by parsing raw data) to suit the expectations of the application;
- They provide access to context data through querying and notification mechanisms available through common interfaces;
- They provide reusable and customizable building blocks of context sensing.
- They will provide context data on demand or supply it when a change has occurred.

The context widgets have a state and a set of behaviors. The widget state is a set of attributes that can be queried by the application. Each widget class defines a set of constant and non-constant attributes that define the context provided by the sensor and callbacks, which are further delegated to the widget's subscribers. They also contain the details about how to connect and query data from the sensor.

Aggregator The aggregator provides the framework with the ability to deliver a specific context information to an application, by collecting related context about an entity that the this application is interested in. It can gather contextual information about a specific entity, and directly provide information related to it or build higher-level context objects. These components alleviate the developer's work because they allow the application to subscribe to a single object that encapsulates the logic of several sensors, making it easier to infer higher level context as well.

To build an aggregator, the developer simply has to specify the entity type and identity. The entity type refers to whether the aggregator represents a person, location, or a device. This component inherits all the methods, properties and callbacks that the widgets have, and

therefore can be subscribed to and pooled. Beside this, new callbacks can be added to an aggregator in order to increase its features.

Interpreter Interpreter transforms context information by raising the level of abstraction. These components abstract low-level raw data or low-level context information into richer and higher level forms of information. Traditionally interpretation is performed by applications, but by introducing interpreters this functionality is brought out of the application level, allowing the reuse of interpreter data by several applications [46].

When creating an interpreter, a developer must provide the designs which are specific to the instance being built. The developers should supply the input and the output attributes and an implementation of the InterpretData() method that effects the translation.

Services The concepts of widget and service are very similar, the main difference being that while a service supports an output (they can execute actions in the behalf of the framework), the widgets provide only input (they receive inputs from the environment). The services implement the same specifics on how to execute the action and how to communicate with the actuator, and this information encapsulated in the component and hidden from the application.

These services can be synchronous (when an ACK success message is sent back to the service after the action execution) or asynchronous (the service notifies the application and feedbacks about results when they are available). To implement a service, the developer must provide a description of the service functionality, the set of parameters, if any, and the code that will interact with the actuator [46].

Discoverer This component is used to locate context components that are of interest of the application as a form of resource discovery. In this way, the application does not need to know where the components are located before it needs them, allowing applications to better adapt to change in the context sensing infrastructure. It contains a registry of what features are available on the platform, maintaining a registry of all the widgets, aggregators, interpreters and services that can be used by an application. The discoverer inherits the widget's class attributes and methods and its basic functionality, therefore widgets can publish, query and subscribe to notifications on this component.

The applications queries a discoverer by using one of two approaches: by choosing a name or Id; by choosing its attributes. When a widget is created it registers itself with the

discoverer (by using multicast) by having the discoverer store its description. So whenever an aggregator or application needs to find and contact another contextual component, they can query the discoverer and match the component by a set/subset of description parameters.

Communication Infrastructure The communication between the different system components is implemented on top of TCP/IP. This approach allows the support of a distributed system and the use of custom built supports such as: wearable computers, handheld computers, mobile phones, custom made sensors, home automation networks, etc.

All the framework components (widgets, interpreters, aggregators) extend the class BaseObject (presented in section 5.5.4). This class encapsulates the communication logic in a distributed environment and all these subclasses inherit its functionality for communicating with the rest context architecture. By default the BaseObject (presented in section 5.5.4) supports HTTP and XML. HTTP is used to send and receive messages, while the XML is used as a language to describe the data being sent. XML and HTTP are chosen because they support lightweight integration of distributed components and enable access to heterogeneous platforms with multiple programming languages. The BaseObject (presented in section 5.5.4) can be extended to support other communication protocols and data structures like CORBA or Java RMI [42].

6.4.5 QuiiQ Automation Foundation & Context Toolkit

These are two different frameworks that can be easily integrated and used concurrently, increasing system functionality. To better understand the integration between the Context Toolkit, the QuiiQ Automation Foundation and the application design guidelines, it is necessary to analyze and suggest some design principles to be followed by the system developer in order to achieve the best system performance and functionality range.

6.4.5.1 Multi-Layer Architecture

As mentioned in the previous chapters, the use of a multi-layer architecture improves the system's flexibility by decoupling the drivers and sensing infrastructure (QuiiQ Communications Foundation), from the abstract system middleware (QuiiQ Automation Foundation), from the system user interfaces (QuiiQ Presentation Foundation). Internally, the QuiiQ Automation Foundation also uses a layered architecture by abstracting the physical devices in a first layer, and providing ambient intelligence support (Environment, Routines, Alerts/Alarms

and Context-Awareness) in a second Layer.

