NTNU

Norwegian University of Science and Technology

Managing spaces in context-aware ubiquitous systems

Waqas Hussain Siddiqui

Master in Information Systems
Submission date: June 2010
Supervisor: Babak Farshchian, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

Context awareness is an important aspect of ubiquitous computing and in order for a system to adapt to user needs or to provide relevant information at right time and place, information about context is required. Due to the advancement in technology, vast amount of information can be gathered from different heterogeneous systems and sensors taking part in ubiquitous scenario. But due to their heterogeneous nature and huge amount of available information it is necessary to extract only useful information which should cover required aspects of context and this information must be in common and predefined format so that semantic meaning can easily be added. The aim of this work is to design and implement architecture for UbiCollab Space Manager which helps UbiCollab user to create and manage spaces, capture and associate contextual information with spaces.

Assignment given: 15. January 2010
Supervisor: Babak Farshchian, IDI

Abstract

In our everyday tasks context plays an important role, we act based on the information we have or based on what we can see, hear or feel about surrounding. Using this information about context we use to adapt ourselves and our behavior for example in class room we usually whisper when we want to communicate with other class fellow, but in cafeteria we talk normally.

Due to the advancements in technology and mobile computing, we are now able to carry computers and smart phones with us, almost everywhere and use them as an alternative to desktop computers. Ubiquitous computing goes step further and refers to the world where computation is being weaved into every day object. In typical ubiquitous computing scenario many invisible computers interact with each other to help user in getting his task done. The ability of being carried easily, i.e. mobility and their presence almost everywhere make it necessary for computer systems, taking part in ubiquitous computing environment, context-aware. If computers can sense the environment they are being used in, they can help user in providing only relevant information, information at correct place and time and such systems can also adapt their behavior according to their surroundings. For example, if would be nice if our mobile phone automatically set to silent profile, whenever we are in class room or in a meeting room.

Ubicollab is a platform for supporting collaboration and is a result of research work done in the areas of mobility and ubiquitous computing. Mobility and ubiquity being the inherent properties of UbiCollab, requires it to be aware of context just like another ubiquitous system. It will help UbiCollab applications to adapt their behavior as per surrounding and will enhance the experience of collaboration by using the resources nearby.

I researched in the area of context-aware ubiquitous computing and used the results of my research to design and implement a solution for making UbiCollab context-aware. The proposed solution answers research problem related with context itself and different aspect of context. Context definition for UbiCollab has already been defined in work previously done; my solution addresses how to represent this contextual information in simple and effective manner, how to gather location information using different and heterogeneous sensors in understandable and standard format.

The outcome of this work comprises of proposed context model, design and implementation of Space Manager for working with spaces, design and usage of flexible data store for storing space information and design and implementation of Location Service Manager for gathering location information using different location sensing technologies.

Keywords: Ubiquitous computing, context-aware computing, context-awareness, context model, location awareness, sensor

Preface

This report was written as a master thesis in Spring 2010, which account towards the final work for the degree of Master of Science in Information System taken at Department of Computer Science, Norwegian University of Science and Technology.

This report contains the contribution I made for UbiCollab platform. In this report the solution I proposed and implemented for making UbiCollab context-aware is presented. My work is based on the previous work done about context and spaces in UbiCollab. This report contains the design and implementation details of UbiCollab Space Manager for managing spaces and design and implementation details of Location Service Manager for getting and using location information as location aspect of context.

I wish to thank my supervisor Babak Farshchian for all the support during the course of this thesis and the valuable feedbacks throughout my thesis.

Thursday, June 10, 2010

Waqas Hussain Siddiqui

Table of Contents

List of Figures

List of Code Listings

List of Tables

Chapter 1 Introduction

Gone are the days when computers were isolated and used for very specific tasks. Now a day a lot of research is being done in the field of computing that deals with the idea to weave computation into everyday objects and activities and make computers invisible to the user. This is known as ubiquitous computing and refers to post-desktop model, making many computers available throughout the physical environment while keeping them invisible to the user.

Presence of computer systems or objects taking part in ubiquitous computing environment, every where possible makes them necessary to be aware of the surrounding they are in so that they can provide relevant information to the user and in the format user can understand. This makes context-awareness an important aspect of ubiquitous computing. By context-awareness we mean the ability of a system to sense and react based on the environment it is in [7]. Using this ability not only system can adapt its behavior according to the environment but can also provide the relevant information in the timely manner and in an understandable format. Context in ubiquitous computing generally refers to the information surrounding a system such as the place where it is located, who else is in the surrounding and what are the available resources in proximity.

UbiCollab is a platform for supporting collaboration that captures the commonality of collaborative applications. This platform is based on the research done in the areas of user mobility and ubiquitous computing. Figure 1-1 depicts a brief overview of UbiCollab as a cross section of research areas.

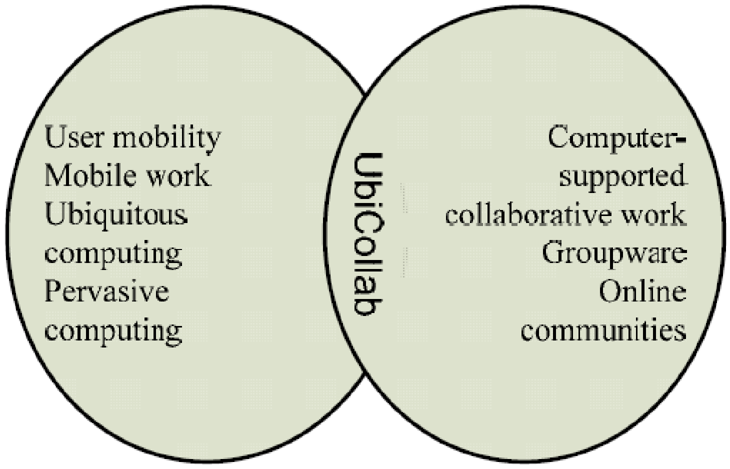


Figure 1-1 UbiCollab

Being a ubiquitous collaborative platform, UbiCollab treats mobility and ubiquity as inherent properties of social interaction [1]. These inherent properties require UbiCollab to be context-aware so that user can benefits from the mobile nature by using it anywhere at any time and can be able to integrate external resources within the surrounding without any prior knowledge or configuration.

1.1 Motivation and Contribution

1.1.1 Motivation

The motivation of this work is to perform research in the area of ubiquitous computing that deals with the context-awareness and to propose a solution for UbiCollab Space Manager that will help in providing the relevant information in an understandable format at the right place.

Context-awareness has gained a lot of attention in ubiquitous computing and has become a primary concern when making applications for ubiquitous computing. By using contextual information systems can adapt their behavior according to environment and can also takes decisions on the behalf of user. But before moving toward context-awareness and behavior adaptation proper definition of context must be specified otherwise system will end up in gathering a lot of useless information from surrounding. After which the context-model is required to structure gathered context information in some concrete format so that it can be used by any interested application.

Up till now most of the research on context-awareness is focused on single user system which either supports very minimum or no collaboration. Research done in the area of Computer Supported Cooperative Work (CSCW) has shown that context plays an important role in cooperation. Therefore, by researching in the area of making UbiCollab context aware, we can help both research communities.

1.1.2 Contribution

In this project we will be extending the UbiCollab by proposing the architecture of Space Manager, a context model for context-data representation and the architecture for extracting high-level information from the low-level sensor readings. This work of ours will lay the foundation for Space Manager and handling context-awareness and location awareness in UbiCollab.

Below are the contributions that will be made during the course of this project:

1. Research: Context-model for context data representation

2. Research: Algorithm for extracting information from low-level location sensors and transforming them into common and understandable format – Location Service Manager

3. Implementation: Developing Space Manager module for creating and managing spaces

4. Implementation: Creating GUI for Space Manager using the previously developed GUI framework for UbiCollab

5. Implementation: Generic database interface for storing and retrieving space information

6. Implementation: Wi-Fi Sensor Plug-in

7. Implementation: Fingerprinting algorithm for Wi-Fi based location detection

8. Implementation: SpaceTwitt Application

1.1.3 Objectives

The notion "computer everywhere" of ubiquitous computing makes up a scenario in which user comes in contact with many heterogeneous devices and these devices also interact with each other to fulfill the need of user. Providing the context related information is the key to ubiquitous computing but this information should also be in a standard and understandable format. UbiCollab being a platform for supporting collaboration and assumes that collaboration can happen anywhere, makes it necessary for UbiCollab to sense the location user is in and provide the information that is only relevant. This is where location awareness aspect of context-awareness comes in as location is one of the key aspects that comprises context.

Earlier research on context-awareness has shown that location can be sensed using different sensors such as wireless sensors, GPS sensors and RFID tags, but none of them alone fulfill the requirement of our UbiCollab project that assumes user can use UbiCollab anywhere and at any time. Therefore, the first research objective is to come out with a context-model that can represent the context in a comprehensive way and must cover basic aspect of context that is *who, what* and *where*.

Second research objectives of this project is to lays the ground by doing for an architecture that provides a standard way of transforming low-level sensor data into high-level meaningful data which then can be used to extract data relevant to particular context.

1.2 Research Method

The research methods we employed throughout the course of this project are: design-science, and literature review.

- ***Design Science Model***
  Author in [8] argues that Information System research involves two complementary but distinct paradigms, behavioral sciences and design sciences. Behavioral science has its roots in natural science research method and seeks to develop and justify theories that explain or predict organization and human phenomena i.e. to investigate the non-technical aspects of information systems. In contrast, design science is a problem solving paradigm which involves the creation, analysis and evaluation of design artifact to gain domain knowledge and propose a solution.

