POLITECNICO DI MILANO
DIPARTIMENTO DI ELETTRONICA, INFORMAZIONE E BIOINGEGNERIA
DOCTORAL PROGRAMME IN INFORMATION TECHNOLOGY

A COMPREHENSIVE FRAMEWORK FOR THE DEVELOPMENT OF DYNAMIC SMART SPACES

Doctoral Dissertation of:
Adnan Shahzada

Supervisor:
Prof. Luciano Baresi

Tutor:
Prof. Carlo Ghezzi

The Chair of the Doctoral Program:
Prof. Carlo Fiorini

2015 – XXVIII

Acknowledgements

First, I would like to thank my advisor Prof. Luciano Baresi for all his time, ideas, and support to make my Ph.D. experience productive and stimulating. I have learnt a lot during our long and lively discussions, and that has contributed significantly to the development of this work.

I am also thankful to Prof. Sam Guinea who has helped me initially to formulate the basis for this research and then worked together with me in pursuing it.

I am grateful to Telecom Italia for funding my Ph.D. studies under the initiative of Joint Open Lab (S-CUBE). This work would not have been possible without their financial and technological support. Specifically, I would like to thank Massimo Valla, Laurent-Walter Goix, Danny Noferi and Naser Derakhshan for the useful and productive collaboration.

A special thanks goes to Prof. Patricia Lago for accepting to review my manuscript and giving her valuable comments.

I also want to offer my gratitude to my tutor Prof. Carlo Ghezzi.

to my wonderful family

Abstract

SMART SPACES have gained a lot of attention during the last few years. These spaces are getting increasingly enriched with a variety of sensors, mobile devices, wearables, actuators and other smart objects. These devices are usually equipped with one or multiple network interfaces that allow them to connect with each other in order to provide useful contextualized services to the smart space inhabitants. Numerous solutions have been proposed by the researchers to solve different aspects related to the realization of these smart spaces. Despite many standalone middleware solutions, development methodologies, IoT-enabled devices, and simulators, there is still a gap between what is available today and the need for an effective end-to-end development/deployment framework.

This study investigates where do existing solutions and approaches lack and what is the right approach for building diversified smart spaces. The fundamental challenge of smart space development is to design a suitable framework that structures and facilitates the conception of these spaces at large, that is, it should be able to support the development of a wide variety of spaces and scenarios. Moreover, there is a big gap between the existing design solutions and their possible transition towards a concrete deployment in the real world. The major reason for this is the inability of the design abstractions (provided by these solutions) to be translated into their proper implementation counterparts.

This thesis bridges this gap by proposing a framework that covers the entire development life-cycle of smart spaces. It offers the same abstractions throughout all the development phases while providing means for both the seamless integration of various components and the utilization of existing systems. The framework provides a fixed software backbone that allows the developer to move seamlessly from a fully virtual, simulated solution to a completely deployed system in an incremental manner. It provides interfaces both to surrogate system components through external simulators, and to ease the deployment of physical elements. The proposed solution also integrates the conventional component-based control (autonomic computing) and the inherent self-adaptive capabilities of the bio-inspired (firefly-based) ecosystem. The framework eliminates the individual shortcomings of these approaches, that is, lack

of inherent situated awareness and mobility, for autonomic approaches, and lack of control over the self-organization in bio-inspired systems. Realistic scenarios are used throughout the thesis to showcase and evaluate the key features of the proposed solution. The results demonstrate that presented solution complements the existing growth of the smart objects and plethora of software solutions by providing a framework to integrate/utilize the available solutions as a step forward towards the efficient end-to end development of smart spaces.

Contents

1 Introduction **1**
1.1 Problem and Research Questions 2
1.2 Research Objectives 4
1.3 Major Contributions 5
1.4 Thesis Structure 5

2 Smart Spaces **7**
2.1 Definition 7
2.2 Diverse Smart Spaces 8
2.2.1 Personal/Restricted Smart Spaces 10
2.2.2 Public/Social Smart Spaces 10
2.3 Example Scenarios 11
2.3.1 Modern Greenhouse 11
2.3.2 Smart Office 12
2.3.3 Public Park 13
2.4 Properties 14
2.5 Challenges 15
2.6 Life Cycle of Smart Spaces 17

3 State of the Art **21**
3.1 Approaches 21
3.1.1 Architecture-Centric Approaches 22
3.1.2 Multi-Agent Systems 24
3.1.3 Nature-Inspired Computing 27
3.2 Solutions 28
3.2.1 Fixed Indoor Deployments 29
3.2.2 Automation (IoT) Hubs 29
3.2.3 Integration Platforms 29
3.2.4 Middleware Infrastructures 30
3.2.5 Complete Development Solutions 31
3.2.6 Validation Tools 32

3.3 Comparative Analysis 32

4 Proposed Framework **37**
4.1 Revisiting the Perspective 37
4.1.1 Incremental Development 37
4.1.2 Integrated Self-Adaptive Approach 39
4.2 Design Abstractions 40
4.2.1 Component 41
4.2.2 Role 42
4.2.3 Group 44
4.3 Semantic Layer 46
4.3.1 Semantic Model 48
4.4 Integration Layer 49
4.4.1 Separation of Functional and Management Design 51
4.4.2 Autonomic Management 53
4.5 Self-Adaptation Capabilities 55
4.5.1 Fireflies Algorithm 55
4.5.2 Adaptation based on Fireflies Metaphor 56
4.5.3 Self-Organization Algorithm 58
4.6 Continuous Validation 62

5 Implementation and Concurrent Execution **67**
5.1 Implementation Model 67
5.2 Component Class 69
5.3 Component Roles and Behaviors 70
5.3.1 Supervisor Role Class 70
5.3.2 Follower Role Class 70
5.4 Asynchronous Message Exchange 71
5.4.1 Message Class 72
5.4.2 Physical Communication 72
5.4.3 Messaging Queues 73
5.4.4 Rendezvous Exchange 73
5.5 Group Coordination Styles 73
5.5.1 Group Class 73
5.5.2 Synchronous Timed Coordination 74
5.5.3 Event-based Coordination 75

6 Evaluation **77**
6.1 Evaluation Plan 77
6.2 Case Study 1: Smart Office 79
6.2.1 Simulation with Native APIs 80
6.2.2 Simulation based on External Simulators 81
6.2.3 Partially Deployed (Simulated) System 82
6.2.4 Fully Deployed System 82
6.3 Case Study 2: Modern Greenhouse 86
6.3.1 Managing incoming carts 89
6.3.2 Sick flowers 89

6.4 Case Study 3: Energy Efficient Buildings 90
6.4.1 Building Model 90
6.4.2 Integrated Control 92
6.4.3 Experimental Details 94
6.5 Case Study 4: Public Park 96
6.5.1 Experimental Setup 96
6.6 Discussion 101

7 Conclusions and Future Directions **105**
7.1 Answers to Research Questions 106
7.2 Future Directions 109

Bibliography **111**

List of Figures

2.1 Constructs of a Typical Smart Space 8
2.2 Smart Space Requirements 9
2.3 Smart Spaces 10
2.4 Greenhouse. 11
2.5 Joint Open Lab Map 13
2.6 Example park scenario. 14
2.7 Smart Space Life Cycle 17

4.1 Incremental Smart Space Development 38
4.2 Incremental Smart Space Development Life Cycle 39
4.3 The Proposed Framework 41
4.4 Component Model 42
4.5 Framework Meta-Model 43
4.6 Group Abstraction 45
4.7 Group Compositions 46
4.8 Design Abstractions 47
4.9 Group Example - Light Management 48
4.10 Sensor Ontology. 49
4.11 RDF graph of a semantic model for sensors. 49
4.12 Taxonomy of semantic model —We have used namespaces instead of URIs for the sake of clarity. 50
4.13 JOL Semantic Model 51
4.14 Example configuration of a Smart Space. 52
4.15 Autonomic Management 53
4.16 JOL- Possible Groups 55
4.17 Integrated Self-Adaptive Mechanism 56
4.18 Fireflies Adaptation Mechanism 58
4.19 Example bootstrapping scenario. 60
4.20 JOL- Topology with external simulators 62
4.21 JOL- High Level View 63
4.22 Design Tasks 64

5.1 Implementation Model 68
5.2 Example Runtime Snapshot of a Component 69
5.3 Component Behaviours 71
5.4 Communication Channels 72
5.5 Asynchronous Messaging Paradigms 74
5.6 Group Coordination Styles 75

6.1 Bootstrapping Time for Components 81
6.2 Simulation in Freedomotic and Siafu 82
6.3 Partially Deployed System (Siafu and real ZigBee enabled lights). . . . 83
6.4 Fully Deployed System 84
6.5 Greenhouse Domain Concept Ontology (DCO). 86
6.6 Group topology: the SSM manages the room supervisors (Room1SV, Room2SV and Room3SV) that play the role of supervisors for carts, but act as followers for the SSM node. 87
6.7 RDF graph for greenhouse group topology. 88
6.8 Discovery and self-configuration of components. 88
6.9 Benchmark office building. 91
6.10 Example control group. 92
6.11 Integrated Control Groups. 93
6.12 Concurrent execution/simulation. 94
6.13 Energy consumption for lighting. 95
6.14 Energy consumption for building. 95
6.15 Park simulation in NetLogo. 97
6.16 Park Topology 98
6.17 Average number of messages per screen. 99
6.18 Average number of messages per user. 100
6.19 Results 100
6.20 Power consumption. 101
6.21 Group size variation. 102

List of Tables

2.1 Diverse Smart Spaces 10
2.2 Smart Spaces - Challenges 17

3.1 Approaches and Solutions 31
3.2 Comparison of most relevant existing systems 35

4.1 Mapping - Component Abstractions and Fireflies Metaphor 57

6.1 Quantitative Evaluation 77
6.2 Qualitative Evaluation 78
6.3 Evaluation Plan 78
6.4 Joint Open Lab - Components 79
6.5 Message Delays 84
6.6 Groups Specification 85

CHAPTER 1

Introduction

We are living in an era where computers are becoming more and more ubiquitous in our everyday lives. There is an enormous technological growth in terms of increase in computing power, memory sizes and miniaturization of devices over the past few years. Many new types of inter-connectable devices such as tablets, smart watches, wearable gadgets, and a plethora of sensing equipments are now available in a ready to use packaging. Moreover, various low cost (single-board) micro-controllers such as Arduino have now also enabled the consumers to have access to various sensors and actuators for their personal automation projects.

With all this advancement, Mark Weiser's [98] vision of ubiquitous and pervasive computing world, where technologies interweave themselves into the fabric of everyday life in an invisible manner and providing computing nearly everywhere, seems to be realized in coming years. A pervasive computing system is situation-aware and can reason intelligently in an autonomous manner to respond to the needs of the users in a given context [29]. The goal of such systems is to create intelligent services within an environment, embedded with various network devices that provide continuous and unobtrusive connectivity to perform useful functions to facilitate their users [38]. Lighting, heating, air conditioning and ventilating control in closed spaces or buildings (with the aim to maintain user comfort and save energy) are the few examples of such services. Moreover, there is also a growing trend of providing ad-hoc services to the users in more open and dynamic outdoor spaces (such as stadiums, malls, and parks) based on their proximity and personal interests.

The manifestation of distributed computing [17, 79, 93] through cloud infrastructures [74], and mobile computing [45] enabling ubiquity of smart inter-connected mobile devices has supported the realization of such pervasive computing systems. The wireless communication technology and portability of various devices also play an im-

portant part in enabling the mobility of pervasive computing infrastructure.

1.1 Problem and Research Questions

Despite all these advancements, the pervasive computing vision is still reality only in parts, as of today. There have been some limited deployments such as smart homes [25, 26] and automation solutions for indoor spaces. But, they are yet to be widely exploited in physical spaces in the same manner as the ubiquitous Web and smartphones have been used – effectively revolutionizing/changing the way we interact with others, work, and conduct our daily lives [57].

Many efforts have been put to manifest pervasive computing in bound (physical or conceptual) environments to convert them into **Smart spaces**. Smart spaces can be regarded as technology augmented intelligent environments with the ability to understand user/human needs within a space and react accordingly in order to provide contextualized services to the inhabitants [28]. Typically, a smart space comprises multiple autonomous entities (with different capabilities) that are heterogeneous in terms of functionality, communication protocols, and execution models. These heterogeneous entities co-exist and collaborate with each other to help inhabitants accomplish their tasks.

Smart spaces are dynamic self-adaptive systems where each entity is characterized by its state (current situation), functionality (affect to/from the system), transitional policies (events/situations that trigger a state change) and the situatedness (position in the environment). These distributed system entities interact with each other (and constitute system events) through direct or indirect communication means. Another important characteristic of these spaces is openness as different entities move while being connected to the space (other entities) and can join or leave the system unannounced and without a central control.

Given the state of all these sophisticated smart objects and technologies, one must ask why do we not already live in smart pervasive spaces (environments) where all these available smart and mobile devices are seamlessly providing us the services to enhance our efficiency and improve our lifestyles? One of the reasons for this is that all these hardware equipment and smart objects are one of the many pre-requisites for the realization of such environments. The growing complexity of the devices also requires the software development process for such systems to be re-modeled. As the smart spaces are inherently complex, dynamic and open in their nature, the required software solution needs to be able to adapt itself to the always changing conditions of the spaces without any (or limited) external human intervention. Another reason for that is the lack of aggregation of the existing solutions, which try to tackle the problem from different angles and do not put an effort to build on top of/in complement to existing solutions. Many smart devices and off-the-shelf autonomous objects are available which have limited sensing and actuating capabilities, but these systems often work in isolation and hence do not enable the implementation of holistic pervasive computing scenarios.

The majority of these solutions are developed through centralized architectural control loops and hence they are prone to have scalability issues. To deal with these issues, some bio-inspired adaptations systems are proposed to move towards complete decentralized and autonomous solutions. Although, all these approaches are interesting and

useful for certain kind of scenarios, there is a need for a decentralized yet manageable solutions.

Following are the three important challenges and open research questions for the development of smart spaces today that are addressed in this thesis.

RQ 1. Do we need to revisit/improve our development processes for the creation of smart spaces in order to realize effective smart spaces and, in case, what would that revision be? (Sec. 4.1)

RQ 2. What kind of software (framework) abstractions and interaction mechanisms are required for the design and implementation of smart spaces to overcome the fragmented (in terms of technologies and functionality) devices and continuously changing user requirements? (Sec. 4.2)

RQ 3. How can we evaluate the effectiveness and capability of such a framework to design, develop and assess highly dynamic diversified smart spaces? (6.1)

The aforementioned questions describe the two major dimensions of the research. RQ.1 is more general and requires an investigation to find out why the existing approaches are not successful in creating these diversified smart spaces. Moreover, it also concerns with studying the need to re-define (or integrate) the current perspectives towards the design and development. Therefore, RQ.1 can further be subdivided into the following questions:

RQ 1.1. What are the requirements for the development of diversified smart spaces and where do existing systems lack? (Sec. 2.5)

RQ 1.2. Which and how can we employ/integrate various principles from the existing design paradigms to address those deficiencies, if possible? (Sec. 4.1.2)

RQ 1.3. What is the suitable development life-cycle for the development of diversified smart spaces? (Sec. 4.1.1)

RQ.2 deals with the implementation of a development framework that would be required to realize the approach which will emerge as an answer to RQ. 1.2. It can be further subdivided into the following questions:

RQ 2.1. How can we build a software framework that fulfils the requirements (Sec. 2.5) identified in answering to RQ.1?

RQ 2.2. How can we use (some of the) existing smart objects, architectures, middleware infrastructures, and simulators to implement smart spaces dynamically, effectively, and opportunistically? (Sec. 4.4 and 4.6)

RQ.3 addresses the problem of assessing development frameworks for smart spaces. It seeks an answer to how one can define challenges that correspond to different scenarios in diversified smart spaces and then being able to evaluate it.

This thesis proposes answers to the above questions by proposing a comprehensive framework, and shows how the proposed framework can be used to design, implement and assess/validate Smart Spaces.

1.2 Research Objectives

The questions and challenges described above can be reformulated as the following:

There is a need to integrate the control mechanisms from architectural solutions/approaches for self-adaptation and inherent self-organizing capabilities from bio-inspired approaches in order to design both effective and continuously evolving smart spaces.

The inability of existing smart spaces to evolve over the time and deal with changing requirements calls for the need of an incremental development framework that offers proper blending of different aspects: provision of appropriate design abstractions to support co-execution of physical and virtual components through proper middleware infrastructures, ad-hoc integration and coordination of heterogeneous components, early evaluation of "incomplete" systems, assessment of alternative solutions and deployments of the system.

This research, first implements the proposed framework and then evaluates it according to the various smart space development challenges. The goal and major objective of the thesis can thus be described as:

"Provide a self-adaptive framework that enables the developers to design, implement and validate dynamic smart spaces by offering programming abstractions that are suitable for the whole development life-cycle."

The main objective can be divided into the following sub-objectives:

O.1. Provide the appropriate abstractions and interfaces for device/system inter-operation

O.2. Provide solutions for the whole development life-cycle of the smart spaces

O.3. Structure and facilitate the dynamic (automated) composition of different systems/devices at run-time

O.4. Facilitate the creation of diverse types of smart spaces and scenarios

The dissertation starts with the analysis of the implementation of smart spaces that exist presently (Ch. 2). The lack and the need for a comprehensive development framework, suitable for the complete life-cycle and applicable for diverse scenarios, are identified. To define what is "comprehensive", requirements for the development of different spaces are analyzed. These requirements have to be addressed/solved to achieve the objectives of the thesis.

A comprehensive study of the related state of the art (Ch. 3) was conducted to understand the pertinence of the identified requirements (in Ch.2) and to ensure that no solutions exist that completely solves those requirements. The short comings of the existing solutions to achieve the objectives were also analyzed along with the possibility of (re)using them in a more comprehensive solution.

A group-based self-organizing framework is designed which is described in Ch. 4. The mapping of the requirements to the solutions provided by the framework is also embedded in the text. Prototypes are developed to demonstrate the feasibility

of the proposed concepts and solutions and they are evaluated, both qualitatively and quantitatively, on the realistic scenarios (Ch. 6).

1.3 Major Contributions

The major contribution of the thesis is a comprehensive development framework for dynamic smart spaces. The proposed framework provides relevant programming abstractions to build a space and also bridges the gap between existing isolated solutions by integrating them to enable a complete end-to-end solution for smart spaces. It comprises several novelties as described in Ch. 4, and summarized in the abstract at the beginning of that chapter.

The most important concepts of the proposed framework are:

- Abstractions that are suitable for the whole development life cycle
- A semantic model for the inter-operation of the components speaking different languages
- A dynamic, self-organizing and extensible middleware/coordination style for the management of smart space components
- Mechanisms for the concurrent execution of heterogeneous components with different time and execution cycles
- Integration of software architectural approach with bio-inspired metaphors for self-adaptation

The side-contributions of the thesis are the following:

- Automated management of components at different levels of abstraction
- Building blocks to build the system from the scratch or integrate different physical or virtual existing systems
- Requirements analysis for different types of smart spaces
- Analysis of the state of the art solutions for the development of smart spaces

1.4 Thesis Structure

The thesis is constituted of seven chapters. Following is the brief summary of each chapter:
Chapter 1 introduces the scope of the work and formulates the research questions with references to their answers presented in the rest of the chapters.

Chapter 2 discusses the theoretical background. It defines and elaborates the concept of a smart space. First, a definition of smart space is provided. Later, a broad classification of diversified smart space infrastructures along with their key characteristics are presented. The chapter is concluded by stating technical requirements for each type of smart spaces. The chapter will serve as a conceptual background and domain

for the thesis.

Chapter 3 surveys the state of the art. It analyzes the existing solutions in terms of the approaches they employ as well as the fraction of target solution space they aim to address. It discusses how approaches such as software architectures, model driven engineering of systems, multi-agent systems and nature-inspired computing are used to develop smart spaces. Moreover, the chapter also provides an overview of the type of solutions currently available for the end-users (off the shelf/ready to use solutions) or the developers (middleware infrastructures and frameworks).

Chapter 4 defines our proposal to adopt an incremental and evolutionary approach to build smart spaces. It provides a conceptual description of the key components of the proposed framework that includes design and programming abstractions, semantic model, integration and collaboration mechanisms and self-adaptive capabilities.

Chapter 5 elaborates the implementation details of the proposed framework. It focuses on how different components of the space, which are conceived through the conceptual framework defined in the previous chapter, can be concurrently executed. It also describes what are the behavioural and coordination styles available for the developers to implement their systems.

Chapter 6 describes the evaluation of the framework through the case-studies and scenarios defined in 2.3. It demonstrates that how the proposed framework can be used efficiently to design diverse smart spaces such as greenhouse, smart office, public buildings and large open spaces like parks.

Chapter 7 concludes the thesis by listing out the contributions and lessons learned. Further, it discusses the possible future work in this area.

CHAPTER 2

Smart Spaces

This chapter defines and elaborates the concept of a smart space. First, a definition of smart space is provided. Later, a broad classification of diversified smart space infrastructures along with their key characteristics are presented. The chapter is concluded by stating technical requirements for each type of smart spaces. The chapter will serve as a conceptual background and domain for the thesis.

2.1 Definition

Many researchers have described the concept of smart spaces from various perspectives [65]. For instance, Ramesh et al. [86] view smart spaces as ordinary environments, equipped with both visual and audio sensing systems, and are capable of perceiving and reacting to the people needs without any wearable or special purpose devices. Saleemi et al. [83] define smart spaces as dynamic environments that change their identity over the time whenever they interact with different entities and exchange (share) information among them. Similarly, Cook and Das [28] perceive smart spaces as physical environments where smart objects collaboratively monitor the environment, interact with the inhabitants, and adapt their behaviour according to the information gathered from the environment. We attempt to incorporate all these views and provide an aggregated definition and constructs of a smart space (see Figure 2.1):

Definition 2.1.1. *Smart space is a technology augmented intelligent environment with the ability to acquire situational data, understand user needs, and react accordingly to provide contextualized services to the inhabitants in order to improve their experience in that environment.*

The aforementioned definition implies the need for a continuous and dynamic interaction between the space and its inhabitants. The construction of smart spaces with the

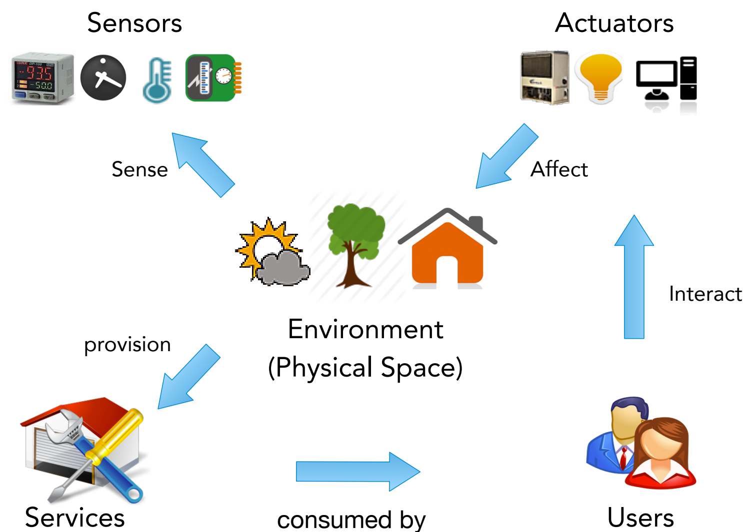


Figure 2.1: *Constructs of a Typical Smart Space*

ability to support such interaction requires the following characteristics:

- *Context Sensing* through continuous monitoring of the environment is required. It will provide the information related to both the physical state and the users within a space.
- A *collaboration model* or a coordination style is needed to form collaborations among different components of the space. It will enable information sharing and implementation of various functionalities and services that the space needs to provide to its users.
- *Self-Adaptation* is required to adjust the smart space according to the changes in the environmental state or the mobility of the components that inhibit the space.
- *Actuating* functionality is required to change or induce new environmental conditions or collaboration topology of the system in response to the changing needs of the space.

2.2 Diverse Smart Spaces

Smart spaces are developed using multiple heterogeneous objects, models, and communication technologies requiring integration and adaptation of many smart space subsystems to the particular user needs [51]. Many of such spaces, with diversified purposes, spatial attributes, and technological infrastructures are being created and deployed [70]. A space can be personal such as home, office, and assisted health care system, or a more open place such as shopping malls, parks, train terminals and airports as shown in Figure 2.3.

This diversity and heterogeneity of real spaces are one of the main hindrances in deploying a research-based system to a practical one. The types of services required by

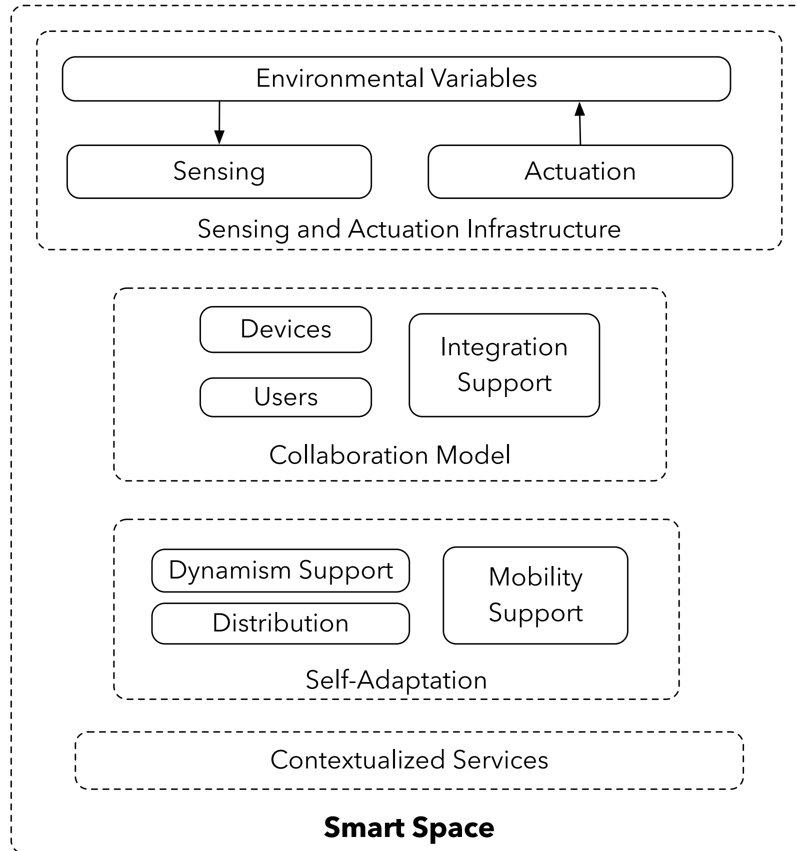


Figure 2.2: *Smart Space Requirements*

the inhabitants vary significantly with the type of space being considered and the way users interact with that space. For instance, users may want the space to automate the typical everyday tasks they perform within the space, or alternatively, they may expect the smart space to optimize the resources usage such as energy and communication bandwidth.

Therefore, It is very important to have some taxonomy to classify the great number of diverse smart spaces created till now and the ones to be created in the next few years [70]. We believe that defining a smart space taxonomy and laying out the corresponding specification criteria is very effective for guiding the development process and systematic evaluation of various smart space systems. Therefore, given the diverse characterization of smart spaces and the various challenges they pose, we can divide the existing spaces broadly into two categories (see Table 2.1):

1. Personal/Restricted smart spaces
2. Public/Social smart spaces

It is important to note that this classification is quite preliminary and it is devised by analyzing various properties of the existing smart spaces. We discuss the properties of each category of spaces, and then, layout some challenges related to the development of these spaces.

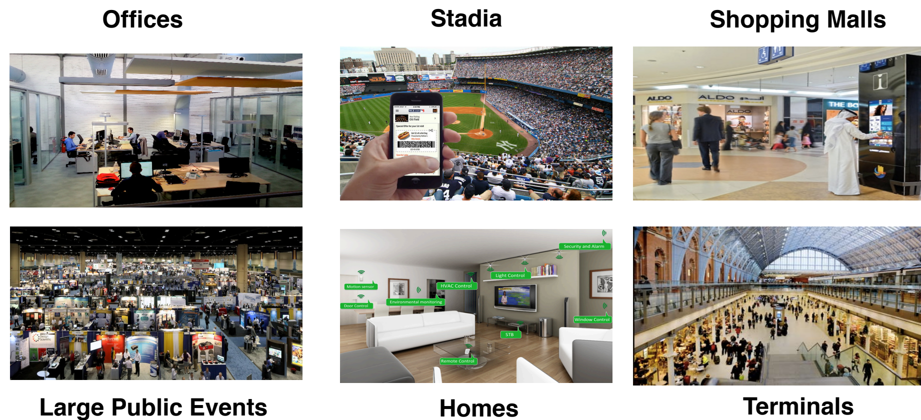


Figure 2.3: *Smart Spaces*

Table 2.1: *Diverse Smart Spaces*

Category	Scope	Interaction Layout	Scale		Example Spaces
			Users	Devices	
Personal/ Restricted	Closed (Mostly Known devices and users)	Fixed Location and context	Small	Small	Smart homes
					Smart Offices
					Assisted Living
			Many	Many	Smart Buildings
Social/ Public	Open (Most of devices/users are not known a-priori)	Dynamic Proximity-based context	Large	Large	Public Places (Parks, Airports, etc.)
					Large Events (Exposition, Football Match)

2.2.1 Personal/Restricted Smart Spaces

We characterize *personal smart spaces* with the following two important properties:

- They have limited/restricted user participation.
- Most of the components are known a-priori.

Typically, these spaces are small/medium scale indoor (buildings) where inhabitants interact with all the devices in a pre-defined manner. These smart spaces comprise static components that do not change (frequently) over the time, and mostly, these spaces are "only” required to connect different smart appliances in closed spaces such as homes and offices. Components of these spaces operate on some (limited) standard protocol(s) and the interaction is based on static layout of the physical space (e,g. floor-plans etc.).

2.2.2 Public/Social Smart Spaces

Social spaces, on the other hand, have components that are highly dynamic (open) and they may enter or leave the space on frequent basis. Moreover, the components are usually mobile and they move within the space. Interaction between users and different

devices is, therefore, ad hoc (proximity-based). For example, one can think of the interactions between humans —through their devices— and the environment to exploit location-based services in big public places like stadia, transport stations and exposition centers. Components of these spaces, mostly, are not known a-priori and they operate on different protocols. Typically, these spaces are big in size and they are shared by large number of users (public). Stadia, large public events (concerts, expositions) and mobile ad-hoc spaces are few examples of such systems.

2.3 Example Scenarios

Some example scenarios are presented here that will be used throughout the thesis to demonstrate various aspects of the proposed solution. Moreover, they have also been implemented for system evaluation.

2.3.1 Modern Greenhouse

The greenhouse is equipped with various kinds of sensors that monitor its temperature, light, and humidity levels. It is essential that the greenhouse be able to utilize its resources optimally, by grouping similar plants together in certain rooms according to their temperature and other physiological needs.

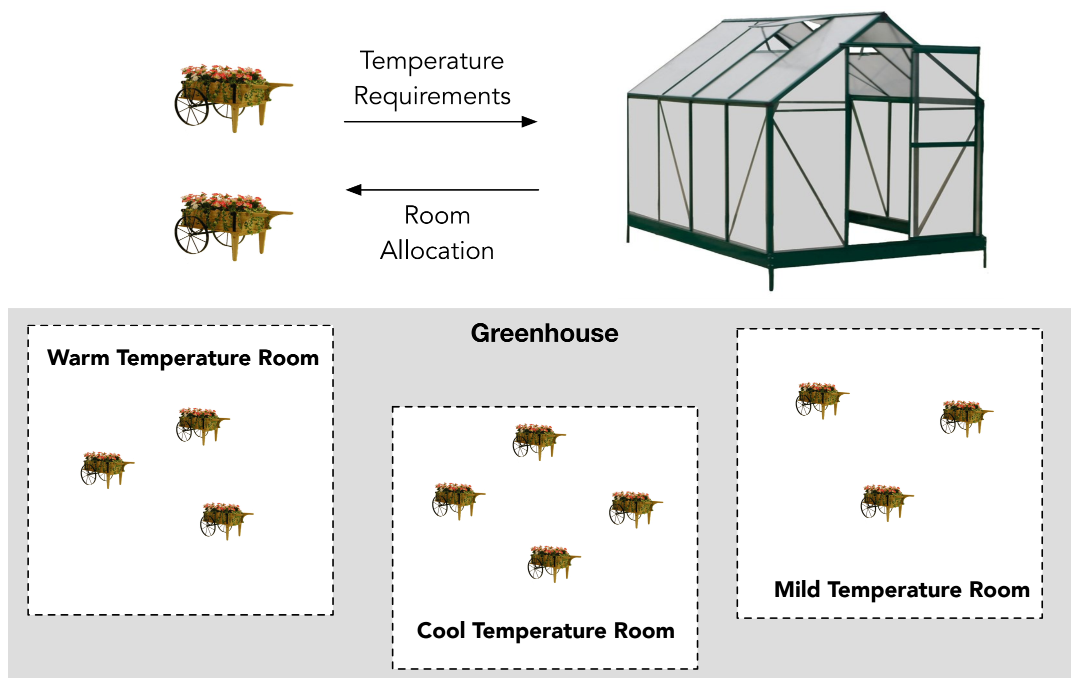


Figure 2.4: *Greenhouse.*

The carts in our system vary in terms of their APIs, and in particular in the structure of the messages they can send and receive. Some carts are fully automatic and do not require any human interface, while others require a human to operate them through a mobile device. The greenhouse is interested in separating plants that get sick from

those that are healthy. The example greenhouse requires the following features to be implemented:

- Automated configuration and management of various types of carts
- Enable effective communication among heterogeneous carts
- Manage carts with sick flowers/plants

Although it might sound like a simple example, we think it embeds many interesting characteristics that belong to different "more conventional" smart spaces such as self-configuration and interoperation among various components.

2.3.2 Smart Office

A typical example of our target systems is a Smart Office where different automation facilities are leveraged for the inhabitants to optimize building processes and user activities. In this thesis, we have used our Joint Open Lab (JOL), a collaboration between Telecom Italia and Politecnico di Milano, as our case study. The lab consists of an open space, a meeting room, demonstration area, two offices and a Kitchen (See Figure 2.5). Rooms of the lab are equipped with controllable lights, HVAC (heating, ventilation, air conditioning), and window shutters. JOL also has luminescence and temperature sensors, and BLE enabled beacons to collect the contextual information about the space. Each person entering the space has a badge (or a mobile device) to identify his/her presence in a room. The space has central HVAC system and one set of windows in Demo area wall. Open space has 12 work desks for the students whereas the two offices have 6 desks in total that are used continuously during the office works hours. There is one controllable light available for every two work desks. Apart from the daily work routine, there are some other scheduled activities that can take place on random days and time such as meetings, demos, and lunch breaks.

The overall goal of the system is to reduce light and heating energy consumption of the lab. The task is to control lights and HVAC in the lab according to the user activities with the aim of increasing energy efficiency. The lights within a room will be switched on or off according to the users entering or leaving the rooms (or the work desks). The workplace occupancy and the general events (meetings, lunch times, non-working hours etc.) are also considered to anticipate the lighting needs in particular rooms. The lights and shutters should also be adjusted based on the intensity of light, and on the activities being pursued. For example, lights can be dimmed, and shutters can be closed, during a presentation. Further, if there is enough daylight available in the lab (or at certain desk), there is no need to turn on the lights. Similarly, the second control aspect deals with the efficient use of HVAC systems. The temperature should be managed according to the thermal comfort level and global energy usage constraints. The thermostat in each room must adopt a compromise between a purely local policy and a building-wide solution, to avoid wasting energy with useless and dangerous spikes. We will use various aspects of this case study to demonstrate features of our proposed framework.