Regarding the context-awareness component (Context Toolkit Framework), it should be divided into three layers: context Provider, Context Interpreter & Aggregator and application Logic (Table 6.1).

Table 6.1: Layered Architecture approach.

Application Logic
Context Interpreter & Aggregator
Context Provider

The context provider follows the widget model and standardizes context acquisition and its decoupling from the sensing infrastructure by applying software components based on widgets. The QuiiQ Automation Foundation already contains this abstraction (used in the higher level concepts and, in the future, to represent the information in the QuiiQ Presentation Foundation) that by extension can be easily integrated with the Context Toolkit. All retrieved data from context providers can then be sent to the context manager (this mechanism is implemented by the Context Toolkit) for further processing. The context interpreter & aggregator layer uses aggregators and interpreters to interpret higher level information and deduce new context from the basic context supplied by the context widgets.

The main components that should be implemented by the system developer are:

- **Widgets, Aggregators, Interpreters.** The main building blocks for the first layer are the widgets, aggregators and interprets provided by the Context Toolkit. The concept of device is extended to have widget like features, and in this way the system can be easily integrated with the context-aware framework. All physical devices are abstracted in a device that through the QuiiQ Communication Foundation encapsulates the communication and driver details. The aggregators allow the use of higher level context data by grouping basic context information or transforming it from one set to another with the help of the interpreters;

- **Sensing Infrastructure Manager.** (Discoverer) This component manages physical resources and keeps track of the state of all the widgets such as their description, location and capabilities;

- **Storage.** This component is responsible for storing the historical data, keeping record of chronological sequence and changes of data retrieved from each widget. The analyses of this data allows the detection of patterns to predict future behavior. The context can be stored using several technologies such as XML Datasets, SQL Databases or text files. The use of SQL databases brings some advantages regarding the performance

of historical context retrieval, alleviating the process of data analysis by allowing the construction of complex queries with minimum querying time.

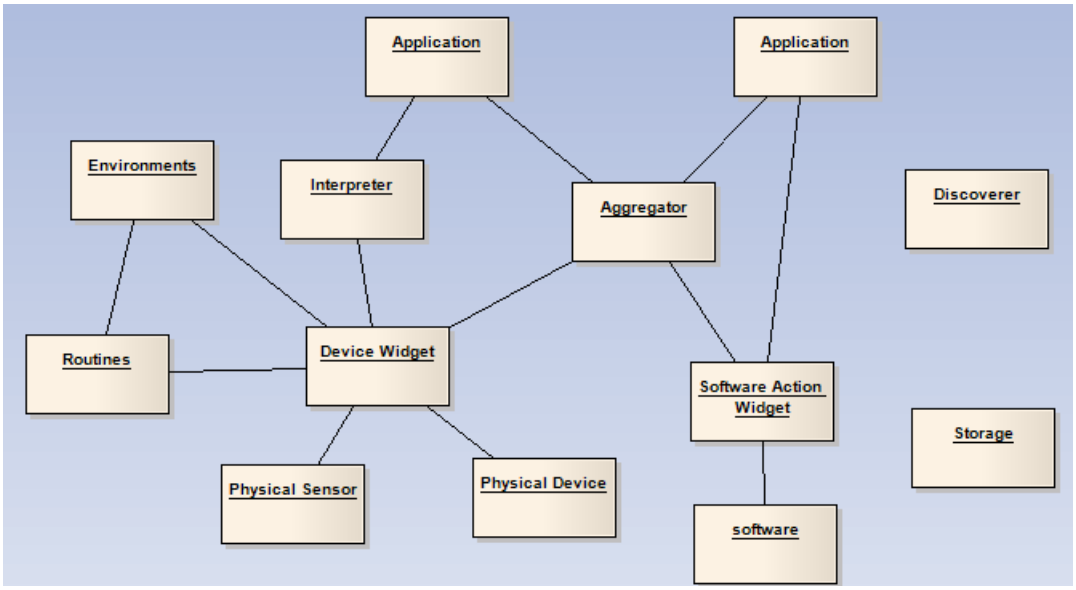

Figure 6.16: Sample Architecture for a QuiiQ Automation Foundation with Context Awareness.

Figure 6.16 represents a possible architecture for a home automation system using the QuiiQ Automation Foundation and the Context Toolkit. It is only representative of the idea for a system of this type, where all the previously described components are presented and the concepts of device as widget, routine, environment and context-awareness are present and related.