  In [8], author also specified seven guidelines that should be taken care of for effective design science research. According to Guideline 1 "design science research must produce a viable artifact in the form of a construct, a model, a method or an instantiation". In computer science artifact can be algorithms, proof of concept applications or prototype (this list is not exhaustive). In my case the artifact is architecture for Space Manager Subsystem. Guideline 2 suggests that "the objective of design science research is to develop technology based solutions to important and relevant business problems." Since in this project my main focus is extending state-of-art collaborative platform for mobile system by using ubiquitous computing, location services and mobile computing, this use of technology relates to guideline 2. Scenarios will be used throughout the course of this project to define problem, sketch functional and non functional requirements and later they will help in evaluating the proposed solution. Also fully functional prototype will be created as the outcome of this project; this final prototype would be refined continuously throughout the course of project by discussing it with supervisor and any other stakeholder. This iterative process of refining artifact is according to Guideline 6 that says "The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment". The evaluation strategy described before conforms to the Guideline 3 which states that "The utility, quality, and efficacy of design artifact must be rigorously demonstrated via well executed evaluation method".

- ***Literature Review***
  Briony J Oates in [9] states that literature review falls into two parts, exploring the literature for selecting research topic and once a topic is chosen. For my project, literature review is very necessary and it will actually derive my research work. Because I will be going to extend the functionalities of an already built system, my proposed solution should follow the current architecture and should not result in removing any functionality system current poses. One of the objectives Briony J Oates mentioned in [9] is "Show that researcher is aware of existing work in the chosen topic area", White paper on UbiCollab and other architectural papers written on UbiCollab would be a good start, not only making me familiar with the UbiCollab platform but will also help me exploring my first research question. Once the key issues regarding context-awareness that are troubling UbiCollab developer/users/researchers are identified, I will extend my literature review to other papers and publications relevant to the topic of location, space and location based services. Within last few years, a lot of work has done in detecting the user's location, although almost all of them are focused to only single user system, but their research has produced some good methods of detecting location. Reviewing literature on this topic will help me in exploring the second research question I have on my list. As the third question is very specific to UbiCollab platform, and I might have gotten to some place by exploring previous two questions, reviewing technical documentation or manual, focusing on the tools and technologies being used by UbiCollab platform will help me exploring my third question.

- ***Usability Testing and Evaluation***
  In [10] author describes usability testing as an evaluation method which deals with evaluating effectiveness, ease of use and satisfaction. "Usability testing is an approach that emphasis the property of being useful and it is conducted in controlled environment". As I have already mentioned in previous section that scenarios will be used throughout the course of this project to articulate the problem and to evaluate the proposed solution prototype. This evaluation will take place in the room reserved for people working on the UbiCollab project because it has all the resources required for testing and creating sample environment, in [10] author has also states that "usability testing in a laboratory is most suitable for testing software upgrades,

prototypes and working systems"; a sample scenario will be used to confirm the functionalities with requirements, that solution provides and to evaluate if the prototype is behaving/performing as expected.

1.3 Report Outline

Given below is the brief overview of remaining chapters:

- **Chapter 2 – Problem Elaboration**
  This chapter describes the research problem and concepts related to it in greater detail. It highlights problems related with context awareness and how they are related with UbiCollab. This chapter also includes the sample scenario that is used to elicit functional and non-functional requirements at the end of same chapter.

- **Chapter 3 – Background**
  This chapter covers through study of current existing UbiCollab system and core concepts related to it. Work done previously in the same research area is also presented in this chapter.

- **Chapter 4 – Proposed Solution**
  This chapter is dedicated toward solution we proposed for the research problem described in Chapter 2. It describes everything that constitutes our solution in greater detail; intended graphical user interface for Space Manager is also presented in this chapter. Overall architecture of proposed system and where everything lies in UbiCollab is presented at the end of this chapter.

- **Chapter 5 – Implementation**
  Implementation chapter takes into account how the solution proposed in Chapter 4 is actually implemented. It describes the inner functionality of every important component in detail using combination of UML class and sequence diagrams.

- **Chapter 6 – Evaluation**
  This chapter describes how the implemented solution is evaluated. Evaluation is done in two different ways and both of the ways with relevant screen shots are discussed in details. This chapter also includes the satisfaction status for requirements.

- **Chapter 7 – Conclusion and Future work**
  This chapter concludes report by discussing the project outcome, summarizing the contribution made and suggesting the ideas for future work. This chapter marks an end to this report.

Chapter 2 Problem Elaboration

This chapter tries to give a deeper understanding of the problem. First it describes the problem in general and continues to relate it with UbiCollab. Then the scenario is described that will be used to elaborate the intended functionalities and elicit functional and non-functional requirement.

2.1 Problem Definition

Context awareness is an important aspect of ubiquitous computing and in order for a system to adapt to user need or to provide relevant information at right time and place, information about context is required. It is usual for ubiquitous computing scenario to have many heterogeneous devices interacting together to fulfill user requirements. This advancement in technologies makes it possible to gather vast amount of information about the devices and surrounding, this information from different devices taking part in ubiquitous system comprises the context. But heterogeneous nature of these objects makes the use of this gathered information almost impossible and there exists a huge gap between raw information gathered from devices and high-level contextual information. Therefore, it is necessary to extract only useful information from the whole lot of information gathered, also this extracted information must be in some proper and predefined format so that semantic meaning can easily be added; Otherwise it will make using and sharing this information really difficult.

2.1.1 Context Model

In ubiquitous computing scenario information about the surrounding and facts can be gathered very easily with the help of different objects taking part in the scenario. But in order to use this information and to add semantic meaning to it, this information first should be transformed to some common format, with common format we mean format should have a predefined structure, making it possible to be used by any object that can understand the structure and should contain enough information about the context.

2.1.2 Location Detection

Location plays an important role in both context-aware and collaborative systems. It enables systems to sense and react based on the environment they are in. Today there are many different locations sensing technologies available such as Low frequency RF, Infrared, Global Positioning System (GPS) or Ultrasonic [15] but no single technology works in every environment, some works indoors and some only works outdoors. Also they give information about location in different formats. Therefore, in order to use this information as an aspect of context, it is necessary to fuse the raw data we get from difference sensors into some standard format.

2.2 Relation with UbiCollab

As UbiCollab promises to provide collaboration opportunity anywhere possible, it makes it necessary for UbiCollab to be aware of the context it is being used in. This has already been realized during previous work on UbiCollab and in UbiCollab Architecture White Paper [1] it is mentioned that “Physical spaces and location plays a central role in UbiCollab and are used as resources for collaboration”. Therefore UbiCollab must keep track of user’s location and the other resources in the surrounding that are of interest to the user. It should be capable of gathering location information from different location sensing technologies and should not rely on any single one because of its peer-to-peer and collaboration anywhere nature, then transforming that information to the simple and standard yet expressive contextual information.

2.3 Scenario Analysis

In this section we will present a scenario that will be used throughout this project as a guideline. This scenario will help us in elaborating the intended functionalities of presented system and how it benefits the UbiCollab user. This scenario will also be used to draw functional and non-functional requirements from it. The scenario is divided into sections in order to highlight the different situations in which UbiCollab user can benefit from the proposed subsystem.

2.3.1 Scenario

Markus works as a VP IT services in the IT department of one of the leading bank. Teams that he manages are geographically distributed within the different departments of bank; therefore a platform that provides collaboration irrespective of geographical position is required. As a company policy they are using UbiCollab as a collaboration platform to fulfill this requirement.

Creating a new space:

Markus uses UbiCollab not only to collaborate with his subordinates and colleagues but also to remain in contact with distant friends and family members. However, he doesn't want to be disturbed by his family or friends, whenever he is in his office, only, collaboration from the people within the workplace is allowed. Using the create new space interface of Space Manager, Markus creates a space in UbiCollab giving it a name 'Markus' Office' and associates it with the physical location of his office. This physical location-to-space mapping is done through location sensor plug-in available in his office, Wi-Fi in this case.

Proactive approach based on context:

Now whenever he enters in his office, he runs an UbiCollab application which basically sets instant messenger profile based on the space information. This application calls Space Manager for current space using the API Space Manager exposes, as a result Space Manager automatically senses the location as 'Markus' Office', sets this space as current space and returns it to the application. In turn application changes his status to 'Not available or Busy' to friends and family and 'Available' to colleagues.

Being a head of IT services, giving presentations at different locations in different departments is his routine job. It is usual for him to visit the same meeting room he was before in, so instead of discovering resources available in the room using UbiCollab Resource Discovery Manager every time and then configuring them for usage, he has created separate spaces using Space Manager for most common room. Now whenever he visits any meeting room which as an associated space, UbiCollab automatically detects the current space, by using information about current space he can browse through the list of available resources using Resource Discovery Manager.

Space within space

For creating new space Markus starts create new space interface of Space Manager, while preparing the create new space interface, Space Manager also loads all the spaces Markus owns and display them in form of selectable list. Using this interface he creates a new space by providing the name 'Markus' Room – Home', other details and selects 'Home' as a parent space.

Last section of the scenario describes a situation which can be a very challenging too. Spaces can exist within subspaces like rooms in an office building or floors in a multistory house. System must provide a way to create and manage subspaces without affecting the main space. The key challenges here are, drawing the boundary lines for subspace and differentiating it from main space.

2.4 Requirement Specification

The aim of this section is to provide a better understanding of the specific requirements of the solution presented in this project. These requirements are drawn from the functionalities of Space Manager as provided in UbiCollab Architecture White Paper [1] and from the scenario described in the previous section. Each requirement follows a common template which consists of ID, brief description and priority.