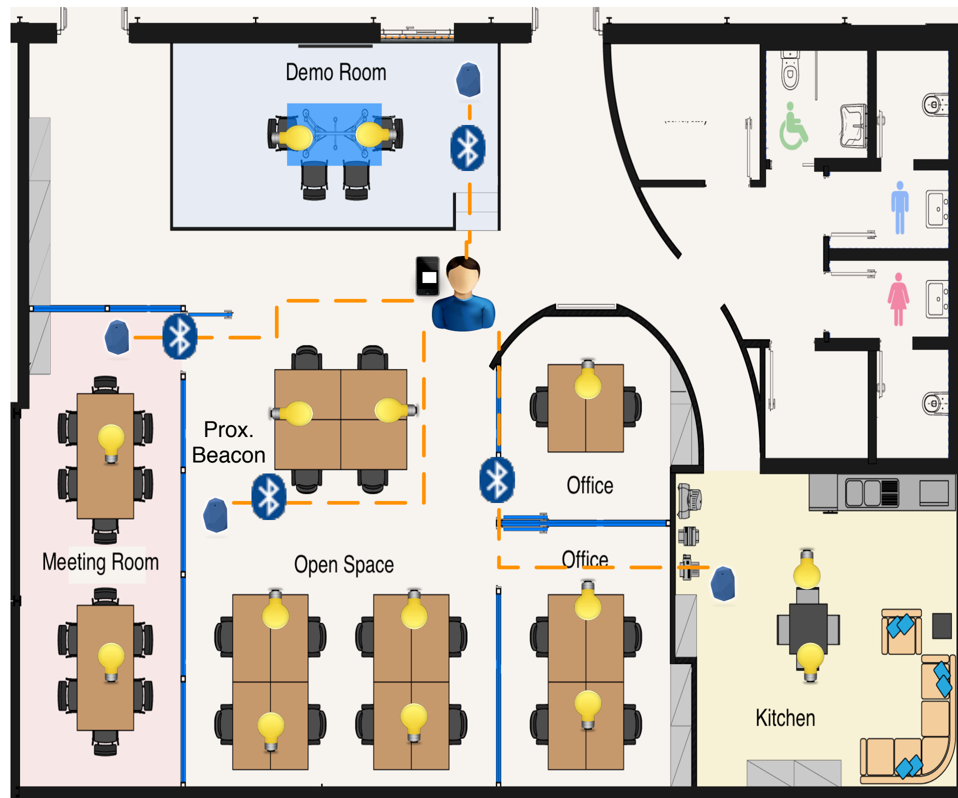


Figure 2.5: *Joint Open Lab Map*

2.3.3 Public Park

Figure 2.6 shows a public park that is divided into sections; each section comprises different attractions (e.g., ferris wheel, kids rail, and cafeteria). Thousands of visitors enter the park every day; each visitor carries a mobile device and is interested in different points of interest. The park is equipped with proximity sensors and large interactive screens that also work as access points for the users to connect with the space.

The park smart space groups the visitors with similar interests —within a certain distance— together by exploiting user profiles and contextual information from the proximity sensors. This special-purpose grouping can enable effective cooperation among the users with similar interests and location by providing efficient data dissemination. Every member is notified in a timely manner about important events and other recommendations so that users can act collaboratively and proactively. Moreover, visitors can also interact (or receive personalized information) with the screens about nearby attractions in the park according to their interests. The same smart infrastructure can also be used to coordinate and evacuate people in the case of an emergency by providing information about the safest and nearest exits.

Given this scenario, the park infrastructure must be capable of adapting to newly joined components and users, and also of self-organizing/self-configuring to ensure continuous service provision. For example, if a new screen is installed, it must become part of the infrastructure seamlessly, and users must be able to utilize it to exploit provided services and interact with the others.

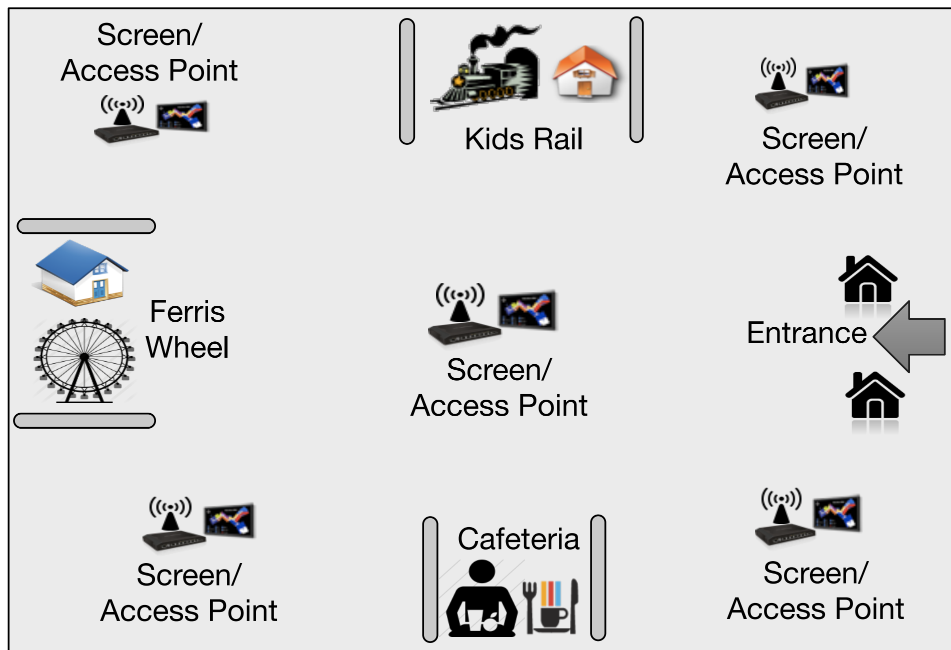


Figure 2.6: *Example park scenario.*

2.4 Properties

It is easy to understand that a home or office spaces are significantly different from a shopping mall or a car space in terms of (properties such as) size, mobility, scale and ownership, and hence, all of these spaces have different development requirements.

Following is the detail of how different smart spaces vary in terms of various properties:

- **Scope.** Personal spaces have limited/closed scope. Scope refers to the possible domain of objects and various protocols that can be employed by a space. Mostly, all the types of objects and users are known a-priori by the personal spaces such as smart homes, offices and other buildings.

  Social spaces, on the other hand, are usually characterized as dynamic spaces as they require management of large domain of objects and various protocols, which may or may not be known a-priori by the smart space.

- **Spatial Boundaries.** Personal spaces are private or restricted, bounded and indoor because they target to provide services to the users in a defined physical space.

  Social spaces are usually public and loosely (or un) bounded because they target to provide services to the users in a shared open place where they have no individual ownership of the physical space.

- **Scale.** Typically, personal spaces are smaller in size as compared to social/public spaces due to the private nature of their usage. They have very few users/inhabitants and the change in the number of users of these spaces does not vary frequently, in case it does at all.

On the contrary, social spaces are larger in size as compared to personal spaces due to the public nature of their usage. There are large number of users of these spaces, which enter or leave the space on frequent basis.

- **Interaction and Spatial Layout.** The distribution of the various components and user activities in personal spaces are usually based on the spatial layout (e,g. floor-plans), which rarely change. Most of the interactions take place between the users and the smart objects around him/her in the given locational context.

  The distribution of the various components and user activities in dynamic spaces are based on the situational (proximity and temporal based) layout, which changes continuously as the users become part of/leave these spaces on frequent basis. There are frequent user to user, and, user to other smart objects, interactions.

2.5 Challenges

The aforementioned properties of different types of smart spaces provide a list of requirements and challenges that need to be solved in order to realize these spaces. The development framework for smart spaces (with diverse properties) is thus required to cater for the following challenges (see Table 2.2):

- **Heterogeneity.** The solution must provide the appropriate abstractions to deal with the heterogeneity of the space components. The components vary in terms of their vendors, platforms they are developed on, and also the communication standards (WiFi, ZigBee, BLE etc.) they support. There is thus the need to support the use of existing IoT enabled objects ranging from sensing devices and smart plugs to home and office appliances.

  Social spaces are usually open systems composed of various subsystems (with varying execution models) operating on different protocols that must be blended together to make the whole system work. Therefore, the development solution should provide the right set of abstraction to cater for following *three dimensions of heterogeneity*: (i) heterogeneous components, (ii) heterogeneity of communication protocols, and (iii) heterogeneity of communication models.

- **Ad-hoc Integration.** Personal spaces require the integration of different components (sensors, appliances) capable of performing different tasks. Both the components and the integration protocol(s) are usually predefined, and the framework needs to provide the mechanism to integrate those components together by providing some simple contextual reasoning and automation (rules) mechanism. On the other hand, social spaces need more adaptive integration mechanisms to deal with the high dynamism of the components.

  Therefore, the system should facilitate the following aspects: (i) the seamless integration/replacement of components at run time, (ii) support for various coordination styles (patterns) for interacting subsystems, and (iii) concurrent and synchronous execution of subsystems with different execution models (time flows).

- **Situatedness.** Various components in the personal spaces have the notion of being situated in certain physical partitions (e,g. room, block, floor, building), which most of the times is very crucial to define how that particular component will behave or how it will coordinate with other components. There is thus this need for the development framework to facilitate and provide abstractions to model fixed situatedness of various components and design the interaction among them.

  Situatedness and location-awareness are very important aspects of social smart spaces as components are highly dynamic and mobile. They continuously enter/leave the space unannounced and change their locations frequently. The framework is required to provide abstractions where location and situatedness are considered as the focal points.

- **Dynamism.** The development of social smart spaces requires a self-configurable coordination mechanism of large numbers of heterogeneous components with high *churn rate* (may enter, leave or fail at any given time). The system should have the ability to self-adapt without (or minimum) human intervention.

- **Scalability.** As social spaces are mostly large and number of components can vary drastically any time, the development framework for these spaces must scale up with increasing (i) number of system components, and (ii) functional requirements (new features and applications) without incurring additional delays or degradation in performance.

- **Flexibility.** The framework should be flexible enough to enable the use of alternative subsystems (e,g. different protocols, simulators, etc.) to explore various possible solutions. It is important to be able to replace set of components with other component(s) without much effort and the need for changing the rest of the system.

- **Incrementality.** Incremental development of smart spaces requires the framework to provide the following features: (i) the continuous evaluation of the system by means of simulation and testing throughout the development life-cycle, and (ii) enabling co-existence of both physical and virtual subsystems at any given stage of development.

The challenges described above answer the first half of **RQ 1.1.** by listing out the requirements for a development framework for smart spaces. We will use these challenges to analyze the capabilities and shortcomings of the existing systems (Table 3.2) to answer the second half of **RQ 1.1.**. Later, we will use these challanges to guide the design of our proposed framework with the aim to bridge the gap in the existing solutions.

Table 2.2: *Smart Spaces - Challenges*

Challenges	Personal Spaces	Social/Public Spaces
Heterogeneity	✓ (Components)	✓ (Components + Protocols)
Integration	✓ (Pre-defined)	✓ (Ad hoc)
Incrementality	✓	✓
Dynamism	✗	✓
Situatedness	✓ (Fixed Locations)	✓ Proximity-based + Mobile
Scalability	✗	✓
Flexibility	✓	✓

2.6 Life Cycle of Smart Spaces

The realization of smart spaces is a multi-phase process where required infrastructure for device collaboration and smart services are designed and implemented. Typically, we have found the life cycle of existing smart spaces to have the following four major phases (See Figure 2.7), where each of these phases is managed in an independent manner:

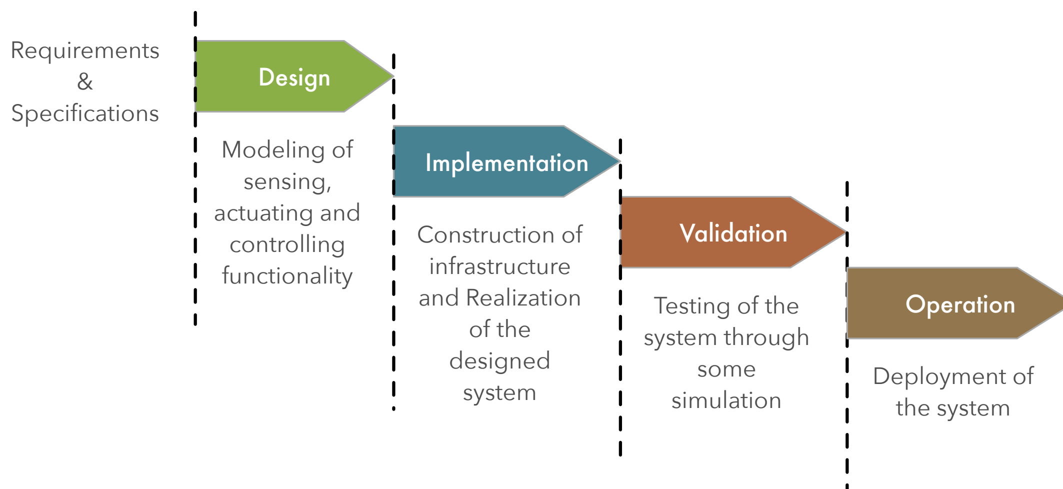


Figure 2.7: *Smart Space Life Cycle*

Design. Design is the first phase that is executed after one establishes the requirements and specification of the system to be. The design phase comprises the following tasks in accordance to scenario-specific requirements:

- Identification of the different constructs (sensors, actuators, controllers, services, etc.) of the smart space
- Definition of the relationships and interactions between those constructs
- Description of continuous services provision to the users in result of those interactions

In this phase, it is analyzed how and what kind of sensors and actuators are needed to be deployed in different spatial locations within the space. Moreover, the required user services, which the space is expected to provide to its inhabitants, are also elicited. As smart space are dynamic and service oriented in nature, it needs to be able to adapt itself according to the changes in the behavior of the users or the space itself. Therefore, after deciding about all the sensing/actuating elements and the required services, one needs to define how different elements will interact with each other in order to provide required contextualized services to the users of the space.

Implementation. The implementation phase encompasses the construction of (both software and hardware) infrastructure. It defines the mechanisms that are required to translate the design into the actual implementation of the system. The ease of this translation is very crucial and it is important to have the same abstractions used for both the design and implementation of the system as it will have a significant impact on the maintainability of the system in later phases. This process may use some programming languages, middleware infrastructures or the support of various development frameworks.

Validation. Validation is one very important phase in the life-cycle of smart spaces as the development process requires integration of large number of heterogeneous elements and devices with each other. In the absence of appropriate validation mechanism, various problems may be discovered in later phases while (deployed and in operation) would be very expensive to be fixed [69]. One of the major reason for that is the inability to generate real environmental settings for evaluation of certain scenarios.

For instance, it is not trivial to create situations to test emergency evacuation or crowd navigation in large dynamic spaces and use of simulation frameworks becomes necessary. Simulators help verify and validate a certain set of features and functionalities of smart spaces without the need to set up real sensors and devices [57]. In this way, evaluation of alternative design decisions can be made to optimize the efficiency and correctness of a smart space.

Operation. Smart spaces evolve over the time due to ever changing and emerging nature of technologies [57]. The technological changes and rapid advancement bring many maintenance challenges for the smart spaces that are already in operation. The operational issues encompass a wide variety of tasks such as replacement of devices and other components, software updates, changing the control logic and other spatial and logical alteration of the system elements. Moreover, as users are the focal point of these spaces, their behaviour, expectations and preferences may change over the course of time. This change, therefore, may trigger the need for adaptation in the existing infrastructure, services or other devices within the space [32].

These are the phases, which are typically used for the development of smart spaces. Most of the existing solutions execute these phases in a linear manner, that is, first the system is designed and implemented. Later, it is validated through some validation tools (mostly simulators) and then physically deployed for the operation. These processes are independent of each other and system is either executed either completely deployed or totally virtual (simulated). This linearity of development phases is a prob-

lem for achieving one of the challenges discussed earlier, that is , *incrementality*. The issue is discussed later in Section 4.1.1.

CHAPTER 3

State of the Art

This chapter surveys the state of the art in the field of smart pervasive spaces (environments) and highly dynamic software systems. Researchers have proposed and developed numerous solutions for the design and deployment of smart environments. Existing solutions and proposed approaches address one or a limited subset of the challenges discussed in the previous section. The survey analyzes the existing literature into two important dimensions:

1. The general approaches and paradigms are discussed in order to understand how they tackle various challenges in the development of smart environments and dynamic systems.
2. Existing concrete solutions, which target to solve one (or more) aspects/development phases of the smart spaces, are discussed along with their short comings.

The discussion is concluded by a comparative analysis of surveyed solutions according to the challenges identified in Section 2.5.

3.1 Approaches

We find various approaches in the literature that have been employed to propose solutions for the creation of smart spaces. Among these, architecture-centric approaches, multi-agent systems (MAS), and nature-inspired computing (NIC) seem to draw more of the researchers attention as compared to other paradigms.

These different research communities target the problem of developing smart pervasive environments (systems) from different perspectives. For instance, software architecture and autonomic computing researchers propose solutions that focus on providing adaptation mechanisms on top of these dynamic and continuously changing systems.

Typically, they provide centralized control loops to monitor the system state and suggest some adaptation or re-configuration directives accordingly. MAS, on the other hand, present a more open and loosely control solution where autonomous agents take local decisions and adapt to their environment. Hence, the multi-agent community provides solutions which have more distributed adaptation mechanisms as compared to autonomic software approaches. NIC puts forth a radically different approach where these systems are modeled on the principles borrowed from the self-organizing natural system and focus shifts from control adaptation to emergence that leads to continuously evolving systems.

We analyze these approaches in terms of the system design, control, organization/-coordinations styles and adaptation mechanisms.

3.1.1 Architecture-Centric Approaches

Architecture-based adaptation is mainly concerned with structural changes at the level of software components [36, 99]. A component should have the ability to configure itself in a manner that enables it to interact with other components and contribute towards achieving the general goal of the system. The architecture of a software system describes its structure that highlights the high-level design decisions about how interacting (system) elements are composed and what interaction principles are employed by them, along with the key properties of both the participating components and the system as a whole [46, 49]. A software architecture also provides the global system level perspective and reduces the complexity of a system through abstractions and separation of concerns and provides a common understanding of system components and their interactions. Moreover, architecture enables the understanding of system's topological and functional constraints at a higher level and hence provides a better way to ensure the validity of system with changing needs [99]. The goal of architecture-based adaptation is to minimize human intervention for managing the system in a way that system should be able to organize itself according to the architectural specification [50]. This makes architecture-centric approaches a popular choice for building self-* systems, and therefore, many researchers from different domains have proposed this type of solutions for conceiving highly dynamic systems.

Architecture-based adaptation offers various benefits [66, 77]:

- The underlying architectural concepts and principles are applicable to a wide range of application domains, which enable software architectures to be the general solution for building systems with different needs.
- It can provide an appropriate level of abstraction to describe the dynamic change in a system by using components, bindings, and composition, rather than handling it at the algorithmic level.
- Architectures generally support various kind of component compositions which are very useful for the development of large-scale complex applications.
- There are lot of architecture description languages and notations which include some support for dynamic architectures and for formal architecture-based analysis and reasoning

Self-adaptation and Organizational styles

There exist a number of architecture-based solutions for pervasive systems, which employ a wide variety of architectural styles, properties, and external control mechanisms. Two of the commonly adopted architecture-based adaptation approaches are: goal driven self-adaptation and model-based self-adaptation.

Goal driven Self-adaptive Architectures

Kramer and Magee proposed an approach [55,66] for self- management at the architectural level in which components configure their interactions autonomously to be compatible with the overall goal of the system. A three-layer reference model is introduced that consists of component control, change management, and goal management.

The component control layer contains a set of interconnected components where each component implements a set of provided services and requires some services to be implemented by other components. In addition, a component has externally visible mode, which is an abstract view of the internal state of the component. Some component level operations are also provided such as creating or deleting components, binding or unbinding connections, and setting values for component mode.

Change management layer is responsible for handling changes by receiving the reported changes in states from the lower level or changes in goals at higher level. This layer contains a set of precompiled set of plans and strategies for predicted class of changes.

The goal management layer processes the state of the system and the goals specification, and tries to generate a plan to achieve that goal. The reference model suggests various research challenges in different layers of the proposed self-management architecture. At the component layer, the most important challenge is the provision of change management capable of component reconfiguration to avoid the undesired transient behavior. Decentralized configuration management that provides the resilience in case of any inconsistent system state, is an important challenge to be managed at change management layer. Similarly, the goal management layer requires constraint-based planning to translate goals into plans.

FlashMob [91], for example, incorporates an aggregate gossip protocol to enable various components of a system to agree on a global configuration (goal). In the proposed solution only one or few (for greater performance) knows about the structural constraints, whereas, all of them are aware of functional and non-functional requirements. On instantiation, components are connected automatically by the specified configuration rules.

External Model-based Self-Adaptation

Many researchers have proposed architecture-centric adaptation approaches in which system architectural models are maintained at runtime and used as a basis for system reconfiguration [34,77]. An architectural model of a system externally captures the overall structure (composition) of interacting components within a running system [48,85]. The external model-based adaptation, such as the one described by Rainbow framework [47], provides reusable infrastructure with some mechanisms for customizing it according to the needs of specific target systems.

The framework is usually constituted of adaptation infrastructure and system-specific adaptation knowledge. The adaptation infrastructure provides some common adaptation functionalities that are reusable across all target systems whereas the adaptation knowledge is always specific to target systems. Probes and Gauges observe the running system and report the observation to the model manager that updates the architectural model. Rainbow, for example, uses the architectural building blocks to create an architectural model by abstracting the behavior of target system. The provided building blocks are component and connector type, constraints, properties, analysis, adaptation operators, and adaptation strategies. The architecture evaluator checks for any violation of constraints and signals the adaptation manager to select an appropriate adaptations strategy. The selected strategy is then executed on the running system, and the action is typically accomplished via system-level effectors.

Architectural Patterns/Styles

The architectural-centric approaches employ different architectural patterns and styles to provide self-adaptive software solutions. A common approach is to exploit a service-oriented infrastructure to provide the right abstractions over hardware (device) heterogeneity and configuration alternatives [92]. The other popular choice is the use of event-based architecture [87, 90] to model the behavior of self-adaptive systems. Many of these approaches make use of publish/subscribe mechanism to manage services or events. We will discuss some of these solutions in the section. 3.2. The problem with these approaches is that services are considered the loci of all the functionality and they do not offer spatiality and situatedness as primary abstractions.

Autonomic computing approaches [18, 39] aim to solve these issues by embedding the adaptation logic within the architecture that monitors the changes in the system and suggests re-configuration (through control loops) accordingly [106]. For example, Gurgen et al. [53] propose an approach for building self-aware cyber-physical systems for smart buildings and cities. The downside of such solutions is that adaptation logic would have to be very complex and heavyweight to ensure the capability of adapting to any foreseeable situation, and especially hard for long-term adaptivity. Moreover, as systems with high churn rate require frequent access to discovery and orchestration mechanisms to get updated contextual information, the extraction of situatedness and proximity related information would be costly.

3.1.2 Multi-Agent Systems

Multi-agent systems (MAS) is another paradigm that has emerged as a preferred precedent to design and develop complex software systems [30]. A multi-agent system typically consists of many collaborative autonomous software agents that work together to achieve overall system goals.

An *agent* is a software entity with human-like properties such as autonomy, reasoning and learning, and sociability through high level knowledge exchange. A widely accepted definition of an agent is defined by Russell and Norvig [82]:

"*An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators*."

From the technological perspective, these software agents are self- contained programs with the ability to control their decisions and actuation with respect to their perception of the environment at a given time in order to achieve personal objective(s) [103]. Whereas from the functional perspective, they can be viewed as software entities that can perform various tasks delegated to them in pursuit of achieving the greater system level goal. This ability of agents convinced researchers to employ multi-agents to design and develop automation solutions and smart environments. MAS provide the following beneficial features for the development of dynamic software systems:

- The agent's inherent capability of monitoring the environment and autonomous decision making is suitable to design dynamic systems with frequent changes.
- Agents have the ability to communicate/collaborate with other agents through high-level asynchronous knowledge exchange that is required for the better understanding of the environment and leads to more informed actuating decisions.
- MAS provide pre-define mechanisms for defining various behaviors and learning methodologies that ease the implementation the application logic of the system.
- MAS also support re-configuration and self-organization in order to recover from partial failures.

Standards. Multi-agent systems got a lot of attention during the 1990s, and many platforms were developed for agent-oriented programming. In result of the growing popularity, the Foundation for Intelligent Physical Agents (FIPA) was formed in 1996 [76] with the aim to promote and standardize agent-oriented systems in order to enable its inter-operation with other technologies. These standards include specification for agent communication language (ACL), agent architectures and their management policies.

JADE (Java Agent Development Framework) [16], for instance, is one of the most widely used distributed agent platform, that is fully compliant with FIPA specifications. It provides developers with a toolkit comprising of agent-platform runtime, Java libraries to create agents and debugging tools. It also provides built-in behavior templates (classes) that can be used by each agent to implement its behavioral logic and capabilities. The message based communication among these agents are ACL (FIPA standard language for communication) compliant.

Self-adaptation

Given that the agents are autonomous goal-directed entities. In a multi-agent system, many agents collaborate to achieve certain goals (tasks) of the system and each agent get its adaptability from these goals [102]. Typically, any goal or task can be achieved dynamically at runtime by selecting from the set of multiple available agents without the need to define exact configurations at design time.

To understand this, let us consider an e-market place where sellers provide a virtual shop for the buyers to purchase items online through delegating requirements to software agents. These agents could negotiate and form a collaboration at runtime based

on the availability of seller (product) availability and certain preferences of the collaborating parties. This goal-directed behavior is dependant on the agent's internal state and processing, which is usually designed on BDI (belief, desire, intention) model. An agent can, therefore, enhance its self-adaptive capabilities by extending its BDI models. This ability provides agents with the flexibility that is required for the self-adaptation.

Adaptive task coordination and management in MAS is a complex issue, particularly in systems where operating conditions change frequently. Approaches such as DynCNET [101] (extension of classic CNET protocol [88, 89]), and FiTA [100] aim to solve the issue of adaptive task coordination in agent-based systems. DynCNET uses the explicit negotiation to assign various tasks to the agents, whereas, FiTA is a field-based approach where tasks emit fields in the environment that guide free agents to perform different functions and tasks as per the requirement at hand.

Organizational Styles

Most of the initial MAS proposals for creating smart spaces and intelligent environments are based on the principle of autonomous individual agents that act to achieve personal objectives, which are in turn designed to move the environmental state towards the system level goal [60]. These systems provide a distributed framework and sophisticated message based communication protocols. However, in order to be an effective paradigm for the development of such dynamic spaces, MAS should be able to work at various abstraction levels and must support openness of the system.

To this end, some middleware platforms [84] were developed for the conception of ubiquitous computing applications with the aim to provide abstractions for sensors, actuators and other elements within a system. These system elements are represented by an agent that acts on its behalf in the MAS. These systems are implemented through development platforms such as JADE, where all agents reside within a runtime environment, and this limits the collaboration between different subsystems. For this, SETH [72] puts forth the concept of hierarchical aggregation and organization of multiple agent-based spaces and the mobility of the agents residing within those spaces.

ASPECS [30] provides a software development process for creating complex systems. It advocates the concept of having agent *holons* for the organization of autonomous agents. A holon is a system (or phenomenon) which is a self- organizing structure, consists of agents or other holons.

The MAS approaches present various solutions for the component organization in dynamic systems, but there are various issues that these approaches do not address:

- Many of the MAS approaches are local, and they do not provide the developer with the ability to define the desired control mechanisms at various levels.
- Most of the agent-based systems still face one issue, and that is the effects of their autonomous interactions are uncertain and hard to predict at the run time.
- They, usually, lack a systematic engineering or architectural view, e,g., ability to define elements required for (agent-oriented) architecture to deal with agent's particular role in certain collaboration or organizational structure.

3.1.3 Nature-Inspired Computing

Over the last few years, there has been a shift towards natural immune systems [21] and bio-inspired approaches that provide inherent adaptation and self-organization capabilities to the components of the system.

Many researchers observed that the complexity of modern pervasive systems is comparable to that of natural ecosystems. Agha [10] suggests the use of natural systems (e,g. physical, chemical, biological, or social) as inspiration to re-model the architectural design of these systems instead of complicating existing solutions. The design of these systems is driven by the idea that instead of putting design time relationships (or static architectural design) among the component of a dynamic system, we should take a more flexible and autonomous approach, similar to the one we observe in highly dynamic massive natural ecosystems. It implies that system will be constituted by fully autonomous agents (having their own beliefs, goals and intentions) and they will operate under the high-level eco-rules that will govern their behaviour within the system.

System Organization and Self-adaptation

The autonomous components of natural systems (e,g., ant colonies [40], or flower pollination [105]) are inherently situated in the space and their behavior is guided through the interaction with other components around them under natural laws. Therefore, the use of natural/bio-inspired systems can provide the following benefits:

- All the system activities and components are inherently situated in a space/location and they are triggered by only the local interactions. This property is naturally suitable for highly dynamic smart spaces.
- The interactions in a natural system are not the pre-defined orchestrations. Typically, they follow a very limited set of general natural laws, which enable them to self-organize in an evolutionary way.
- Due to the loose control rules, they have the inherent capability to adapt according to natural patterns, and hence, they can reconfigure their structural formation in a given space according to the environmental needs.

NIC is relatively a new paradigm (approach) as compared to its counterparts (architectural approaches and MAS), and therefore, different proposals and research perspectives exist on how can one build software systems modeled on the principles of natural systems. The following is the brief description of some of the significant perspectives in this area:

Variants and Reserach Directions

Zambonelli and Viroli [106] highlight that biological metaphors are suitable for modeling the spatial relationships of components and enable both localized and distributed social behaviors. They introduce a reference architecture for implementing bio-inspired pervasive ecosystems where service components are described as *species*, chemical or biological *signals* are spread to and perceived from the environment and used as interaction mechanism, *eco-laws* define the interaction rules for the species, and the *space* itself is the software infrastructure.

Fernandez-Marquez et al. [43] present the notion of *core bio-inspired services*, which provide various bio-inspired mechanisms, such as evaporation, aggregation or spreading to be used in various applications. The aim of their work is to ease the design and implementation of the self-adaptive system by enabling reuse of various reoccurring behavior in different applications. The proposed solution is an execution model, namely *BIO-CORE* that provides these bio-inspired patterns as services to be used by any other middleware framework.

Di Nitto et al. [37] investigated how decentralized bio-inspired self-organization algorithms can be used/exploited to develop distributed software architectures for dynamic systems. They studied how various bio-inspired principles (such as emergence, evolution, etc.) can be used to develop the heterogeneous decentralized load balancing in the self-organizing system topology.

Many pervasive frameworks [24, 62] borrow their key principles from natural systems and focus on providing localized self-adaptive capabilities to conceive dynamic software systems. SAPERE [24] is the chemical-inspired middleware solution which is proposed with the aim to address the dynamism in large number of (mobile) components. It models a pervasive service framework as a distributed MAS such that the coordination among various application agents is based on their situated interactions.

Another interesting work in this area introduces the concept of *augmented ecologies* [94]. It also takes inspiration from natural ecosystems and maps components of a pervasive environment to a virtual ecosystem of organisms interacting in a spatial and context-dependent way.

The above mentioned approaches present some very novel and useful concepts for the development of smart environments and dynamic software systems. However, there are certain issues that need to be considered:

- Local decision making and self-organization fail to guarantee the desired control over self-* mechanisms to ensure reliable, contextualized services to the users.
- This is an evolving paradigm and more rigorous experimentation and evaluation is required (to prove its ability to converge in very large systems) in order to use it for diverse real systems.

3.2 Solutions

The second dimension, which we use to study existing systems, is to analyze the type of solutions proposed by the researchers. It is interesting to note that various solutions employ different approaches (described above) and target certain type(s) of smart spaces (discussed in Sec. 2.2). Moreover, these solutions provide support for different phases of space development life-cycle.

Current systems include some ready to use solutions for fixed spaces, middleware infrastructures, comprehensive development toolkits and solutions, validation tools, and few integration platforms. We observed that, mostly, fixed or ready to use solutions (integration hubs) are employed for static spaces such as smart homes, offices and assisted health-care. As one moves towards more mobile spaces and scenarios, such as large events and stadia, middleware infrastructure or other development frameworks and tools are proposed that require scenario-specific customizations by the developer.

We present a brief summary of these solutions along with their shortcomings in solving challenges defined in the previous section. Table 3.1 lists existing solutions according to the approach they adopt, whereas, Table 3.2 compares few of the *representative* solutions according to the described challenges.

3.2.1 Fixed Indoor Deployments

Many indoor (fixed) spaces such as *smart homes, offices*, and *hospitals* have been proposed and deployed with the aim to deliver corresponding domain specific customized services to the inhabitants [13, 58]. An industry report [63], published by Raymond James and Associates, shows the growing trend of smart homes and e-health deployments over the last few years. Microsoft Easy Living project [15], iDorm [60] and Gator Tech Smart House [56] are examples of significant smart home and assisted living architectures that laid the foundation for succeeding systems such as [97]. Easy Living and iDorm employ the architectural approach, whereas, Gator is an agent-oriented solution. Shamim et al. [62] present an ant-based (bio-inspired) service selection framework for a smart home monitoring environment that enables different residents of a home to access various media services in such a way that their experiences are optimized.

These systems focus on providing abstractions over the heterogeneity of different sensors, home appliances and communication protocols that are known a priori. They do not support the co-existence of simulated and physical objects in order to provide incremental development of the system, which results in costly updates in case of changing components types or adding new devices. The reason for this is the inability to validate the solution through simulations and partially deployed systems.

3.2.2 Automation (IoT) Hubs

Besides these deployed spaces, there is now a growing trend of developing *ready to use* solutions (sometimes refered to as IoT or automation hubs) [75]. The idea is to provide users with a pre-configured central controlling hub that is capable of interacting and connect various smart objects present in a fixed space (homes in particular). Many large manufacturing companies have launched similar products such as Haier (U+ Smart Living) [4], Samsung (Smart Home) [7], Google (Nest and Revolv) [3] and Apple (HomeKit) [1]. Most of these systems follow the architectural approach and make use of cloud infrastructure to deploy their services.

Most of these systems target to solve two of the challenges for fixed spaces: (i) connecting heterogeneous components (smart objects) with different communication standards, and (ii) providing automation rules for their integration.

3.2.3 Integration Platforms

There are some other systems that take a more practical approach towards the development of smart spaces. Unlike the ready to use solutions, they provide frameworks and platforms to develop smart spaces by providing support to integrate different smart objects and existing protocols for device communication. Freedomotic [2] is one of such mashup-oriented platforms that facilitate the development and integration of standard building automation protocols to realize smart buildings. Freedomotic is very useful for building automation and supports the integration of external objects, but it lacks

user mobility and self-organization mechanisms to deal with dynamism. OpenHAB is another similar [5] integration platform to connect heterogeneous components in smart homes and offices context. These systems follow architecture-centric approach to integrate different smart objects together and define rules on top of that.

BOSS [35] provides a set of building operating system services to program various fault-tolerant applications on top of distributed components in large buildings. It offers abstractions to define building resources, real-time processing and a semantic model for implementing the communication among the components.

Ambient Dynamix [22] is a context framework (for Android) where sensing and acting capabilities are deployed on-demand on mobile devices to let the devices adapt to the environment. Vykon [8] is another integration solution for building automation that supports communication and control of devices running on heterogeneous protocols. AllJoyn [6] is a platform that supports ad hoc communication and resource sharing between smart devices in a physical proximity. It is independent of the operating systems and wireless network protocols used by the participating devices.

These platforms can be used to bridge the gap between physical devices and the corresponding virtual components defined to model the execution and interaction of these devices within a space.

3.2.4 Middleware Infrastructures

A lot of work has been done to provide a middleware infrastructure for enabling the development of pervasive systems in general, and smart spaces in particular. In literature, we find that all three (described) approaches are adopted by researchers to design middleware infrastructure.

Most of the earlier work in this field has been done to devise service-oriented middleware solutions [92]. AMIGO [71] is a middleware that operates over different application domains and support context information to facilitate users with intelligent services. The AMIGO middleware is platform-dependent and also lacks the support for heterogeneity and fault tolerance. SOCRADES [20] is a service oriented middleware that provides a service-based interface to interact with heterogeneous devices over the network. It offers a sophisticated event-driven messaging system. It does not, however, support contextual information, and does not show how the system will scale and organize itself in dynamic environments. Reyes and Wong also propose a service-oriented middleware [81] for integrating various sensors and actuating devices. The proposed middleware abstracts heterogeneous devices as services that can be invoked by other devices. AlfredO [80] provides an OSGi-based architecture for flexible interaction with electronic devices to construct applications in a modular way. The capabilities of devices are represented as services that can be accessed dynamically by mobile phones. RUNES [31] offers a publish-subscribe middleware solution for wireless sensor networks and embedded systems. It offers dynamic reconfiguration capabilities to cater heterogeneity, resource scarcity, and dynamism in embedded networks. Although it does not aim to provide a whole development framework for smart spaces, it has useful device coordination mechanisms and abstractions that can be used to program primitive devices.

From Table 3.2, it can be observed that these middleware solutions do not support continuous validation of the designed system by allowing a mix of physical and virtual

Table 3.1: *Approaches and Solutions*

Approach	Solutions				
	Fixed Deployments	**IoT Hubs**	**Integration Platforms**	**Middleware Infrastructures**	**Complete Solutions**
Architecture-centric	EasyLiving iDorm	Samsung Home Apple Kit	Freedomotic OpenHAB Vykon	SOCRADES AMIGO AlfredO RUNES	DEECo DiaSuite
Multi-agent	Gator			MUSA SETH Ubiware	ASPECS
Nature-inspired	Ant-based Smart Home			SAPERE Middleware	SAPERE

components. Further, they do not provide any mechanism for the synchronization of the different subsystems with different time flows.