6.4.5.2 Widget Architecture Summary

QuiiQ Automation Foundation already abstracts and converts the raw data provided by the physical sensors, and uses it to report information to every component in the system, so this will be a good point for integration between the QuiiQ Automation Foundations and the Context Toolkit, providing the final architecture with context aware capabilities. By extending the concept of device (as represented in figure 6.17, it is possible to convert it into a context widget allowing it to provide context information to the context-aware system. A lot of other widgets can be developed by the system developers (e.g., using the SOAP technology and web services to access information, or by subscribing RSS feeds, etc.) creating in this way the best system for a particular situation.

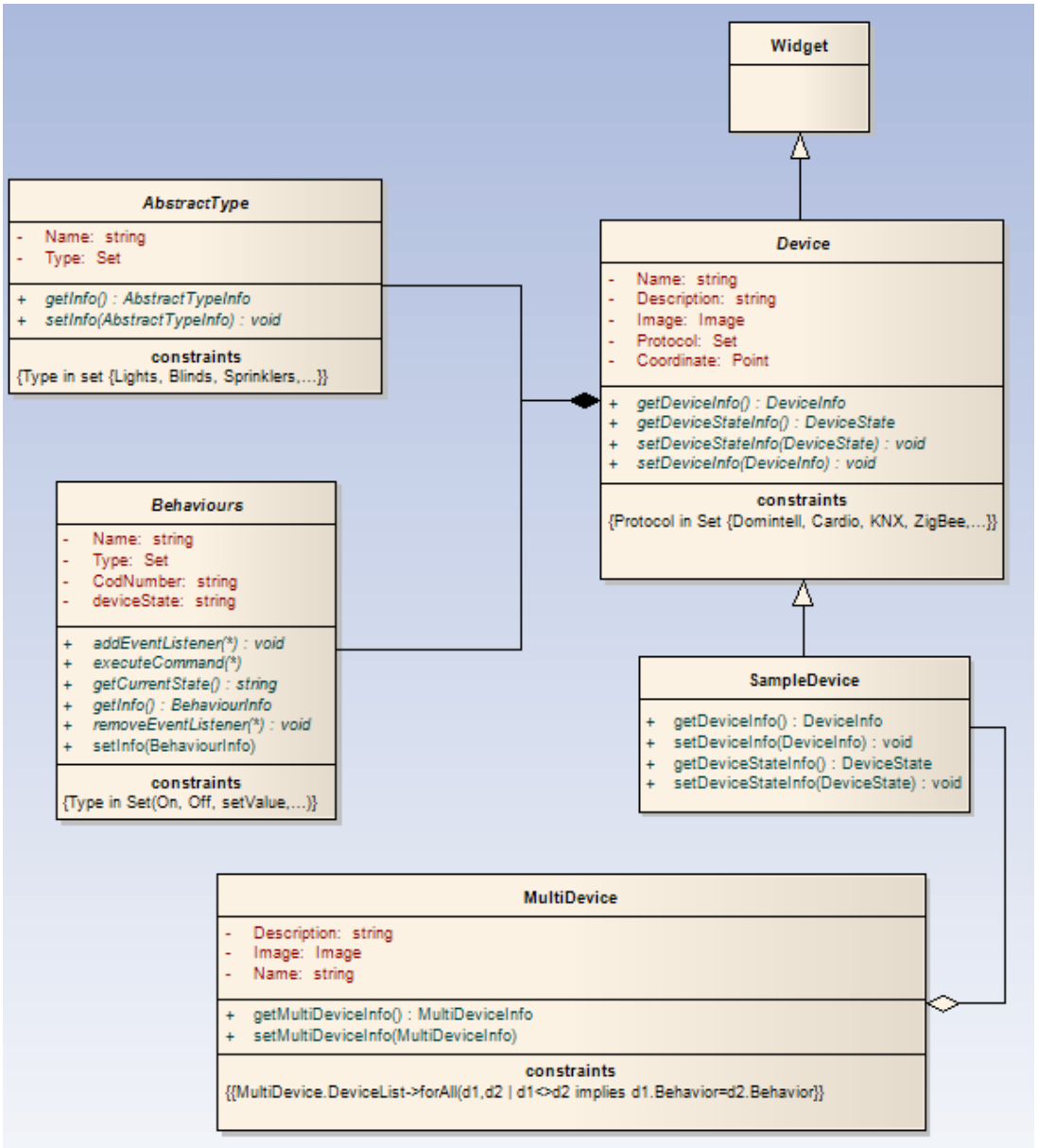

Figure 6.17: QuiiQ Automation Foundation devices class model.