Each specification ID also is a form of template which tells about what kind of specification requirement it is e.g. functional or non functional, is requirement is specifics to some component or general and the sequence number. For example NFR-G1 and FR-SM1 represents non-functional general requirement 1 and functional requirement 1 related to Space Manager, respectively.

NFR stands for Non-functional requirement

FR stands for Function requirement

G stands for General

SM stands for Space Manager

LSM stands for Location Service Manager

2.4.1 General Requirements

ID	Brief Description	Priority
NFR-G 1	The system must run on mobile and handheld devices which is in our case s UbiNode and must integrate with UbiCollab platform.	H
NFR-G2	The system must conform to the technical constraints put by UbiCollab.	H

NFR-G4	The system must provide an API for other components of UbiCollab.	H
NFR-G5	System must at least run on a CDC Java Virtual Machine.	H
NFR-G6	System's components should run on different OSGi Implementations.	M

Table 2-1 Non functional General Requirements

2.4.2 Specific Requirements – Space Manager

ID	Brief Description	Priority
FR-SM 1	User must be able to create new spaces and modify existing spaces.	H
FR-SM 2	Space manager must provide a way to associate context related information with the space.	H
FR-SM 3	Space manager must also allow user to associate location information with the space.	H
FR-SM 4	User must be able to identify current space automatically, where possible.	H
FR-SM 5	User must be able to set any space as current space.	H
FR-SM 6	User must be able to browse all the spaces he has created.	H
FR-SM 7	Space Manager must provide a way to share spaces and also provide a way to download spaces.	M
FR-SM 8	User must be able to create sub spaces within spaces.	L
FR-SM 9	Space Manager must be able to deal with different type of location information	M

Table 2-2 Specific Functional Requirements - Space Manager

ID	Brief Description	Priority

NFR-SM 1	Space Manager must at least provide the graphical user interface for basic operations.	H
NFR-SM 2	Space Manager must expose API interface for applications and other components of UbiCollab.	H
NFR-SM 3	Space Manager must provide space information in XML format with predefined structure	H
NFR-SM 4	Information captured as a part of space must conforms to the proposed context model	H

Table 2-3 Specific Non functional Requirements - Space Manager

Chapter 3 Background

3.1. UbiCollab context and background

3.1.1 Ubiquitous Computing

Ubiquitous computing is relatively a new field of research that was first articulated by Mark Weise and his colleagues in 1988 at the Computer Science Lab at Xerox PARC. The main idea behind ubiquitous computing is to weave computation into everyday objects and activities and make computers invisible to the user. To some ubiquitous computing is considered to be as Third Wave of computing, following "many people per computer" and "one person per computer" as First Wave and Second Wave of computing, respectively.

3.1.2 Context-aware computing

Context-awareness refers to the idea of such systems that are aware of their surrounding and can adapt their behavior depending on the changes in environment. This term is originated from ubiquitous computing [5]. Now a day mobile phone is no more a device for receiving and making calls only, it employs new technology, greater power and far more functionalities – actually a mobile computer system. Gone are the days when interaction between user and computer was only happen in static context such as home or office. Also the advent of wearable smart devices gives user the ability to access different computational resources through wireless network.

This advancement in technology and increase in mobility makes it important for applications running on mobile system to sense the environment they are in and to adapt to ever changing environment. These context-aware applications adapt themselves according to the location of use, information about other people in proximity and accessible devices or resources [7].

Following three are important aspects of context: *location of use*, *information about other user in surrounding* and *the nearby resources* [7]. In the context aspects mentioned location is the most common and important aspect but in this thesis we will focus on the system that not only captures the location information but keep track of other information as well.

3.2. Preliminary Study

The purpose of this section is to get information about the system and get an understanding of system's current situation. We start this section by briefly describing what UbiCollab is and its core concepts that are relevant to this project.

3.2.1 UbiCollab

UbiCollab, which stands for Ubiquitous Collaboration, is a platform for supporting collaboration on the internet. It is a service platform for provision of basic services for supporting collaboration among people. UbiCollab make extensive use of earlier CSCW research, and extends this research with insights from ubiquitous computing and wireless services. UbiCollab assumes that collaboration can happen in any place, not restricting users to one virtual or physical shared space. It assumes that users involved in collaborations will not necessarily be collocated, and will need access to various virtual and physical resources. UbiCollab collects and uses context about a group of people (e.g. their location) and promotes the usage of physical artifacts in collaboration wherever necessary [3].

UbiCollab provides a platform that captures the commonalty of collaborative applications and provides generic mechanism for applications to be built without extensive coding. UbiCollab tries to be domain-independent and providing only the basic functionality, is therefore following an open innovation approach where third party applications play an equally central role as the platform itself. Integration with physical environment where collaboration happens is a key aspect of UbiCollab [1].

UbiCollab architecture follows the Service-Oriented Architecture (SOA) approach. UC is implemented as a collection of independent components in form of dynamically deployable services that can be deployed and used independently on a mobile device. Each UC component is being developed to cover a very specific area of responsibility in UC. Components can be mixed and used together in different configurations (compositions) decided by the application using them. Only those components that are needed by a specific user (and his/her applications) will be deployed on his/her mobile device [1].

3.2.1.1 Human Grid

UbiCollab is based on the notion of human grid. A human grid is a collection of people and their artifacts/resources connected together using UC platform technology. Interactions in a human grid are supported using resources, artifacts, services, etc. imported into the grid by its participants. UC assists its users in building a human grid and supports communications among them, and they can be distributed geographically.

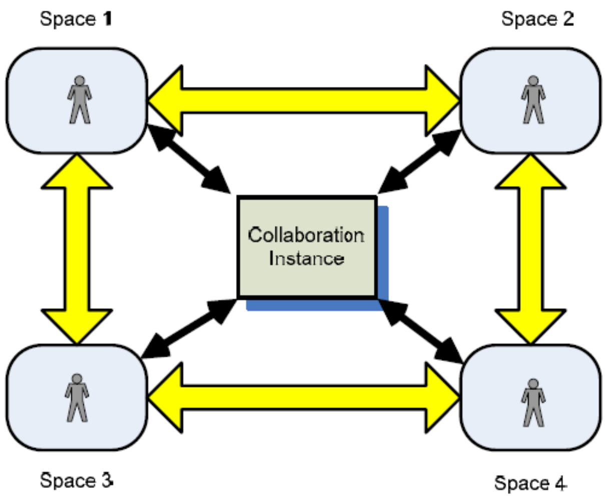


Figure 3-1 Human Grid

Figure 2.1 shows the concept of Human grid with Collaboration Instance in middle providing the context for collaboration. Spaces represent physical spaces (such as meeting rooms, offices, streets, homes) where UC user resides. Spaces contain physical resources and artifacts or digital services in the immediate surroundings of the users. These physical artifacts and services are tools or resources for interacting with the collaboration instance and other users in the grid.

Human grid is adaptive and reconfigurable in that it will change its configuration in order to best context, services and artifacts that users have available in any given space. It may change its configuration and deployment configuration in order to assists users in a lot of different scenarios, from work-collaborative related to health care assistance, as the user moves from a space to another [1].

3.2.1.2 UbiNode

Each user in UC is represented and assisted by a mobile device called a UbiNode as shown in figure 2.2. UbiNode is a network-enabled device that acts as a personal server, running a subset of the main UC components and some of user's application designed for it. This means that each user has his/her own instance of a Resource Discovery Manager, Service Domain Manager, Space Manager, CI Manager etc. running locally on his/her UbiNode. UbiNode is organized in a "platform space" where core components reside and a "user space" where each user can store and run his/her application which communicates with external devices. All the components allow interaction with other applications exposing Web Service interfaces. Complete independence among UC components allows us to outsource all composition tasks to the applications and guarantees a high level of modularity in the architecture of a UbiNode, in accordance with the Service-Oriented Architecture (SOA) approach [2].

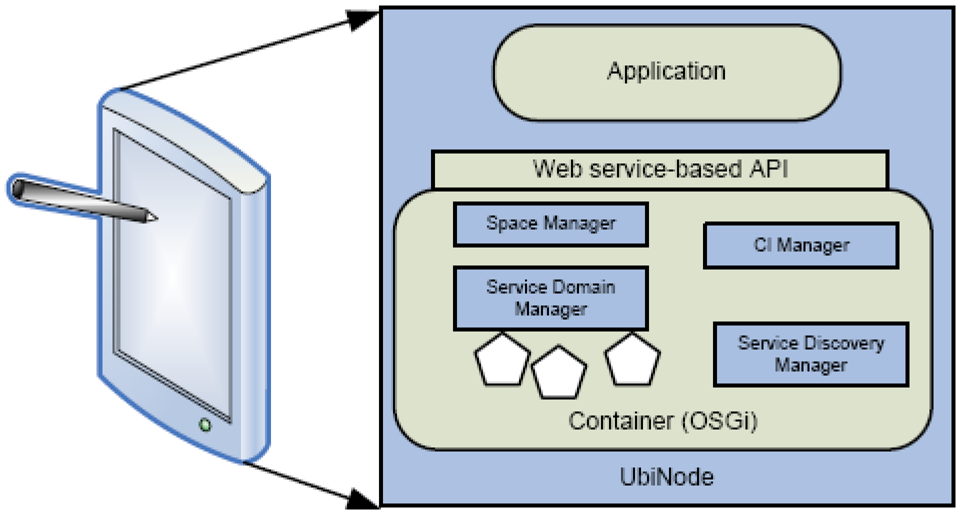


Figure 3-2 Architecture of UbiNode

3.2.1.3 Context in UbiCollab

In [5] author defines the context as:

"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."