Researchers from multi-agent systems (MAS) community have also proposed solutions for the development of smart environments. For instance, iDorm project aims to realize intelligent healthcare environments by associating embedded agents with various sensors and actuators [60]. Ubiware [84], a middleware platform, is developed for the conception of ubiquitous computing applications. It aims to enable the creation of self-managed complex systems in general and, industrial systems in particular. Each of the sensors, actuators and other elements within a system is represented by an agent that acts on its behalf in the MAS. The middleware provides declarative rule-based language (S-APL) to define these agents.

ASPECS [30] is an agent-oriented software process for complex systems but it also provides a self-organizing middleware for user-driven service adaptation. SETH [72] puts forth the concept of aggregation of multiple agent-based spaces and mobility of these agents. The architecture supports a layered organization to create complex smart environments by means of inheritance and aggregation relationships. It assumes a well-defined system requirements to work with and is unable to adapt itself to evolving user needs and system requirements. Therefore, it is not a feasible choice for real-time dynamic and large scale systems.

Similarly, there are some nature-inspired middleware frameworks (e.g., SAPERE [24]) proposed with the aim to address the dynamism in a large number of (mobile) components. This is an evolving paradigm and more rigorous experimentation and evaluation is required (to prove its ability to converge in very large systems) in order to use it for diverse real systems.

3.2.5 Complete Development Solutions

ASPECS [30] defines a comprehensive agent-oriented process for engineering complex systems. It proposes a holonic organizational style, and also provides tools for all the development phases. It offers abstractions to design the system from scratch and does

not offer integration framework to use existing solutions. DEECo [19] is an ensemble based component system where an ensemble represents dynamic binding of a set of components and thus determines their composition and interaction. DEECo provides useful tools for the design and deployment of the system, but it lacks the ability to integrate external systems and does not support continuous evaluation by enabling both virtual and physical components to coexist. Further, it does not have an optimal knowledge exchange mechanism and components share all the data while communicating irrespective of the needs. DiaSuite [23] is a development framework to build pervasive control systems and offers the following features: a language to design system taxonomies (through entities), an architectural pattern to model application specific data-flows, and an embedded simulator called DiaSim that is a customization of Siafu. This methodology provides developers only with the fixed entities like contexts, controllers, and a particular context simulator. Further, it does not deal with mobility and dynamism of the system component and lacks the support for coexistence of various physical and virtual subsystems together.

3.2.6 Validation Tools

Besides all these concrete solutions, we think that it is important to mention various validation tools that are also available for evaluating the system performance. Usually, different simulation frameworks are used for mimicking the behavior of the components (in a virtual environment) to understand how the system will behave in the real scenario. Siafu [73] is a context simulator that provides a way to generate different contextual information in a space. It also supports the modeling and visual simulation of the dynamic user behavior within the space. Few other simulation frameworks [68,78,96] for creating virtual smart homes also exist. EnergyPlus [9] is another widely used system that can be used to simulate the energy needs within a building under given conditions. Ptolemy II [67] is an actor oriented open source framework to model and simulate different models of computations through defined interfaces and timed synchronization. Ptolemy II is primarily a tool for experimentation and does not provide any deployment and run-time support for the designed systems. Further, the models are defined at the design time, and they do not support any change, reconfiguration or dynamic behavior of the involved components.

3.3 Comparative Analysis

The first dimension of analysis is the adaptation approach that is employed by various research communities and present solutions. Majority of these solutions are developed through centralized architectural control loops and hence they are prone to have scalability issues. To deal with these issues, multi-agent systems proposed to use autonomous agents that collaboratively form distributed systems with runtime self-adaptation delegated to individual entities of the system. These agent-based systems, then, laid the foundations for bio-inspired self-adaptive eco-systems, which are proposed to move towards completely decentralized and autonomous solutions. Although, all these approaches are interesting and useful for certain kind of scenarios, there is a need for decentralized, yet manageable solutions.

Secondly, Table 3.2 presents a comparison of the **representative** systems from each

class of solutions (approaches) with respect to the challenges identified in Sec. 2.5. Following is the brief summary how different solutions address those issues:

Heterogeneity. It is observed that most of the existing solutions cater for the heterogeneity in terms of component types and communication protocols. However, almost all of them (Ptolemy being the exception) does not explicitly provide mechanisms to handle system elements with heterogeneous execution models.

Situatedness. Although, there are solutions that provide fixed situated properties and location management, situatedness and mobility are not provided as first class abstractions by any of the systems, except nature-inspired eco-systems (e,g. SAPERE). This is very important property of a system where components are highly dynamic, and they frequently move around the space. The mobility changes the context, and hence, requires the spaces to adapt accordingly to provided location aware services. This aspect is ignored by many systems and hence it is required to be incorporated in the development solution.

Ad-hoc Integration. All the surveyed systems provide the integrated automation of various components of the smart space. DiaSuite and Ptolemy also provide support for the integration of heterogeneous coordination patterns for interacting components/-subsystems. Although, Ptolemy is the only solution that allows concurrent and synchronous simulation of subsystems with different execution models, but unfortunately it is a simulation tool and used only for the validation (and not the deployment) of the system.

Dynamism. Most of the systems lack the ability to cater for the highly dynamic nature of components. Among all the solutions, only SAPERE, DiaSuite and ASPECS provide explicit mechanisms to deal with the dynamism. They offer sophisticated self-organizing (re-configuration and self-healing) mechanism that are capable of adapting system without human intervention.

Scalability. Most of the existing solutions deal with adding new features and requirements, but they do not address the issue of scalability in terms of very high number of components. SAPERE approach provides enough evidence of catering with the issue and their design choice inherently deals with large self-adaptive systems.

Flexibility. Many of the surveyed systems are flexible when it comes to change or re-configure some components of the systems, but there is not much support available for the replicability of different system elements to enable the testing of various alternative subsystems.

Incrementalitly. This is the property that is lacking in almost all the existing solutions as they do not support the incremental development and continuous validation of the system to be designed.

The survey shows that most of the existing solutions and approaches are targeted

to solve a particular subset of the problems (challenges) related to smart space development. One can make the following high-level observations from the comparison presented in this chapter:

- Each of the employed design approaches has some relevant features that are useful for the development of the systems under consideration, yet they fall short in providing a holistic solution that covers all the challenges.
- These solutions, many of which are useful in limited scenarios, work in isolation and do not make any effort to integrate other developed elements to support the complete life-cycle of these systems.
- This isolation results in a sequential development process as one has to develop the complete system, and then the system is validated through simulators.

We, on the other hand, claim that all these solutions should be employed within a development framework and integrated through the same middleware to allow continuous evaluation of the system while moving seamlessly from design to actual deployment. Given the wide range of external simulators, communication protocols, distributed technologies, a useful framework should be more generic and be able to interact with almost any external system or simulator. To conclude (the answer of **RQ 1.1.**), the existing systems and approaches fall short in following aspects:

- Provision of a generic framework that is applicable for all diversified spaces and that supports all the development phases for the creation of smart spaces.
- Lack of balance between the control and the autonomy of smart space components
- Inconsideration towards inherent evolutionary nature of the development process of the smart spaces
- Inability to define/control ensembles of components at various granularity levels.

Challenges		IoT Hubs	Middleware Infrastructures				Methodologies			Integration Platforms	Simulators
		[75]	[20]	RUNES	SETH	SAPERE	ASPECS	DEECo	DiaSuite	Freedomotic	Ptolemy II
Heterogeneity	Components	✓	✓	✓	✓	✓	✓	✓	✓	✓	
	Communication	✓		✓		✓	✓		✓	✓	✓
	Models							✓			✓
Adhoc Integration	Automation	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Patterns								✓		✓
	Time Flow										✓
Scalability	Components					✓					
	Functionality		✓		✓	✓	✓				
Incrementality	Coexistence									✓	
	Evaluation										
Dynamism	Churn rate					✓		✓	✓		
Flexibility	Configurability			✓		✓	✓	✓	✓		
	Replacability								✓	✓	✓
Situatedness	Proximity-based					✓					

Table 3.2: *Comparison of most relevant existing systems*

CHAPTER 4

Proposed Framework

This chapter discusses the proposed perspective shift in the development of dynamic smart spaces. It also describes a comprehensive development **framework** (Figure 4.3) to design, prototype, and assess alternative solutions for smart spaces. The framework provides design abstractions to abstract over the heterogeneity of devices and manage the intricacies of distributed components. It also leverages effective architectural coordination styles and integration mechanisms to facilitate the incremental validation and deployment of the space.

4.1 Revisiting the Perspective

Two major revisions (in answer to **RQ 1.**) of the current approaches are proposed and advocated in the thesis. The first one is the **incremental development** (Figure 4.1) of the smart spaces where different phases co-exist with each other and where it is possible to work with 'incomplete' systems at any given type. The second proposal is related to self-organization/adaptation mechanisms. We advocate that there is a need to have a self-adaptive approach that takes the best of the existing software architecture control loops and bio-inspired emerging algorithms to develop a system, which has both the control over system processes and the certain level of autonomous emergence properties.

4.1.1 Incremental Development

It is important to re-iterate the fact here that it is very hard to predict at the design time about the sensors, actuators, controllers and data that are required for the space-to-be. This is true because of the following reasons:

- Change in functional or service requirements as an overall evolution of the system.

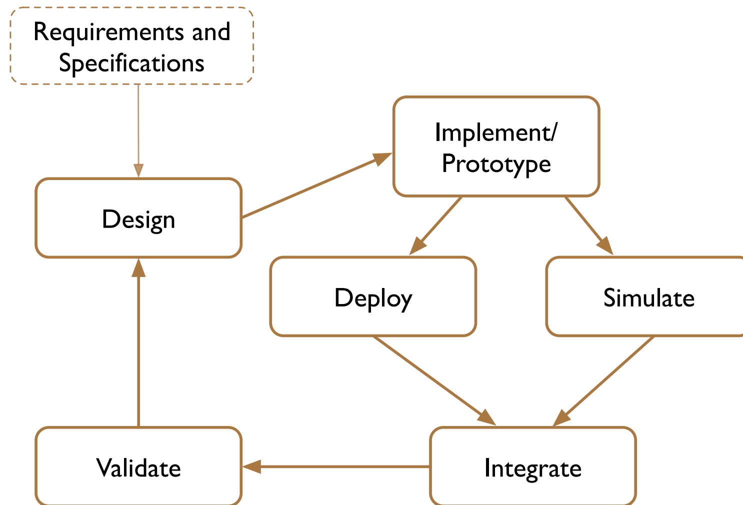


Figure 4.1: *Incremental Smart Space Development*

- The great variety of smart objects and communication technologies generates the need for the assessment of alternative solutions.
- Inability to produce certain test case scenarios in the real environment and the need to test simulated components (or subsystems) with partially deployed system.

Therefore, we claim that it becomes very important for a designer to be able to try different alternative solutions easily and quickly, and the continuous evaluation allows their "immediate" assessment before committing to the physical deployment of the system. We argue that the community has mainly followed a code and fix solution, more aimed to make things work instead of moving a step back and devising a more comprehensive and principled approach.

RQ 1.3. is concerned about finding a suitable development life-cycle for the development of diversified smart spaces. We propose a framework that provides the backbone for all the other features and stays unchanged till the final deployment to allow for an early evaluation of "incomplete" systems. This enables application developers to use or replace one solution to the other without the need of changing the overall organizational structure and topology of the system. It provides interfaces to surrogate system components through external simulators, to foster rigorous evaluation of designed systems, and to ease the deployment of physical elements.

In result of this, an incremental development process for the dynamic smart spaces is presented, which is shown in Figure 4.2.

In the *design* phase, all the elements of the smart spaces and their interactions are modeled. These elements may be physical (to be deployed) or virtual (to be simulated). The next phase of the process is *implementation/prototyping* of the designed system, which may be physically deployed or simulated through external simulators. In the *integration* phase, both simulated and virtual components are executed concurrently. It means that the running system comprises of many physically deployed real elements and some simulated behaviors and components, which are integrated together by the framework. This property of the framework enables continuous system valida-

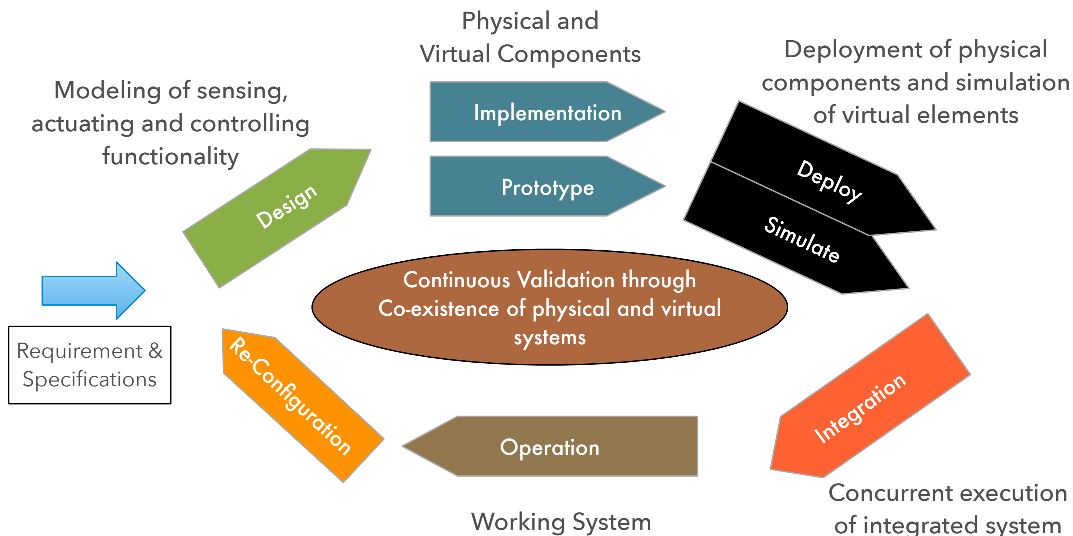


Figure 4.2: *Incremental Smart Space Development Life Cycle*

tion throughout the development process as it enables the early evaluation of various system elements through simulations. This, in turn, allows one to modify/configure different elements of the space to test various alternative solutions. It also helps the developer to move seamlessly from the completely simulated solutions to the fully deployed one in an incremental fashion.

4.1.2 Integrated Self-Adaptive Approach

Section 2.5 and 3.3 answer **RQ 1.1.** about the shortcoming of the existing systems. The **RQ 1.2.** inquires about the possibility to employ/integrate various principles from the existing design paradigms to address those shortcomings and deficiencies. This section aims to find an answer for this question and puts forward an integrated self-adaptive approach towards the development of dynamic spaces.

The autonomic computing community has been focusing on super-imposed adaptation mechanisms by adding further, dedicated components to the (software) architecture of the system. State of the system is monitored and in case of any changes, the super-imposed adaptation mechanism analyzes whether the system is still in "desired" state of operation or it needs to adapt according to the observed changes. As this adaptation logic is usually external to the system and needs to continuously monitor the state of the system, it adds to the complexity of already complicated systems.

In contrast, bio-inspired solutions provide inherent self-organization support and they do not need super-imposed mechanisms in order to able to react to the continuously changing environments. However, this autonomy and self-organizing capability, on the other hand, results in systems, which fail to guarantee the desired level of reliability and control.

This thesis aims to blend the two views and proposes an architecture-centric solution that synthesizes component-based control and bio-inspired (fireflies-based) mechanisms to exploit the best characteristics of the two paradigms. Suitable abstractions

help conceive self-organizing, ad-hoc collaborations among the components of a space. The idea is to define the functional control logic through architectural loops and interdependencies of various system elements through role-orientation. Whereas, the runtime adaptation and collaboration formation will be handled through embedded fireflies-based (bio-inspired) adaptation mechanisms that would use the defined interdependencies of elements and then guide system to efficient/desired system topology and configuration. The synthesized approach enables the following:

- Components are provided with the capability to self-adapt in dynamic situations according to their continuous changing spatial and proximity context, while keeping the best possible system configuration.
- The developers, at the same time, are still able to define control mechanisms to achieve application goals despite the self-organization of components without a central controlling entity.
- It provides application developers with more flexibility to define control logic through component control loops, or alternatively/complementary through bio-inspired organizational rules.

The details of the proposed adaptation approach are provided in Sec. 4.5.

4.2 Design Abstractions

This section answers **RQ 2.** by proposing abstractions and integration mechanisms for the design and implementation of smart spaces. The aim is to overcome the issue of fragmented (in terms of technologies and functionality) devices and support the self-organization to deal with the continuously changing user requirements.

The survey of the state of the art highlights the issues (Sec 3.3) with the current approaches and points out the areas that require better or improved solutions. Following observations were made that will be used as the guiding principles for the design of a new development framework:

- It is easier to work with small ensembles of devices at functional and management level and then *evolve* the system over the time.
- Think big, start small approach is required for the development of Smart Spaces. It means that, although, the system should be designed considering the broader and higher level design, but should be implemented in an incremental manner and step-wise deployments.
- Abstractions are required for the continuous integration and concurrent execution of physical and virtual components. The provided abstractions are required to be applicable/usable for all the development phases of smart spaces.

The process of *designing* a smart space through the proposed framework can conceptually be divided into two distinct activities. First, the designer needs to define the various elements of the space through provided abstractions defined in the meta-model in Figure 4.5; second s/he needs to layout how these elements will coordinate together to achieve the system goals. The proposed framework offers role-oriented components

modeling that allows one to successfully capture the contextual and functional relations among the different components of a smart space.

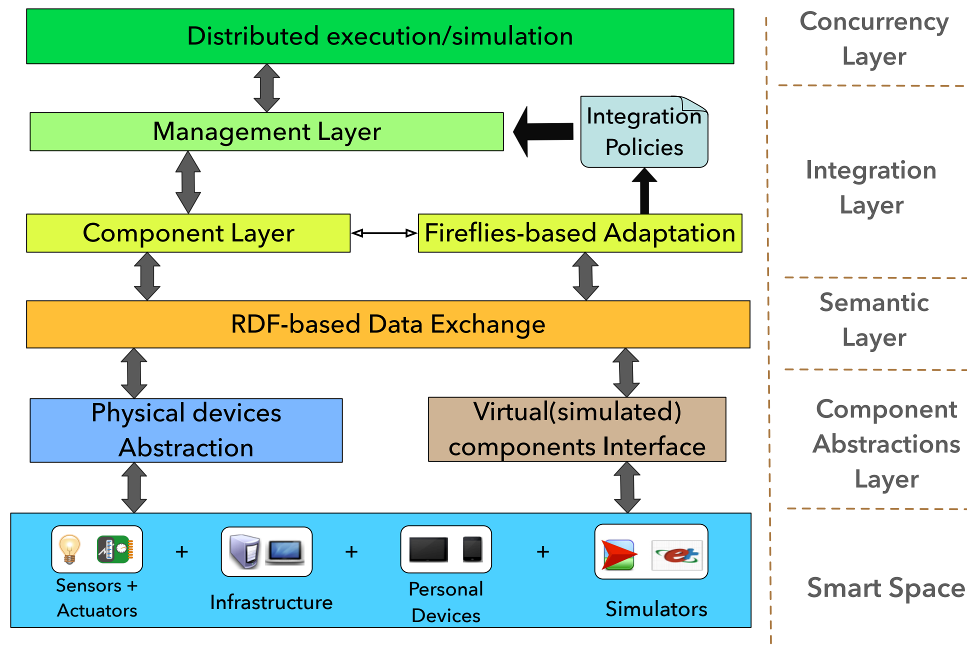


Figure 4.3: *The Proposed Framework*

4.2.1 Component

The *Component Abstraction Layer* defines the *Component*, the basic design level abstraction of the framework.

- Components capture the essential attributes of space entities and maintain their state over time.
- A component may refer to any physical or virtual entity that has some inherent capabilities (control, sensing, actuating, or simulation) or functionality (room/floor controller in an office) that may be needed by other components.

All the sensors, actuators, controllers, users or external systems (low level physical layer in Figure 4.3) within a smart space will be modeled as components. The benefit of having this component abstraction is that a developer will not have to deal with any physical level details about the devices (i.e., whether a component is related to any physical device, any web-service or a virtual source), while defining rules and behaviors for the components. This plugin ability will, in turn, enable the developers to replace one component (or any subsystem) with another one in a very easy and seamless manner.

Figure 4.4 shows the component model employed by the proposed framework. It consists of the following elements:

- The components maintain identity that is used to identify a specific component within the system.

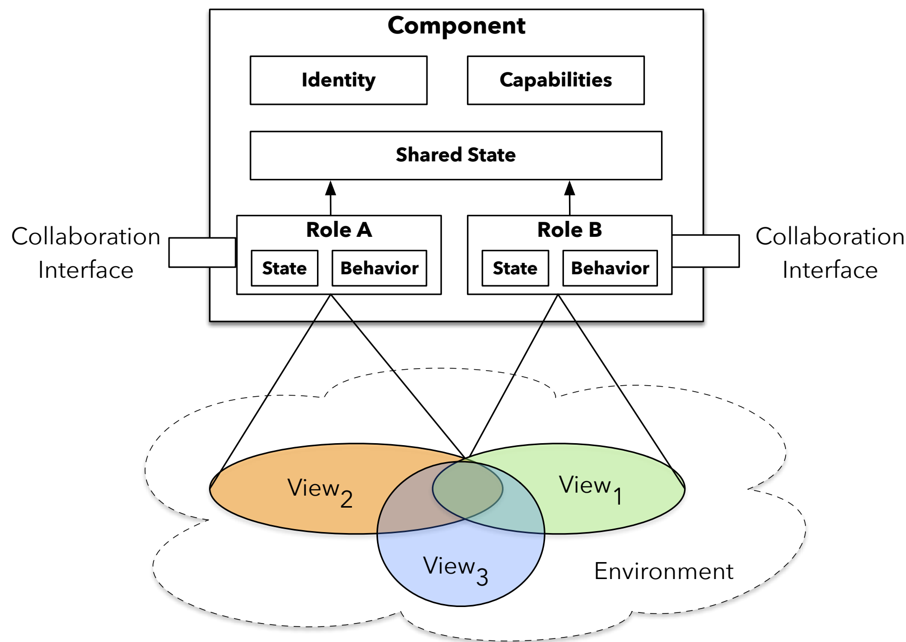


Figure 4.4: *Component Model*

- It defines some capabilities, usually programmed functionalities, that may be required by some other components.
- It collaborates through other components through role (described later) interfaces, which allow a component to be (temporarily) a part of functionality-specific collaboration/view of the space.
- The component also maintains a state, that can be shared among various roles.

This abstraction allows the developer to concentrate on the properties of interest and define suitable behaviors accordingly. The *same* component can act as a proxy of both a physical entity and a simulated one with no external changes. The same architecture is kept throughout the whole development process: it can evolve by both decomposing existing components and replacing simulated behaviors with real ones. Proper simulators can be used for the early validation of conceived artifacts.

4.2.2 Role

Roles are scenario-specific views and behaviors of components. Each component can play/exhibit different roles at a given time, and in a given context or situation as shown in Figure 4.4. Role-orientation of the framework components offers the following benefits:

- Components are typed and as such they cannot change their behavior easily. In contrast, roles provide dynamic views on a component in a specific context, and thus they are useful for creating context-oriented behaviors and data exchange among components.

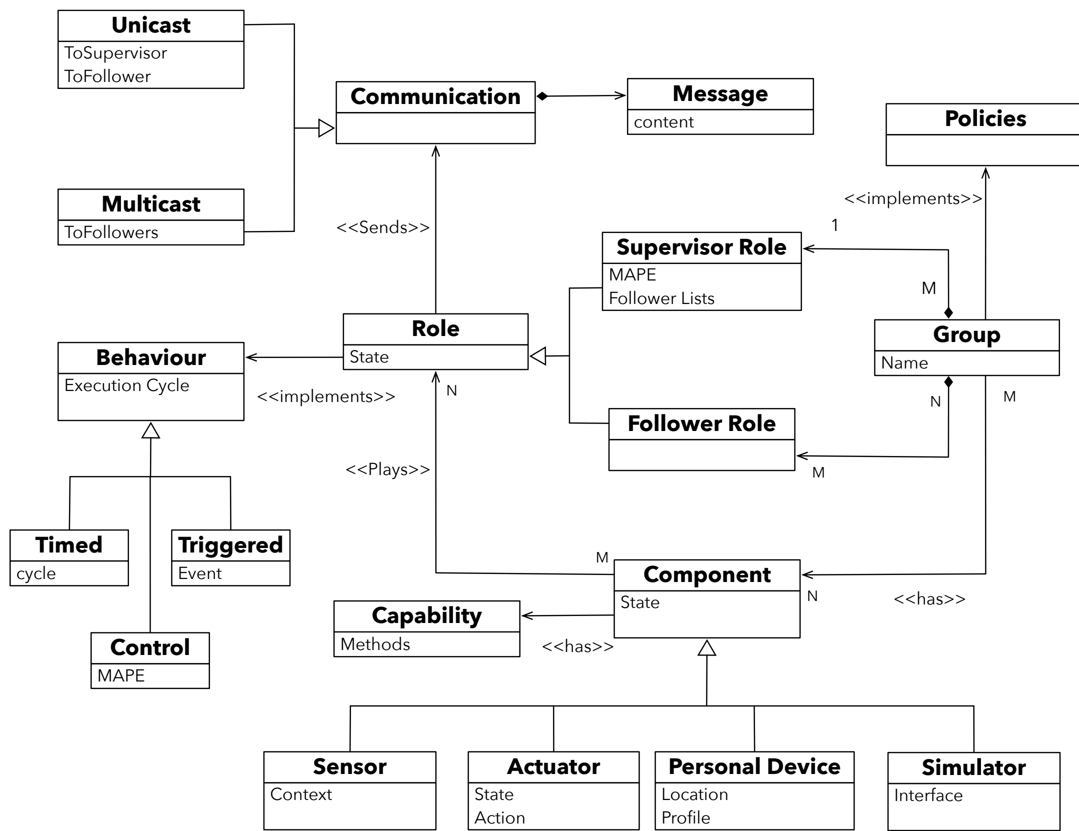


Figure 4.5: *Framework Meta-Model*

- Roles provide separation of concerns between the component identity (and data) and its behavior and collaborations.
- They are superimposed on components and can be changed or removed at runtime according to contextual needs.
- The dynamic role selection can help specialize the information exchanged between components, and thus reduce the amount of transferred data.
- The collaborations set between components on the basis of their roles help control domain dependencies and provided/required features in a fine-grained manner.

A role comprises a *type* and *behaviors*. The type defines how these roles could be used to form collaborations, whereas the behaviors are used to define the tasks that a role is supposed to perform.

Role Types

Role types are used to define the capacity in which one component can interact with other components. There are two types of roles that a component can play:

- *Follower role*, which corresponds to a worker
- *Supervisor role*, which oversees workers

These role types are used to form groups (Sec. 4.2.3) and ad hoc collaborations. In each collaboration, one component is chosen to play the supervisor role, which is selected dynamically at runtime, while the other components are followers. The supervisor component will be responsible for management whereas followers will act upon the directives they receive from the supervisor. Each component can play many roles in different groups, enabling information sharing across multiple groups (exemplified and discussed in next section).

Behaviours

Behaviors are application-specific functionality that a component performs in certain situations. There may be different types of behaviors such as:

- Timed (cyclic) behaviour, which repeats itself after a defined time interval.
- Triggered behavior, which is executed when a certain (specified) event takes place.
- Controller behavior, which refers to MAPE control loop for decision making.

For instance, a luminance sensor component may implement cyclic behavior that reads/sends luminance information after every 60 seconds to other interacting components. On the other hand, a lamp component may implement a triggered behavior that is invoked every time an event takes place that requires an action (turn on/off) to be performed. Controller behavior can be implemented by a computational component that controls and mediates between sensory information and actuator actions in order to maintain the desired lighting conditions. Section 5.3 will discuss in detail how these behaviors can be implemented and coordinated together to achieve certain functionality.

4.2.3 Group

In our framework, components are clustered into **groups** based on their functionality, location or other logical factors and dependencies. The rationale for employing the group-based coordination of smart space components is manyfold:

- Arguably, management of groups of components is intrinsically more meaningful and less dynamic than managing individual components.
- Groups enable more efficient organization of the components as rules can be defined at various granularity levels to deal with high dynamism and changing contexts.
- A group acts as a facilitator to integrate various components and form ad-hoc collaborations.
- Groups also enable the possibility of surrogating some components to realize the transition from a simulated entity to the real one.

Figure 4.6 describes the working of a group. There is one component with the supervisor role that implements the MAPE (Monitor, Analyze, Plan, Execute) control loop [39]. All the other components in a group participate as followers. Followers (e,g. sensors) provide environmental context/data to the supervisor which is responsible to

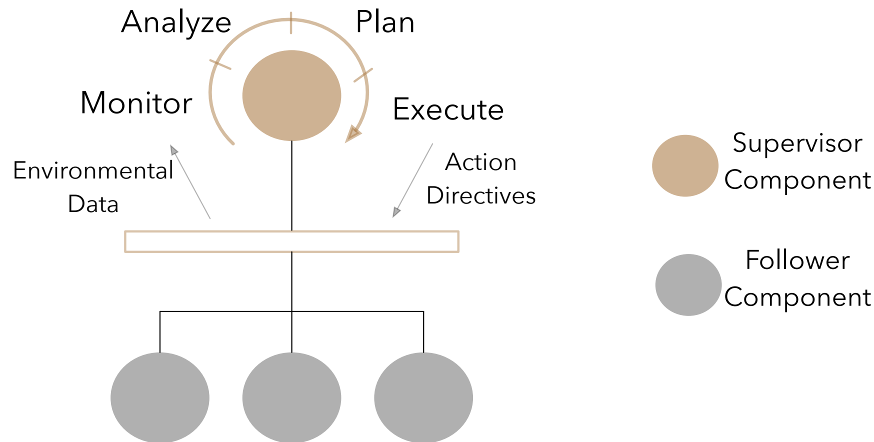


Figure 4.6: *Group Abstraction*

analyze and decide about certain actions that are required to be executed. These action directives are then sent by the supervisor to the relevant followers (e,g. actuators).

A group provides an abstraction for reasoning about the coordination and management of a single set of nodes. It does not, however, allow us to coordinate or manage the entire system. To enable global coordination, the framework allows groups to be "composed". Group composition is achieved by allowing a node to belong to more than one group at a time, and by allowing it to play different roles in different groups (e.g., a node can be a supervisor in group A and a follower in group B). This way designers can construct different kinds of organizational structures, depending on the application's coordination needs. Figure 4.7 shows some of the possible organizational structures (e.g., hierarchical, circular, flat). These structures allow groups to share "local" knowledge, and, through appropriate compositional design, reach a "global" understanding and coordination of the entire system. In the figure, black circles represent supervisors, white circles represent followers, and circles that are half black and half white are components that play both the supervisor and the follower role, in different groups.

Figure 4.8 describes how all the elements of a space (lamp, sensor, server, simulator, smartphone) can be abstracted as *components* that are capable of playing multiple (supervisor and follower) roles in certain groups (proximity and lighting). As a component, lamp and sensor play the *lighting group follower* role, simulator and smartphone play the *proximity group follower* role, and the server plays both *lighting group supervisor* and *proximity group supervisor* roles. Based on these role playing capabilities, all the physical elements (components) are then organized into two groups called lighting and proximity.

Figure 4.9 explains how information (contextual data) is exchanged and decisions are made within a group. An example light management group is presented which consists of the following components: (i) a control component (light manager) , (ii) two sensors (light and proximity sensor), and (iii) two actuators (light and a window). The light manager plays the supervisor role in the group and hence is responsible for all context management and decision making. The two sensors and the actuators, on the other hand, play the follower role in the group and hence send/receive contextual information or directives from the light manager. To be precise, light and proximity sensors

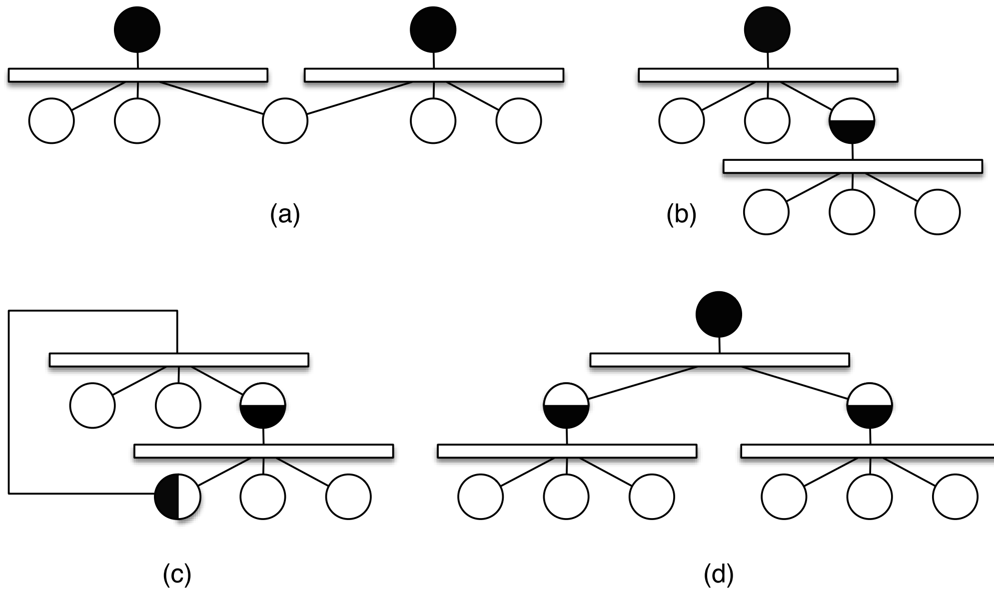


Figure 4.7: *Group Compositions*

generate light and location context, respectively, and send it to the light manager. The light manager analyzes the light and proximity information on receiving, decides actuation directives for light and window, and send it to turn on/off the light or open/close the window shutter.

The ability of a component to be part of multiple groups at a time raises the problem that it may receive conflicting directives from the different supervisors. For example, it might be the case that the light management group may prefer a shutter to be opened whereas the temperature adjustment group wants it to be closed. The framework fosters the idea that a well designed system would not have to face these situations. However, the interactions among the different groups, and the system-specific composition of the commands issued by the different supervisors provide a further way to manage them. The last option is that the internal logic of each single component can always specify how to deal with these spurious, conflicting cases. This means that either the groups are composed in such a way that the commands issued by the temperature supervisors can filter those issued by the light managers. Otherwise, the shutter itself can be programmed to decide that the commands received from a temperature supervisor override the commands received from the other supervisors.

4.3 Semantic Layer

Interaction and information exchange among large numbers of heterogeneous devices is a major issue while developing applications for smart spaces. The proposed framework provides a Resource Description Framework (RDF) based *Semantic Layer* to harmonize heterogeneous components in a smart space. RDF-based ontological representation of information that makes interoperability among components easy because of its inherent support for data linking between different ontologies.

RDF [64] is a conceptual modeling approach in which information is described un-

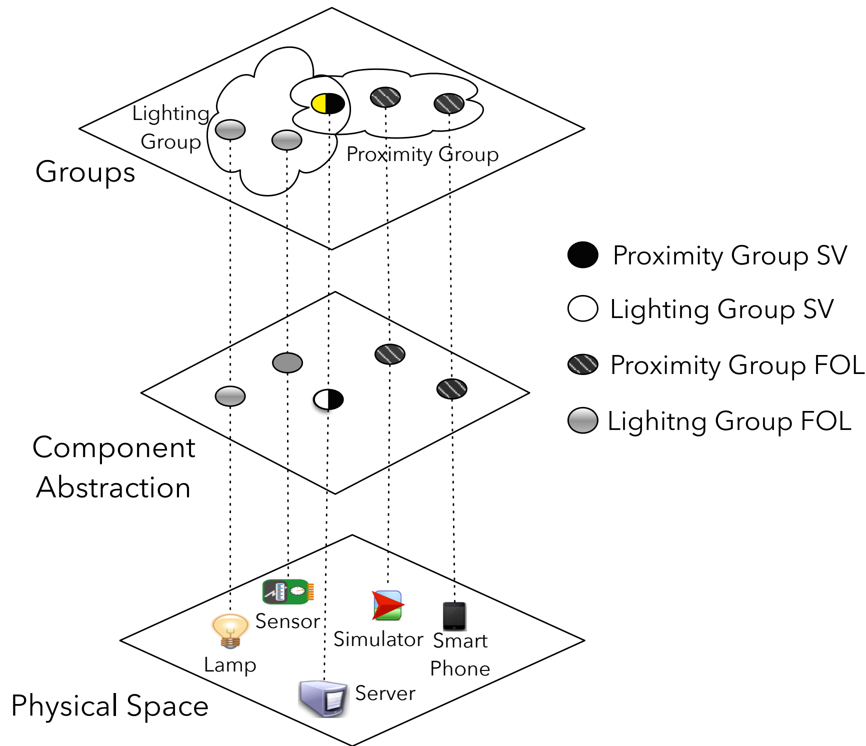


Figure 4.8: *Design Abstractions*

der the form of *Subject – Predicate – Object* triples. The *subject* field refers to a resource, whereas the *predicate* field describes how the subject relates to an *object*. The subject and predicate are both defined by URIs, whereas objects can be either resources (URIs) or literal values.