A basic Widget that subclasses the BaseObject (presented in section 5.5.4) is already present in the Context Toolkit, handling all the features that are common across widgets. It is responsible for maintaining a list of subscribers to which each widget must respond, store all data that is sensed, mediating access to the widget's services and allowing inspection of the widget's internal information [46].

A widget automatically inherits (from the BaseObject) all the communication functionalities, being able to act as a client and as server. When a message is received, the BaseObject tries to handle it. If it is unable to handle the message, the widget will handle the message and respond. A widget can handle the following types of messages:

- **Widget Inspection.** These messages allow the system to inspect the widget capabilities;
- **Widget Subscriptions.** This type of message allows a component to subscribe to the widget context information;
- **Widget Storage.** By default a widget stores its own data, without the need for any code from the system developer, allowing other components to access its historical information;
- **Widget Services.** Allows the request for the widget to execute an asynchronous or synchronous service.

6.4.5.3 Extensions

Ana Hristova in her masters thesis [42] proposed some extensions to the Context Toolkit that addresses some of its weaknesses. She presents new concepts such as the Quality of Context, an improvement in the Resource Discovery mechanism, an ontology based model and inference Engines and user behavior analyzer.

- **Context Provider Registry.** The context provider registry keeps record of all the context providers that subscribe to it;
- **Quality of Context Manager** This component evaluates the captured context data and assesses it on several parameters. It takes the application service request and analyzes the demands referring the quality of the context and processes the context information to meet these requirements;
- **Conflict Resolution Engine** After the execution of the Quality Context Manager, contradictory values can be detected, captured from different physical devices that measure the same parameter. The Conflict Resolution Engine provides a way of dealing with the

ambiguity of the context and makes a decision regarding which sensor to use. These preferences can be based in user decision, quality of context and analysis of historical data;

- **Inference Engine** In order to discover higher level context information, the Inference Engine, infers new context information based on the knowledge obtained from the sensors and the historical data stored by the system. In order for these component achieve the best performance possible, an upgrade to the key-value context model should be made and the use of ontology's to describe context information is necessary;

- **User Behavior Analyzer** By using a sub-area of machine learning, reinforcement learning, the system can automatically track user feedback, and modify system behavior according to it. If a user turns on a light and then turns the light off for a longer period of time, the later might be considered as a preferred and switching to the other might be considered a negative feedback. The analyzer registers the user's behavior and its interaction with the system and improves the system's performance increasing user satisfaction;

- **Privacy and Security Module** By providing authentication, the system guarantees that only the registered users/components have access to the context information. Another functionality to guarantee the privacy of the context information is to provide an access control mechanism that assigns a owner and a set of permissions to a component and restricts the access to information depending on the access level the user has.

Using this approach a new layer is created between the sensing infrastructure (Figure 6.16) and the application.

Figure 6.18 introduces a new layer in the previously presented architecture, integrating new concepts that will significantly increase the system's features and system performance.