And context-awareness as:

“A system is context-aware if it uses a context to provide relevant information and/or services to the user, where relevancy depends on the user’s task.”

As described in previous sections, UbiCollab is a platform for supporting collaboration over internet and based on the notion of human grid. The resources being used for collaboration and participants of human grid constitute the context in UbiCollab. Resources can be anything from spaces and physical locations to artifacts.

3.2.1.4 Physical Locations and Spaces

In order to define what the space is, consider an example of lawnmower, which is purely a physical device that resides and being used in the real world. This physical device exists only in a single space and its relationship is only with the physical world it resides in. Not only its location is unique to that space but its influences and effects are only through the physical space within which it resides [6].

In contrast ubiquitous systems can inhibit more than one space and at the same time they can consider their presence in both physical and virtual space. Surfing the web, using FTP to access remote files or exploring the file system are some example of virtual space and system having some computation power can do all these things simultaneously, while also being located somewhere in physical world. On the other hand physical space can be considered as an area with some fixed physical location, containing users and devices that somehow are related with other and with the space and can be the subjects influencing the space and devices and users within the space [6].

In the context of UbiCollab space represents an area of interest to the user that contains resources and users containing UbiNode, these resources can be used for collaboration by the users of UbiCollab. Although like any other context-aware platform supporting collaboration, in UbiCollab physical location plays a central role and is used as a resource for collaboration. But UbiCollab space may or may not be associated with some physical location. In the earlier case either the geometric coordinates of physical location or any other information that can be used to differentiate among different physical locations are stored.

A space can be an office, home etc. When space is used as a resource for collaboration (e.g. shared with participants in meeting), it is called a Collaboration Space [1].

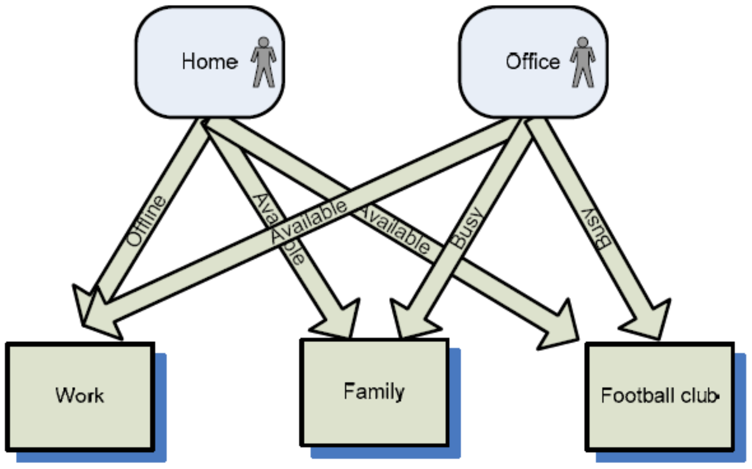


Figure 3-3 Spaces

Figure 3-3 depicts more than the idea of spaces described above, as shown in figure user has defined two spaces “Home” and “Office” where space home is the representation of home’s physical location and similarly office space is the representation of his office’s physical location. As a resource for collaboration these two spaces are used in three Collaboration Instances that the user owns or participates in.

3.3 Related Work

In this section we will briefly go through the related work previously done in the same research field. Although no previous work discussed here provides the complete solution to our problem but not only provide the inspiration for our contribution to this research field but also lays the ground for our proposed solution.

3.3.1 W4

Their work has the same objective as ours: A model for representing contextual information that is simple but expressive and extracting useable information from low-level context data [11]. W4 proposes a model for representing context-model and few algorithms for extracting high-level information from low-level information, enriching them with semantic meaning. Figure 3-4 depicts the general architecture of W4 that comprises of 3 layers: service layer, W4 context layer and data production Layer. Objects in data production layer generate W4 tuples, W4 context layer separates services in service layer from the raw context data and service layer creates context-aware services [11].

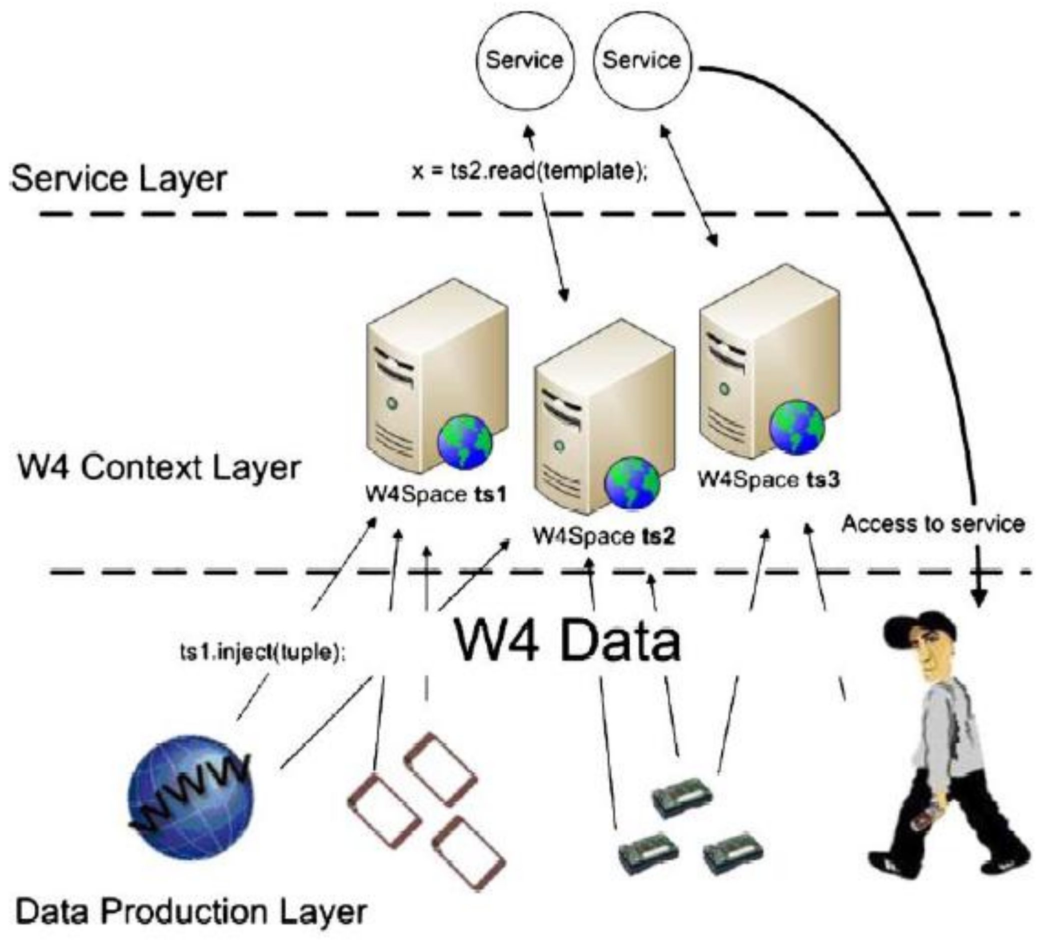


Figure 3-4 Architecture of W4

The context-model presented is the inspiration to work of ours in this project as the proposed contextual model is simple yet expressive. It represents context in the form of four Ws i.e. *Who*, *What*, *Where* and *When*. Each field in this tuple represents different aspect of context information, *who* represents the subject, *what* describes the activity being performed by the subject, *where* tells the location in which the action is being performed and *when* deals with the date and time of the action.

3.3.2 Place lab

Place lab is an open source initiative to help building location-aware applications. It falls into the category of fusion architecture that deals with refining raw sensor data into high-level information [16]. It is based on mediator/observer design pattern and follows layered event streaming fusion architecture [16], the Place Lab architecture is shown in Figure 3-5. Motive behind Place lab project is to provide a very generic, modular and cross platform toolkit that helps in making location-ware applications. While supporting modularity Place lab makes it sure to provide data from different sensor in common but distinguishable format so that no essential detail can be lost. By providing the support for sensing both indoor and outdoor environments using 802.11 access point information and Global Positioning System

(GPS) and Global System for Mobile Communications (GSM) respectively, Place lab is able to cover wide area [17].

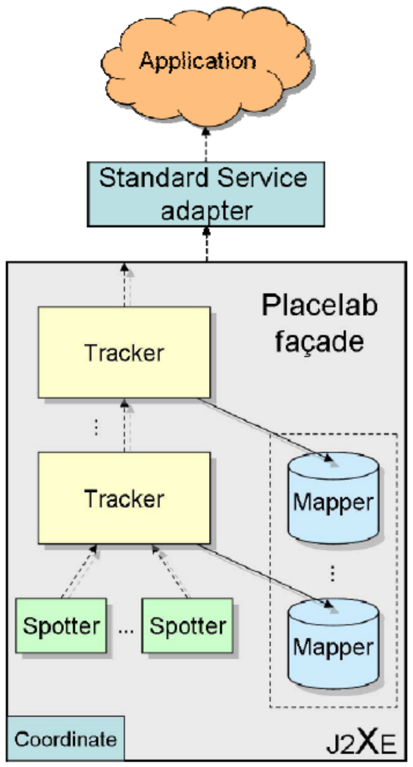


Figure 3-5 Place Lab Architecture

Figure 3-5 shows the architecture of Place lab, where main components are shown using boxes [16]. Spotter here represents the components responsible for abstracting away environment sensing hardware, Place lab comes with implementation of four standard spotters 802.11, GSM, Bluetooth and GPS. Tracker is the component responsible for position estimation; it gets spotter data in form of standard format and uses persistent data from Mapper, we will describe Mapper next, to calculate single position. Static databases that are used to store location information are called Mappers, they store location coordinate as well as the radius of coverage area. In order to use these components, Place lab provides classes with the name same as of component and each exposes generic methods to support operation they perform.