In the smart office scenario, each room has its own set of sensors; they feed information into the system using a specified model, and, thanks to that model, all the other components can interpret their values. Figure 4.10 shows a simple example ontology model for the sensors. Each sensor is defined by its location, the physical property it measures, the range within which its values can be output, the actual reading, and a timestamp.

An example of how this sensor ontology can be used to represent information in RDF is shown in Figure 4.11. The RDF graph shows the property measurements of two sensors $S1$ and $S2$, located in $MeetingRoom$ and $DemoRoom$, respectively. Both these sensors measure temperatures; $S1$ can sense temperatures between -100 and 100 degrees, whereas $S2$ can measure temperatures between -50 and 100 degrees. The graph also shows one measurement for each sensor (i.e., the value and its timestamp). Thanks to this model the sensors can “speak” the same language, and the system can digest the information they generate. Similarly, the semantic model is also used to define roles for various components, so that the SSMs can associate the components with relevant groups by matching relevant ontological information.

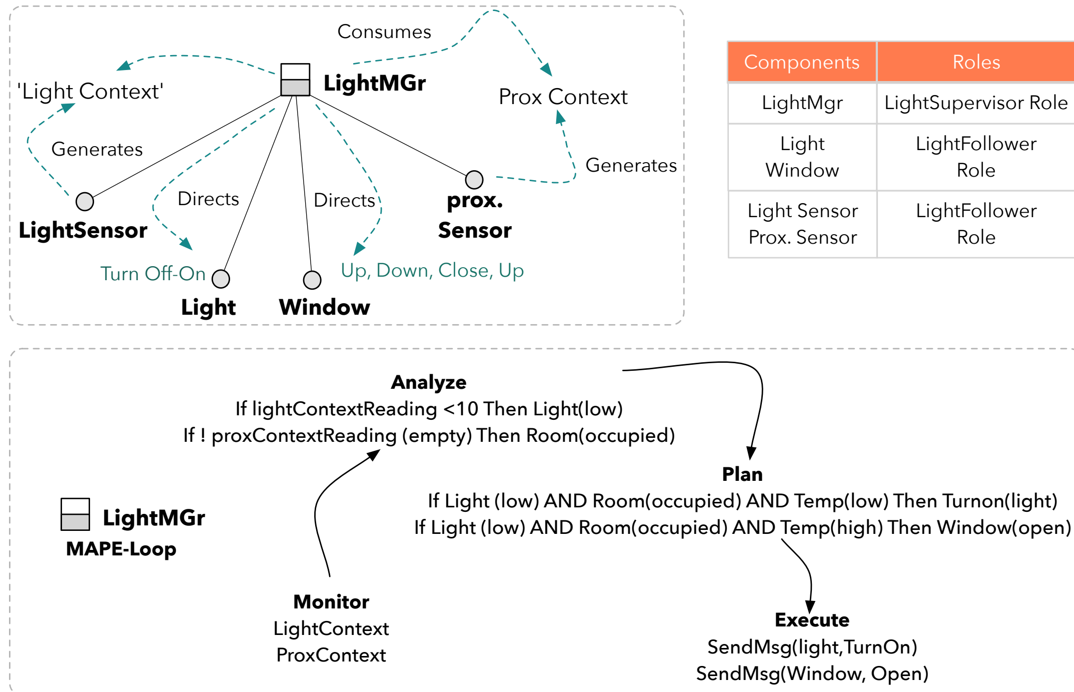


Figure 4.9: *Group Example - Light Management*

4.3.1 Semantic Model

Use of a common semantic model helps solve the integration problem as different components use the same defined standard for sending messages to other components. The proposed semantic layer can conceptually be divided into a *Group Coordination Ontology (GCO)*, a *Message Protocol Ontology (MPO)*, and a *Domain Concept Ontology (DCO)* (see Figure 4.12). The GCO defines the generic concepts used by the framework for automated group formation and self-configuration. According to the GCO, every *Group* has multiple *Components*, and a *GroupInfo*. A *GroupInfo* describes the *Supervisor* and *Follower* roles used in that specific group, whereas *Components* are system elements that conform to the semantic model. The inter-communication among components is carried out according to the MPO. The MPO defines the general messaging protocol. Not only does it enforce rules such as “followers can only send messages to their supervisors” (described by Figure 4.12), it also defines message exchange types that simplify the interpretation of the messages themselves. Besides having a message protocol, we also need to standardize the content of the system’s messages; this usually depends on the application domain. The DCO defines the vocabulary that is used to solve this particular issue.

Figure 4.13 shows the DCO for the JOL case-study that defines the application specific semantic model. Each device and appliance of JOL is defined as a component and has an associated location and communication protocol. There are two types of devices namely *sensors* and *actuators*. Sensors are responsible for generating contextual information (luminance, temperature and proximity) whereas actuators (smart plugs in our case) are used to control the power state of the *appliances*. Components can be identi-

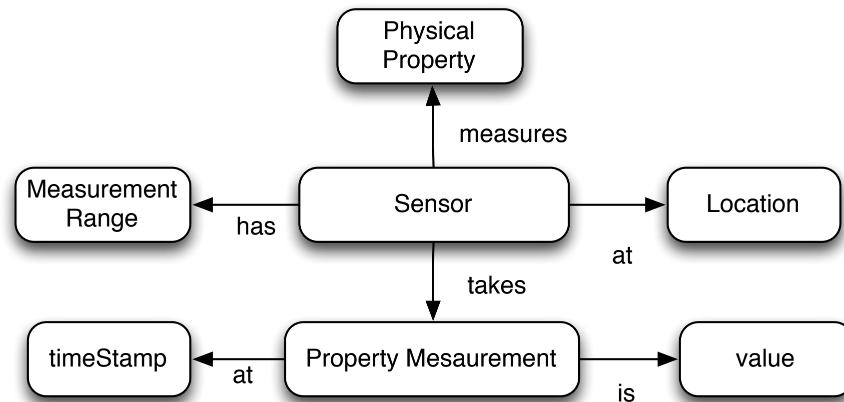


Figure 4.10: *Sensor Ontology.*

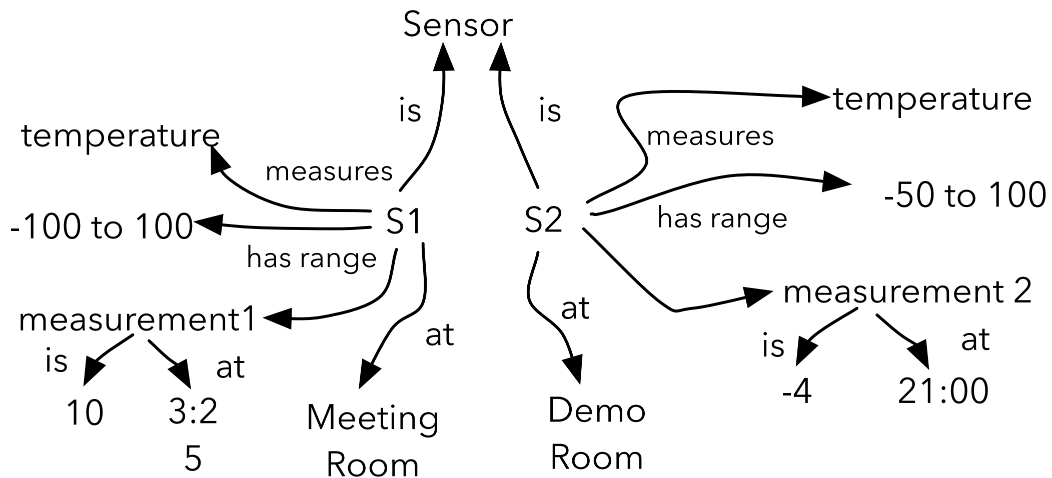


Figure 4.11: *RDF graph of a semantic model for sensors.*

fied and located based on their area, room and position coordinates. All the components at JOL can communicate via any of the three (WiFi, ZigBee, and BLE) protocols.

4.4 Integration Layer

The Integration Layer is responsible for coordinating heterogeneous components and subsystems by organizing (integrating) all of them in various self-organizing groups. As described earlier, components collaborate with each other through these *groups*, which are formed on the basis of common/related functionality, location, or other logical factors and dependencies. In each group (corresponding to a collaboration), one component is chosen dynamically to play the role of supervisor while the other components take part in the group as followers. Each component can play different roles in diverse collaborations (groups), and enable information sharing across multiple adhoc collaborations. It ensures the separation of concerns between the sensing/actuating infrastructure and the components responsible for autonomic management (discussed

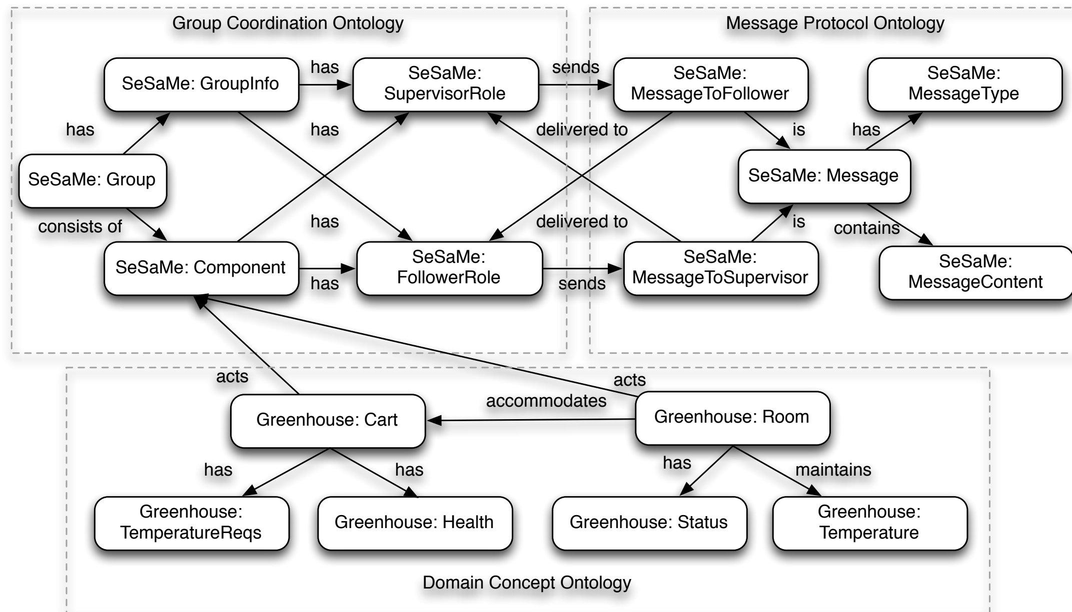


Figure 4.12: *Taxonomy of semantic model —We have used namespaces instead of URIs for the sake of clarity.*

later).

For example, one can integrate and group all the luminance sensors, lamps and window shutters at JOL (or a room) for lighting control and management, whereas another group can be formed to manage temperature by combining temperature and radiation sensors, external weather data, windows and all HVAC systems. Alternatively, there can be a room (space) management group that consists of all the devices (sensors and actuators) in that room to make an integrative (both temperature and lighting) control system.

Moreover, it is important to note that the component abstraction and the role orientation allow one to blend physical and simulated elements in a system together. One can also replace a group of components with another component that simulates them. For instance, a group of lamps can be replaced or simulated by Freedomotic which can be modeled as one or multiple components. At the architectural level, a component can be one or a set of element(s) that has a state and some functionality that it exhibits under certain environmental conditions. This ability of the architecture enables the co-execution of the system at different levels of maturity and in different versions. Physical entities and external simulators (emulating one of more entities) can work together to allow a developer to move seamlessly from a simulated solution towards a concrete deployment of the system. This is important for spaces such as our public park as it is not easy to evaluate alternatives and create situations with hundreds of people to test the designed system. For example, one may need to test the physical infrastructure (screens, sensors, alarms, lamps, etc.) of the park before the real deployment and may need to simulate user behaviors through an external simulator while analyzing the results on the real screens.

The proposed framework caters for the integration requirements for both fixed and

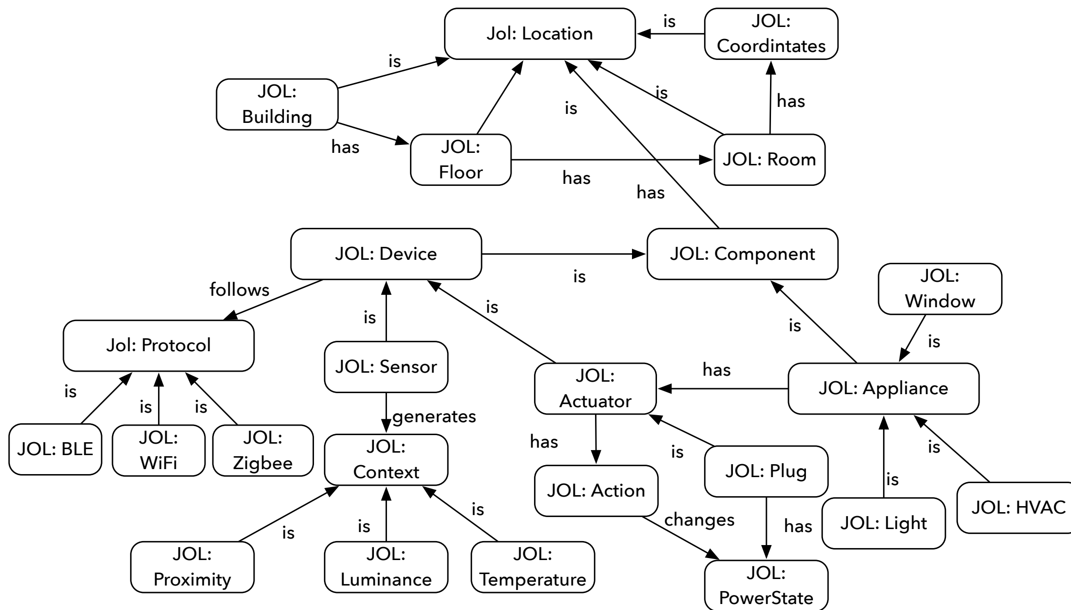


Figure 4.13: *JOL Semantic Model*

dynamic spaces. *It provides special purpose components for the autonomic management of functional component groups in case of fixed/static smart spaces. However, for the scenarios which are more dynamic and where components need to be more autonomous and self-adaptive, it offers bio-inspired integration mechanism*. Following are the details of autonomic management and self-adaptive capabilities of the proposed framework:

4.4.1 Separation of Functional and Management Design

The framework offers separation of concerns between the sensing/actuating infrastructure and the components in charge of its efficient, autonomic management. Sensors, actuators, displays and computational elements define the *Components Groups*, while special-purpose components *Smart Space Manager*s (SSM) are responsible for the (distributed) management of *Components Groups*. In Figure 4.14, $S1, S2, S3$, and $S4$ are the supervisors for the component groups whereas $SSM1$ and $SSM2$ are the two supervisors responsible for the management of these functional groups.

At both levels, components are still divided into supervisors and followers, grouped together according to common needs and goals, and groups are composed properly. The two layers impose that each lower-level group be (indirectly) managed by an upper level node. This means that each supervisor of a group of components is connected, as a follower, to an SSM. This way it can receive updates from the management layer as to how it should coordinate its group. The SSMs can also be connected among themselves to share information, which can then be used to better manage the groups. The autonomic group formation is guided by the *integration policies*.

It is important to note that use of SSMs is not mandatory, rather they provide de-

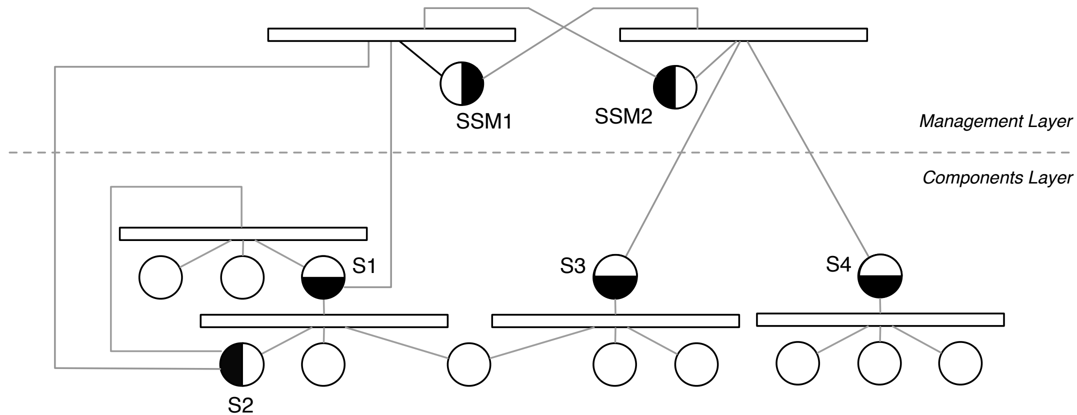


Figure 4.14: *Example configuration of a Smart Space.*

veloper an added functionality for the spaces with fixed infrastructures to have more robust solutions. They may not be used for dynamic or first-responder scenarios where no fixed infrastructure is available.

Integration Policies

Integration policies consist of rules and information that is used as a guideline by the framework to form groups at runtime. These policies are used to define the following configuration rules:

- Group types information (along with their role dependencies)
- Group formation criteria (e,g. location based, proximity based, heuristic functions)
- Location map of the space, which declares the spatial distribution
- Heuristic functions for group formation
- Self-organizing parameters (SOPs) for groups such as: maximum group size, maximum number of messages processed by the groups, etc.

Location map could be based on both physical distribution (e,g. floors, rooms etc.) or functional requirements (e,g. a zone with same lighting or heating dependencies) of the space. Figure 4.15 shows an example of location map and the group information description. In the given example, a location is characterized by name, attribute (physical/logical), type (room/floor/zone) and topology (autonomous/composite). Group information defines the group name, organization style (e,g. supervisor-follower), required supervisor and follower roles, grouping type (location/heuristic based), grouping criteria based on the grouping type defined (e,g. room/floor/zone in case grouping type is location), and values for SOPs.

Heuristic functions can also be used to optimize the topology of the system by finding the best group for each component according to the needs of time. The self-configuring group formation based on fireflies-based heuristic functions are described in Section 4.5.

4.4.2 Autonomic Management

The autonomic management is based on the following two activities: (i) specification of the roles for each component and (ii) definition of the group formation criteria (integration policies) to integrate the components in the desired topology using the declarative language. Once they are defined, the framework then takes care of all the integration and coordination of components accordingly. One benefit of having this self-configurable system is that by just changing the group formation criteria, we can simulate various possible topologies of the system to determine the best configuration for the system.

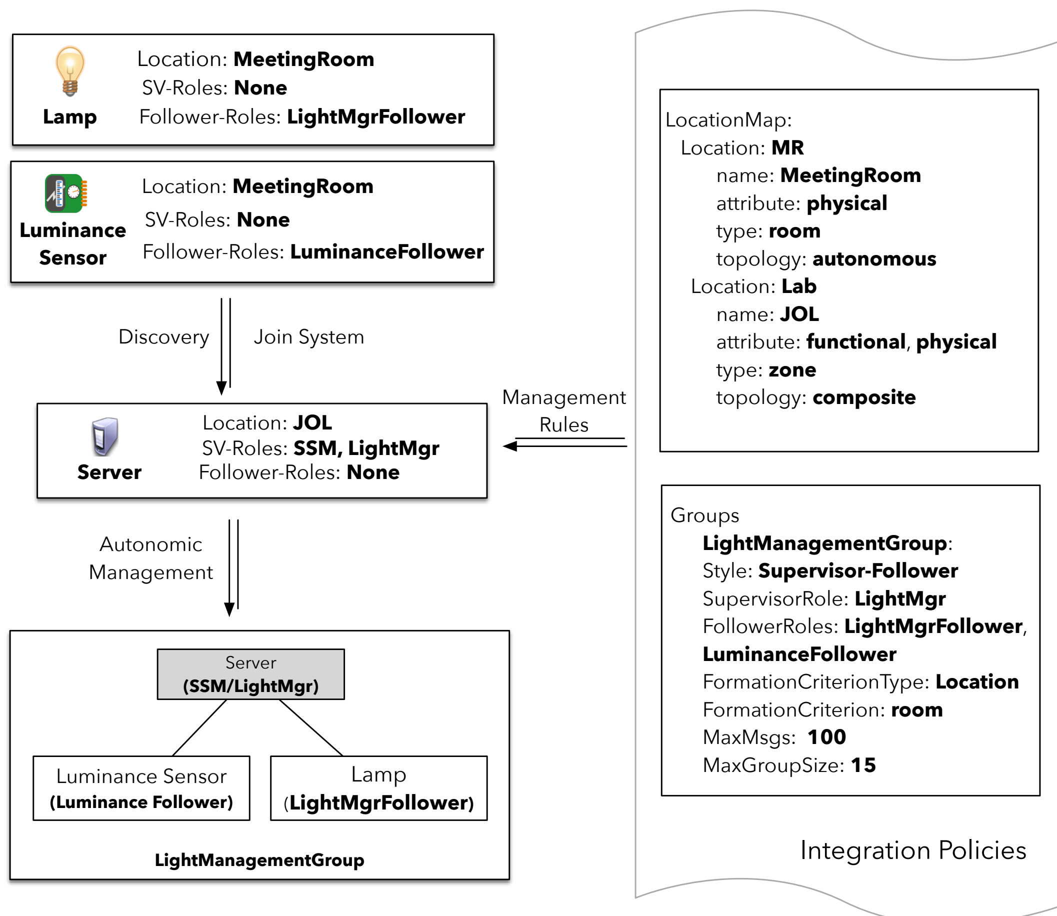


Figure 4.15: *Autonomic Management*

The autonomic management deals with dynamism by supporting the automated formation of new groups. A new component can connect to any existing group, or create a new one. This means that the framework automatically identifies the right group supervisor and links the new component to it. If the component can play different roles, that is, it can belong to different groups, the framework will decide its group memberships according to all its roles and other different aspects.

To understand this, let us first consider the case in which the new component can only play a single supervisor role. When it connects, the SSM searches for the cor-

responding group in the shared group list. If such a group does not exist, it creates a new group with that component as the supervisor, asks the component to activate its (SSM) follower role, and updates the lists of shared groups and supervisors. If, on the other hand, one or more acceptable groups already exist, the SSM will decide, based on the system's desired configuration, how to add the component to the already existing groups. Based on performance and resource utilization, the management layer will select the group with less followers, and less message exchanges (*fireflies-based adaptation for systems without SSMs is presented in the next section 4.5*).

If the new component can play multiple supervisor and follower roles, the integration layer enacts the above procedures for each and every one of the component's roles. This implies that the component will become part of all the groups that it can be part of, at that particular time. If a component can play both the supervisor and the follower role in a specific group, the management layer will decide its role based on the system's overall needs. The component will become a supervisor if any of the existing groups is congested. Otherwise, the new component will become a follower.

Figure 4.15 shows a simplified (partial) description of the integration policies that is used to define grouping criteria for lamps and luminance sensor. The first block describes the location map that consists of two locations namely *MeetingRoom (MR)* and *JOL*, where MR is a physical room and JOL is both functional and physical zone. The second block defines the formation criteria for one of the groups called *LightManagementGroup*. It describes that a component with *LightMgr* role can supervise this group, whereas, components with *LightMgrFollower* and *LuminanceFollower* roles can be part of the this group as followers. Moreover, it instructs the framework to group these components on the basis of their location with room as a granularity.

As described in Figure 4.15, integration policies are fed to the server (component), which is capable of playing SSM and LightMgr roles. Therefore, it starts acting as an SSM on its initiation, and later on when lamp and luminance sensors join the system, it creates the LightManagementGroup by activating its LightMgr role as well. It is important to note that it puts both these components in the same groups as they are located in the same room (MeetingRoom) and grouping criteria is set to room.

Let us now consider the JOL building to understand the integration mechanism. JOL can be divided in different lighting zones and two thermal zones in context of energy usage. It can be seen that each room has its own lighting controls and hence corresponds to distinct lighting zones. Whereas, it has only two thermal zones and single set of windows. Group formations in infrastructure-based spaces (such as JOL) usually have defined physical locations (rooms) that are used as primary criteria for grouping.

Figure 4.16 shows one of the possible configurations for the JOL scenario. Rectangles represent controllers (supervisors) and circles represent devices and appliances (followers). In this configuration, each room has its manager for lighting control whereas there are two temperature managers for the rooms where we have controllable HVAC. This implies that components are grouped according to the zone location (rooms). There is one SSM for the JOL that is responsible for all the group management and high level decision making.

The integration mechanism also lays the foundation of the possibility to use external systems and simulators within the framework to execute/simulate some of the function-

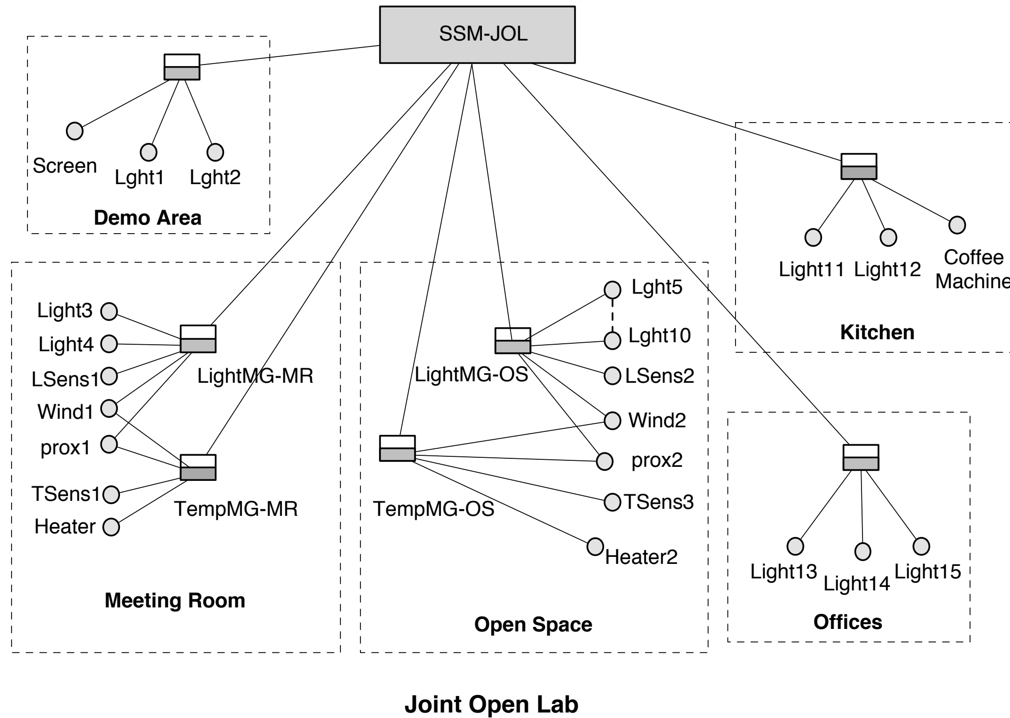


Figure 4.16: *JOL- Possible Groups*

ality (in response to **RQ 2.2.**). The issue is discussed in detail in Section 4.6.

4.5 Self-Adaptation Capabilities

The framework provides a self-adaptation mechanism, which is inspired by the firefly algorithm [104] and mimics the behavior of fireflies to form groups. The idea is to guide the formation of role-oriented component groups through fireflies-based algorithm, and hence, achieve **integrated adaptation mechanism** as proposed in section 4.1.2. Figure 4.17 describes the proposed integration of component based control and the fireflies-based adaptation.

4.5.1 Fireflies Algorithm

Yang [104] was the first who studied the flashing behavior of fireflies and used it for solving optimization problems. The firefly algorithm (FA) is a meta-heuristic algorithm that has proven its efficiency [44].

Fireflies are characterized by the *flashing light* they produce through bioluminescence. Such flashing light serves as primary courtship *signals* for mating. Typically, flying *males* are the first signalers and try to *attract* flightless *females* on the ground. In response to these signals, females emit continuous or flashing lights. Both mating partners produce distinct flash signal patterns that are precisely timed in order to encode information about species identity and sex. Females are attracted by behavioral differences in courtship signals. Typically, females prefer *brighter* male flashes. The flash intensity varies with respect to the distance from the source. To summarize, the

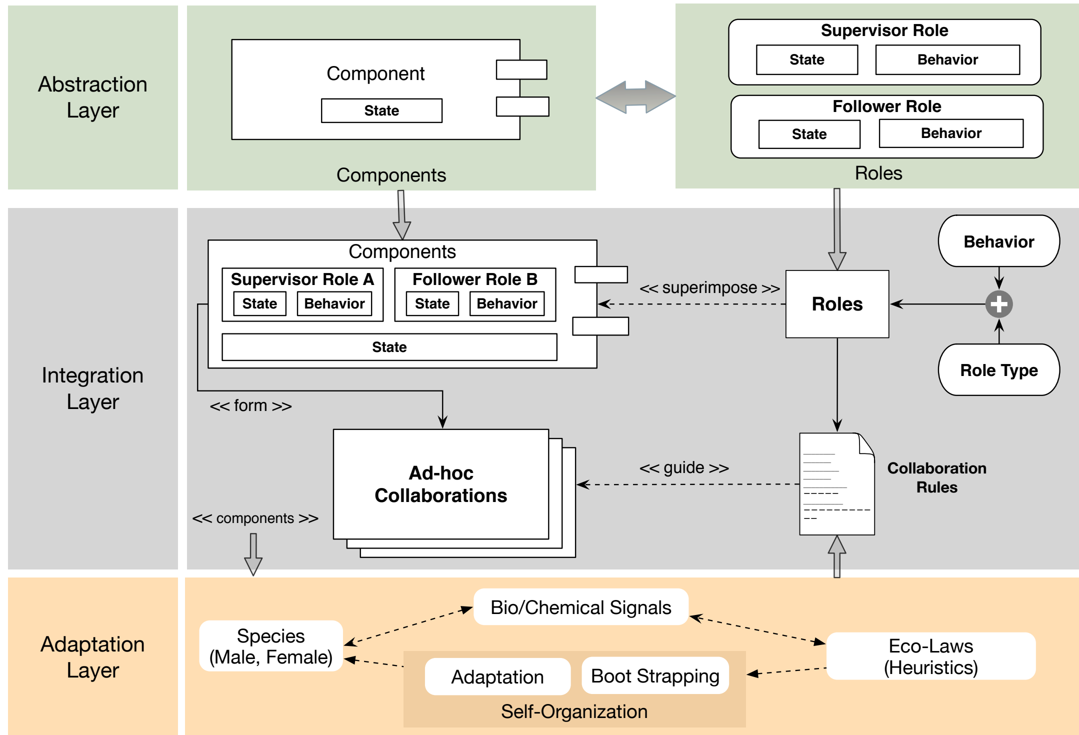


Figure 4.17: *Integrated Self-Adaptive Mechanism*

FA follows the following rules [104]:

- Fireflies are attracted to each other regardless of gender.
- The brighter a firefly is, the more attractive it is and thus less attractive fireflies must move towards more attractive ones.
- The brightness of a firefly is calculated by means of an objective function.

4.5.2 Adaptation based on Fireflies Metaphor

Since fireflies offer natural clustering capabilities that take distance into account intrinsically, they can be used as metaphor for guiding self-organization in dynamic self-adaptive software systems. We do not use that algorithm as is, rather we use some of its aspects and the rest of the self-organization is based on the general behavior of fireflies.

Zambonelli and Viroli [106] introduce a reference architecture for implementing bio-inspired pervasive ecosystems where service components are described as *species*, chemical or biological *signals* are spread to and perceived from the environment and used as interaction mechanism, *eco-laws* define the interaction rules for the species, and the *space* itself is the software infrastructure. We have paired these guidelines and the firefly metaphor for implementing our self-adaptation layer.

The component abstractions are mapped (see Table 4.1) to the firefly metaphors as follows:

1. Component roles are abstracted as flies (species) and role types are male and female flies.

Table 4.1: *Mapping - Component Abstractions and Fireflies Metaphor*

Software Abstraction	Fireflies Metaphor
Component	Nest
Role	Fly
Supervisor Component	Male Fly
Follower Component	Female Fly
Message	Light Flashes
Integration Rules	Eco-Laws

2. Flies live in nests (components) that are used to represent all the entities and interfaces of a system.

3. Light flashes (signals), which are generated (as brightness) and perceived (as attraction), guide the interaction mechanism and flies react to these signals according to their current state.

4. Rules (eco-laws) — such as: ”brighter fly will be more attractive, and brightness decreases with distance”— define the existence of flies and their social interactions, and provide guidelines to form collaborations.

5. The collaboration layer (space) enforces spatial relationships among various flies.

Unlike the original firefly algorithm, we consider bi-sex species of flies. Components are nests and each nest can host many flies. Male flies act as supervisors, while female flies are followers, and hence a component can play multiple roles in various groups. Supervisors flash light to attract followers. The brightness of a supervisor defines its suitability to act as supervisor whereas followers are attracted by supervisors around them according to their distance and brightness. We have also incorporated different light colors used to differentiate roles.