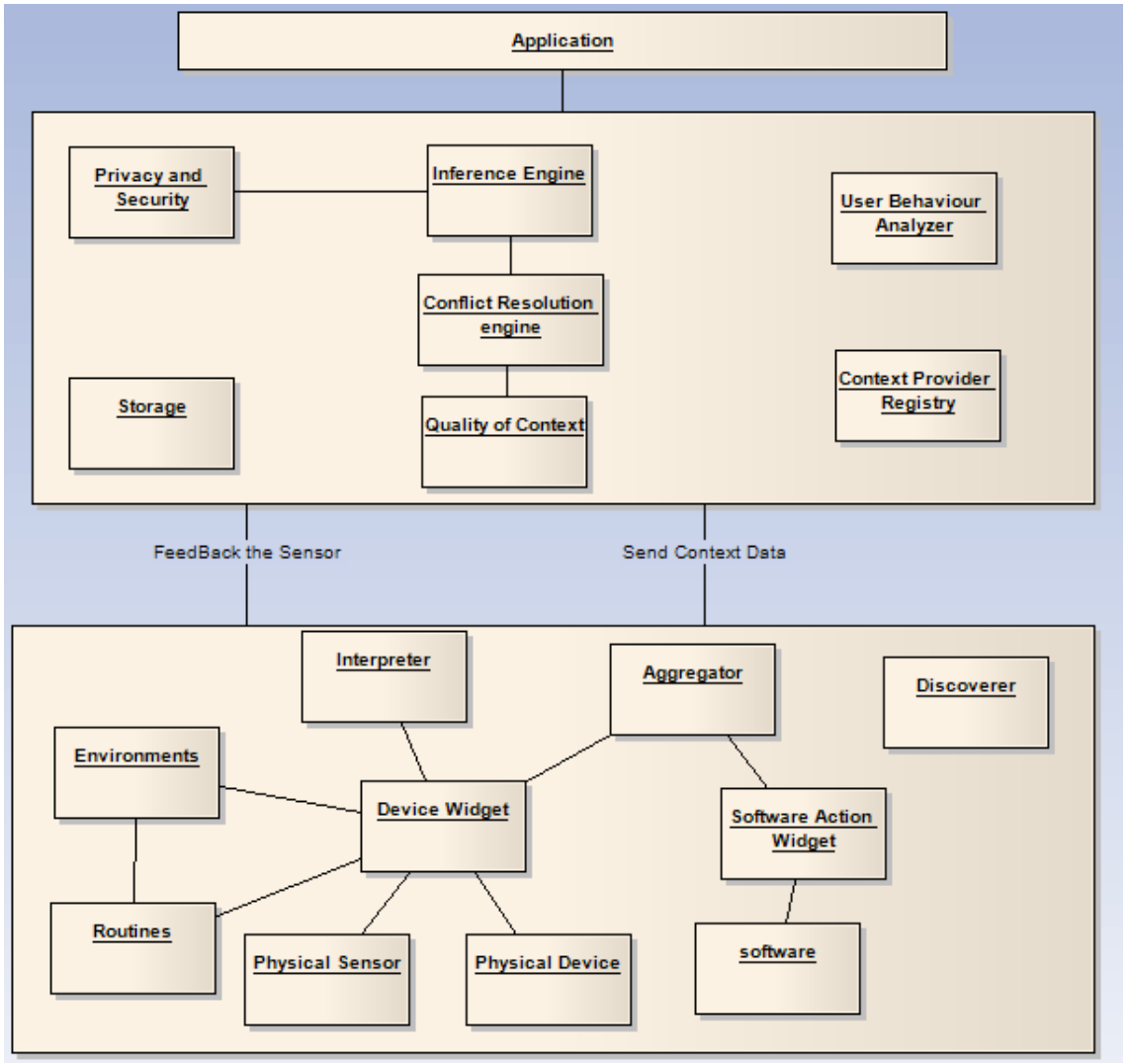

Figure 6.18: QuiiQ Automation Foundation devices class model.

Chapter 7

Conclusions and Future Work

QuiiQ Automation Foundation is a considerable improvement over current practice for addressing several problems about home automation software development. QuiiQ Automation Foundation offers a complete new outlook on home automation and through the abstraction of home automation's basic concepts (division and device) allows the development of entirely new concepts of ambient intelligence in a easy, scalable and adaptable way.

It is an evolutionary approach since it is based on existing knowledge, practices and concepts, thus enabling a smooth adaptation by application developers. Allows the development of home automation applications in a fast and easy way, adapting each to a particular reality and allowing implementation of an all new ambient intelligence concept to better suit the final user's needs.

From the preceding work, and based on the merits of other frameworks, it can be envisaged that the QuiiQ Automation Foundation approach should help to reduce the development time of home automation applications, providing higher level concepts and a smooth integration with a context-aware framework, the Context Toolkit.

This chapter presents the main contributions made on this dissertation and outlines future research.

7.1 Research Summary

From a home automation perspective, the main result coming out of this research is the contribution to the definition of higher level abstract concepts which allow the representation of several devices that work on top of various home automation protocols. It proposes some

ambient intelligence concepts (like environments, routines and operation modes) which reduce the effort expended by the final user to execute everyday tasks, and define particular environments, for example, to simulate human presence in the house. It presents a survey on context-aware frameworks, and defines a method for integrating the Context Toolkit and the QUiiQ Automation Foundation. By using a context-aware framework some intelligent decisions can be made, such as to place the house under a reduced energy consumption mode, or automatically maximizing user comfort.

Another important contribution to the home automation software development industry is that it presents all the previously mentioned concepts as a framework proposal which will, in the future, integrate with the QuiiQ Automation Framework and allow the development of home automation software while expecting considerable savings in time and money. When finished, the QuiiQ Automation Framework will allow a company, for example, to develop generic intelligent home software, and another company to develop the driver for its particular protocol, allowing several companies to use this software on top of their own protocols.