Although, the architecture of Place Lab makes it very good candidate to be used as a toolkit for enabling location-awareness, but in the context of UbiCollab using Place Lab as it is, is not feasible at all as it provides much more functionality than required. Our only requirement is to sense the environment and cover as wide as possible; we don't need any components like Mapper or Tracker. Therefore, instead of using Place Lab, we have proposed our own fusion system which is heavily based on the idea of Place Lab. The Location Service Manager that we are going to propose and implement has the same motives as of Place Lab and uses Place Lab as practical guideline.

3.3.3 Other related work

Context Models:

Context-toolkit [18] is a java based toolkit which aims toward providing help in developing context-aware applications. This toolkit provides widgets, aggregators and interpreters to abstract away context-sensing mechanism, aggregating contextual information about places and users and interpreting low-level contextual information into higher level information, respectively [19].

Context Fabric [20] is very similar to Context-toolkit with an exception that it focuses more on modeling and storing context data. Context Fabric also specifies Context Specification Language (CSL) similar to structured query language (SQL) to provide programming abstraction over context data. Their aim of representing context information in a format that any application can use and a flexible data store to store this information is similar to ours, their context model represents context using entities (people, place, thing), attributes (entity property), relationship among entities and aggregates. But because of no standard structure and complexity between contexts makes it difficult to browse, extract and use context information.

Location Detection:

The Active Badge Location System [21] is used to provide information about where people are, it uses a especially designed small size hardware, called badge that transmit infra-red signal every second. Receiver is being placed in the area which is being observed and when ever any person wearing badge moves between observed areas, respective signal receivers updates the location of that particular person in central system. Very specific to infrared signals, need special hardware, receivers need to be installed throughout the area are some major drawbacks.

Radar [22] is an indoor tracking system for location people inside of the building. The main advantage of Radar is it works on pre laid network of wireless location area network (WLAN). The main disadvantage of Radar, in the context of our work, is that it requires an offline calibration phase during the installation of system. This requirements kills the basic idea of peer-to-peer system as system in advance need to know about different 802.11 access point and their signal strength in order to use Radar.

Chapter 4 Proposed Solution

This chapter is dedicated towards the contribution we made and provides the description of our proposed solution for UbiCollab Space Manager.

For creating and managing spaces architecture for Space Manager has been presented and UI interface based on UbiCollab eWorkbench has also been provided for easy use of Space Manager. Space Manager also provides the API for other components of UbiCollab. A context model to represent contextual information has also been proposed, we have tried to keep this context model as simple and expressive as possible. For dealing with the location awareness aspect of context-awareness architecture for location sensing technologies has also been presented, whose main goal is to convert low-level sensor reading into common and understandable high-level data that can be used by Space Manager to add location information to the context.

4.1 Solution Overview

The purpose of the system presented here is to give users of UbiCollab the ability to create and manage spaces. Since every space also represents the current context user is in, it must also provide a way to associate contextual information with every space. Just like any other ubiquitous system, in UbiCollab context information can be gathered from multitude of resources taking part in UbiCollab. In order to avoid excessive and unnecessary information, we have presented a context model which is very simple yet expressive; this context model works as a blueprint for the information that every space must capture.

Next section will describe our proposed context model in more detail; we will continue this chapter by describing Space Manager, Location Service Manager, Space database and GUI respectively. We will wind up this chapter by combining all of the proposed solutions in overall system structure section that will also relate our work with the current UbiCollab system.

4.2 Context Model

The context model we are going to present here is inspired from the work presented in [11]. The main idea behind this model is to represent world's facts in simple and expressive way and they must be structured in a standard way to be used easily. In ubiquitous computing the main aspect of context are: who is the user, what he is doing, where he is and what the resources around him.

Our context model takes care of these aspects by using a very simple three field structure: Who, What and Where.

Who: This field of context model represents the subject of the fact; in our subject it is the user of UbiNode.

What: This field represents the performed activity such as “work” or “relax”. In our case this information is provided by user himself.

Where: This field relates the location to the fact; in our project location information is gathered through different location sensors.

4.2 Space Manager

4.2.1 Introduction

Space Manager is based on the idea presented in previous works done on UbiCollab; see [1]. The idea is to benefits the UbiCollab users and applications to use and provide information that is only relevant to the current context and this current context is being identified by the current space user is in. This makes the context and space interchangeable in the context of UbiCollab. Therefore, our proposed structure of space must take into account not only the information that are relevant to space but also the information that represents the fact that happen inside the space.

4.2.2 Space

Figure 4-1 depicts the two different but typical spaces in the context of UbiCollab. One is marked as Home and other is marked as Office. Although both of them are geographical collocated and are totally different from each other but they both contains resources that user can use to perform some activities or to use them as the resources for collaboration.

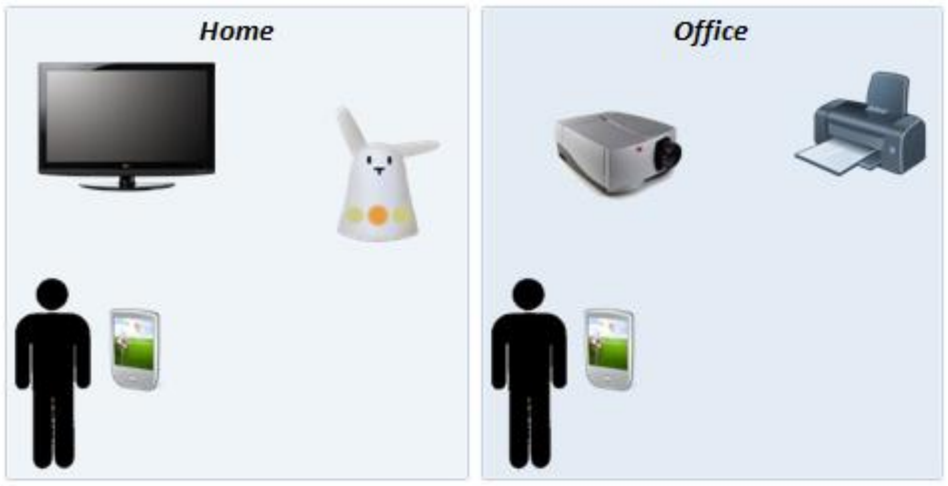


Figure 4-1 UbiCollab user in two different Spaces

User's presence in either space tells more than the current location of user, it tells about the fact like who is the user, what he is doing, where he is and what are the available resources around him. Simply the current context user is in.

To capture all these important details for determining the context we have proposed the structure of UbiCollab space similar to the context model described in previous section. Table 4-1 Structure of Space in UbiCollab shows the different fields UbiCollab space contains, their relation with the context model and their explanation.

Space field	Explanation	Context model field
Space ID	*A unique identifier to make distinction among multiple spaces.*	N/A
Name	*Name of space, this field can also serve as the purpose of space. For example space marked with "Ubiquitous Lab" says that user is in lab working on ubiquitous computing.*	What
Owner	*Field contains information about the user who owns this space.*	Who
Location Information	*Information about the physical location that this space represents in form of location sensor readings. More information about location sensors is given in next section.*	Where
Description	*Description about the space.*	N/A
Parent Space ID	*A unique identifier of space that contains this space.*	N/A
Date Created	*Date and time when the space was created by the user.*	N/A
Date Last Used	*Date and time when the space was last set as current space.*	N/A
Date Set	*Date and time when the space is set as current.*	When

Table 4-1 Structure of Space in UbiCollab

The structure of space defined above tells almost everything about the space but you would have noticed that it doesn't tell anything about the resource available within space. This is because it's the responsibility of Resource Discovery Manager of UbiCollab to keep track of resources available within particular space. When discovering the new resource Resource Manager can query Space Manager to get information about current space of user.

4.2.3 Space Manager API

In this section we will describe the basic functionalities that Space Manager will provide to deal with the spaces and the API it exposes for other components of UbiCollab. All of the API methods that described here are implemented as a part of our project work and described in greater detail in the next chapter.

CreateSpace: This method is used to create new space with the details provided as arguments. On successful creation of new space the unique identifier is returned.

SetSpace: This method is used to set a particular space as a current space. This can either be done manually by the user or Space Manager can automatically do this on the behalf of user.

GetSpace: Get Space method returns the Space in an xml format. Either the unique id of space or the location information can be provided to find the space and return.

GetCurrentSpace: This method returns space that was set as current space in an xml format.

DownloadSpace: Download method downloads and saved the method in space database from the provided URL as an argument. The URL must be to the valid XML definition of Space.

4.3 Location Service Manager

Location awareness is an importance aspect of context aware computing and also plays an important role in the context of our work. Each space in UbiCollab can be associated to some physical location; therefore Space Manager must provide a way to detect the location user is in. Today there are vast numbers of location sensing technologies are available but the problem is that different technologies give location information in different format. Also some of them provide information without any user interaction such as Wi-Fi or GPS sensors and for some of them user’s interaction is required such as RFID tags.

To deal with such a situation we have proposed a solution that transforms the information gathered from different location sensors into to common and understandable format. This solution of ours also provides a way for other UbiCollab resources to be used as positioning reference without any extensive coding, they just have to implement an interface and provide some basic information. This basic information can be anything that can be used to distinguish among different contexts.