Algorithm 1 Self-organization algorithm

```
1:  Define functions for brightness and attraction
2:  Define initial nests for fireflies N_i, i := 1, 2, ..., n          ▷ define components
3:  Assign fireflies (male/female and color) to each nest  ▷ define roles for components
4:  for each nest n ∈ N do
5:      Call BOOTSTRAPPING(n)
6:  end for
7:  while System.state := running do
8:      Check for new nests and fireflies and update population
9:      for each nest n ∈ N do
10:         if n[male] := ∅ then                   ▷ if component has no supervisor
11:             Call BOOTSTRAPPING(n)
12:         else
13:             Call ADAPT(n)
14:         end if
15:     end for
16: end while
```

4.5.3 Self-Organization Algorithm

The self-organization algorithm (Algorithm 1) consists of two phases:

1. Bootstrapping for new or orphan (with no supervisor) components
2. Adaptation for the components with an active collaboration

The self-organization algorithm requires that the functions for calculating brightness and attraction be defined. To this end, the following heuristics have been used for organizing our park:

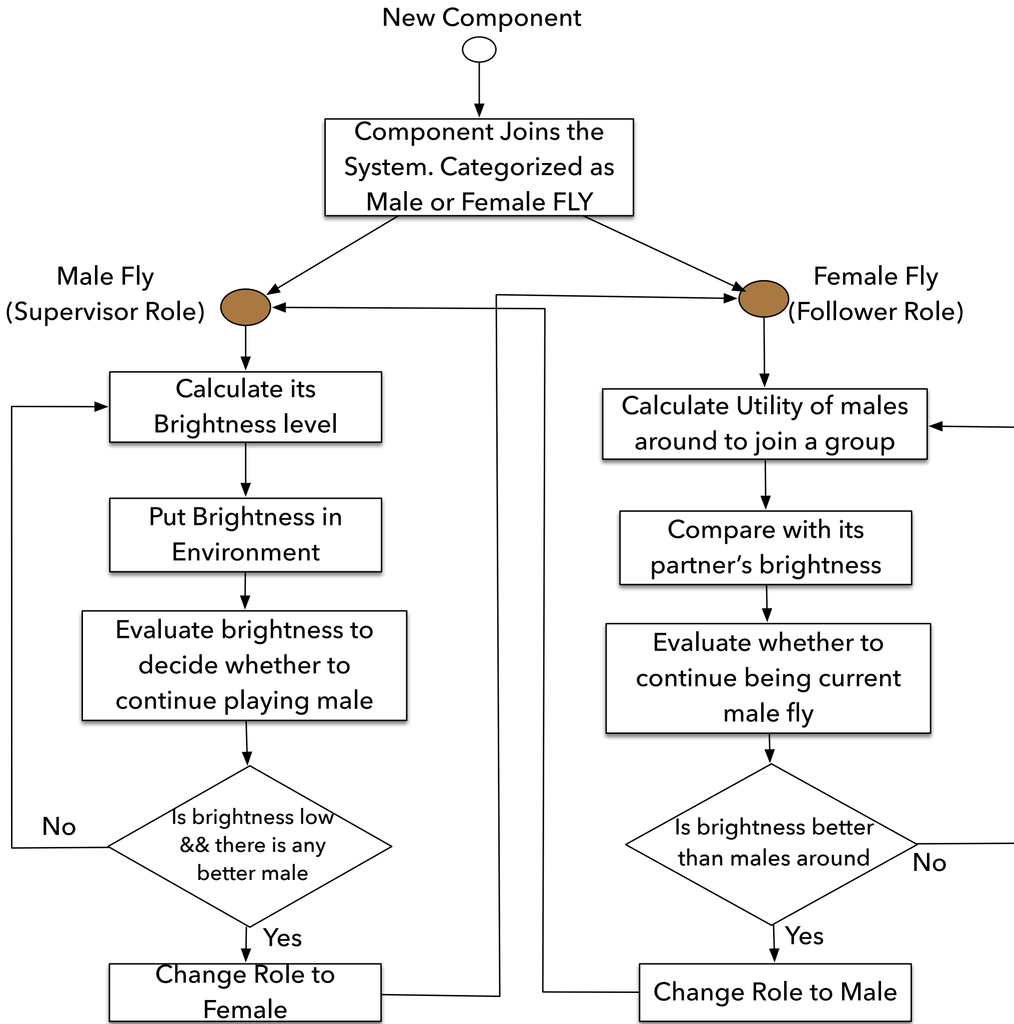


Figure 4.18: *Fireflies Adaptation Mechanism*

$$\beta_k(t) = \frac{c1}{\sum_{i=1}^{n} L_{ik}} + c2 \times P_k \tag{4.1}$$

- $\beta_k(t)$ is the brightness value for supervisor (male fly) k at time t
- n is the number of followers (female flies) for node k

Procedure 1 bootstrapping(nest c)

```
1: let M be the set of male-flies in c          ▷ set of supervisor roles of component c
2: let F be the set of female-flies in c        ▷ set of follower roles of component c
3: if M != ∅ then
4:     for each (male) fly m ∈ M do
5:         Calculate brightness[m] according to Equation 4.1
6:     end for
7: end if
8: if F != ∅ then
9:     for each (female) fly f ∈ F do
10:        let CList be the male-flies with color := color[f] within radius X of c
11:        if CList.size != 0 then
12:            for each fly k ∈ CList do
13:                calculate attraction[f,k] according to Equation 4.2
14:                let q be the component with max (attraction[f, CList_k])
15:                set f[mate] := q
16:            end for
17:        end if
18:    end for
19: end if
```

- P_k is the energy of supervisor k
- L_{ik} is the communication load between follower i and supervisor k
- $c1$ and $c2$ are the constants used to assign weights to the corresponding parameters (communication load and energy)

$$\alpha_{ik}(t) = \frac{c3}{d_{ik}^2} + c4 \times \beta_k \tag{4.2}$$

- $\alpha_{ik}(t)$ is the attraction of supervisor k towards follower i at time t
- β_k is the brightness level of supervisor k, d_{ik} is the distance between follower i and supervisor k
- $c3$ and $c4$ are constants used to assign weights to distance and brightness.

Dynamism and Self-Configuration

Whenever a new component joins the system, the bootstrapping procedure (Procedure 1) is called upon, iterates over the set of roles the component can play, and searches for the best possible collaboration for it according to defined heuristics (brightness and attraction) functions. Figure 4.19 shows an example bootstrapping scenario where a follower role tries to connect to the space by finding the best supervisor around. The follower component (female fly) $FF1$ can be part of collaboration 1, 2, or 3 by following (mating) supervisors $MF1$, $MF2$, and $MF3$, respectively. The decision of

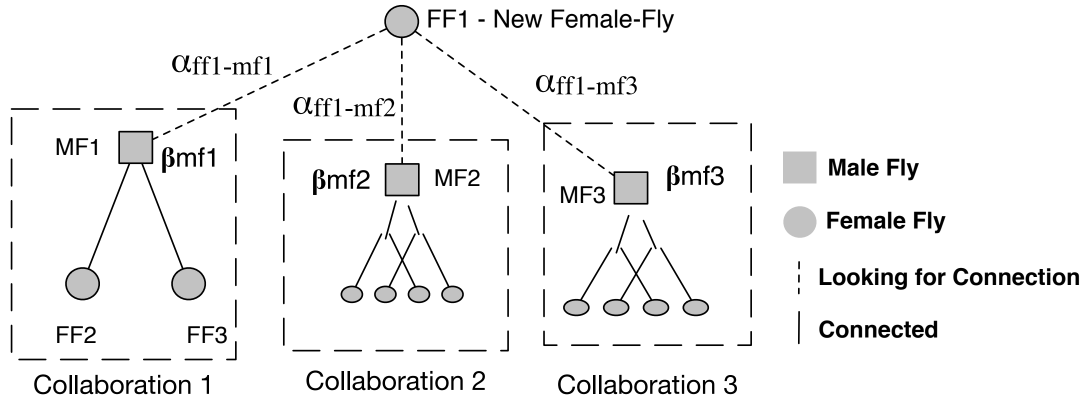


Figure 4.19: *Example bootstrapping scenario.*

joining a particular collaboration will be based on the maximum attraction it perceives: $max(\alpha_{FF1,MFk})$, where α, with $k \in 1, 2, 3$, is calculated by using Equation 4.2.

The effectiveness of the above function can be understood through the public park scenario. For instance, whenever a user connects to the space, the user's interests are matched to the various existing groups, and s/he is assigned to a group (or to a set of groups) accordingly. If a user can be part of both the "Kids Rail" and the "Ferris Wheel" groups, and the group for kids rail is congested, the framework will start by adding the user to the ferris wheel group. Thanks to the inherent self-organizing capabilities of fireflies, this is achieved autonomically as per the heuristic functions and one does not need to explicitly define this behavior.

Adaptation

The second step of self-organization is adaptation (Procedure 2). It first checks if the component is not part of any collaboration, and in case, it bootstraps it. Otherwise, it does the following:

- For each supervisor, it checks whether there is another supervisor available with better suitability, and if it is the case, it changes its role to follower.
- For each follower, it calculates the suitability of the supervisors around and if it finds a better supervisor then the current one, it switches to it.

The suitability of supervisor is calculated by its brightness (heuristic function), whereas, the follower calculates the attraction based on the brightness of the supervisor components around.

Reliability

The fireflies based algorithm is capable of automatically managing situations in which components leave the system unexpectedly. If a supervisor component leaves the system, all the followers become orphans as the group cannot exist without a supervisor. These components just trigger the bootstrapping procedure again to reconnect and select the next best supervisor in their proximity. This automated re-configuration of the

Procedure 2 adaptation(nest c)

```
1: let S be the male-flies of c
2: let F be the set of female-flies of c
3: if S := ∅ then
4:     if F := ∅ then
5:         Call BOOTSTRAPPING(c)
6:         Exit Procedure
7:     else
8:         Connect to nearest fly
9:     end if
10: else
11:     for each (male) fly s ∈ S do
12:         Calculate brightness[s] according to Equation 4.1
13:         let CList be the male-flies with color := color[f] within radius X of c
14:         let q be the fly with max (brightness[CList_k])
15:                                         ▷ where q is number of flies in CList
16:         if brightness[s]<brightness[q] then
17:             set s[mate] := q
18:         end if
19:     end for
20:     for each (female) fly f ∈ F do
21:         let CList be the male-flies with color := color[f] within radius X of c
22:         Calculate attraction[f, CList_k] according to Equation 4.2
23:         let q be the fly with max (attraction[f, CList_k])
24:         if attraction[f, supervisor[s]]<attraction[f, q] then
25:             set f[mate] := q
26:         end if
27:     end for
28: end if
```

system (components) topology in case of failures or unexpected events ensures the reliability of the whole system. This issue is discussed in detail with our case study 4 (public park) in evaluation section.

Congestion management

The heuristic functions defined for the selection of a groups in our proposed groups consider the supervisor workload at the given time. It reduces the chances of a component to be selected as a supervisor whose workload (in terms of number of messages or group size) is greater than other components in its proximity. In the public park scenario, the system guides the group of users to their desired destinations by connecting people with similar interests, but if the system finds out that a group (of people) is exceeding the set highest limit, it can split the group into multiple sub-groups, and guide them to the locations in a way that avoids message, and people congestion. This splitting of groups is not instant/explicitly defined, rather the system gradually converges to the more uniform distribution of people as the components try to self-optimize their

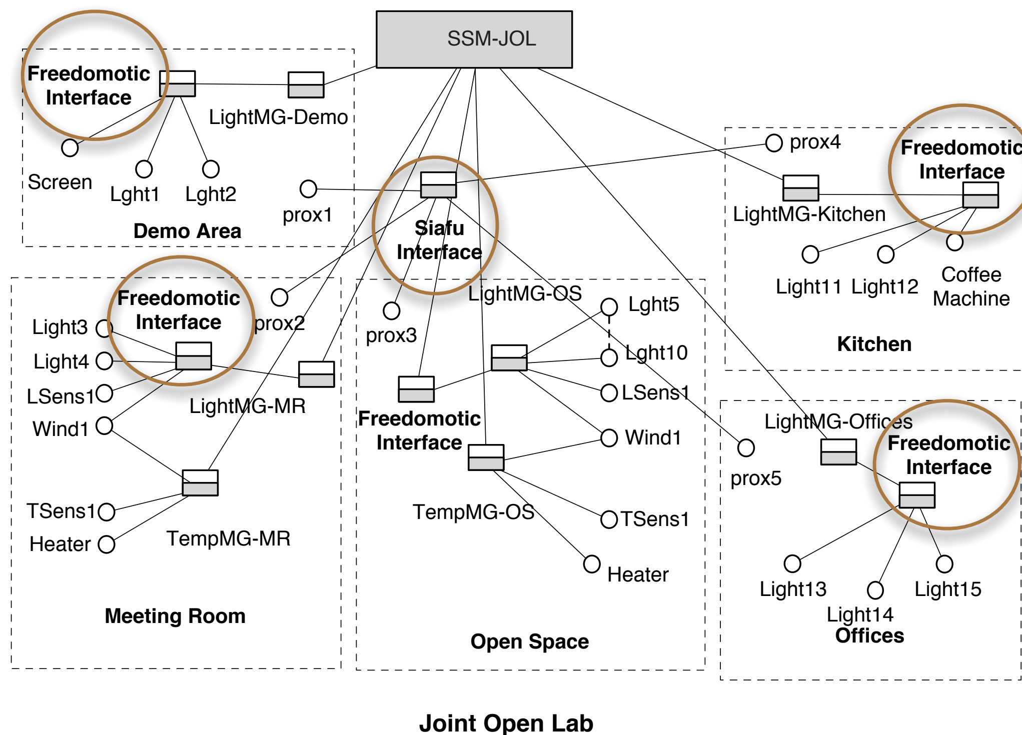


Figure 4.20: *JOL- Topology with external simulators*

performance by the set heuristics.

An evaluation of these characteristics is presented in section 6.5.

4.6 Continuous Validation

One of the significant challenges for **incremental development** of smart spaces is the continuous validation of the system at various development phases. In order to move from the simulated solution towards a concrete deployment, it is very important to test the system in a hybrid way by combining simulated entities and physical components together. One of the major reason for that is the inability to generate real environmental settings for evaluation of certain scenarios.

For instance, it is not trivial to create situations to test emergency evacuation or crowd navigation in large dynamic spaces. Moreover, testing with large number of components, after their deployment, would be very costly and it becomes hard to change the system/components in case system validation results are not good. Therefore, use of simulation frameworks becomes necessary to test the implemented system components, ideally with other deployed ones, to understand how they behave in realistic scenarios. Based on the validation results, a developer should be able to deploy the real components with their simulated behaviors without changing the rest of the system. More precisely, the framework offers two options to simulate the different entities:

1. The developer can define special-purpose roles that implement the foreseen behavior, as long as the real components are not available.

2. Any external simulator can be plugged-in to mimic the behavior of a single component or of an entire subsystem

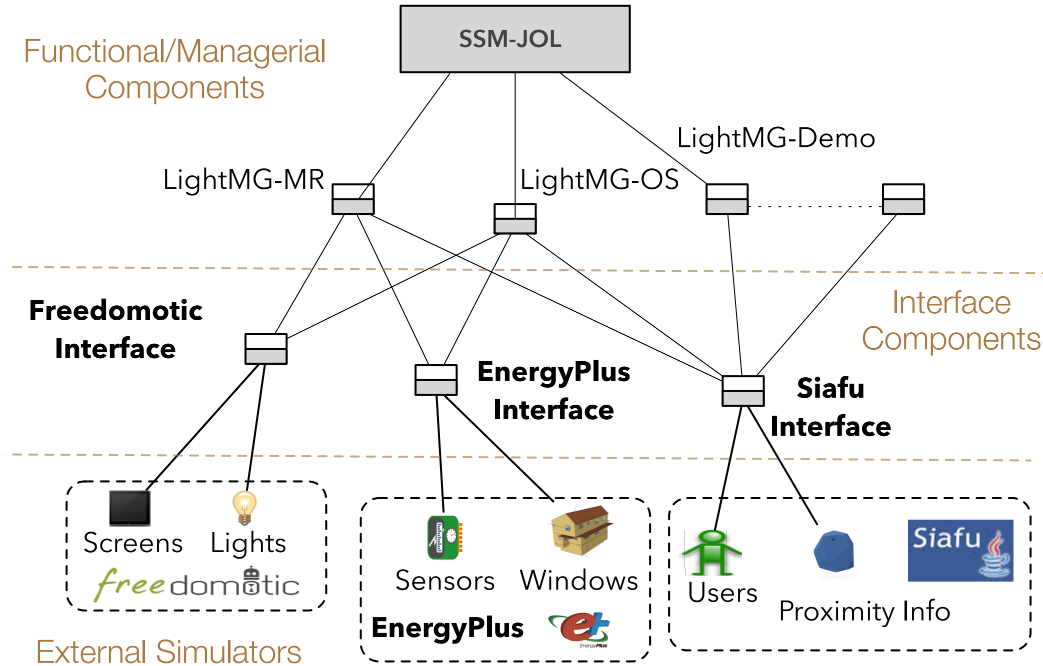


Figure 4.21: *JOL- High Level View*

The proposed framework, therefore, provisions the use of an external system (also to address the issue raised in **RQ 2.2.**) in the same way as any other component. It provides the component interfaces to maintain the state and identity of a (set of) components and at the same time being able to plug them to virtual or physical objects. All the component roles, behaviors and collaborations throughout remain independent of the type (physical or virtual) of the object associated to it. In this way, an application programmer can test and validate various configurations of the system using external simulators without incurring any additional cost.

For example, one can use Freedomotic's ability to communicate with actual devices within the lab through ZigBee. Figure 4.20 describes how Freedomotic can use the provided abstractions to act as an interface between real devices deployed at JOL. Similarly, to simulate user activity and mobility for testing, we can use Siafu simulator. Siafu will provide the proximity information and the rest of the system will act as described in Figure 4.16. This ability of the framework to replace a set of components with external simulators and concurrently execute both virtual and physical elements allows for the continuous validation of the system.

Figure 4.21 shows the high level view of the JOL design. It shows that how one can test the system by simulating the communication with all the lights and screens through Freedomotic, windows and energy needs through EnergyPlus, and the user proximity information with Siafu. It helps understanding and validating the system design and once a system component or subsystem is tested virtually, it can be replaced by actual

physical components. This hybrid simulation and deployment of the system will enable the incremental development and evaluation of the designed spaces.

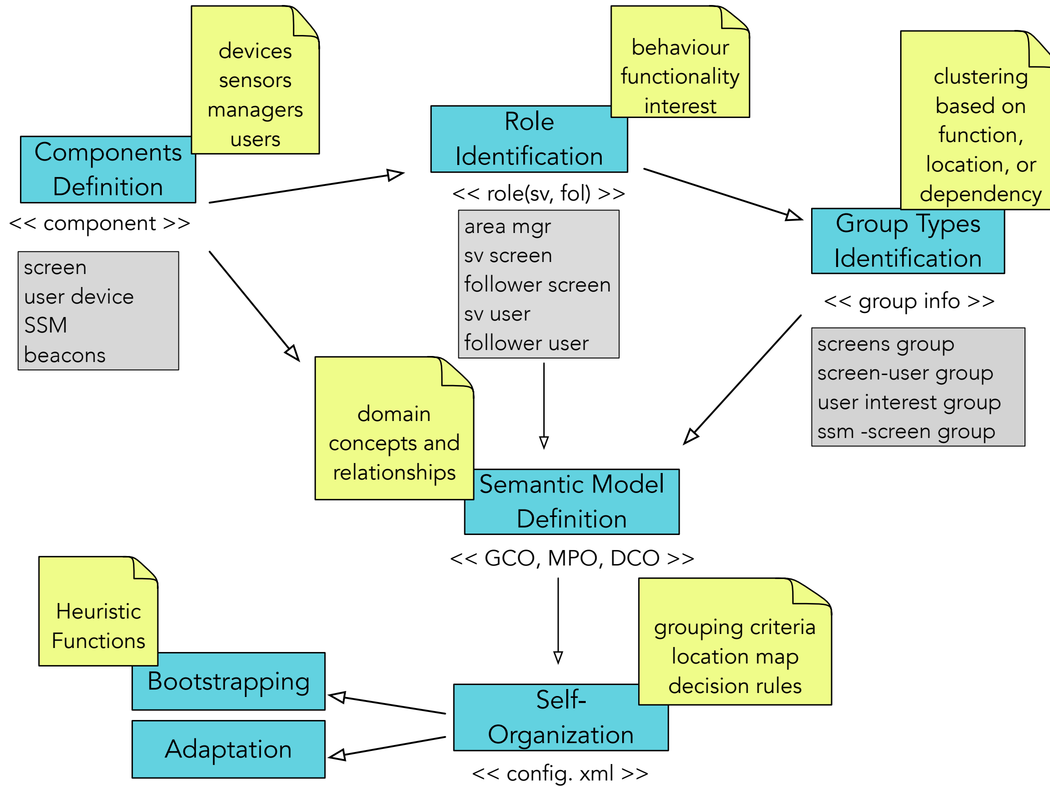


Figure 4.22: *Design Tasks*

Summary

Figure 4.22 summarizes all the high-level tasks pertaining to the design of a smart space using the proposed framework. The application programmer needs to carry out the following steps:

- Identify the components of the system and the corresponding capabilities that can be translated as framework defined "roles".
- Identify the group types and the formation criteria to form groups at runtime.
- Define the domain concept ontology, based on the component, role and group type identification, to harmonize various heterogeneous components.
- The self-organization takes place at runtime and organizes the components according to the defined grouping criteria and heuristic functions (used by firefly-based self adaptive algorithm).

Figure 4.22 also describes some of the identified component types (e,g. screen, user device, SSM etc.), roles (area manager, supervisor screen, follower screen, supervisor user and follower user), and group types (screens, screen-user group, user interest group

etc.) for our public park scenario. Chapter 5 explains how these components, roles and groups can be implemented and executed concurrently.

CHAPTER 5

Implementation and Concurrent Execution

This chapter explains the implementation details of the framework. It describes the APIs and concurrent execution mechanisms provided by the proposed solutions. The framework is responsible for coordination, control, and simulation of all the components that execute concurrently and communicate through messages. It models the concurrent heterogeneous components and systems such that their execution and interdependencies can be synchronized. The implementation of the group-based communication uses A-3 middleware [14] which is built on top of JGroups [12].

5.1 Implementation Model

As described in chapter 4, the framework allows a developer to implement various components through provided component abstraction or simulate/bridge it through any external system.

The proposed framework has been developed in Java and it provides APIs to define both the functional and managerial components of smart spaces. Application programmers can define components, roles (including behavioral logic), and groups information programmatically according to the implementation model shown in Figure 5.1. The implementation model describes the classes (APIs) provided by the framework, which are required to be used by developers to implement their system and exploit various framework capabilities.

Each component maintains its inherent data that can be used (or shared) by multiple roles played by that component. Each role, implemented as a separate thread, executes according to the defined behavior. Supervisor role, in addition to the follower role, also has an implemented MAPE (monitor, analyze, planing and execution) loop that may be used by the programmer to define control logic.

Each (external) system is also defined as a component that is interfaced with other system components to exchange data and information among them. The process of designing the concurrent execution flow can be divided into the following three steps (levels):

1. Definition of component and their behaviors
2. Selection of a message exchange paradigm
3. Employment of a group coordination style

There are many issues that need to be taken care of in order to achieve this concurrent execution, such as: (i) scheduling of these concurrently executing subsystems and information dissemination among them, and (ii) management of the data exchange peculiarities.

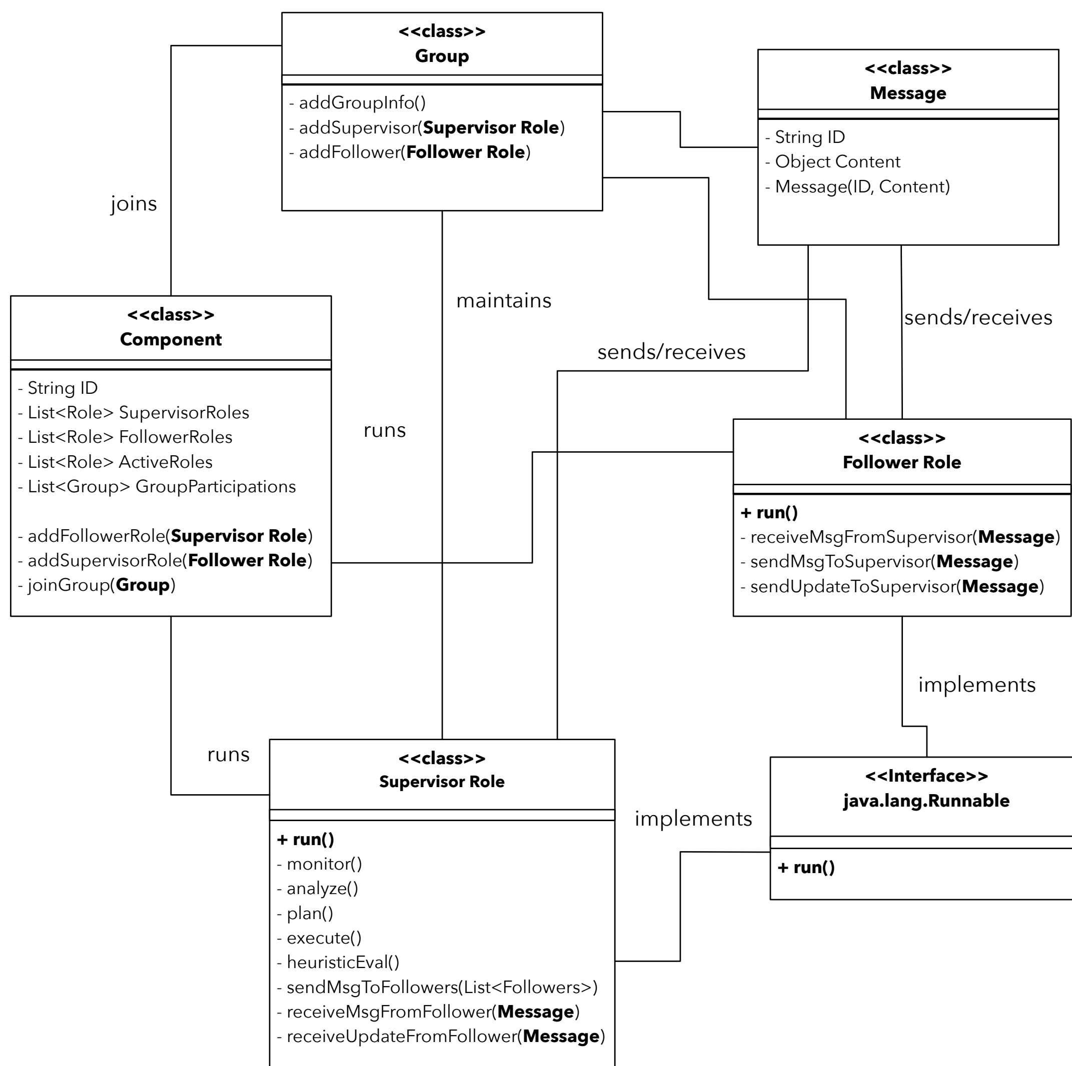


Figure 5.1: *Implementation Model*

5.2 Component Class

Component is the basic abstraction of the proposed framework and, therefore, a developer needs to extend *component class* to implement each sensor, actuator, device or controller that is part of the smart space. A component is identified by an ID and it maintains a shared state that is used by different roles played by it. As a component can participate in one or more groups depending on the situation, it is possible to define multiple roles which the component can employ at runtime.

To achieve this, a component object manages different lists: two of them for defining supervisors and follower roles (capabilities) that it can play, and the third one for maintaining the list of groups it is participating at a given moment. Therefore, after a component is created, its supervisor and follower roles (at least one) must be *added* through provided methods. It is important to note that component behavior/functionality is defined through the dynamic roles, and therefore, components will not be able to execute if there are no appropriate roles added to their role capability lists. A component also has a *joinGroup* method that will be discussed later.

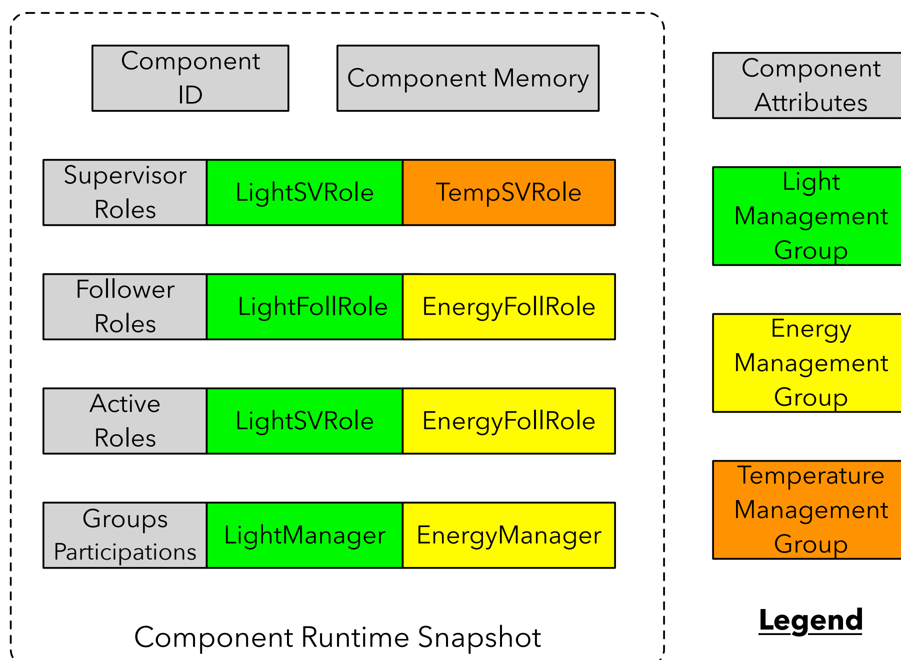


Figure 5.2: *Example Runtime Snapshot of a Component*

Figure 5.2 shows an example runtime snapshot of a component. The lists maintained by the component for supervisor and follower roles describe that the component is capable of playing the following roles: LightSVRole, TempSVRole, LightFollRole, and EnergyFollRole. These roles belong to three different group types, namely Light Management, Energy Management and Temperature Management groups. The component has two active roles in the given example which are LightSVRole and EnergyFollRole. Therefore, through these lists, components maintains the information about its participation in various groups in different capacities (i,e. supervisor or follower).

5.3 Component Roles and Behaviors

The first step is to define the behaviors of all the roles corresponding to each space component. A behavior represents a task, rules and execution logic to be carried out by a component (through roles) in a certain group. Figure 5.3 describes three possible behaviors that can be defined for a component role.

5.3.1 Supervisor Role Class

The framework also provides the Supervisor role abstract class that may be extended to define various supervisor roles. As shown in Figure 5.1 that the supervisor role class implements java runnable, which enables to execute instances of the supervisor class by a thread. The class must define a method of no arguments called run. This method is used to define role-specific component behaviors.

It is mandatory to always have an active supervisor in each group and if the supervisor of the group fails, the group stops to exist. Therefore, it only the supervisor component that can create a new group and followers can then join that group. As the supervisor component is responsible for managing the group, framework allows it to implement the MAPE control loop.

The supervisor class also implements two important functions (*receiveMessage-FromFollower and sendMessageToFollowers*) for the communication between members of the same group. The first function is called whenever a follower sends a message to the supervisor. So, this function needs to be implemented for each supervisor class in order to manage the received message. Similarly, the second function is used to send (unicast/multicast) messages to the followers. There is also a *receiveUpdateFrom-Follower* function that can be implemented to handle various updates received from the followers. We will discuss messages in the later section.

The last function is the *heuristicEval* function that is used when there is a need to define supervisor's suitability to supervise a group in order to select the best leader among various possibilities. For example, firefly-based adaptation can make use of this function to select the appropriate groups for different components.

5.3.2 Follower Role Class

The follower role class is used to define the behavior of follower components. There can be many followers in a group which can only communicate with supervisors. The structure of the follower class is similar to the supervisor one. Any class extended from follower role class is required to provide implementation *run* method to define desired behavior of the follower. It also has *receiveMessageFromSupervisor*, *SendMessageTo-Supervisor*, and *SendUpdateToSupervisor* functions that are used to communicate with the group supervisor.

As described earlier, both types of roles need to provide implementation for the *run* function, which defines the behavior. Following are the three important behaviors that are supported by the framework:

- **State-oriented (triggered) behavior** is used to program a component that has some inherent states such as lamps (on/off state) and window blinds (open/close state). The components wait in certain state and on some events (or triggers) take

the transition to the next state that would result in communicating that information to relevant peers. The triggered behavior is used when there is a need to report (or request) certain events (changes in state) related to a particular component. Figure 5.3 shows how one can model a two state lamp through triggered behavior.

- **Cyclic (timed) behavior** repeats itself after a defined time interval. This behavior type can be used to create behaviors that keep executing continuously and perform certain tasks after a defined time (e,g. sensors that send/generate periodic data). In context of the JOL, we can use the cyclic behavior to model temperature and luminance sensors that send data periodically, say every 60 seconds, to the room manager.

- **Control (MAPE) behavior** corresponds to a control loop [27] where certain data is monitored (collected) and analyzed, and, then the required actions are planned and executed. This behavior is orthogonal to the cyclic and event triggered behaviors as it can be used in complement to those behaviors. For instance, this behavior may be incorporated by the supervisor role as it needs to collect certain information from the followers (e,g. running cyclic behaviors), analyze that data, plan the set of actions that are required, and then send the acting (executing) directives to the followers (e,g. running trigger behaviors). In case of the JOL, a room manager will implement this behavior and it can *monitor* the data collected from sensors and external simulators, *analyze* it, *plan* the actions that are required, and send *execute* commands to lamps, windows, or external simulators.

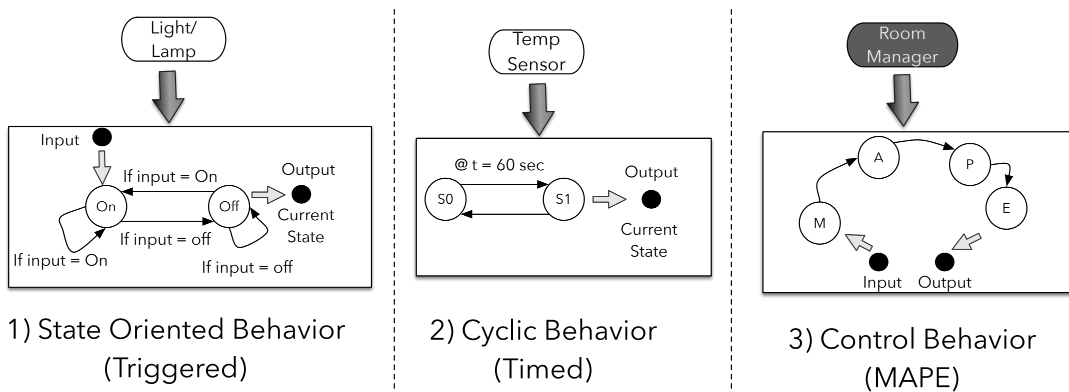


Figure 5.3: *Component Behaviours*

5.4 Asynchronous Message Exchange

The second step is the selection of messaging paradigm for the different components of a space to communicate with each other. The components in our framework may be deployed on different nodes and they generate data flow dependent on concurrent but asynchronous or untimed processes (roles).

5.4.1 Message Class

The message exchange between different components (within a group) is done through the send/receive message functions implemented by supervisor and follower roles. The *Message* class defines the body of the messages to be exchanged among roles. A message is a container which has an ID and the content (implemented as a generic object). The developer is free to define the type of the content. The message objects are serialized in order to perform the exchange of object on the network.

5.4.2 Physical Communication

The framework uses two different mechanisms to model the communication between the two type of implementations as shown in the Figure 5.4.

As stated earlier, the group communication uses Jgroups, which enables multicast communication among components and messages can be exchanged using both IP and TCP. A cluster (group in our case) can be formed with members (components) joining from both Local Area Networks (LANs) and Wide Area Networks (WANs). It supports the mechanism to inform each component of the group about the changes within the cluster. Each component is associated with a channel (per group) that is used to receive all the communication to/from the component in a group. The supervisor can send messages in the following ways:

- Broadcast, to each follower of the group
- Multicast, to some of the followers of the group
- Unicast, to only one of the followers of the group

On the other hand, the communication with the framework components and the external systems is done through sockets. The external system (usually a simulator) is plugged in with the framework and continuous exchange of data takes place through the dedicated socket connection. The data refers to the sensing and actuation directives that can be sent to/from the simulator and the framework.

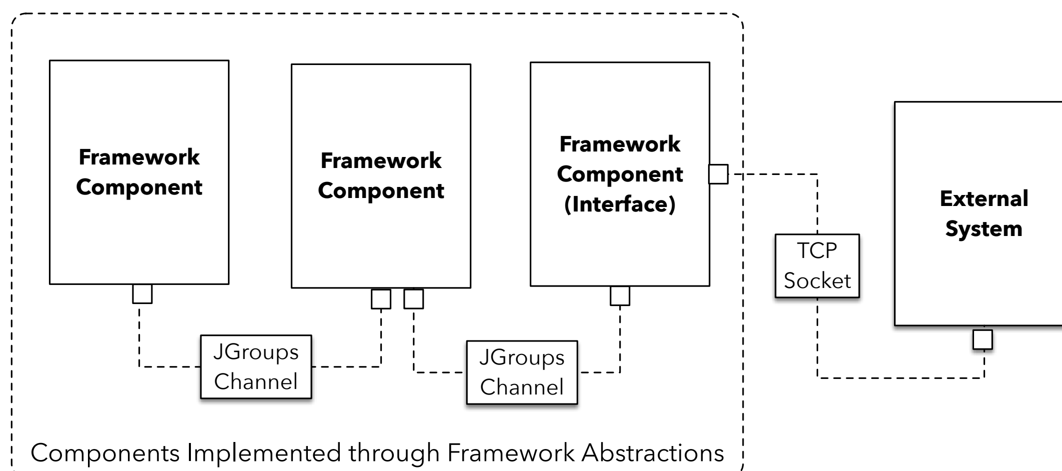


Figure 5.4: *Communication Channels*

The communication and data exchange between the two components is achieved through asynchronous messages [52]. We support the following two messaging paradigms within our framework: (i) messaging queue and (ii) rendezvous.

5.4.3 Messaging Queues

Messaging Queues [33] allow one to model the communication between components based on "Send and Forget" pattern [59]. In the proposed framework, the components send messages to each other via distributed framework and these messages are stored in queue associated with each component from where it dequeues them one by one for the processing.

For the JOL example, let us consider the room manager (supervisor role) that receives status updates from different sensors and actuators. All the messages directed to room manager will be stored (put) in the messaging queue (by the framework) associated with it and will be processed sequentially.

5.4.4 Rendezvous Exchange

Rendezvous data exchange is instantaneous in nature [41]. The framework emulates a pairing between the two components (sender and receiver) which remain in the block mode during the communication. Although, the messages are still asynchronous (as components keep on executing their behaviors), but framework ensures a turn-based interaction between the two components. In context of the smart spaces, rendezvous is very useful when two isolated (external) systems need to talk to each other. They can create a fast paired connection (e.g. sockets) to send and receive data according to defined information exchange rules.