Now, going back to the beginning of this dissertation, we proposed three research questions:

1. *what are the main theoretical concepts directly present in home automation systems?* This question was completely answered by defining the concepts of division and device. This study began in chapter 4, with the definition of software variation and the representation of the physical identification, the functional identification and the space identification properties. Next, it was materialized in the models present in section 6.3 (based on the properties previously defined, it is possible to represent the entire home automation network);

2. *how can we create new concepts and adapt the existing ones in order to give added value to the system?* By analyzing people's way of living and interacting with the technology, and by identifying one's needs and wishes, some concepts were defined which work on top of the division and device concepts. These concepts provide the system with some kind of ambient intelligence; for example an environment, which allows the user to define some specific ambiance (a sequence of actions that is executed with the "push of a button"), and use them to increase his comfort and enjoyment. Routines allow some basic repeatable actions to be performed automatically (based on the existence of several triggers - date, state of other device, etc.), while the operation modes allow the definition of more complex routines to, for example, provide presence simulation. Finally, the alarm allows the use of an intelligent security system, providing not only the use of simple alarms (where all the zones are armed in the same mode), but the definition of security profiles, in which each zone has its own security mode. All

of these concepts are essential to the user's daily life and are completely abstract, and therefore completely protocol independent, with their models being defined in sections 6.4.1, 6.4.2 and 6.4.3;

3. *how can we give some behavioral/functional "intelligence" to the system?* This is partially fulfilled by the concepts previously presented on environment, routine and operation mode (see sections 6.4.1 and 6.4.2), but we wanted to take this one step further, and so a survey on context-aware frameworks was conducted (in chapter5) in order to find the best option for integrating the device concept with a context-aware framework (due to the proliferation of context-aware framework proposals in the research world, there was no need to define a new one). With the integration of context-awareness we achieved an higher level of "intelligence" in the QuiiQ Automation Foundation. This will allow the system to adapt its behavior to the user's needs, like for example to increase user's comfort by attempting to anticipate their actions, reduce energy consumption, or sprinkle the grass according to different sequences based on the weather and the time of day. Section 6.4.4 and 6.4.5 presents the Context Toolkit, some upgrades to the Context Toolkit and its possible integration strategies with the QuiiQ Automation Foundation.

In summary, this dissertation provides answers to the previously defined research questions, but it presents these answers in such a way that the integration of these models with the future QuiiQ Automation Framework is easy and direct. This is the beginning of a new and revolutionary way of looking at and developing home automation software.

7.2 Future Research Directions

Beside the previously mentioned contributions, there is a lot of work yet to be done on the QuiiQ Automation Framework project where this sub-project is included. From designing the driver's structures and communications approach, to the analysis and definition of graphical user interface components to allow the dynamic development of graphical user interfaces.

Regarding the QuiiQ Automation Foundation, when the first system is in development and field-tested (it was not possible to develop an application and field-test due to time and financial limitations; the only field-test was made with the QUiiQ AUTOMATION product), an adjustment to the theoretical concepts will probably be made in order to better suite the final user needs or the application developer needs. Regarding the context awareness, there is a lot of evolving yet to occur. From the definition of a context ranking, to a context triggered action ranking, which will allow the definition of automatic actions that have a degree of certainty (if the action is overruled by a user's action, the system's confidence decreases,

otherwise it is reinforced),etc.

Another essential evolution in the QuiiQ Automation Foundation is the abstraction of the concept of user, and the possibility to define all kinds of profiles (access profiles, security profiles, climatization profiles, etc) and integrate these profiles into the rest of the system.

References

[1] G. Conte, D. Scaradozzi, A. Perdon, M. Cesaretti, and G. Morganti. A simulation environment for the analysis of home automation systems. In *Control & Automation, 2007. MED '07. Mediterranean Conference on*, pages 1–8, June 2007.

[2] S. Lorente. Key issues regarding domotic applications. In *Information and Communication Technologies: From Theory to Applications, 2004. Proceedings. 2004 International Conference on*, pages 121–122, April 2004.

[3] Barry W. Boehm. Improving software productivity. *Computer*, 20(9):43–57, 1987.

[4] Ademar Aguiar. *Framework documentation*. PhD thesis, Departamento de Engenharia Electrotecnica e Computadores, Porto, 2003.

[5] Han Albrecht Schmid. Systematic framework design by generalization. *Commun. ACM*, 40(10):48–51, 1997.

[6] Graby Booch. Designing an application framework. http://www.ddj.com/architect/184409177?pgno=8, 2008.

[7] Hans-Erik Eriksson, Magnus Penker, Brian Lyons, and David Fado. *UML 2 Toolkit*. John Wiley and Sons,, 2 edition, 2004.

[8] Johann Bourcier, Antonin Chazalet, Mikael Desertot, Clement Escoffier, and Cristina Marin. A dynamic-soa home control gateway. In *SCC '06: Proceedings of the IEEE International Conference on Services Computing*, pages 463–470, Washington, DC, USA, 2006. IEEE Computer Society.