Figure 4-2 shows the overview of our proposed architecture for fusing location information from different location sensors into common format.

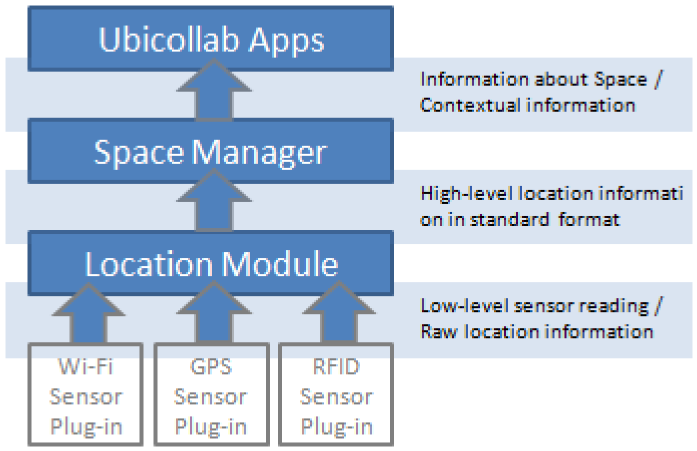

Ubicollab Apps
Information about Space / Contextual information
Space Manager
High-level location informati on in standard format
Location Module
Low-level sensor reading / Raw location information
Wi-Fi Sensor Plug-in
GPS Sensor Plug-in
RFID Sensor Plug-in

Figure 4-2 Overview of Location module

The reason behind the idea of keeping location module separate to Space Manager is ensure the loose coupling between the location sensors and Space Manager so that Space Manager can still work if there is not sensor plug-in is present on one hand and on the other hand any resource can work as a positioning reference without having any information about Space Manager.

Type of Location Sensors

Based on such a nature of locations sensors that some requires interaction and some don't, we have divided sensors in two different types: Non-observable sensors and Observable sensors.

Table 4-2 below shows the difference between types of sensors and also lists some examples.

Non-observable Sensor	Observable Sensors
1. They require user's interaction to get location information	1. They can provide location information with or without any interaction from the user
2. They can't be monitored periodically	2. Their readings can be taken on periodic basis
3. Example: RFID tags	3. Example: Wi-Fi, GPS

Table 4-2 Difference between Non-observable and Observable Sensors

Every location sensor will be considered as a non-observable sensor unless and until it implements the interface for observable sensors, which is just a extension of location sensor interface providing a little more information about the readings sensor provides about the location.

Making this separation between location sensors makes it possible for Space Manager to take certain decision on the behalf of user automatically by sensing the location where possible.

4.4 Space Database

In order to save and use spaces, they need to be stored either in some file or in database. Besides saving information about space we will also be saving location information that we get from different location sensors, therefore making it necessary that the data structure for storing this information must be very generic. SQL database seems to be the natural choice for achieving this. Also, for extracting stored information of this kind SQL queries will play an important role and will make things easier. Figure 4-3 shows the ER diagram of space database.

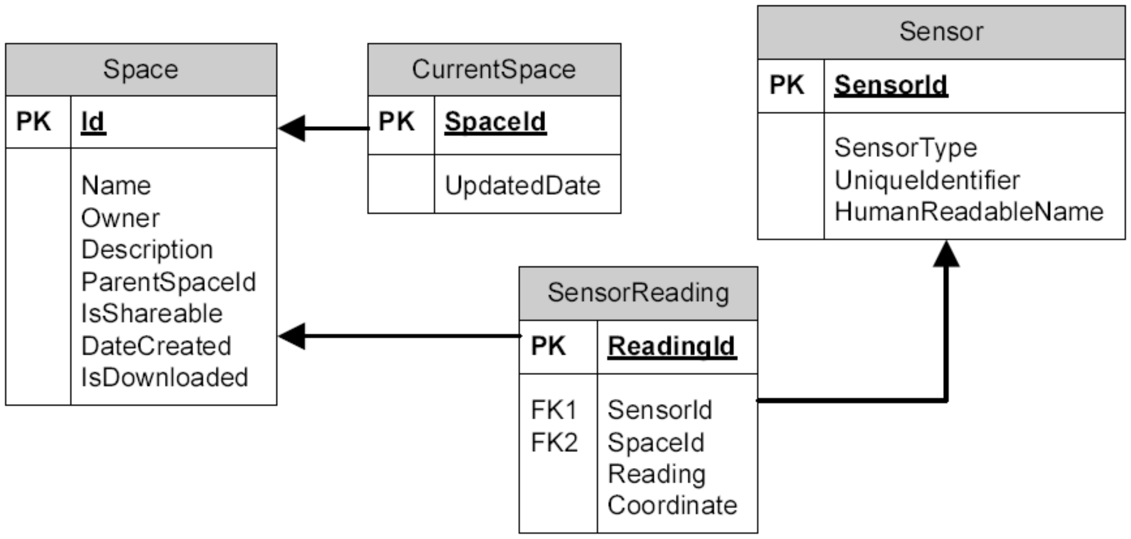


Figure 4-3 ER diagram of Space database

Space table represents the space user has created using Space Manager and this table stores all information about space except, information about location associated with it. Information about location is saved in form of information about location sensor and their readings and is stored in table *Sensor* and their readings are saved in a separate table called *SensorReading*. Location Service Manager converts the information that it gets from location sensor into standard format before inserting them into Sensor and SensorReading table. *CurrentSpace* table stores the information about current space and it will have only one record at a time.

The table below shows the sample data in the database tables described above.

ID	Name	Owner	Description	ParentSpaceId	IsShareable	DateCreated	IsDownloaded
001	My Home	Alex	This space is my home	-1	False	03/04/2010 12:23:00 PM	False
002	Gym	Alex	My GYM	-1	True	05/04/2010 08:00:01 PM	False

Table 4-3 Database table representing Space

SensorId	SensorType	UniqueIdentifier	HumanReadableName
001	Wi-Fi	01:1e:e5:64:f1:15	Linksys
002	Wi-Fi	01:0b:85:89:b9:fd	Ntnu
003	Wi-Fi	01:0b:85:89:b9:ff	ntnuguest
004	GPS	GPSSensor01	N/A

Table 4-4 Database table representing Sensor

ReadingId	**SensorId**	**SpaceId**	**Reading**	**Coordinate**
001	001	1	-36	NULL
002	002	1	-62	NULL
003	003	1	-62	NULL
004	004	2	NULL	49.951220, 2.197266

Table 4-5 Database table representing Sensor Readings

Sensor and SensorReading tables are generic in nature and can store information gathered from different kind of location sensors in same format. Every location sensor will be added only once to the database and if in case it is found again during creating a new space or updating an existing space, only the new record in the readings table will be added.

4.4 GUI Mockups

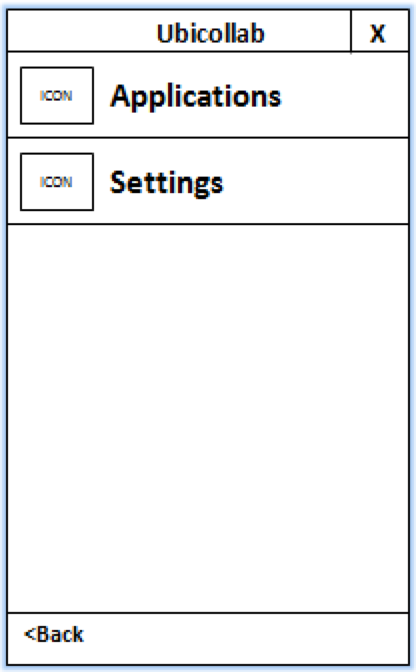
This section will present mockups of intended graphical user interface (GUI) of Space Manager. It should be noted that they are mockups and do not represent the look of final GUI but they are made to illustrate what kind of functionalities the various GUI screen will offer.

- ***Use Case 1: Starting Space Manager***

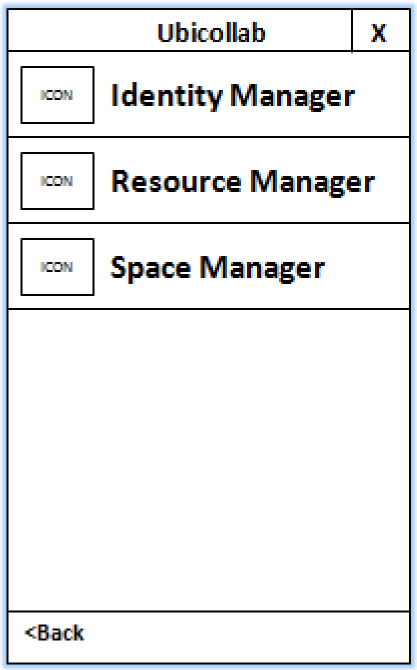
  Goal: User wants to start Space Manager

  Steps to follow:

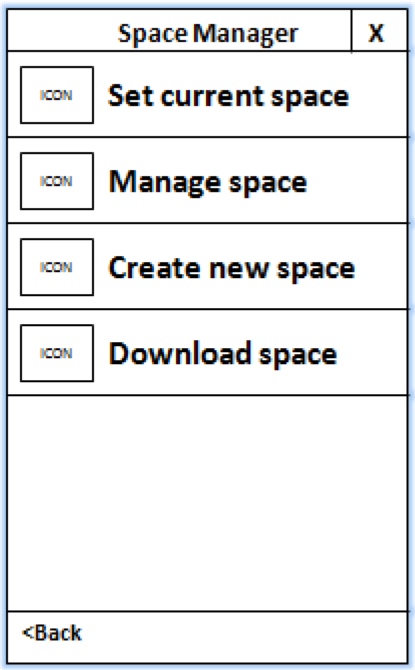
  1. User starts UbiCollab on his UbiNode
  2. User taps on Settings option
  3. Now user taps on Space Manager to start the Space Manager interface