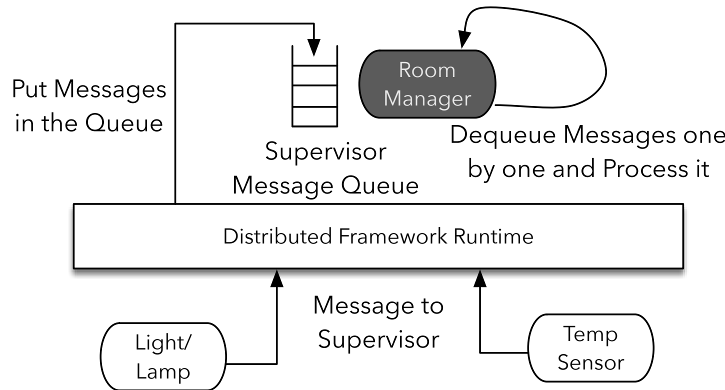
For example, one can use rendezvous message exchange between the physical devices and an external simulator (such as Siafu) that simulates the mobility of the users within the space.

5.5 Group Coordination Styles

Scheduling the communication and Coordination among the participating components within a smart space is the final step. In order to model the dependencies and information exchange among various independent components, one needs to define the order in which these components receive or send messages and then execute their desired computations and behaviors. Groups can follow different computational and communication models for their execution that can be customized according to the system and scenario needs.

5.5.1 Group Class

The *Group* class is used to define a group. The developer needs to create an instance of this class with the relevant group information, that is, supervisor and follower roles specification. The group instantiation requires the name of the role classes that will be used as default roles for the group. These roles will be used, when any component joins the group, to determine the capacity (supervisor or follower) in which the joining component will participate in the group. If the roles that can be played by the components



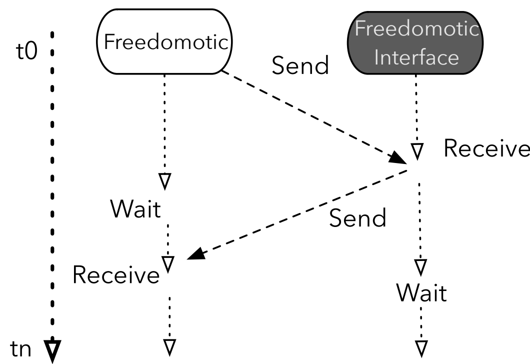


Figure 5.5: *Asynchronous Messaging Paradigms*

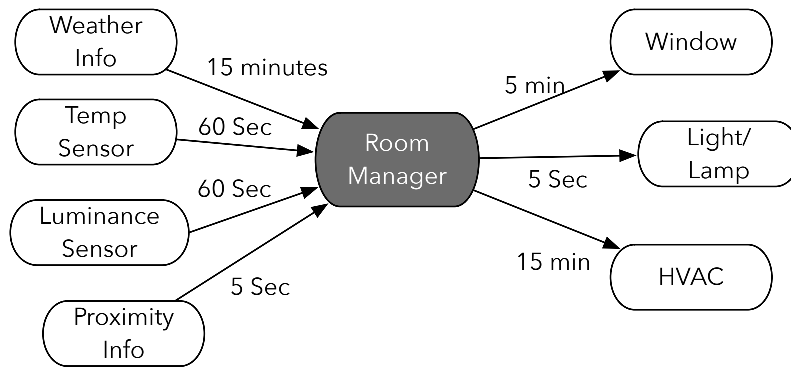
are different then the roles required by the group, the component will not be able to join the group.

A component then, by knowing the details of group information, can join the group through *joinGroup* method. Therefore, before a component tries to join a particular group, it must have received the information concerning the group, and the developer needs to design the right set of roles and data that are required in order to make a group work as desired.

5.5.2 Synchronous Timed Coordination

Synchronous Timed Coordination has the notion of common time, and different components synchronize among them according to a global (or distributed local) clock [54]. In context of our framework, it refers to the ordering of multiple independent components according to the time duration of their execution. Components send and receive messages to each other (via roles) on clock tick(s) in order to synchronize the information.

To understand this, let us consider the JOL scenario where movement of windows shutters are controlled on the basis of light and temperature in the lab. Weather and en-



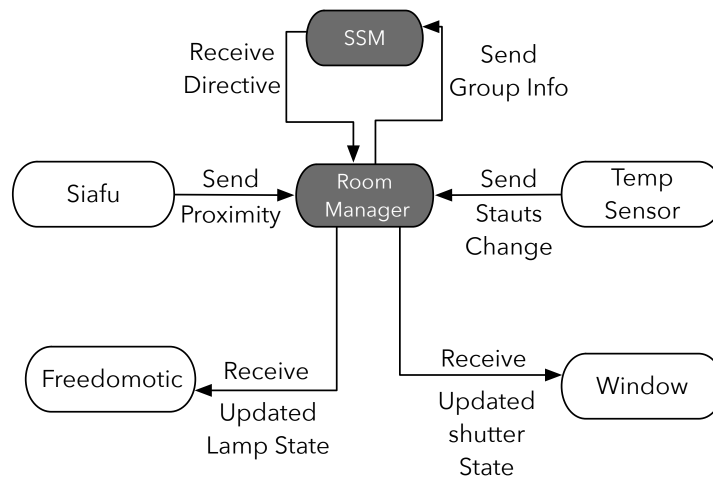


Figure 5.6: *Group Coordination Styles*

ergy consumption constraints are also considered during the decision making process. It can easily be seen that weather information (sometimes updated after 15 minutes), internal light and temperature (available instantly) and windows movement (should or may take minutes to change its status from open to close or vice versa), all have different time steps and data granularity and it becomes absolutely necessary to synchronize all of them based on some global time. The group supervisor in this case would take the responsibility of collecting the information from followers at different time intervals and then synchronize them according to a common time (step) to make valid decision.

5.5.3 Event-based Coordination

Event-based Coordination works similar to publish-subscribe [42] as some components sends the messages (or events) to a mediator (supervisor), whereas the other components who want to receive that particular information can follow (or subscribe to) that

and are notified when any message is generated. The messages are published without knowing about the subscribers (or receivers) and similarly, they are received by the subscriber without knowing about the sender. In the proposed framework, sensory information can adapt this style as sensors (periodically) publish contextual data to the supervisor which in turn delegates that information (or corresponding directives) to other components that require it.

In the JOL, room manager can work as a mediator between different components that need information generated by one another. For example, temperature sensor and Siafu can send (publish) temperature and user proximity information to room manager, whereas Freedomotic and window shutters receive (or subscribe to) the relevant information from room manager.

A coordination pattern is not enough for data exchange between heterogeneous components. Roles explicitly define which information will be received and sent between them to work (more or less as plugs). We also need to consider the following issues:

- Multiplicity of data that needs to be exchanged between the two involved components
- Data dependency and flow rate
- Time steps for execution (no. of iterations in case of simulation)

The first thing we need for efficient data exchange is to define what is the type and structure of the data that needs to be communicated between two components. For example, a sensor component would send the context reading to the room manager, a window shutter needs to get action command (the slat angle) from the manager and the manager needs to send the status of all lights, windows, sensor and HVAC status to SSM. The multiplicity of data is defined through roles and supervisor and follower roles must agree on the data exchange model. As described earlier, roles correspond to the programmable behaviors and you need to define the execution loop for these behaviors. The execution loop will represent the data flow rate for a specific component behavior. The next step is to use one of the communication patterns described above to instruct the framework to schedule the execution of those components as per the requirements.

CHAPTER 6

Evaluation

This chapter addresses **RQ 3.** that is related to the evaluation of the effectiveness the proposed solution. It describes the assessment and validation of the proposed framework through different case studies and scenarios. The selected case studies are chosen from different application domains, and they vary in terms of the user requirements, involved devices and the type of user interactions. First, various quantitative and qualitative evaluation metrics are described, which are then used to demonstrate how the proposed framework can be used efficiently to design diverse smart spaces such as greenhouse, smart office, public buildings and public parks.

6.1 Evaluation Plan

The evaluation will consider both qualitative and quantitative performance parameters as summarized in Tables 6.1 and 6.2. All of these measures are defined in correspondence to the challenges identified in section. 2.5.

Table 6.1: *Quantitative Evaluation*

Measure	Metrics
Efficiency	Bootstrapping time, Delay (impact on performance)
Scalability	Additional delay introduced when number of components grows
Reliability	Time required to recover (re-configure) from a component failure
Validity	Error rate in synchronizing heterogeneous systems

The quantitative evaluation aims to measure the efficiency, scalability, reliability, and validity of the proposed framework. Efficiency is measured in terms of the bootstrap time of a component when it wants to join the system and the overall delay that is induced by the framework. Scalability measures the additional delay introduced by

the framework when the number of components grows. Reliability of the framework is analyzed in term of time required by the system in case a component fails or leaves the system unexpectedly, whereas, validity metric shows the soundness of synchronization of heterogeneous systems models.

Table 6.2: *Qualitative Evaluation*

Measure	Assessment
Interoperability	Is the framework able to handle heterogeneity in terms of devices, protocols, and external systems?
Integration	Can the framework concurrently execute (interacting) systems with different execution models?
Dynamism	Does the framework self-organize, in case components join or fail, without halting?
Extensibility	Is it possible to add new functional components (roles) without changing/halting the system?
Incrementality	Does the framework support both physical and virtual components together for continuous evaluation?

Similarly, following qualitative evaluation metric will be used to analyze the system: interoperability, integration, dynamism, extensibility, and incrementality. Interoperability and integration measures would assess framework's ability to handle heterogeneous devices, communication protocols and systems (e,g. external simulators), integrate them and then co-execute them together in a synchronized manner. The framework will also be evaluated for its capability to manage high dynamism of the components without halting the system or other user services. Incrementality of the system would also be analyzed, that is, whether the framework is able to support continuous validation and incremental development through ensuring co-existence of both physical and virtual components together.

Table 6.3: *Evaluation Plan*

Evaluation Metric	Case-Study
Efficiency	Smart Office, Public Park
Flexibility	Smart Office, Energy Efficient Building
Reliability	Public Park, Greenhouse
Scalability	Smart Office, Public Park
Interoperability	Smart Office, Greenhouse, Energy Efficient Building
Integration	Smart Office, Public Park, Energy Efficient Building
Dynamism	Public Park, Greenhouse
Extensibility	Smart Office, Public Park
Incrementality	Smart Office

Four different prototypical implementations for diverse smart space scenarios are conducted to evaluate the proposed framework. Table 6.3 provides a mapping between the goal of each implemented case-study in terms of the evaluation metrics it intends to validate:

- Smart Office (JOL) case-study targets to provide efficient lighting control system within the lab according to use proximity. It is used to demonstrate the framework's ability to cater for all the challenges (except for high dynamism).

- Smart greenhouse case-study demonstrates the autonomic configuration of dynamic heterogeneous components.
- Implementation of energy efficient building case-study aims to showcase that framework is flexible, valid, supports interoperability, and integration of heterogeneous systems.
- Public park scenario is implemented to test the ability of proposed synthetic approach (firefly with architecture level control) to deal with highly dynamic systems, where different components need to be integrated, without much loss of efficiency.

6.2 Case Study 1: Smart Office

This case-study is about provisioning automated lighting control in our Joint Open Lab (JOL). This case-study focuses on the following aspects:

- How can we integrate heterogeneous (in terms of type, function and communication protocols) components for interoperation through the proposed framework?
- How much time it is required for a component to join the system (bootstrap) and how does this delay scale up with the increasing number of components?
- How much delay is caused by the framework when components send messages to each other?
- Is the framework able to co-execute both virtual and real elements together and how flexible is the framework in terms of replacing one (or set of) virtual component(s) with real ones?

Table 6.4: *Joint Open Lab - Components*

Component	Type	Communication Technology
Lights	Physical	Zigbee
iBeacons	Physical	BLE
Freedomotic	Virtual	TCP Sockets
OpenHAB	Virtual	REST Service
Siafu	Virtual	TCP Sockets
Server	Computational	WiFi

As a first step, we started with prototyping a virtual system using the provided design abstractions (native APIs) both for implementing and simulating components (defined in Table 6.4) behaviors. We used the same group configuration as described in Figure 4.16.

The next step was to use external simulators (Siafu and Freedomotic) to simulate the behavior of users and devices, while the room and lab level management used the same abstractions. In the end, we experimented with real ZigBee enabled lights and BLE proximity beacons (to detect the presence of real users) to understand system performance in the real scenario.

Experimental Setup:

We used three different machines and a smart phone for conducting our experiments. We deployed the SSM, room managers and external interfaces on a Windows 8.1 (64 bit) virtual machine (WinVM) with 2 GB of RAM, running on an HP workstation (Intel Xeon CPU, E3-1245, 3.40 GHz) with OS Windows Server 2012. The Freedomotic and openHAB instances were hosted by an Intel Atom dual core (D2550, 1.86 GHz) machine (IntelAM) with 2 GB of RAM and running Windows 7 Home Edition (64 bit). The Siafu was run on a Macbook Air with Intel Core i7 (1.7 GHz) with 8 GB of RAM running OSX 10.9.5. Estimotes[1] (BLE enabled beacons) and Google Nexus 5 (Android 4.4) were used to get the proximity information of the real users. Bticino's radio/Zigbee control switches[2] and Open Web Net/Zigbee gateway[3] (ZigBeeGW) with Radio 2.4 GHz standard ZigBee technology were used to communicate with lights.

6.2.1 Simulation with Native APIs

In this implementation, we modeled all the JOL components only with framework's native APIs. Roles were used to define both the functional and simulation behaviors of the components. For instance, a luminance (light) sensor role was not only responsible for generating random luminance level in the room, but it also sent the generated context to the room managers. We clustered the components in three different type of groups:

1. Smart Space Management Group (MG), where SSM is the supervisor whereas each room manager acted as a follower
2. Context Management Group (CG), where room managers are the supervisors and all the temperature and light sensors are followers
3. Appliances Management Group (AG), where room managers are supervisors and all the appliances are followers

Table 6.6 summarizes that we have used event-based group coordination for components of MG as room managers update the SSM only if they have new context information to share or some other change has been observed within the group. Room managers send the summarized context information to the SSM, which in turn send corresponding control commands, if required. CG uses synchronous timed coordination as it needs to synchronize the environmental conditions in the JOL with control decisions to maintain the desired lighting and temperature state within the lab. The time step of the execution cycle is set to 60 seconds. AG follows event-based coordination to transit between different appliance states (e,g. luminance level or thermostat setting) according to the current context generated through CG after 60 seconds.

The delay in message passing between the components was negligible (less than 1 millisecond) in this case as summarized in 6.5. Moreover, each component has to join a group to be part of the system, and there is some associated bootstrapping cost. Experimentation with various number of heterogeneous groups (varied from 1 to 20 components), we recorded that it takes each component around 1.4 seconds on average

[1] http://estimote.com/api/
[2] http://www.bticino.it/comandi-luce/comandi-radio
[3] http://www.myopen-legrandgroup.com

to join a group and start its execution (See Figure 6.1). Each case was run for 5 times and results are averaged out.

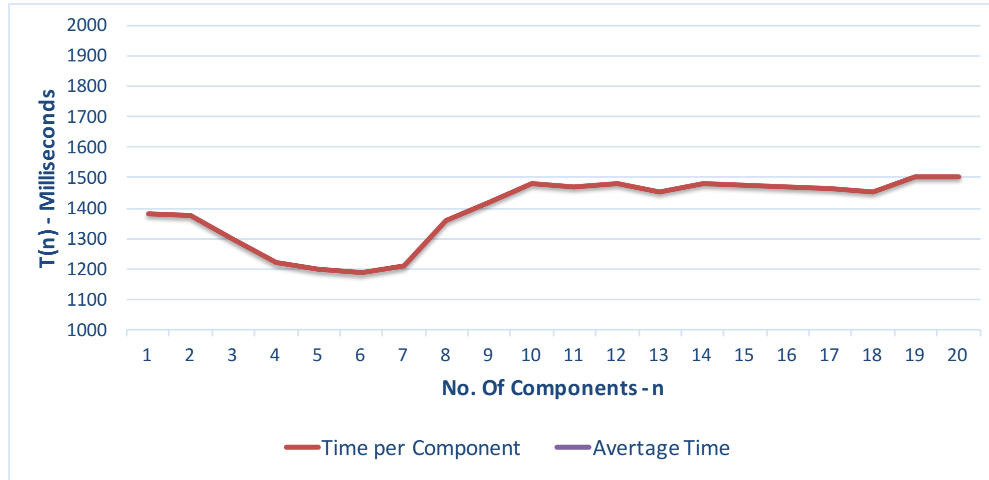


Figure 6.1: *Bootstrapping Time for Components*

6.2.2 Simulation based on External Simulators

In order to test and validate the system, one needs to have a more sophisticated and realistic scenarios. There are some context and device simulation frameworks available that enable you to test your system in realistic scenarios and environmental conditions. We used Siafu to model the detailed user daily routine in JOL. We also designed three activities, in addition to daily work schedule, namely meeting, demo and lunch time. Each user is assigned a work desk and room lights are controlled based on the desk occupancy and overall energy requirements. Similarly, the activity context information is also used to preempt the lighting or other functional needs in a particular room. For example, if the context declares that it is meeting time, lights and HVAC in the meeting room are turned on in advance to prepare for the meeting. The lights and other actuators are simulated through Freedomotic that also presents a visual representation of all the objects[4] .

We created two interface components to communicate with external simulators and they form External Interface Group (EIG). Table 6.6 shows that EIG inter-group communication follows synchronous timed coordination style and rendezvous messaging paradigm (implemented through sockets). External Interface, as a supervisor of the group, provides the server socket and various external systems can communicate with it through the defined socket. Both supervisors and followers need to agree on the data to be exchanged. The communication between room managers and external interfaces was done through synchronous timed coordination with a time step of 60 seconds.

Figure 4.21 describes the high level view of the system, whereas, Figure 6.2 shows a snapshot of the co-simulation with Siafu and Freedomotic during a meeting. In this implementation, a message is passed to other components within 1 millisecond whereas the bootstrapping cost remains the same. It is important to note that message delay refers to the time consumed between room manager deciding to change the status of a

[4]The demo is available online at: `http://home.deib.polimi.it/shahzada/JOLDemo/demo-JOL.mov`

Figure 6.2: *Simulation in Freedomotic and Siafu*

light (for example) and the control command received by the light. It does not account for the delays caused by simulation frameworks and data flow restrictions (e,g. time steps).

6.2.3 Partially Deployed (Simulated) System

The next iterative step is to test the designed system on real objects. We used openHAB [5] as a bridge to communicate with ZigBee enabled lights in the JOL and replaced the virtual lights with real ones. We used the same Siafu simulation to mimic user behavior and accordingly controlled real lights in the lab. We just needed to change the configuration mapping, that is, from virtual lights to real lights in order to make the previous prototype work on real lights. There is no change required in group formation or component roles whatsoever. The delay in message communication is 3 to 4 milliseconds in most of the cases as there is also a bridge to ZigBee controller involved while sending commands to the lights. It is important to note that there is also some delay caused by the polling time due to the message queue at openHAB HTTP service interface (external interface in our case).

6.2.4 Fully Deployed System

The final step is to fully deploy the system without simulating user and device behaviors. We used real ZigBee enabled lights, BLE proximity beacons and Android based

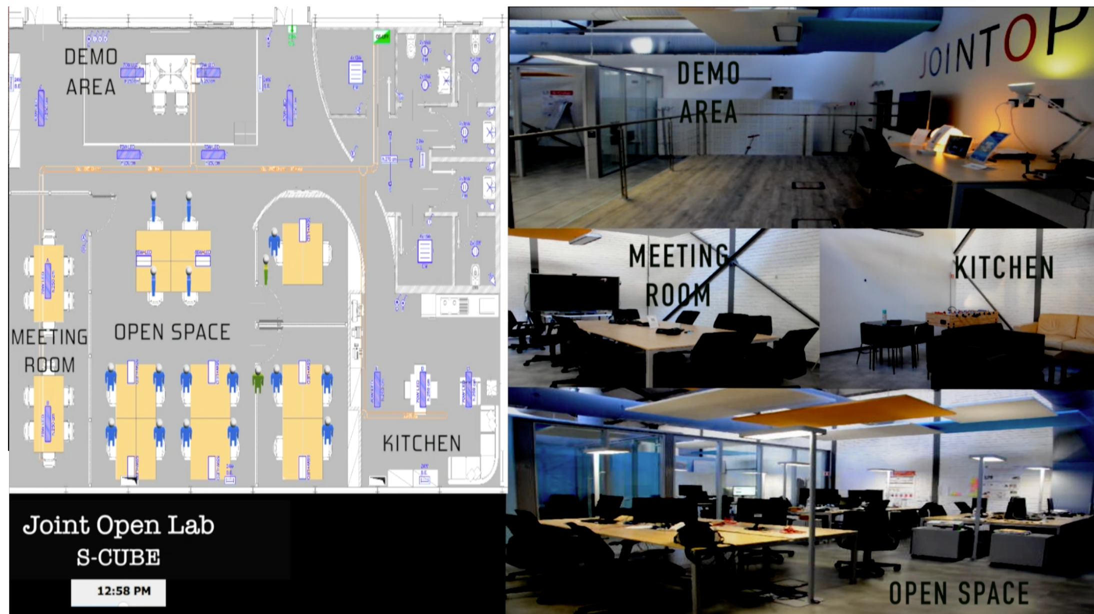


Figure 6.3: *Partially Deployed System (Siafu and real ZigBee enabled lights).*

application that communicates with beacons to understand a user's location within JOL. There is no change in how we addressed ZigBee lights in previous partially deployed system. For user location, an android application interacts with the BLE proximity beacons placed in all rooms in JOL and sends this context information to room managers using event-based coordination. Figure 6.4 describes the working of the final deployed system that is based on the following steps:

1. User proximity is sensed through BLE beacons and mobile device (Nexus 5)
2. The proximity info is sent to room manager (WinVM) through Wifi
3. Room manager sends the corresponding lighting control command to external interface that delegates it to Freedomotic (IntelAM)
4. Freedomotic sends the command to actual light through ZigbeeGW

We used the event-based coordination for that as there is no need to cater for the proximity information periodically if there is no change. The framework only needs to be notified if any of the users enters or leaves a room. Again, the group formations remain the same, and we just required to configure the supervisor address information. The message delay, in this case, is 5 milliseconds which is slightly more than the partially simulated system.

The results above demonstrate that framework can integrate (at runtime) heterogeneous components (see Table 6.4) that vary in terms of their type, function and communication protocols. Further, the framework was able to support the incremental development of the JOL smart space, that is, moving from the completely simulated solution (using native APIs and external simulators) to the fully deployed one without the need to change the high-level component organization. This is possible because of the framework's ability to co-execute both virtual and real system elements together and

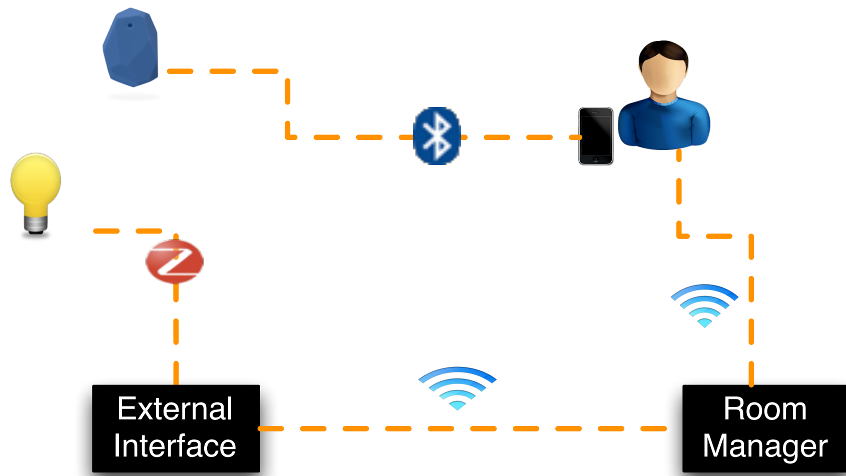


Figure 6.4: *Fully Deployed System*

Table 6.5: *Message Delays*

Average Message Delay (milliseconds)			
Native APIs	External Simulators (Siafu + Freedomotic)	Partially Simulated (Siafu + real lights)	Fully Deployed
<1	1	3-4	5

the flexibility to replace one (or set of) virtual component(s) with real ones over the time.

The framework takes around 1.4 milliseconds on average to bootstrap a new component and this time does not fluctuate too much even when you increase the number of components deployed (on a single machine). Moreover, the delay incurred by the framework is negligible for developed case-study as it takes from less than a millisecond to 5 seconds to deliver one message from one component to other depending on the communication technologies used and their corresponding limitations.

Table 6.6: *Groups Specification*

Group	Supervisor			Follower			Data Exchange		Messaging Paradigm	Group Coordination
	Device	Role	Behavior	Device	Role	Behavior	Sent by Supervisor	Sent by Follower		
Management Group (MG)	WinVM	SSM	Control	WinVM	Room Manager	Triggered	Control Commands	Context Change	Messaging Queue	Event-based
Context Group (CG)	WinVM	Room Manager	Control	IntelAM	Sensor	Cyclic	✗	Context Info	Messaging Queue	Synchronous Timed
Appliances Group (AG)	WinVM	Room Manager	Control	ZigBeeGW	Lamp Screen	Triggered	State Change Commands	Status Updates	Messaging Queue	Event-based
Simulation Group (SG)	WinVM	Room Manager	Triggered	WinVM	External Interface	Cyclic	Control Commands	Context + Status	Messaging Queue	Event-based
External IF Group (EIG)	WinVM	Interface SV	Cyclic	IntelAM Macbook	FD IF Siafu IF	Cyclic	Context	Context	Rendezvous	Synchronous Timed
Proximity Group (PG)	WinVM	Room Manager	Control	Nexus 5 (Beacon)	Proximity Interface	Triggered	✗	Proximity Events	Messaging Queue	Event-based

6.3 Case Study 2: Modern Greenhouse

This case-study aims to develop an automated organization of heterogeneous flower carts according to their thermal (or other) needs within a modern greenhouse. It focuses on the following two aspects:

- How can we harmonize heterogeneous components for interoperation through the proposed semantic model?
- How can the proposed semantic model help in self-configuration of the groups?

Experimental Details

The first step was to develop the domain concept ontology (DCO) for the greenhouse scenario. Figure 6.5 describes greenhouse DCO that consists of domain concepts such as cart, room, temperature requirements and health status of a cart, and room status and temperature. The group coordination ontology (MPO) and the message protocol ontology are the same as described in Figure. 4.12.

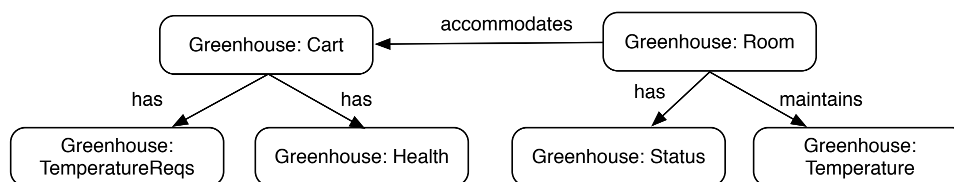


Figure 6.5: *Greenhouse Domain Concept Ontology (DCO).*

The greenhouse is modeled in a way that each room is represented by a group in the Components Layer, and the carts are components that become followers of these groups (see Figure 6.6). In the Management Layer we have a single Greenhouse SSM that is responsible for supervising the room supervisors. Room supervisors are responsible for collecting data from their followers. Their goal is to identify their follower carts' needs and report to the SSM if they see any abnormal behavior or data. The room supervisor can ask a flower cart to leave the room at any time, either because its flowers get sick or because the SSM decides that it should no longer be part of that group. The SSM, on the other hand, issues directives that can change the supervisor for a particular room. This in turn may have an impact on the type of flower carts that can be accepted in that room.

We used Siafu to build a model of our greenhouse, and to populate it with agents for carts and room supervisors, and one for the SSM: the agents are virtualized elements. We then associated each element with a framework component. The components were defined using the GCO, communication followed the MPO, whereas special-purpose greenhouse DCO (see Figure 6.5) is used for application-specific message content.

Components discovery and self-configuration of groups

The framework offers a mechanism for components to connect to the system without having any previous knowledge about the topology. When different components want to join a system, they insert their role information (described using the GCO) into

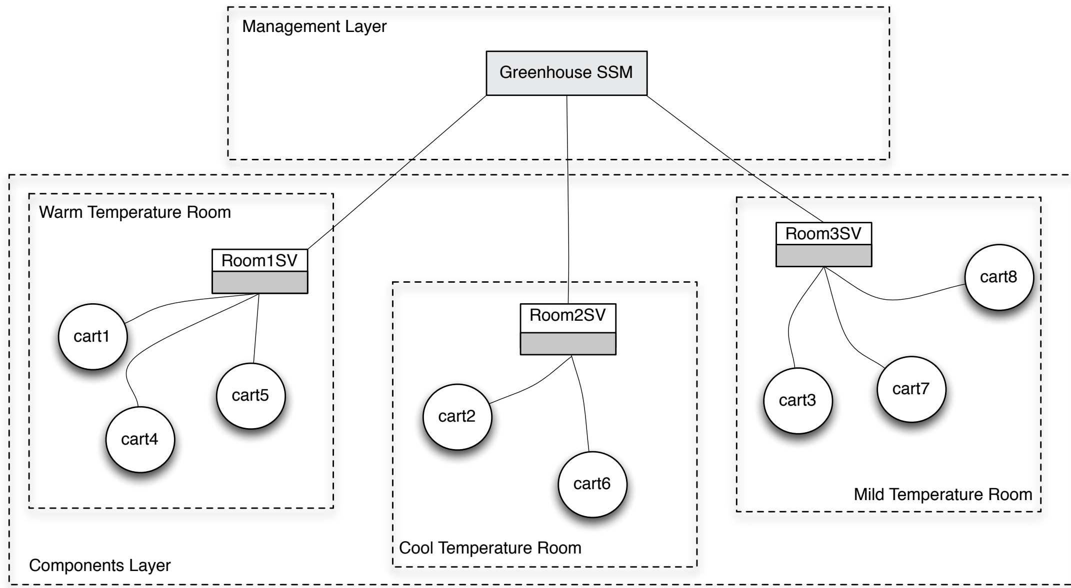


Figure 6.6: *Group topology: the SSM manages the room supervisors (Room1SV, Room2SV and Room3SV) that play the role of supervisors for carts, but act as followers for the SSM node.*

the knowledge base in the form of RDF triples. For example, a component with id $cart1$ with the follower role *CoolTempeatureRole* will insert the following triple into the knowledge base:

$$\langle SeSaMe : cart1, SeSaMe : FollowerRole, \text{``}CoolTemperatureRole''\rangle.$$

SSMs subscribe to this information (see Figure 6.8), so they are notified as soon as a triple is inserted into the knowledge base. They then collaborate to chose a group for the component. If we consider the above example triple, the SSMs will find a group that has listed *CoolTempeatureRole* as its follower role. After choosing a group for the component, the SSMs insert their decisions into the knowledge base in the following form:

$$\langle SeSaMe : cart1, SeSaMe : Group, \text{``}CoolTemperatureRoom''\rangle.$$

The component is made aware of this input, and it connects to the assigned group. If a component can play multiple roles in multiple groups, it will add multiple tuples to the knowledge base. The SSMs can then decide to add the component to as many groups as needed.

Semantic group communication

Message exchanges between two components are carried out using the *Message* concept defined in the MPO. Message communication involves many different aspects, such as message content encoding, the transportation and communication mechanisms that should be used, and how message content should be interpreted. Message transportation is managed by the group-based infrastructure, but message content requires

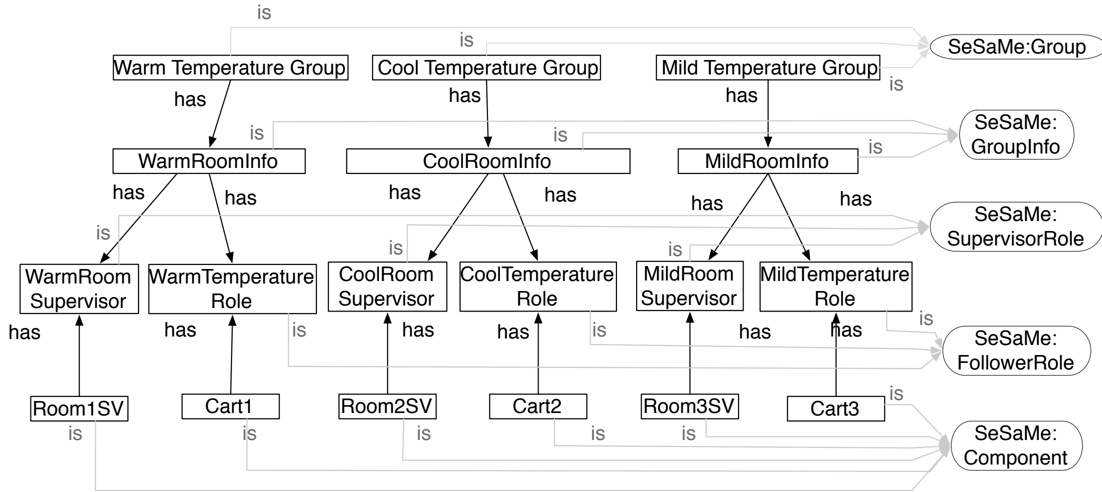


Figure 6.7: *RDF graph for greenhouse group topology.*

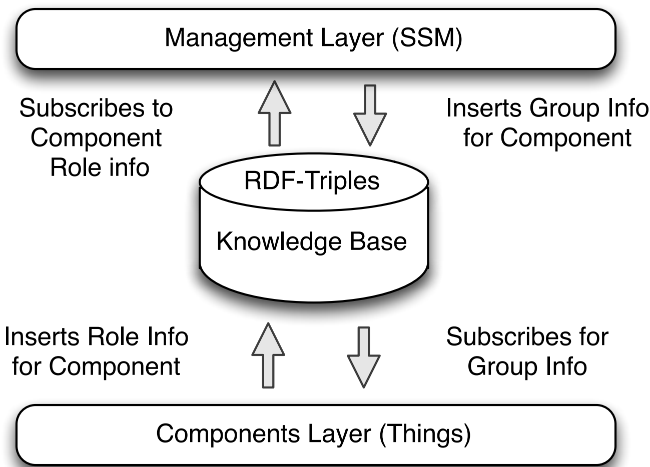


Figure 6.8: *Discovery and self-configuration of components.*

an ontology that defines the concepts that are relevant to the application domain. This is covered by the DCO. Figure 4.12 shows the simple DCO ontology that we have used in our smart greenhouse. Carts and rooms within a greenhouse become components if they can play certain roles. Message passing between carts and rooms are achieved between their supervisor or follower roles. The message contains an instantiation of the cart's *Health* concept, or an instantiation of the room's *Status* concept. If wanted, the DCO can be extended to define more detailed aspects of the health and status messages.

We used a context simulator for ubiquitous systems called Siafu [73] to help demonstrate our framework. Siafu was run on a Macbook Air with Intel Core i7 (1.7 GHz) with 8 GB of RAM running OSX 10.9.5. We have tested the framework on two specific scenarios.

6.3.1 Managing incoming carts

The first scenario we tackle represents the case in which three new flower carts, named $cart1$, $cart2$, and $cart3$, enter the greenhouse. $Cart1$ is carrying flowers with high temperature needs, and hence implements the *WarmTemperature Role*. Similarly, $cart2$ and $cart3$ have *CoolTemperature Role* and *MildTemperature Role*, respectively. Group Information (Supervisor and Follower Role) about all the potential group types needs to be pre-registered with the Greenhouse SSM to enable autonomic discovery and group assignment.

Upon reaching the greenhouse, the carts insert their role information into the knowledge base to know which room they should go to. The information is received by the Greenhouse SSM that finds the appropriate rooms for the carts according to their roles (corresponding temperature needs). In the current example, the SSM allocates $cart1$ to the *Warm Temperature Group*, $cart2$ to the *Cool Temperature Group*, and $cart3$ to the *Mild Temperature Group*. The SSM also takes into consideration the number of carts that are present in each of the system's groups.

6.3.2 Sick flowers

The second scenario describes what happens when certain flowers become sick. For instance, if some flowers on $cart3$ get sick, $cart3$ uses the DCO to notify the $Room3SV$ through the following RDF-triple:

$$\langle SeSaMe : cart3, Greenhouse : Health, \text{“}Sick\text{''}\rangle,$$

which in turn notifies the Greenhouse SSM. In the meanwhile, $Room3SV$ asks $cart3$, which contains the sick flowers, to activate the sick flower role and to reconnect to the system through the SSM, which in turn decides where it should put the sick flowers. It can allocate a new room for the sick plants, or transform the room in which the cart currently is, into the room for sick flowers. In the latter case, the room's supervisor has to change its role, and all the other flowers and plants need to leave the room. In this example, the SSM decides to make the *Mild Temperature Room* the room for all sick plants, and changes the role of $Room3SV$ to that of the sick flower supervisor. This will leave all the nodes following the previous supervisor role in an orphaned state. According to proposed framework, all the nodes whose supervisor node fails or leaves the system, automatically try to reconnect to an SSM. The result is that the SSM will allocate them to a new room, according to the newly defined system topology.