[9] R.J.C. Nunes. A web-based approach to the specification and programming of home automation systems. In *Electrotechnical Conference, 2004. MELECON 2004. Proceedings of the 12th IEEE Mediterranean*, volume 2, pages 693–696 Vol.2, May 2004.

[10] V. Miori, L. Tarrini, M. Manca, and G. Tolomei. An open standard solution for domotic interoperability. *Consumer Electronics, IEEE Transactions on*, 52(1):97–103, Feb. 2006.

[11] A. Alheraish. Design and implementation of home automation system. *Consumer Electronics, IEEE Transactions on*, 50(4):1087–1092, Nov. 2004.

[12] Paolo Pellegrino, Dario Bonino, and Fulvio Corno. Domotic house gateway. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 1915–1920, New York, NY, USA, 2006. ACM.

[13] G. Acampora and V. Loia. Fuzzy control interoperability and scalability for adaptive domotic framework. *Industrial Informatics, IEEE Transactions on*, 1(2):97–111, May 2005.

[14] Andrea Valerio, Giancarlo Succi, and Massimo Fenaroli. Domain analysis and framework-based software development. *SIGAPP Appl. Comput. Rev.*, 5(2):4–15, 1997.

[15] Toni Jussila, Jori Dubrovin, Tommi Junttila, Timo Latvala, Ivan Porres, and Johannes Kepler. Model checking dynamic and hierarchical uml state machines. In *in: Proceedings of MoDeV 2 a (2006*, pages 94–110, 2006.

[16] M.V. Zelkowitz and D.R. Wallace. Experimental models for validating technology. *Computer*, 31(5):23–31, May 1998.

[17] S. Kuo, Z. Salcic, and U. Madawala. A real-time hybrid webdent access architecture for home automation. In *Proceedings of the 2003 Joint Conference of the Fourth International Conference on Information, Communications and Signal Processing, 2003 and the Fourth Pacific Rim Conference on Multimedia*, volume 3, pages 1752–1756, 2003.

[18] James Gerhat. *Home Automation and Wiring*. McGraw-Hill, 1999.

[19] Tony Northrup and Eric Faulkner. *Home Hacking Projects for Geeks*. O'Reilly, 2004.

[20] X10. X10. http://www.x10.com, 2007.

[21] Connected Home Media. Exploring x10 technology. http://www.connectedhomemag.com/homecontrols/articles/index.cfm?articleid=22668, September 2001.

[22] Konnex Association. http://www.knx.org/knx-standard/introduction/, April 2008.

[23] Crestron. http://www.crestron.com/, 2008.

[24] Echelon. http://www.echelon.com/, 2008.

[25] Zensys. http://www.z-wave.com/modules/Z-Wave-Start/, 2007.

[26] ZigBee Alliance. Zigbee overview. Technical report, ZigBee Alliance, 2007.

[27] Thierry Bodhuin, Gerardo Canfora, Rosa Preziosi, and Maria Tortorella. Hiding complexity and heterogeneity of the physical world in smart living environments. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 1921–1927, New York, NY, USA, 2006. ACM.

[28] QuiiQ. http://www.quiiq.com/QuiiQHADetail.aspx, 2008.

[29] Rui Barreira, Rui Almeida, Luis Neves, et al. *QuiiQ AUTOMATION - User Guide V1.8*. QuiiQ, June 2008.

[30] StarDraw. http://www.stardraw.com/products/stardrawcontrol/default.asp, 2008.

[31] Embedded Automation. http://www.embeddedautomation.com/, 2008.

[32] André Bottaro and Anne Gérodolle. Home soa -: facing protocol heterogeneity in pervasive applications. In *ICPS '08: Proceedings of the 5th international conference on Pervasive services*, pages 73–80, New York, NY, USA, 2008. ACM.

[33] M. E. Fayad, D. C. Schmidt, and R. E. Johnson. *Implementing Application Frameworks — Object-Oriented Frameworks at Work*. John Wiley & Sons, 1999.

[34] Mohamed Fayad and Douglas C. Schmidt. Object-oriented application frameworks. *Commun. ACM*, 40(10):32–38, 1997.

[35] Charles W. Krueger. Software reuse. *ACM Comput. Surv.*, 24(2):131–183, 1992.

[36] T. Biggerstaff and C. Richter. Reusability framework, assessment, and directions. *Software, IEEE*, 4(2):41–49, March 1987.

[37] Ralph E. Johnson and Brian Foote. Designing reusable classes. *Journal of Object-Oriented Programming*, 1:22–35, 1988.

[38] Nadia Bouassida, Hanêne Ben-Abdallah, Faiez Gargouri, and Abdelmajid Ben Hamadou. Stepwise framework design by application unification. *IEEE International Conference on Systems, Man and Cybernetics*, 6, October 2002.

[39] Todd Hansen. Development of successful object-oriented frameworks. In *OOPSLA '97: Addendum to the 1997 ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (Addendum)*, pages 115–119, New York, NY, USA, 1997. ACM.

[40] Klaus Pohl, Gunter Bockle, and Frank van der Linden. *Software Product Line Engineering*. Springer, 2005.

[41] David R. Morse, Stephen Armstrong, and Anind K. Dey. The what, who, where, when, why and how of context-awareness. In *CHI '00: CHI '00 extended abstracts on Human factors in computing systems*, pages 371–371, New York, NY, USA, 2000. ACM.

[42] Ana Hristova. Conceptualization and design of a context-aware platform for user-centric applications. Master thesis, Norwegian University of Science and Technology, June 2008.

[43] B.N. Schilit and M.M. Theimer. Disseminating active map information to mobile hosts. *Network, IEEE*, 8(5):22–32, Sep/Oct 1994.

[44] P.J. Brown, J.D. Bovey, and Xian Chen. Context-aware applications: from the laboratory to the marketplace. *Personal Communications, IEEE*, 4(5):58–64, Oct 1997.

[45] N. S. Ryan, J. Pascoe, and D. R. Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. In V. Gaffney, M. van Leusen, and S. Exxon, editors, *Computer Applications in Archaeology 1997*, British Archaeological Reports, Oxford, October 1998. Tempus Reparatum.

[46] Anind K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, Georgia Institute of Technology, November 2000.

[47] Jason Pascoe, Nick Ryan, and David Morse. Using while moving: Hci issues in fieldwork environments. *ACM Trans. Comput.-Hum. Interact.*, 7(3):417–437, 2000.

[48] Stephen Fickas, Gerd Kortuem, and Zary Segall. Software organization for dynamic and adaptable wearable systems. In *ISWC '97: Proceedings of the 1st IEEE International Symposium on Wearable Computers*, page 56, Washington, DC, USA, 1997. IEEE Computer Society.

[49] Harry Chen, Tim Finin, and Anupam Joshi. An ontology for context-aware pervasive computing environments. *Knowl. Eng. Rev.*, 18(3):197–207, 2003.

[50] Terry Winograd. Architectures for context. *Hum.-Comput. Interact.*, 16(2):401–419, 2001.

[51] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, 2(4):263–277, 2007.

[52] Thomas Strang and Claudia L. Popien. A context modeling survey, September 2004.

[53] Barbara T. Korel and Simon G. M. Koo. Addressing context awareness techniques in body sensor networks. In *AINAW '07: Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops*, pages 798–803, Washington, DC, USA, 2007. IEEE Computer Society.

[54] Patrick Fahy and Siobhan Clark. Cass - middleware for mobile context-aware applications. In *Mobisys*, 2004.

[55] Lalana Kagal, Tim Finin, and Anupam Joshi. A policy language for a pervasive computing environment. In *POLICY '03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, page 63, Washington, DC, USA, 2003. IEEE Computer Society.

[56] P. Korpipaa, J. Mantyjarvi, J. Kela, H. Keranen, and E.J. Malm. Managing context information in mobile devices. *Pervasive Computing, IEEE*, 2(3):42–51, July-Sept. 2003.

[57] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The context toolkit: aiding the development of context-enabled applications. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 434–441, New York, NY, USA, 1999. ACM.

[58] Gregory Biegel and Vinny Cahill. A framework for developing mobile, context-aware applications. In *PERCOM '04: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, page 361, Washington, DC, USA, 2004. IEEE Computer Society.

[59] Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74–83, 2002.

[60] Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann, and Werner Retschitzegger. Context-awareness on mobile devices - the hydrogen

approach. In *HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9*, page 292.1, Washington, DC, USA, 2003. IEEE Computer Society.

[61] Tao Gu, Hung Keng Pung, and Da Qing Zhang. A service-oriented middleware for building context-aware services. *J. Netw. Comput. Appl.*, 28(1):1–18, 2005.

[62] Sven Meyer and Andry Rakotonirainy. A survey of research on context-aware homes. In *ACSW Frontiers '03: Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003*, pages 159–168, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[63] Johan Plomp and Paolo Tealdi. Ambient intelligent technologies for wellbeing at home. In *EUSAI '04: Proceedings of the 2nd European Union symposium on Ambient intelligence*, pages 81–82, New York, NY, USA, 2004. ACM.

[64] Alan Newberger and Anind K. Dey. Designer support for context monitoring and control. Technical report, Intel Research, June 2003.