The framework exploits *Semantic Ontologies* to achieve interoperability both between the components of the same layer and between components of different layers. We used Smart-M3 [61], a platform operating on principles of space-based information exchange, to embed semantic models into our middleware. Smart-M3 is based on two kinds of components: Semantic Information Brokers (SIB) and Knowledge Processors (KP). The former are used to store a system's semantic information; the latter insert or retrieve data from a SIB. In our framework, we use SIBs to develop a knowledge base that supports RDF-based data storage, while components, both from the Components Layer and from the Management Layer, act as KPs.

Even if this is only a simple preliminary example, the integration of ontologies into the core middleware infrastructure demonstrated significant benefits in the integration of truly heterogeneous elements. In general, one may notice that a proper partitioning of the ontology into coherent pieces helps the designer modularize the whole system and organize the communication among the different elements. It is also important to notice the use of a domain specific ontology to take into account the peculiarities of the particular application domain, and thus a clear and sound design of this ontology is crucial for the correct and efficient operation of the system.

Our experiments have only been limited to a simulated environment, but even in this context, where communication delays are negligible, we did not experience significant delays introduced by the semantic layer. This allows us to be pretty confident that the framework will also be able to service more complex and complete systems efficiently.

6.4 Case Study 3: Energy Efficient Buildings

The development of energy-efficient buildings is a multi-faceted problem. Besides the proper design and construction of the building, one must also take into account its operation, and manage the components in charge of energy efficiency. This implies that one needs to do the following:

1. define the control components,
2. design the control system (and strategies), and
3. validate it —maybe through simulation— properly

Currently, different software systems such as EnergyPlus (for simulating the energy requirements of a building) and Matlab/Simulink (for conceiving the control system) are used to design and validate energy control systems. All these tools, which correspond to different phases, work in isolation.

In this case-study, the proposed framework is evaluated in terms of bridging the gap between existing design, testing, and simulation solutions, and providing an integration mechanism to conceive control systems that blend diverse aspects, co-simulate them through the coordinated use of different simulators, and deploy them in real contexts.

The goal is to provide integration solution for control systems and simulators and also ease the design and experimentation of different control strategies that can be changed or modified at runtime.

6.4.1 Building Model

To exemplify the "general" problem, this thesis uses the DOE Commercial Medium Office Building (Benchmark V 1.0_3.0[5]) as concrete and well-known building model (see Figure 6.9). The building has the following characteristics:

Building Geometry: The building has a rectangular shape with aspect ratio of 1.5, and it spans 4,982 square meters (m^2). The building has three floors, where each floor consists of one core, which covers 38% of the area, and four perimeter zones (62%), for a total of 15 zones in the building. Each perimeter zone is 4.57 m deep, with

[5] http://energy.gov/eere/downloads/archive-reference-buildings-building-type-medium-office.

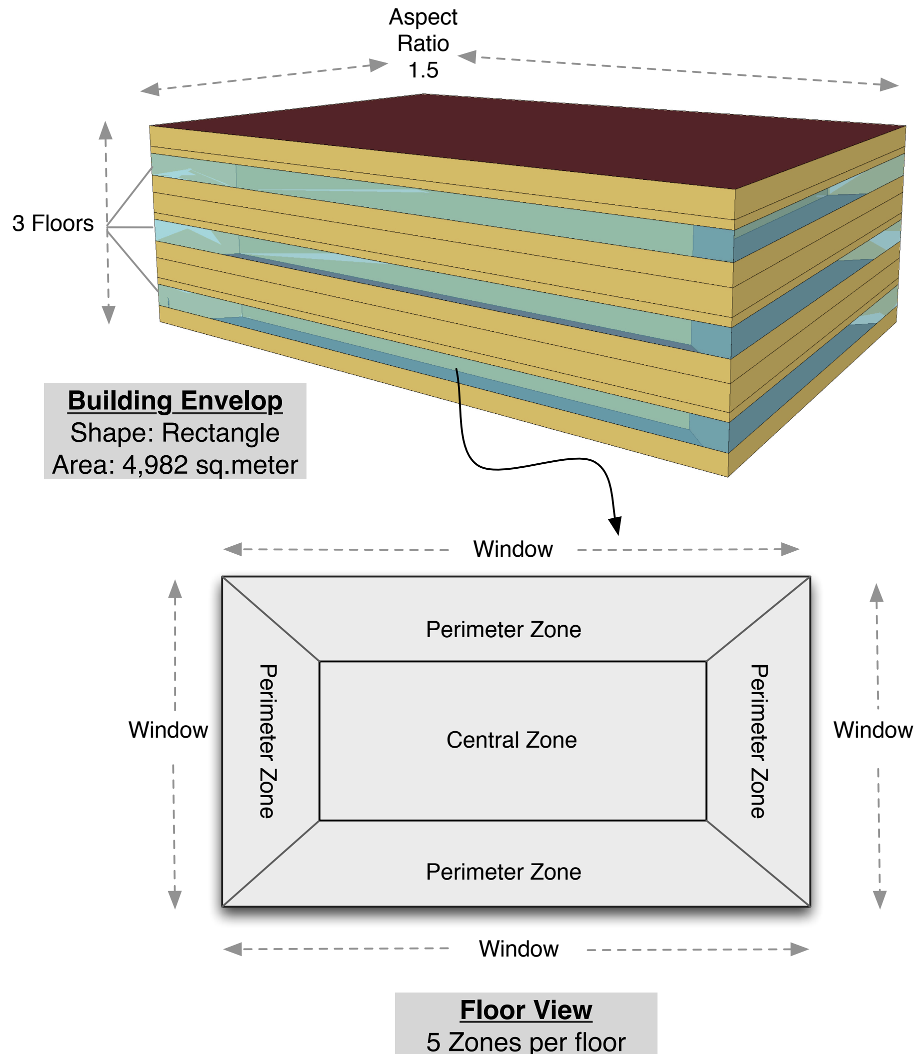


Figure 6.9: *Benchmark office building.*

windows facing each direction, which receive daylight; the core zones do not receive any daylight.

Building Envelope: The thermal properties of the building envelope vary with respect to climate according to ASHRAE Standard 90.1-2004 [11]. The exterior walls are steel-framed, roof is flat, whereas floor is based on concrete slab. There is equal distribution of windows, and window-to-wall ratio is set to 24.3%. Windows use single pane glazing and horizontal blinds with 0.025 m width. Infiltration takes place in perimeter zones and the HVAC system of the building is based on gas furnace with economizer as per [11].

Peripherals: Lighting power density is set to 10.76 W/m^2 whereas the electrical plug loads are set to 8.07 W/m^2. The office building accounts for 195 people in total with the occupancy density set to 3.91 people/100 m^2. Occupancy is primarily controlled by a dynamic statistical occupancy model derived from actual data from a typical office building.

We have extended the building model by adding dimmable lights in each zone that are able to react according to the specified luminance level in a zone. EnergyPlus is then used to simulate the HVAC and lighting conditions within the building to calculate the corresponding lighting, cooling and heating energy requirements. We are using the Baltimore climate and weather conditions from TMY3 (Typical Meteorological Year version 3)[6] data set for our experimentation as some other existing solutions use the same data.

6.4.2 Integrated Control

The integrated control within the building is achieved by organizing the building control in multiple control groups through the abstractions provided by the framework. It means that a control group comprises components with two distinct types of roles: (i) *supervisor*, that is, a controller, and (ii) *follower*, that is, sensors and actuators —or their corresponding interfaces if simulated by external systems (See Figure 6.10).

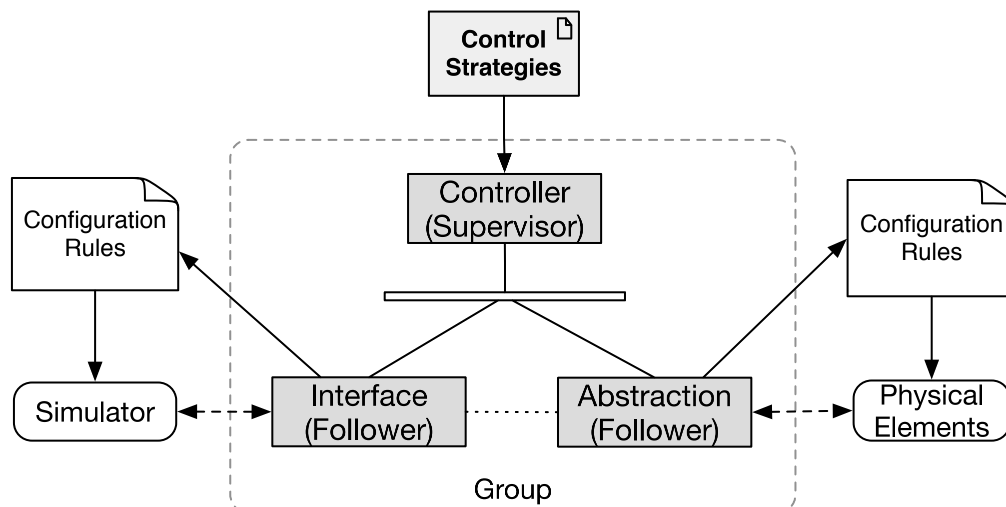


Figure 6.10: *Example control group.*

We have implemented the control groups (controllers, sensors and actuators) as described in Figure 6.11 initially, and then conducted experiments with different grouping criteria to change the decision making granularity (e,g., floor/zone/building level, isolated lighting and HVAC control, or integrated control at various levels). Further external simulators (EnergyPlus in this case) is interfaced through our middleware to the relevant groups. EnergyPlus is also used for simulating the occupancy patterns that are devised from statistical data.

Let us now consider the zone scheme described in Figure 6.9 to understand the design of an integrated control; each floor has five zones, where four of them have windows with controllable blinds, but there is a common shared HVAC control for each floor. Figure 6.11 shows the control configuration where one can have 5 groups (one for each zone), each having a zone manager controlling all the components in the zone according to defined control strategies. These 5 zones act as followers for the lighting

[6] http://doe2.com/Download/Weather/TMY3/

and thermal management groups which in turn are managed by the floor manager. Each zone manager monitors the luminance (measured in lux) and temperature in the zone and informs the thermal and lighting control groups, which then will decide whether the window shades or lights could be adjusted to meet the requirements. Besides taking the requirements of a particular zone under consideration, a thermal manager must also balance the thermal needs of the whole zone while controlling the HVAC system. Based on the decision of the thermal control, the zone manager adjusts the lux by turning on and off some lights adaptively. Different zones collaborate to achieve the optimized control state.

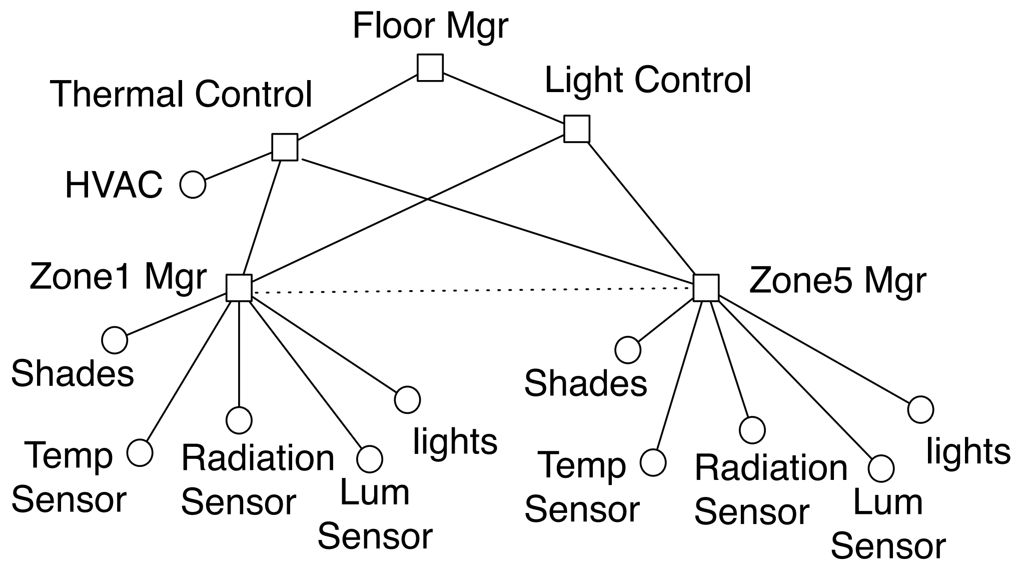


Figure 6.11: *Integrated Control Groups.*

Scheduling the communication among the components is an important task as it defines the order in which these components receive or send messages.

As each component has its own execution cycle and duration, they must be synchronized, that is, the execution of the different components must be organized, to let the system evolve through consistent states. Components send messages to/receive messages from other components (via roles) on clock tick(s) to synchronize the information exchange. Data exchange with external simulators is instantaneous; sender and receiver are paired, and they remain blocked during the interaction.

For example, let us consider that in our example building, the movement of window blinds is controlled on the basis of the light and temperature in the zone. Weather conditions and constraints on energy consumption are also considered during the decision making process. It can easily be seen that weather information (updated every 15 minutes), internal light and temperature values (available instantly) and window movements (that may take thirty seconds to change status from open to close or vice versa) have different frequencies and execution times: their synchronization on some global time is mandatory. The group supervisor (controller) in this case takes the responsibility of collecting the information from followers on different time intervals and of

synchronizing it on a common time (step) to make valid decisions (See Figure 6.12).

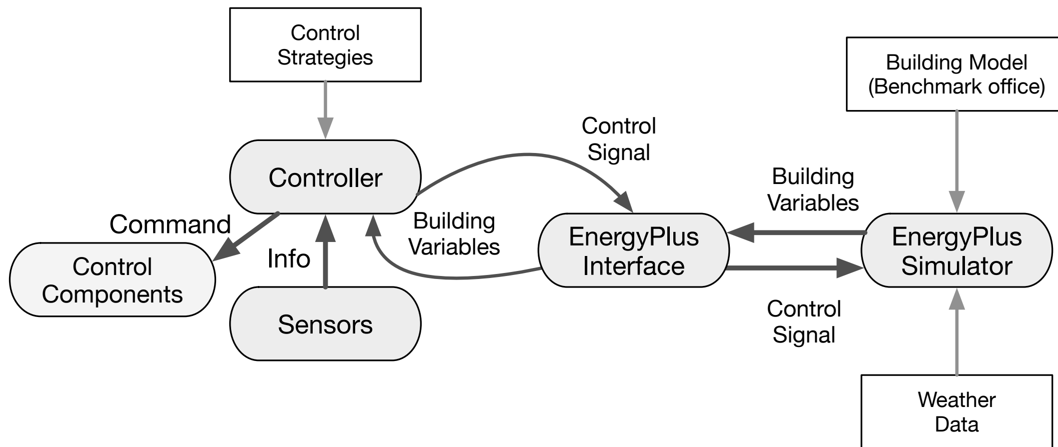


Figure 6.12: *Concurrent execution/simulation.*

Besides synchronizing the information flow, we also need to take into account the type, structure, and multiplicity of the data exchanged between parties. For example, a sensor component sends read values to the zone manager, while a window blind needs the action command (the slat angle) from the manager. Required and produced data are then stated in an external configuration file; involved roles (both supervisors and followers) must agree on the same data exchange model.

6.4.3 Experimental Details

We have experimented with different control strategies for windows shading, HVAC, and artificial lights. The following two window shading control strategies are used for experiments:

1. Shades with fixed slat angle
2. Dynamic control of the shades

In the first strategy the slat angle will be fixed (e,g., at 0°, 45°, and 90°) whereas in the second strategy the shading is adjusted (5° shifting step) according to the lighting and thermal needs within the building.

For HVAC, we are using the following thermal strategies:

1. On/off control, which is managed on the basis of current zone temperatures and heating/cooling schedules (set-points)
2. Optimum start with constant temperature gradient, which anticipates the HVAC needs based on daily schedules and performs pre-heating and cooling with fixed temperature changes per hour

For artificial lighting, we are using two control strategies:

1. Discrete on/off control, where lights can only be turned on and off based on the current luminance level and the set-points, and

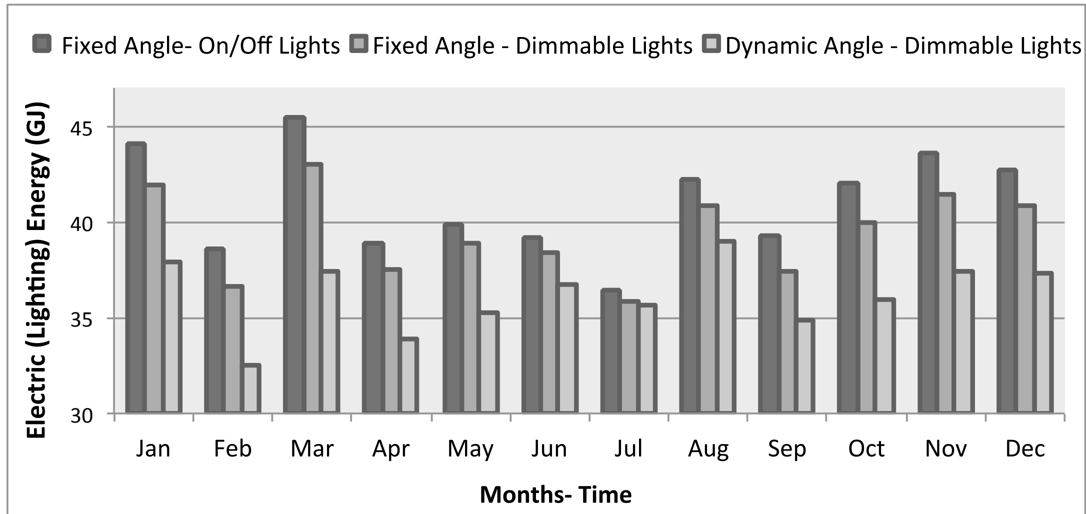


Figure 6.13: *Energy consumption for lighting.*

2. Continuous (dimming) control, where lights can be dimmed step-wise from completely bright to turned off.

We have tested the following three integrated control strategies in our preliminary experiments: the first strategy ($S1$) uses fixed slat angle (45°) and on/off lighting control, the second strategy ($S2$) also uses fixed slat angle but incorporates dimmable lighting control, whereas the third strategy ($S3$) uses dynamic slat angle and dimmable lighting control. Moreover, we have used on/off control for HVAC system for all three strategies where heating and cooling set-points, respectively, are 21° and 24° C for working hours and 15.6° and 30° C otherwise. The luminance set-point for each zone is set to 500 lux.

The simulation is run for a whole year and the time step is set to 5 minutes in EnergyPlus, that is, the control system interacts with the building every 5 simulated minutes. Figure 6.13 shows that $S3$ always has a better energy efficiency for lighting than $S2$, which in turn outperforms $S1$. $S3$ saves 12% lighting energy (annually) with respect to $S1$, whereas $S2$ saves 4% in comparison to $S1$.

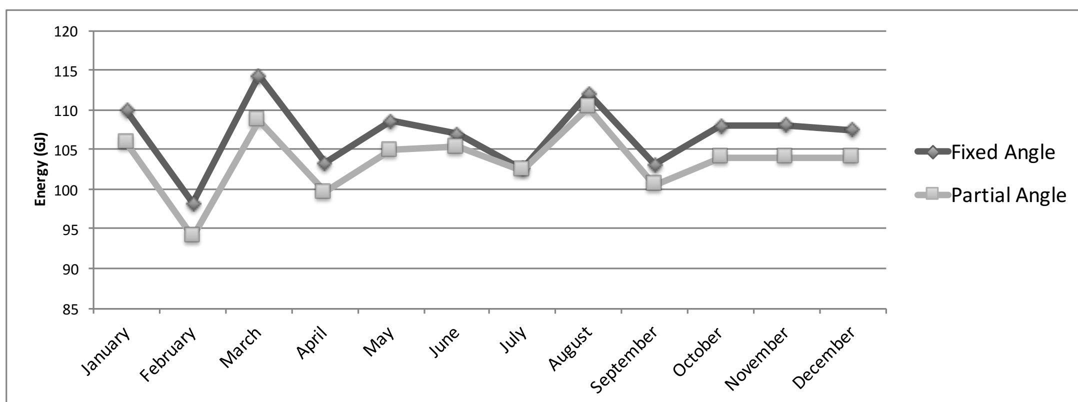


Figure 6.14: *Energy consumption for building.*

The partial and fixed angle strategy also had a significant impact on the total energy usage of the building. Figure 6.14 shows that partial angle strategy almost always outperforms the fixed angle strategy throughout the year. The only period where dynamic control of the window shutters (slat angle) is not significant is the month from July to September, where shutters are usually closed during the daytime to save the cooling energy.

The primary purpose of this case study was to evaluate the ability of the integration mechanism of the framework to conceive control systems that blend diverse building dynamics and co-simulate them through the coordinated use of different simulators. The framework was successfully able to coordinate building simulation and implementation of control decisions that take place in different systems in order to enable energy efficiency in office buildings.

As integrating heterogeneous systems alone is not enough. Provision of a valid synchronization among the various execution models running in different subsystems is also very important for a useful solution. These experiments also demonstrate the framework's **validity** in terms of synchronizing multiple components and simulators together according to their execution and (data) communication cycles. The results for different strategies were matched to the expected results (through running static strategies in EnergyPlus without external control system) and they found to be in line with them.

6.5 Case Study 4: Public Park

This section describes the experiments we conducted to evaluate the scalability, reliability, and efficiency of the proposed architecture by designing and simulating our public park. It focuses on evaluating the following aspects:

- How well the framework can handle dynamism and mobility of the users/components while keeping the reliability of the services?
- How can the proposed framework balance energy consumption of each device?
- How can the proposed framework uniformly distribute the communication load (group size)?

6.5.1 Experimental Setup

The park is setup and simulated[7] (see Figure 6.15) through the Netlogo simulator [95]. NetLogo is a programming language and modeling environment for simulating natural and social phenomena. It is widely used by researchers for modeling complex systems that evolve over time and for analyzing micro-level behaviors of autonomous entities along with the corresponding macro-level patterns that emerge from the mutual collaborations. The environment (or simulated world) in Netlogo is spatially organized in *patches* and the individuals that live in the simulated world are termed *breeds*.

Our public park is then structured around a grid of 30×30 patches, where each patch can be a pathway, a screen, or a point of interest. Components (breeds) can be of three types: people, screens and points of interests (POI). We started the simulation with a

[7] The simulation video is available online at: `http://home.deib.polimi.it/shahzada/ParkSimulation/NetLogo-ParkSim.mov`

population size of 120 and decided that 5 new people had to join, and 1 person had to leave, the system every 25 time units. People move towards the directives received from the screen 50% of the time and take random movements otherwise.

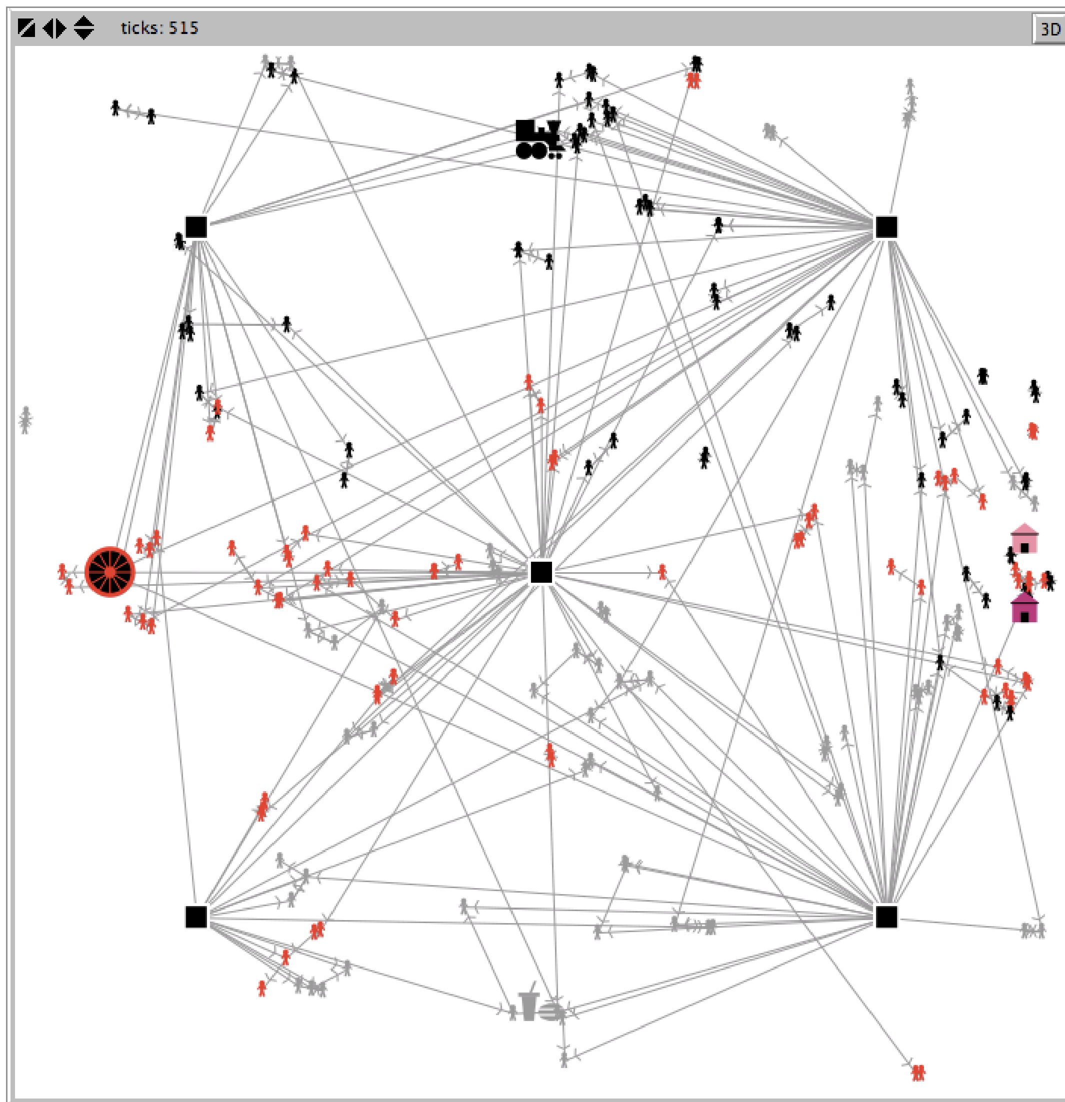


Figure 6.15: *Park simulation in NetLogo.*

We have run two different simulations for the example scenario. The first one considers the following role and collaboration types as shown in Figure 6.16:

1. Screen to screen collaboration, which enables them to exchange localized data.
2. Screen to user collaboration, which enables users to connect to the screen and get personalized information.
3. User to user interest-based collaboration, which allows users with similar interests to connect together for possible social interactions and exchange of helpful information.

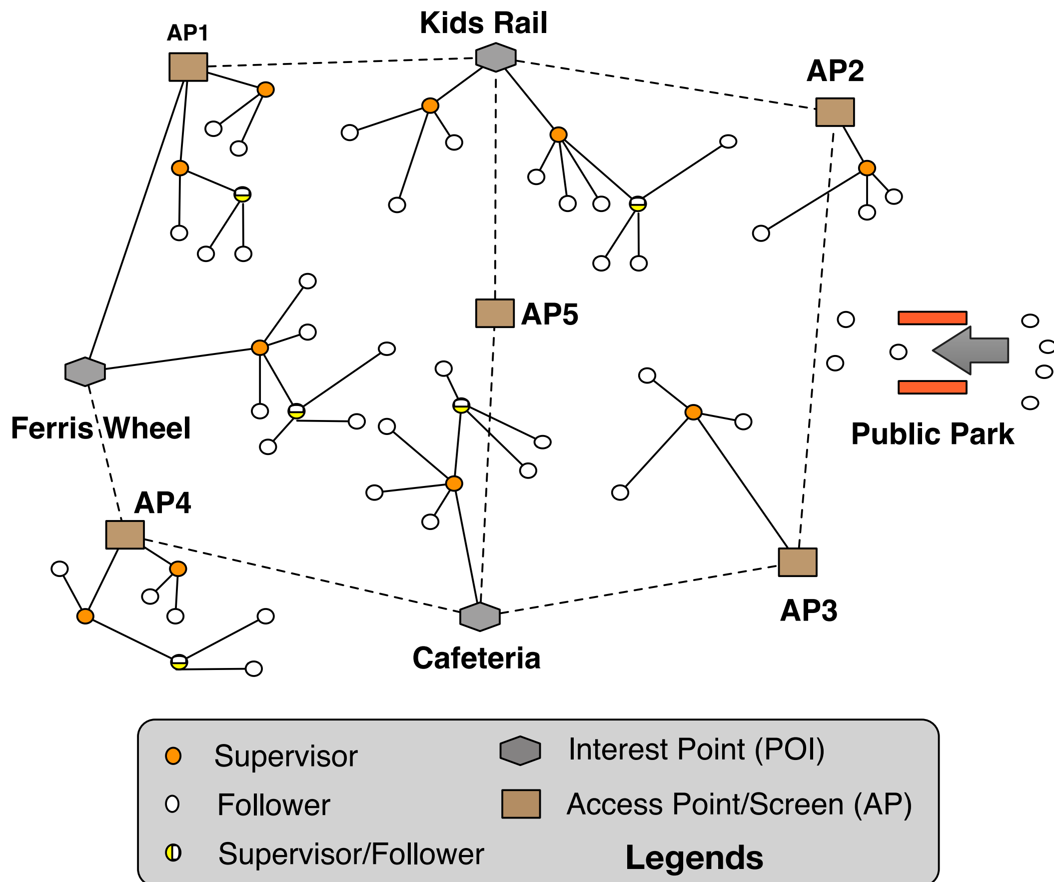


Figure 6.16: *Park Topology*

We form user collaborations based on the three interest types. The interests of a user are defined through the corresponding roles, and each user tries to connect to another user with the same interests that is in turn connected to a screen. The decision of being part of a collaboration (group) is done through the fireflies-based algorithm. The second simulation is carried out without any role-orientation (similar to a centralized approach): the knowledge exchange among components is not filtered by user interests, and each component receives the same amount of information from the space. The different components simply try to connect to the first component they find in their proximity and do not use fireflies-based adaptation.

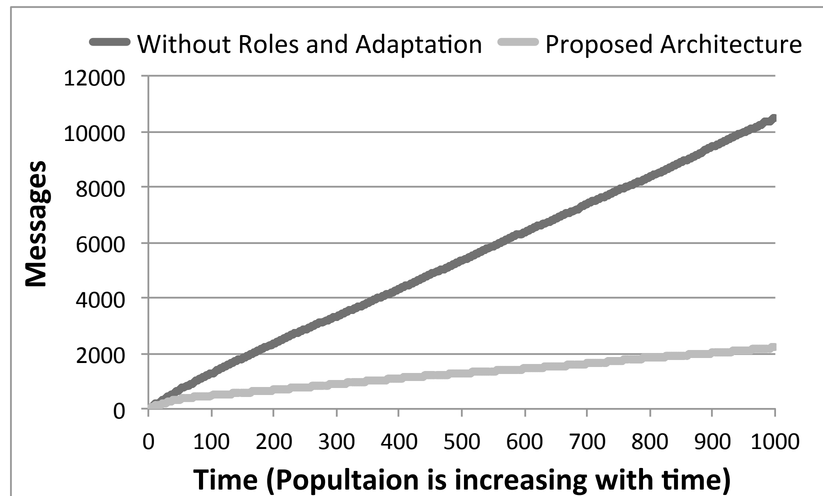


Figure 6.17: *Average number of messages per screen.*

The goal is to evaluate the impact of our architecture on the number of exchanged messages within the space, while keeping the reliability of services. We recorded that in the case of supervisor failure, all corresponding follower components of the group are re-assigned to a new group within 3 time units. We have also measured the energy and communication load (dependent on the group size) for each (user) supervisor component. The energy consumption of each device is measured as a (constant) penalty for each message sent or received by the device. The proposed solution was able to balance both the energy consumption and the size of different user groups (communication load).

Figures 6.17 and 6.18 show the results of the experiments. The simulations were run for 1000 time units (simulation cycles), and the population was initialized with 120 components and dynamically increased to 280 elements. We have measured the number of messages for the two cases; messages are the directives that are sent from the screens or from the users that act as supervisors. In case of simulation without roles and adaptation, a component receives messages for all the points of interest and then it selects those to take into account. The absence of special-purpose collaborations costs more messages since each component receives information about each POI, even if it is not interested in it. Besides their number, these messages are also a good indicator of the amount of knowledge that is shared among components. For example, Figure 6.21 shows that on average the proposed architectural solution allows for a reduction of 78% of the number of messages that each screen receives and of 65% for each user.

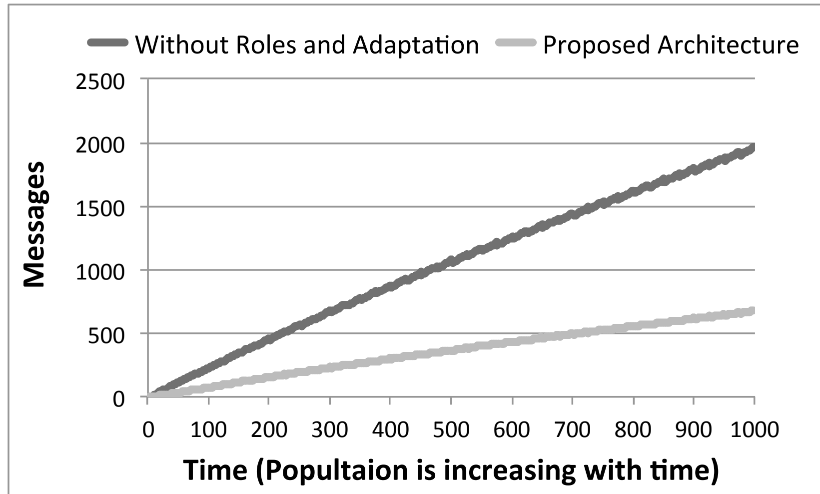


Figure 6.18: *Average number of messages per user.*

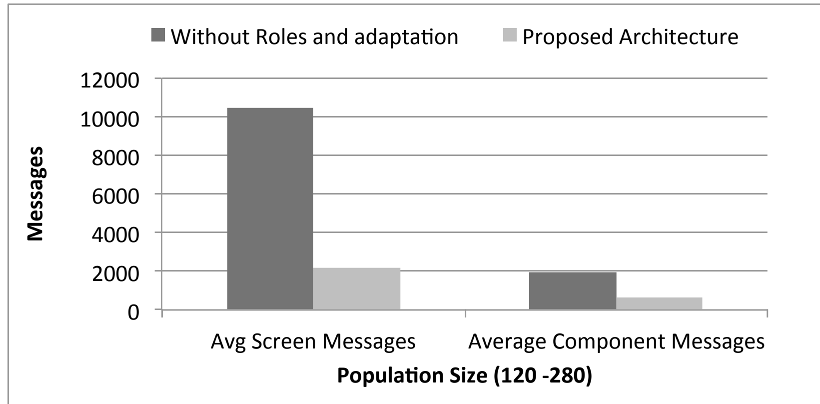


Figure 6.19: *Results*

It reduces the load of "access point" (screens in our case), which is one of the critical factors with centralized ways of communication, by distributing and delegating supervision to some of the users. Moreover, it also limits the amount of knowledge that is shared among these distributed and autonomous components that form collaborations at run-time. The unnecessary knowledge exchange decreases the performance of the system as it leads to additional execution (computing) cycles and network saturation.

We have also measured the energy and communication load (dependent on the group size) for each (user) supervisor component. The energy consumption of each device is measured as a (constant) penalty for each message sent or received by the device. The proposed solution was able to balance both the energy consumption and the size of different user groups (communication load). Throughout the simulation, we observed that the variability of energy consumption for each device remains within the standard deviation of 1% whereas the difference in group size has the deviation of less then 1 component across all groups.

The results highlight that the proposed architecture maintains reliable service provision in a scenario with highly dynamic and mobile components. Moreover, it distributes the communication load uniformly across components by delegating supervision to

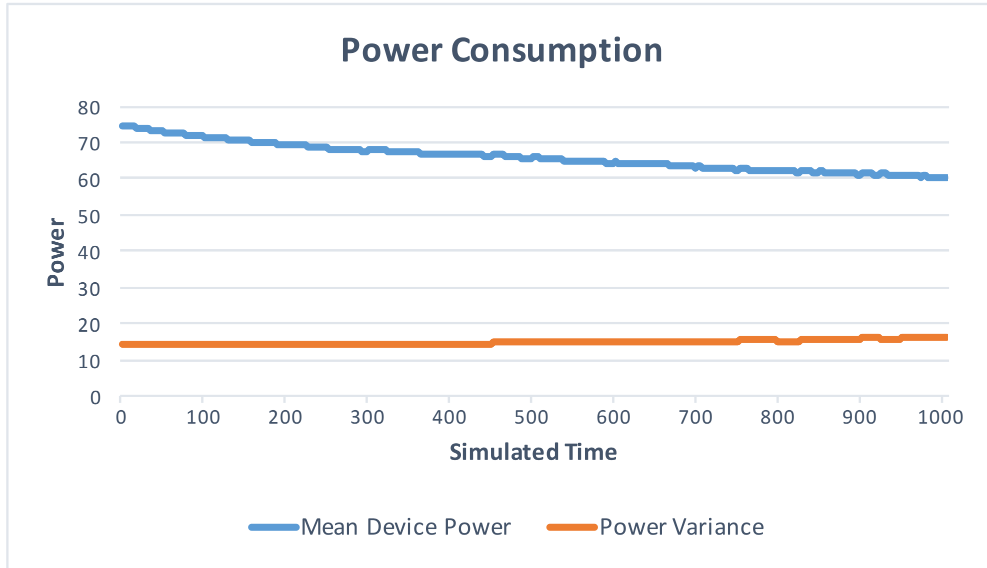


Figure 6.20: *Power consumption.*

some users and hence reduces the overall network congestion.

6.6 Discussion

The above four case studies have demonstrated the effectiveness of the proposed framework in various dimensions. We have conducted experiments to evaluate different quantitative and qualitative characteristics of the network as described in Tables 6.1 and 6.2. Following is the summary of the results corresponding to the various evaluation metrics:

- **Efficiency.** JOL and public park case studies were used to measure the efficiency of the framework in terms of bootstrapping time, delay (impact on performance), and congestion. Our experiments show that a component joins the system within 1.4 seconds on average and sends a message within 5 milliseconds in the worst case (fully deployed system using various stand alone subsystems). Moreover, the framework was able to balance the load of the various groups in a uniform manner where the difference in group size had the deviation of less then 1 component across all groups in a system of 280 components.

- **Reliability and Dynamism.** Public park and greenhouse case studies were used to demonstrate that the framework is able to re-configure the system topology in case of components frequently joining/leaving the system. We have measure the time required by the system to recover (re-configure) from a component failure/leave. We observed that all corresponding follower components of the group with a failing supervisor component were re-assigned to a new group within 3 time units.

- **Scalability.** We conducted experiments through both a relatively static space (JOL), and a very dynamic space (public park) to observe how the proposed solution scale up in terms of its efficiency. In JOL, we experimented with (up to) 20

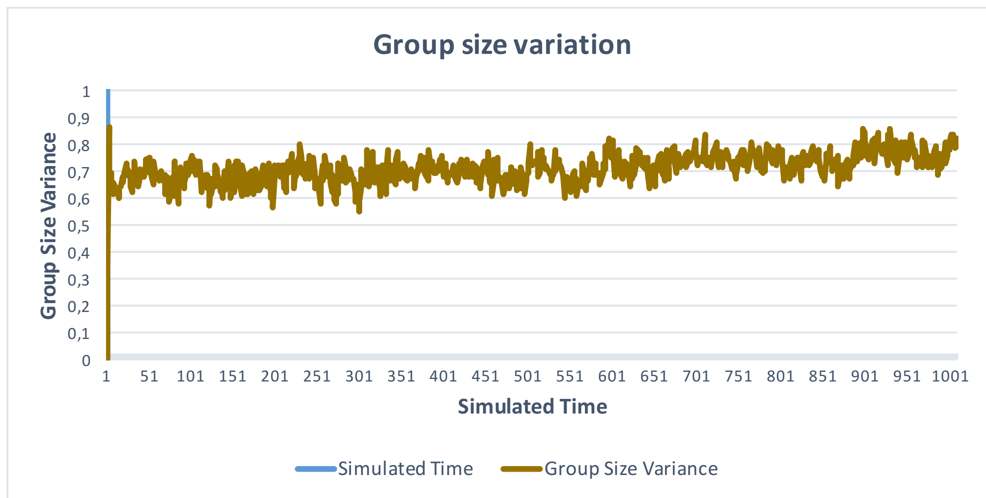


Figure 6.21: *Group size variation.*

components (on a machine) to see how much time is required for each component to bootstrap. It was observed that after deploying 7 components on a machine, the average bootstrap time per component shows a linear behaviour. Similarly, for public park scenario it was observed that each component bootstraps within 3 time units where 5 new people had to join, and 1 person had to leave, the system every 25 time units.

- **Interoperability.** All the case studies demonstrate interoperation of the components with heterogeneous functions, execution cycles and communication technologies they use. For example, Table 6.4 describes that various components used in JOL differ from each other in terms of the type and communication protocols. The framework was able to ensure that these components can interoperate and communicate with each other to execute functional logic.

- **Integration.** Similarly, the experiments have also showcased the ability of the framework to integrate different components together in logical self-organizing groups at the runtime. The self-organization is triggered by either the static location-based criteria (enforced by SSMs as in case of JOL) or the heuristic functions that are used by the individual components (in an ad hoc manner as in the public park) to decide which group or component to connect with. Moreover, JOL and energy efficient building scenarios also demonstrate that framework was able to concurrently execute (interacting) systems with different execution models. For example, in energy efficient building, the control system was successful in synchronizing control decisions with sensor inputs and the energy requirements gathered from an external simulator (EnergyPlus), where all of these elements had different execution cycles.

- **Extensibility and Flexibility.** JOL and public park case studies demonstrated that it is possible to add new functional components (roles) without changing/halting the system. Moreover, it was trivial to replace one or set of component(s) without changing the high-level functional and organizational logic. For example, set of virtual lights (simulated through Freedomotic) in JOL were replaced with real

ZigBee enabled lights (using openHAB as a bridge) just by changing the interface component. Similarly, the framework was flexible enough to change the user proximity data simulated by Siafu with the events generated by smartphone sensing BLE beacons.

- **Incrementality.** JOL case study showcases that the framework supports concurrent execution of both physical and virtual components together for continuous evaluation. We developed the JOL smart office space in an incremental fashion, which is, starting with completely simulated solution through external simulators, testing and validation the functional and managerial logic and then moving on towards the fully deployed solution in various transitional phases.

The above analysis suggests that the framework is very effective for the incremental development of both static and dynamic smart spaces.

CHAPTER 7

Conclusions and Future Directions

This chapter summarizes the contents of this thesis and provides an outlook on the directions for the future work. The thesis started with the challenge/objective to provide a development framework that enables the developers to design, implement and validate various dynamic smart spaces by offering programming abstractions that are suitable for the whole development life-cycle. The challenge was formalized into three high-level research questions, in result of which, different challenges related to smart space development were identified and revisiting of the development approach was proposed.

The work presented in this thesis introduces a development framework that enables developers to design, implement and validate diversified smart spaces. The thesis employs incremental development process to create smart spaces in order to fill in the gap caused by the inability of existing smart spaces to evolve over the time and to deal with changing requirements. The framework offers proper blending of different aspects:

- Provision of appropriate design abstractions to support co-execution of physical and virtual components through proper middleware infrastructures
- Ad-hoc integration and coordination of heterogeneous components
- Early evaluation of "incomplete" (partially deployed) systems
- Assessment of alternative solutions and deployments of the system

The framework allows the developer to move seamlessly from a fully virtual and simulated solution to a completely deployed system while enabling continuous validation at any given stage of development. It offers the same abstractions throughout the whole development process to provide means for both the seamless integration of various components and the utilization of existing systems. Interfaces are provided both to

surrogate system components through external simulators and to ease the deployment of physical elements.

The thesis synthesizes the control mechanisms from architectural components for self-adaptation and inherent self-organizing capabilities of fireflies (bio-inspired) in order to design both effective and continuously evolving smart spaces. The synthetic approach provides the required autonomy to the individual components/groups and at the same time ensures the desired level of distributed control at various granularity levels. Four different prototypical implementations of diverse smart space scenarios are conducted to assess the feasibility of proposed concepts, fulfilment of system requirements, and various quantitate and qualitative measures (metrics).

7.1 Answers to Research Questions

This section reviews the research questions and discusses the contribution of this thesis towards them.

RQ 1. Do we need to revisit/improve our development processes for the creation of smart spaces in order to realize effective smart spaces and, in case, what would that revision be?

Two major revisions of the current approaches are proposed and advocated in the thesis. The first one is the **incremental development** (Figure 4.1) of the smart spaces where different phases co-exist with each other and where it is possible to work with 'incomplete' systems at any given time. The second proposal is related to self-adaptation mechanisms. We advocate and demonstrate that there is a need to have a self-adaptive approach that takes the best of the existing software architecture control loops and bio-inspired emerging algorithms to develop a system that has both the control over system processes and the certain level of autonomous emergence properties.

RQ 1.1. What are the requirements for the development of diversified smart spaces and where do existing systems lack?

We conducted a comprehensive survey of the state of the art to understand the requirements for the development of various types of smart spaces. We have established that in order to conceive all kinds of spaces, a framework needs to have the following properties: ability to abstract over heterogeneity, support to integrate different components and protocols at runtime, facilitation of incremental development of the smart spaces, cater scalability and dynamism of the components and the flexibility to use or replace alternative solutions or subsystems. Most of the existing systems lack in terms of providing support for the continuous validation and testing of alternative solutions in order to move seamlessly from design to actual deployment. Moreover, existing solutions also lack the balance between the control and the autonomy of smart space components.

RQ 1.2. Which and how can we employ/integrate various principles from the existing design paradigms to address those deficiencies, if possible?

Many researchers have proposed the solutions for the design of smart spaces. The autonomic computing community has been focusing on super-imposed adaptation mechanisms by adding further, dedicated components to the (software) architecture of the system. In contrast, bio-inspired solutions provide inherent support to self-organization but they fail to guarantee the desired level of reliability and control. This thesis presents a hybrid approach that blends the two views and proposes an architecture-centric solution that merges component-based control and bio-inspired (fireflies-based) mechanisms. Suitable abstractions help conceive self- organizing, ad-hoc collaborations among the components of a space.

RQ 1.3. What is the suitable development life-cycle for the development of diversified smart spaces?

It is very hard to predict at the design time about what kind of sensors, actuators, controllers and data are required for the space-to-be. Therefore, it is very important for a development framework to be able to test different alternative solutions before committing to the physical deployment of the system. This implies that an incremental development approach is required for the smart spaces that enables continuous validation of the system at different stages of the development. Moreover, many concrete scenarios may also be difficult to reproduce just for testing the system. For example, it is not trivial to create situations to test daily user activities in the example smart office or crowd navigation in large events such as the public park. The proposed framework eases the task and allows for early and continuous validation by combining simulated entities and physical components together. For instance given a space, one can use Siafu, or any other simulator, to represent real users and mimic their behavior to evaluate the developed solution. We have demonstrated that the hybrid execution of heterogeneous components is very effective tool for the transition from a simulated environment towards its concrete deployment. It enables entities to be added incrementally to the system over the time.

RQ 2. What kind of software (framework) abstractions and interaction mechanisms are required for the design and implementation of smart spaces to overcome the fragmented (in terms of technologies and functionality) devices and continuously changing user requirements?

The thesis has presented a self-adaptive framework for highly dynamic and large smart spaces. It presents a group-based abstraction for coordinating distributed components. Each group clusters components with similar or relevant characteristics. These groups can be exploited in various ways to cluster devices, and/or users, so that they can act as single coordinated entities. For each group, framework chooses a supervisor component (role) that is in charge of coordinating the group's elements, and of communicating with other existing groups (i.e., with other supervisors). Since each component can play different roles in the system, a node can participate in different groups at the same time. The framework also offers basic self-organizing primitives and fireflies-based heuristics that can be used to specify how groups can be re-organized at runtime.

It provides means for components to connect to the system with no prior knowledge of its topology, and it provides means to ensure reliability and avoid message congestion in case of high component churn rates.

RQ 2.1. How can we build a software framework that fulfils the requirements identified in answering to RQ.1?

The component, role, and group abstractions are complemented with semantic model and self-organization mechanisms (that integrate component control with firefly-based adaptation) to ensure communication among heterogeneous components, ad-hoc integration of dynamic components, and self-configuration/healing in case some of the components fail or leave the system unexpectedly. Moreover, reconfiguration of the grouping, as defined in the proposed firefly-based adaptation, ensures the uniform distribution of workload for each component in order to ensure efficiency of the system and avoid congestion.

RQ 2.2. How can we use (some of the) existing smart objects, architectures, middleware infrastructures, and simulators to implement smart spaces dynamically, effectively, and opportunistically?

The proposed framework provisions the use of an existing or external system in the same way as any other component (implemented with native APIs). It provides the component interfaces to maintain the state and identity of a (set of) components and at the same time being able to plug them to virtual or physical objects. All the component roles, behaviors, and collaborations remain independent of the type (physical or virtual) of the object associated to it. The framework, hence, offers two options to simulate the different entities: i) the developer can define special-purpose roles that implement the foreseen behavior, as long as the real components are not available; ii) any external simulator can be plugged-in to mimic the behavior of a single component or of an entire subsystem. In this way, an application programmer can test and validate various configurations of the system using external simulators without incurring any additional cost.

RQ 3. How can we evaluate the effectiveness and capability of such a framework to design, develop and assess highly dynamic diversified smart spaces?

In our analysis of the state of the art, we first identified and listed out all the significant and necessary dimensions/requirements, which should be addressed by the development framework. An important aspect was the ability of this framework to be applicable to all kinds of smart spaces, that is, provide abstractions and tools suitable for whole development life-cycle of both dynamic and static spaces. The evaluation plan describes all the metrics (corresponding to the identified dimensions) that needs to be measured and observed to understand the effectiveness of the proposed framework. The next step was to design (or select) appropriate case-studies (with varying needs and characteristics – from static and closed spaces to more open and dynamic space) to better demonstrate various aspects and features of the framework. Four different di-

verse smart spaces were used to assess the feasibility of proposed concepts, fulfilment of system requirements, and various quantitate and qualitative measures (metrics).

The results suggest that the framework is able to fulfil the identified requirements that are required to develop different types of spaces. The framework incurs small delay which is not significant for the systems of this scale and hence it provides a practical solution for the issues identified in the survey of the state of the art. Moreover, the framework also complements the existing growth of the smart objects and plethora of software solutions, and provides a framework to integrate/utilize the available resources as a step forward towards the efficient development of smart spaces.

7.2 Future Directions

The work presented in the thesis opens many interesting research directions for the future work. The focus of this thesis has been the integration of various physical and virtual subsystems together to develop and validate individual smart spaces. The proposed framework can be further extended to integrate/aggregate various smart spaces together and move towards the concept of *hyper-spaces* [70]. Arguably, incremental aggregation of smart spaces into hyper-spaces (and then aggregation of these hyper-spaces) is one potential way to realize **smart cities**. There are many research challenges involved such as:

- Conflict resolution in component control
- Integration of information sources through a common data exchange (semantic) model
- Self-organizing mechanisms with the ability to adapt according to changes occurring in external (other integrated spaces) systems.
- Security and privacy of smart spaces

Another direction for future work is to experiment with metaphors from other natural systems that include chemical, physical, social and biological self-organizing systems. Although the proposed framework has implemented the fireflies-based algorithms for self-adaptation, the provided abstractions are general enough to be extended by implementation of other biological metaphors, for example. This would provide the developer with more freedom to tailor adaptation mechanism to the one that best suits their system requirements.

At the micro level, some of the possible extensions of the work are:

- Provision of a descriptive language to define/model the system through provided abstractions
- Various ready to use solutions for decision making related to group formation and supervisor control logic
- Validation of the system design; currently it is up to the developer to ensure that s/he provides the appropriate roles and group definitions to make the system work. By a proper validation mechanism, a developer can verify if all the required components, roles, and groups are in place.

Bibliography

[1] Apple Home Kit: A High-Level Device Connectivity Framework, [online]. available: https://developer.apple.com/homekit/.

[2] Freedomotic: Open IoT Framework, [online]. available: http://www.freedomotic.com/.

[3] Google Nest, [online]. available: https://nest.com/.

[4] Haier (U+ Smart Living), [online]. available: http://mg.haier.com/index.php?m=content&c=index&a=show&catid=42&id=47.

[5] OpenHAB, "openHAB -empowering the smart home,", [online]. available: https://github.com/openhab/openhab.

[6] Proximal Connectivity Platform, [online]. available: https://allseenalliance.org/developer-resources/alljoyn-open-source-project.

[7] Samsung Smart Home, [online]. available: http://developer.samsung.com/smart-home/.

[8] Vykon: Integrating Building Automation, [online]. available: http://www.tridium.com/galleries/brochures/VYKON-FINAL.pdf/.

[9] D.B. Crawley, L.K. Lawrie, F.C. Winkelmann, W.F. Buhl, Y.J. Huang, C.O. Pedersen, R.K. Strand, R.J. Liesen, D.E. Fisher, M.J. Witte, J. Glazer. EnergyPlus: Creating a New-Generation Building Energy Simulation Program. *Energy and Buildings*, 33(4):319 – 331, 2001.

[10] G. Agha. Computing in Pervasive Cyberspace. *Communications of the ACM*, 51(1):68–70, 2008.

[11] ASHRAE Standard ASHRAE. Standard 90.1-2004, Energy Standard for Buildings Except Low Rise Residential Buildings. *American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc*, 2004.

[12] Bela Ban et al. Jgroups, A Toolkit for Reliable Multicast Communication. *URL: http://www. jgroups. org*, 2002.

[13] J. Bardram, T. Hansen, M. Mogensen, and M. Soegaard. Experiences from Real-World Deployment of Context-Aware Technologies in a Hospital Environment. In *Proceedings of 8th International Conference on Ubiquitous Computing*, pages 369–386, 2006.

[14] L. Baresi and S. Guinea. A-3: An Architectural Style for Coordinating Distributed Components. In *Proceedings of 9th Working IEEE/IFIP Conference on Software Architecture*, pages 161–170, 2011.

[15] B. Barry, M. Brian, K. John, K.Amanda, and S. Steven. Easyliving: Technologies for intelligent environments. In *Proceedings of International Symposium on Handheld and Ubiquitous Computing*, pages 12–29. Springer, 2000.

[16] F. Bellifemine, A. Poggi, and G. Rimassa. JADE–A Fipa-Compliant Agent Framework. In *Proceedings of PAAM*, volume 99, page 33. London, 1999.

[17] G Blair, G Coulouris, J Dollimore, and T Kindberg. Distributed Systems: Concepts and Design, 2012.

[18] Frances M. T. Brazier, Jeffrey O. Kephart, H. Van Dyke Parunak, and Michael N. Huhns. Agents and Service-Oriented Computing for Autonomic Computing: A Research Agenda. *IEEE Internet Computing*, 13(3):82–87, 2009.

[19] T. Bures, I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit, and F. Plasil. DEECO: An Ensemble-Based Component System. In *Proceedings of the 16th International ACM Sigsoft Symposium on Component-based Software Engineering*, CBSE '13, pages 81–90, New York, NY, USA, 2013. ACM.

[20] A. Cannata, M. Gerosa, and M. Taisch. SOCRADES: A Framework for Developing Intelligent Systems in Manufacturing. In *Proceedings of IEEE International Conference on Industrial Engineering and Engineering Management*, pages 1904–1908, 2008.

[21] N. Capodieci, E. Hart, and G. Cabri. Designing Self-Aware Adaptive Systems: From Autonomic Computing to Cognitive Immune Networks. In *Proceedings of International Conference on Self-Adaptation and Self-Organizing Systems Workshops*, pages 59–64, 2013.

[22] D. Carlson and A. Schrader. Dynamix: An Open Plug-and-Play Context Framework for Android. In *Proceedings of IEEE International Conference on the Internet of Things (IOT)*, pages 151–158, 2012.

[23] D. Cassou, J. Bruneau, C. Consel, and E. Balland. Toward a Tool-Based Development Methodology for Pervasive Computing Applications. *IEEE Transactions on Software Engineering*, 38(6):1445–1463, Nov 2012.

[24] G. Castelli, M. Mamei, A. Rosi, and F. Zambonelli. Pervasive Middleware Goes Social: The SAPERE Approach. In *Proceeding of 5th IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops*, pages 9–14, 2011.

[25] E. Chan, M.and Campo, D. Esteve, and J. Fourniols. Smart Homes-Current Features And Future Perspectives. *Maturitas*, 64(2):90–97, 2009.

[26] M. Chan, D. Esteve, C. Escriba, and E. Campo. A Review Of Smart Homes-Present State And Future Challenges. *Computer Methods and Programs in Biomedicine*, 91(1):55–81, 2008.

[27] Autonomic Computing et al. An Architectural Blueprint for Autonomic Computing. *IBM White Paper*, 2006.

[28] Diane J. Cook and Sajal K. Das. How Smart Are Our Environments? An Updated Look at the State of the Art. *Pervasive Mobile Computing*, 3(2):53–73, March 2007.

[29] Diane J. Cook and Sajal K. Das. Pervasive computing at scale: Transforming the state of the art. *Pervasive and Mobile Computing*, 8(1):22 – 35, 2012.

[30] M. Cossentino, N. Gaud, V. Hilaire, S. Galland, and A. Koukam. ASPECS: An Agent-Oriented Software Process for Engineering Complex Systems. *In Autonomous Agents and Multi-Agent Systems*, 20(2):260–304, 2010.

[31] P. Costa, G. Coulson, R. Gold, M. Lad, C. Mascolo, L. Mottola, G: P. Picco, T. Sivaharan, N. Weerasinghe, and S. Zachariadis. The RUNES Middleware for Networked Embedded Systems and its Application in a Disaster Management Scenario. In *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications*, pages 69–78. IEEE Computer Society, 2007.

[32] M. Crotty, N. Taylor, H. Williams, K. Frank, I. Roussaki, and M. Roddy. A Pervasive Environment Based on Personal Self-Improving Smart Spaces. In *Constructing Ambient Intelligence*, pages 58–62. Springer, 2009.

[33] E. Curry. Message-Oriented Middleware. *Middleware for Communications*, pages 1–28, 2004.

[34] Eric M. Dashofy, A. Van der Hoek, and Richard N. Taylor. Towards architecture-based self-healing systems. In *Proceedings of the first workshop on Self-healing systems*, pages 21–26. ACM, 2002.

[35] S. Dawson-Haggerty, A. Krioukov, J. Taneja, S. Karandikar, G. Fierro, N. Kitaev, and D. Culler. BOSS: Building Operating System Services. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, pages 443–458, 2013.

[36] R. De Lemos, H. Giese, Hausi A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, Norha M. Villegas, T. Vogel, et al. Software Engineering for Self-Adaptive Systems: A Second Research Roadmap. In *Software Engineering for Self-Adaptive Systems II*, pages 1–32. Springer, 2013.

[37] E. Di Nitto, Daniel J. Dubois, and R. Mirandola. On Exploiting Decentralized Bio-Inspired Self-Organization Algorithms to Develop Real Systems. In *Proceedings of ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 68–75. IEEE, 2009.

[38] COOK J. DIANE and K. SAJAL. Smart environments: Technologies, protocols, and applications. *Hoboken: John Wileyand Sons*, 2005.

[39] S. Dobson, R. Sterritt, P. Nixon, and M. Hinchey. Fulfilling the Vision of Autonomic Computing. *Computer*, 43(1):35–41, 2010.

[40] M. Dorigo and C. Blum. Ant Colony Optimization Theory: A Survey. *Theoretical Computer Science*, 344(2):243–278, 2005.

[41] Bruce Powel Douglass. *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*, volume 1. Addison-Wesley Professional, 2003.

[42] Patrick Th. Eugster, Pascal A. Felber, R. Guerraoui, and A. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003.

[43] Jose L. Fernandez-Marquez, Giovanna Di M. Serugendo, and S. Montagna. Bio-Core: Bio-Inspired Self-Organising Mechanisms Core. In *Bio-Inspired Models of Networks, Information, and Computing Systems*, pages 59–72. Springer, 2012.

[44] I. Fister, X. Yang, and J. Brest. A Comprehensive Review of Firefly Algorithms. *Swarm and Evolutionary Computation*, 13(0):34–46, 2013.

[45] George H. Forman and J. Zahorjan. The Challenges of Mobile Computing. *Computer*, 27(4):38–47, 1994.

[46] D. Garlan. Software Architecture: A Travelogue. In *Proceedings of the on Future of Software Engineering*, pages 29–39. ACM, 2014.

[47] D. Garlan, Shang-Wen Cheng, An-Cheng Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *Computer*, 37(10):46–54, 2004.

[48] D. Garlan and B. Schmerl. Model-Based Adaptation for Self-Healing Systems. In *Proceedings of the first workshop on Self-healing systems*, pages 27–32. ACM, 2002.

[49] D. Garlan, B. Schmerl, and S. Cheng. Software Architecture-Based Self-Adaptation. In *Autonomic computing and networking*, pages 31–55. Springer, 2009.

[50] I. Georgiadis and J. Magee, J.and Kramer. Self-Organising Software Architectures for Distributed Systems. In *Proceedings of the First Workshop on Self-healing Systems*, WOSS '02, pages 33–38, New York, NY, USA, 2002. ACM.

[51] V. Gorodetsky. Agents and Distributed Data Mining in Smart Space: Challenges and Perspectives. In *Agents and Data Mining Interaction*, pages 153–165. Springer, 2013.

[52] A. Gul. Abstracting Interaction Patterns: A Programming Paradigm for Open Distributed Systems. *Formal Methods for Open Object-based Distributed Systems*, 1, 1997.

[53] L. Gurgen, O. Gunalp, Y. Benazzouz, and M. Gallissot. Self-Aware Cyber-Physical Systems and Applications in Smart Buildings and Cities. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1149–1154, 2013.

[54] V. Hadzilacos and S. Toueg. A Modular Approach to Fault-Tolerant Broadcasts and Related Problems. Technical report, NY, USA, 1994.

[55] W. Heaven, D. Sykes, and J. Magee, J.and Kramer. A Case Study In Goal-Driven Architectural Adaptation. In *Software Engineering for Self-Adaptive Systems*, pages 109–127. Springer, 2009.

[56] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, and E. Jansen. The Gator Tech Smart House: A Programmable Pervasive Space. *Computer*, 38(3):50–60, 2005.

[57] S. Helal and S. Tarkoma. Smart Spaces. *IEEE Pervasive Computing*, (2):22–23, 2015.

[58] Y. Hen-I, C. Chao, A. Bessam, and H. Sumi. A Framework for Evaluating Pervasive Systems. *International Journal of Pervasive Computing and Communications*, 6(4):432–481, 2010.

[59] G. Hohpe and B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., 2003.

[60] A. Holmes, H. Duman, and A. Pounds-Cornish. The Idorm: Gateway to Heterogeneous Networking Environments. In *Proceedings of International ITEA Workshop on Virtual Home Environments*, volume 1, pages 20–21, 2002.

[61] J. Honkola, H. Laine, R. Brown, and O. Tyrkko. Smart-M3 Information Sharing Platform. In *Proceeding of IEEE Symposium on Computers and Communications (ISCC), 2010*, pages 1041–1046, 2010.

[62] M. S. Hossain, S. Alamgir. Hossain, A. Alamri, and M. A. Hossain. Ant-Based Service Selection Framework for a Smart Home Monitoring Environment. *Multimedia Tools and Applications*, 67(2):433–453, 2013.

[63] Raymond James and Inc. Associates. The Internet of Things, A Study in Hype, Reality, Disruption, and Growth. *US Industry Report*, 2014.

[64] G.and Jeremy J. Klyne. Resource Description Framework (RDF): Concepts and Abstract Syntax. 2006.

[65] Dmitry G. Korzun, S. Balandin, and A. Gurtov. Deployment of Smart Spaces in Internet of Things: Overview of the Design Challenges. In Sergey Balandin, Sergey Andreev, and Yevgeni Koucheryavy, editors, *Internet of Things, Smart Spaces, and Next Generation Networking*, volume 8121 of *Lecture Notes in Computer Science*, pages 48–59. Springer Berlin Heidelberg, 2013.

[66] J. Kramer and J. Magee. Self-Managed Systems: An Architectural Challenge. In *Proceedings of Future of Software Engineering*, pages 259–268. IEEE, 2007.

[67] Edward A. Lee, II. Davis, L. Muliadi, S. Neuendorffer, J. Tsay, et al. Ptolemy II, Heterogeneous Concurrent Modeling and Design in Java. Technical report, DTIC Document, 2001.

[68] W. Lee, S. Cho, W. Song, K. Um, and K. Cho. UbiSim: Multiple Sensors Mounted Smart House Simulator Development. In *Proceedings of IEEE International Conference on Dependable, Autonomic and Secure Computing*, pages 450–453, 2013.

[69] Z. Lei, S. Yue, C. Yu, and S. Yuanchun. SHSim: An OSGI-based Smart Home Simulator. In *Proceedings of 3rd IEEE International Conference on Ubi-media Computing (U-Media), 2010*, pages 87–90. IEEE, 2010.

[70] J. Ma, Laurence T. Yang, Bernady O. Apduhan, R. Huang, L. Barolli, and M. Takizawa. Towards A Smart World and Ubiquitous Intelligence: A Walkthrough from Smart Things to Smart Hyperspaces and Ubickids. *International Journal of Pervasive Computing and Communications*, 1(1):53–68, 2005.

[71] C. Magerkurth, R. Etter, M. Janse, J. Kela, O. Kocsis, and F. Ramparany. An Intelligent User Service Architecture for Networked Home Environments. In *2nd IET International Conference on Intelligent Environments, 2006.*, pages 361–370, July 2006.

[72] I. Marsa-Maestre, M.A. Lopez-Carmona, J.R. Velasco, and A. Paricio. Mobile Devices for Personal Smart Spaces. In *Proceedings of 21st International Conference on Advanced Information Networking and Applications Workshops*, volume 2, pages 623–628, 2007.

[73] M. Martin and P. Nurmi. A Generic Large Scale Simulator for Ubiquitous Computing. In *Proceedings of 3rd Annual International Conference on Mobile and Ubiquitous Systems*, pages 1–3, 2006.

[74] P. Mell and T. Grance. The Nist Definition Of Cloud Computing. 2011.

[75] G. Mone. Intelligent Living. *Communications of the ACM*, 57(12):15–16, 2014.

[76] Patrick D. O Brien and Richard C. Nicol. FIPA - Towards a Standard for Software Agents. *BT Technology Journal*, 16(3):51–59, 1998.

[77] P. Oreizy, Michael M. Gorlick, Richard N. Taylor, D. Heimbigner, G. Johnson, A.and Rosenblum David S. Medvidovic, N.and Quilici, and Alexander L. Wolf. An Architecture-Based Approach To Self-Adaptive Software. *IEEE Intelligent systems*, (3):54–62, 1999.

[78] J. Park, M. Moon, S. Hwang, and K. Yeom. Cass: A context-aware simulation system for smart home. In *Proceedings of ACIS International Conference on Software Engineering Research, Management Applications*, pages 461–467, Aug 2007.

[79] D. Peleg. Distributed Computing. *Siam Monographs On Discrete Mathematics And Applications*, 5, 2000.

[80] Jan S. Rellermeyer, O. Riva, and G. Alonso. AlfredO: An Architecture for Flexible Interaction With Electronic Devices. In *Proceedings of the ACM/IFIP/USENIX International Conference on Middleware*, pages 22–41, 2008.

[81] J.M. Reyes Alamo and J. Wong. Service-Oriented Middleware for Smart Home applications. In *Proceedings of IEEE Wireless Hive Networks Conference*, pages 1–4, 2008.

[82] S. Russell and P. Norvig. Artificial Intelligence: A Modern Approach. 1995.

[83] M. Saleemi, N. Díaz Rodríguez, E. Suenson, J. Lilius, and I. Porres. Ontology Driven Smart Space Application Development. *Semant Interoper. Issues Solut. Chall*, 101125, 2012.

[84] Vasile M. Scuturici, S. Surdu, Y. Gripay, and Jean-Marc Petit. UbiWare: Web-Based Dynamic Data & Service Management Platform for Ami. In *Proceedings of the Posters and Demo Track*, page 11. ACM, 2012.

[85] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*, volume 1. Prentice Hall Englewood Cliffs, 1996.

[86] R. Singh, P. Bhargava, and S. Kain. State Of The Art Smart Spaces: Application Models and Software Infrastructure. *Ubiquity*, 2006(September):7, 2006.

[87] T. Sivaharan, G. Blair, and G. Coulson. Green: A Configurable and Re-Configurable Publish-Subscribe Middleware for Pervasive Computing. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, pages 732–749. Springer, 2005.

[88] R. G. Smith. The Contract Net Protocol: High-Level Communication And Control In A Distributed Problem Solver. *IEEE Transactions on computers*, (12):1104–1113, 1980.

[89] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. In Alan H. BondLes Gasser, editor, *Readings in Distributed Artificial Intelligence*, pages 357 – 366. Morgan Kaufmann, 1988.

[90] P. Spiess, S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L. Souza, and V. Trifa. SOA-Based Integration of the Internet of Things in Enterprise Services. In *IEEE International Conference on Web Services*, pages 968–975, 2009.

[91] D. Sykes, J. Magee, and J. Kramer. FlashMob: Distributed Adaptive Self-Assembly. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '11, pages 100–109, New York, NY, USA, 2011. ACM.

[92] T. Teixeira, S. Hachem, V. Issarny, and N. Georgantas. Service Oriented Middleware for the Internet of Things: A Perspective. In *Towards a Service-Based Internet*, volume 6994 of *Lecture Notes in Computer Science*, pages 220–229. Springer Berlin Heidelberg, 2011.

[93] D. Thain, T. Tannenbaum, and M. Livny. Distributed Computing In Practice: The Condor Experience. *Concurrency-Practice and Experience*, 17(2-4):323–356, 2005.

[94] F. Tisato, C. Simone, D. Bernini, Marco P. Locatelli, and D. Micucci. Grounding Ecologies on Multiple Spaces. *Pervasive and mobile computing*, 8(4):575–596, 2012.

[95] S. Tisue and U. Wilensky. Netlogo: A Simple Environment for Modeling Complexity. In *Proceedings of International Conference on Complex Systems*, pages 16–21, 2004.

[96] T. Van Nguyen, J.G. Kim, and D. Choi. Iss: The interactive smart home simulator. In *Proceedings of . ICACT International Conference on Advanced Communication Technology, 2009*, volume 3, pages 1828–1833. IEEE, 2009.

[97] E. Warriach, E. Kaldeli, A. Lazovik, and M. Aiello. An Interplatform Service-Oriented Middleware for the Smart Home. *International Journal of Smart Home*, 7(1):115–141, 2013.

[98] M. Weiser. The Computer for the 21st Century. *Scientific American*, 265:94–104, 1991.

[99] D. Weyns and T. Ahmad. Claims And Evidence for Architecture-Based Self-Adaptation: A Systematic Literature Review. In *Software Architecture*, pages 249–265. Springer, 2013.

[100] D. Weyns, N. Boucké, and T. Holvoet. Gradient Field Based Transport Assignment in AGV Systems. In *Proceedings of 5th International Joint Conference on Autonomous Agents and Multi-Agent Systems, AAMAS, Hakodate, Japan*, 2006.

[101] D. Weyns, N. Boucke, and T. Holvoet. DynCNET: A Protocol for Dynamic Task Assignment in Multiagent Systems. In *Proceedings of 1st International Conference on Self-Adaptive and Self-Organizing Systems*, pages 281–284, July 2007.

[102] D. Weyns and M. Georgeff. Self-Adaptation Using Multiagent Systems. *Software, IEEE*, 27(1):86–91, 2010.

[103] M. Wooldridge. Agent-Based Computing. *Interoperable Communication Networks*, 1:71–98, 1998.

[104] Xin-She Yang. Firefly Algorithms for Multimodal Optimization. In *Stochastic Algorithms: Foundations and Applications*, pages 169–178. 2009.

[105] Xin-She Yang. Flower Pollination Algorithm for Global Optimization. In *Proceedings of International Conference on Unconventional Computation and Natural Computation*, pages 240–249. 2012.

[106] F. Zambonelli and M. Viroli. A Survey on Nature-Inspired Metaphors for Pervasive Service Ecosystems. *International Journal of Pervasive Computing and Communications*, 7(3):186–204, 2011.