1. Ubicollab interface



2. Settings interface



3. Space Manager interface

Figure 4-4 Use Case 1 GUI Mockups

- ***Use Case 2: Create New Space***

Goal: User wants to create a new space

Steps to follow:

1. User starts Space Manager Interface as illustrated in use case 1
2. User taps on create new space option
3. User fills in the required information about the new space he is creating
7. User taps on create button to create a new space saving the information to space database

Optional steps:

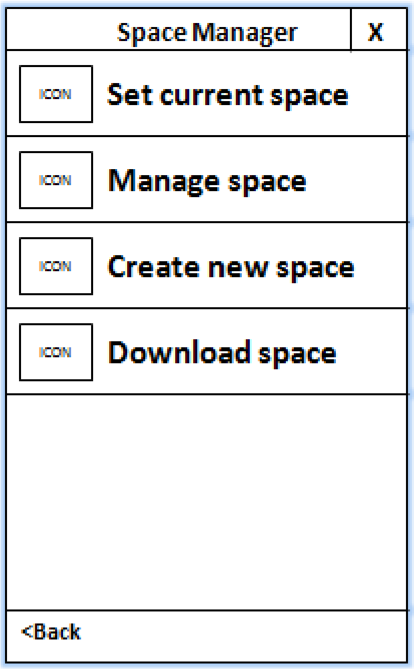
To associate location information to the space

4. User taps on Position Me button
5. Space Manager tries to get readings from available location sensors

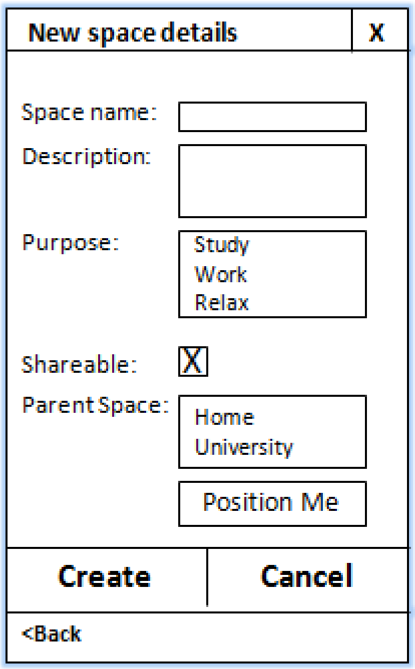
5a. Space Manager saves the location sensor reading and displays the successful positioning message

5b. Space Manager doesn't find any location sensor that can be read automatically. It then asks user to help in positioning by either reading the RFID tag or by punching a number to the resource that he wants to use as location reference.

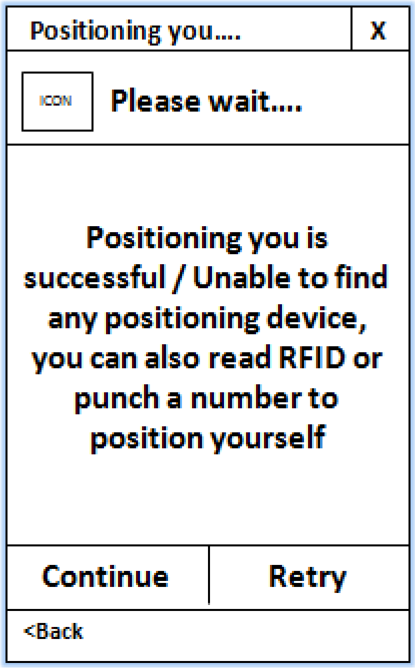
6. User taps continue to return to screen 3



2. Functionalities of Space Manager

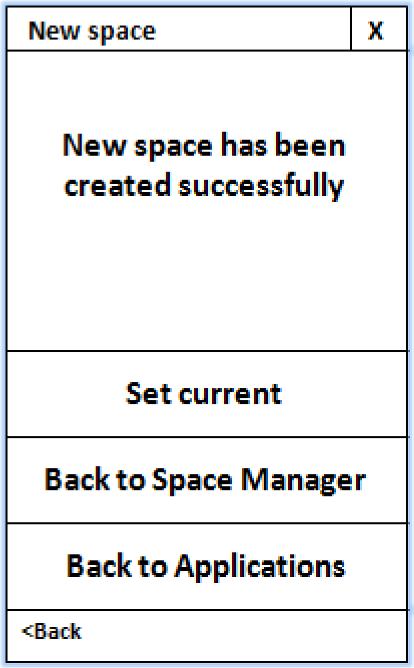


3. Details about new space



Positioning you is successful / Unable to find any positioning device, you can also read RFID or punch a number to position yourself

4, 5, 6. Position user using available location sensing plug-in



New space has been created successfully

7. New space created screen

Figure 4-5 Use Case 2 GUI Mockups

- *Use Case 3: Set Space*

Goal: User wants to create a new space

Steps to follow:

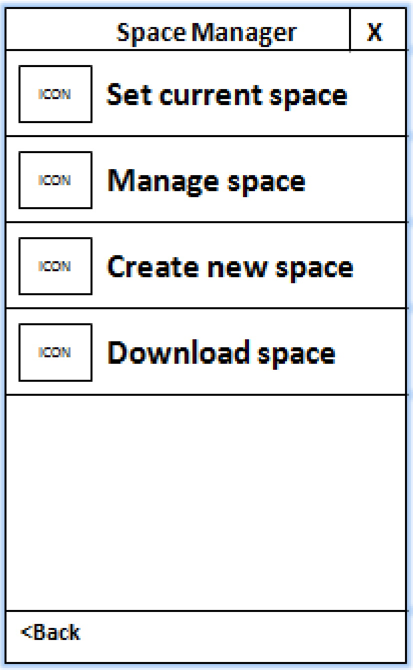
1. User starts Space Manager Interface as illustrated in first scenario
2. User taps on set current space
3. Space Manager tries to get readings from available location sensors to find matching space

4a. Space Manager founds the space and set it as current space automatically and displays the confirmation message to the user

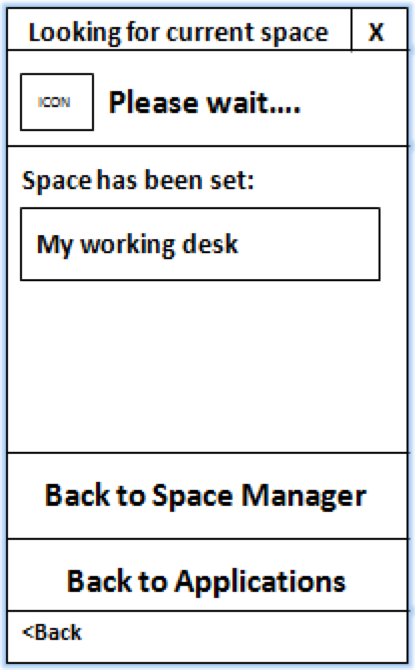
4b. Space Manager founds more than one space associated with the sensed location and displays them as a choice for user to set the most appropriate one

4c. Space Manager unable to find any space automatically therefore displays all the spaces user have in his space database as choice for user to set the current space he wants to

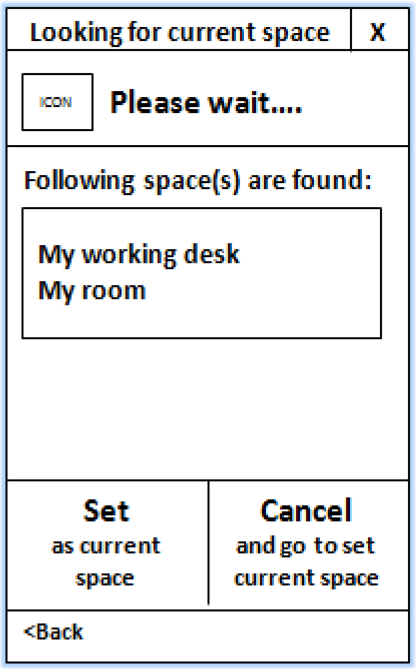
4d. Space Manager unable to find



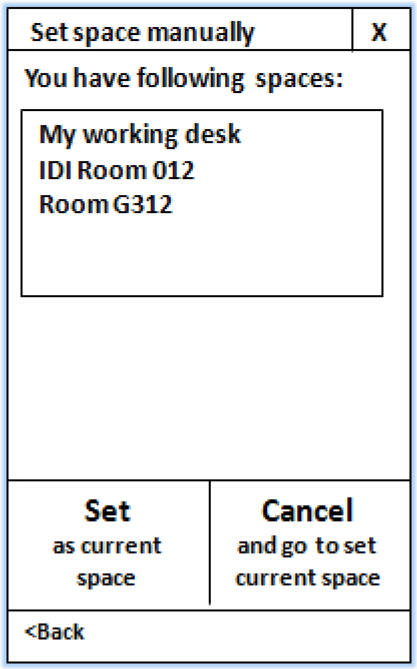
2. Functionalities of Space Manager



4a. Found space automatically


Looking for current space
X
ICON
Please wait....
Following space(s) are found:
My working desk
My room
Set
as current space
Cancel
and go to set current space
<Back

4b. Found more than one space



4c. Displays all spaces to set manually

Figure 4-6 Use Case 3 GUI Mockups

- ***Use Case 4: Download Space***

  Goal: User wants to download already created space to his UbiNode

  Steps to follow:

  1. User starts Space Manager Interface as illustrated in use case 1
  2. User taps on Download space
  3. Download space interface with three options to choose

  Option 1

  4a. User taps on Enter URL

  User enters the URL to xml file that contains the space information and tap on download

  Option 2

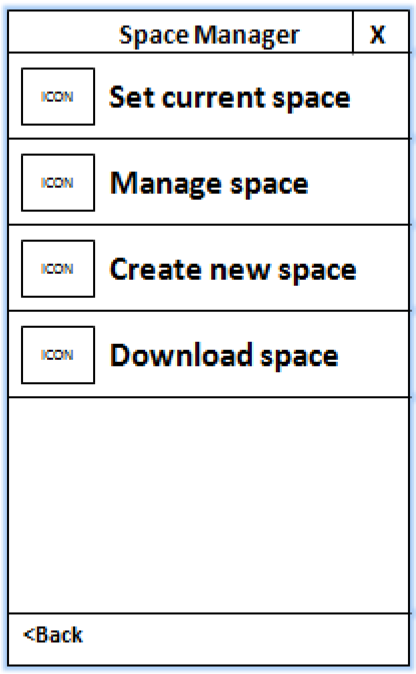
  4b. User taps on via Bluetooth

  User receives space xml file via Bluetooth from another user

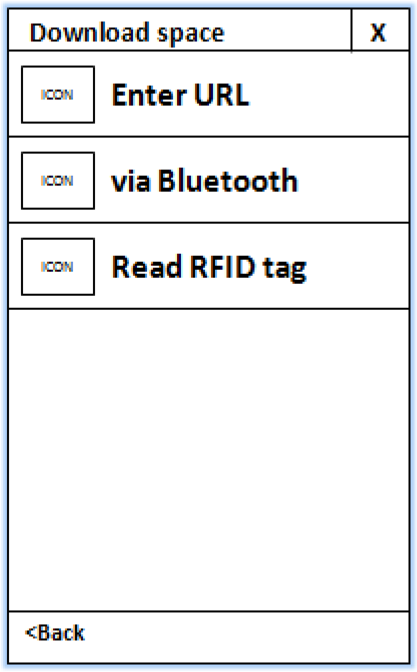
  Option 3

4c. User taps on Read RFID tag

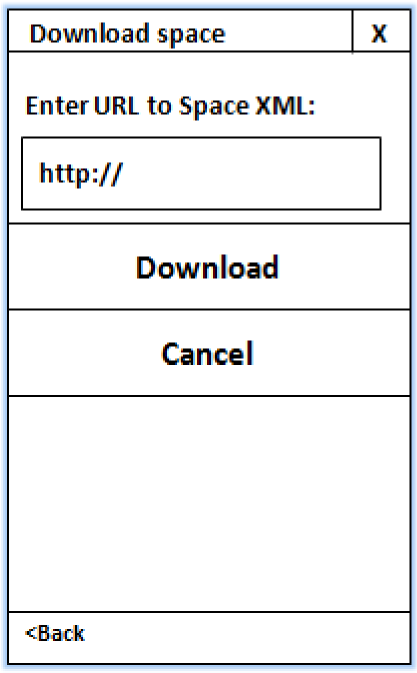
User reads an RFID tag with the help of tag reader and downloads the space xml from the associated URL



2. Functionalities of Space Manager



3. Download space Interface



4a. Download space xml from URL

Figure 4-7 Use Case 4 GUI Mockups

4.5 Overall system structure

Figure 4-8 shows where everything that we have discussed in previous section of this chapter fits in.

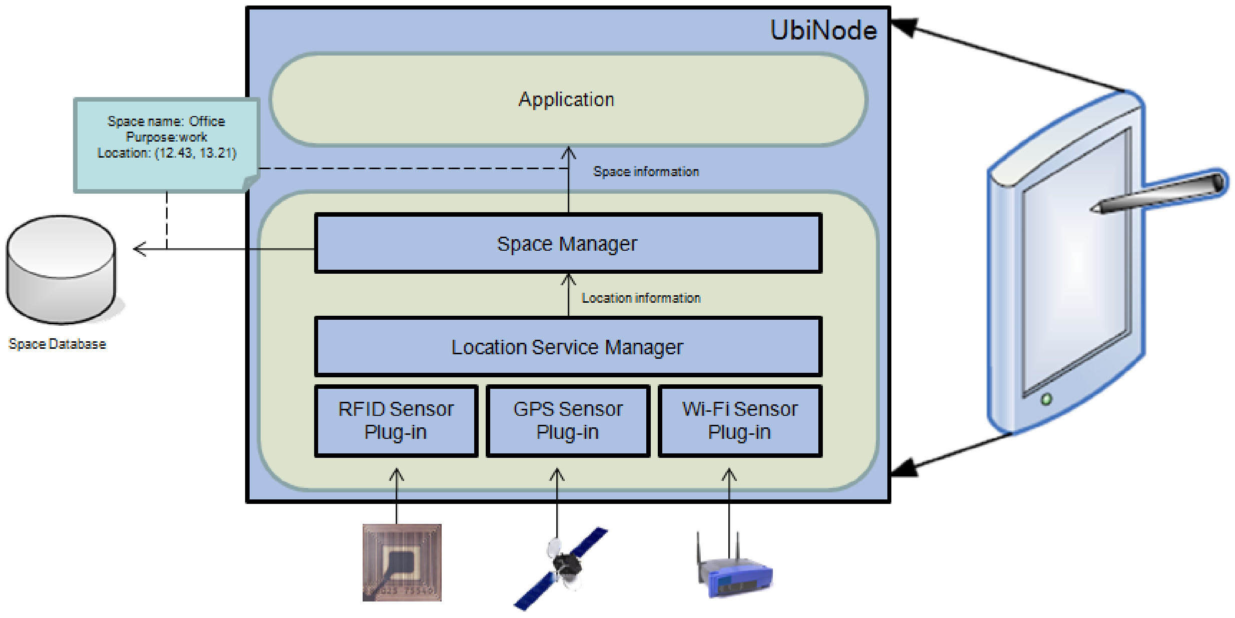


Figure 4-8 Overall system architecture

Chapter 5 Implementation

This chapter deals with the implementation done as a part of this project work. Although the main aim of this project is to propose and implement the architecture for Space Manager but we will start this chapter by getting into the details of Location Service Manager's implementation. Location sensing is an optional but an important aspect of Space Manager and in order to describe the implementation of Space Manager it is first necessary to get introduced with Location Service Manager that provides ways to sense the location using different location sensing technologies.

Therefore, we will start this chapter by going through in details about the tools and technologies that we have used during the course of implementation, then continue this chapter by describing the detailed architecture of Location Service Manager, different interfaces and classes it implements, implementation details of Wi-Fi Sensor Plug-in using Location Service Manager and how any new or existing component of UbiCollab can be used as a positioning reference. Then we will end this chapter by describing the implementation of Space Manager and how it fulfills all the user requirements that were described in requirements section 3.4 of chapter 3.

5.1 Tools and Technologies

In this section of implementation chapter we will describe in more detail about the tools and technologies that are used during the course of this project. UbiCollab being an open source platform makes it necessary in a way to use open source tools and technologies only.

5.1.1 Java Mobile Edition

Java mobile edition is a subset of Java Platform from Sun, providing the limited set of runtime and API for creating applications for small devices such as mobile phones and PDA. Although Java ME run on Java virtual machine like the other Java editions but its works totally different way as compare to other Java editions. In order to support wide number of devices, Java ME consists of configurations, profiles and options packages.

There are two configurations for Java ME that are listed below:

- **CLDC** stands for Connected Limited Device Configuration, this configuration is to support less powerful device with limited set of functionalities.

- **CDC** stand**s** for Connected Device Configuration and is to support more powerful devices such as Smart Phones and PDAs.

5.1.2 OSGi

In order to support modularization in system development, simple Java language is not enough. Due to the flat classpath structure of coding in Java and the absence of dependency management fails it to fit properly to the requirement of Service-Oriented Architecture (SOA).

OSGi solves this problem by providing the framework for modular development in Java. "It defines a way to create true modules and a way for those modules to interact at runtime." [12] OSGi creates modules known as bundles, which are jar files with some additional manifest. This module system is dynamic in nature and bundles can be installed, updated and uninstalled without taking down the entire application. These modules are decoupled through service interface. This service interface wired bundles in a dynamic way.

Currently there are different implementations of OSGi, such as Equinox, Knopflesfish, Felix and Concierge.

The current version of UbiCollab is using the Equinox implementation of OSGi and is also the most widely deployed OSGi framework today [12].

5.1.3 eRCP / eSWT

eRCP stands for Embedded Rich Client Platform for building and deploying rich client applications on embedded device such as mobile phones. This is an extension to Eclipse Rich Client Platform (RCP). eRCP enables the same application model used on desktop machine to used on mobile devices [13]. eRCP is a set of components that are a subset of RCP components, applications that are developed for mobile phones using eRCP will automatically run on desktop platform.

eRCP consist of components [13]:

- Core runtime
- eSWT
- eJFace
- eWorkbench
- eUpdate

eSWT stands for Embedded Standard Widget Toolkit, is a technology to build native looking Graphical User Interface (GUI) for variety of mobile phones. It comes as a part of eRCP framework.

In previous work on UbiCollab [2], this toolkit is evaluated to be the best choice for creating native looking GUI.

5.1.4 HyperSQL Database

HyperSQL database or HSQLDB is an open source relational database engine written in Java. HSQLDB is very suitable for applications targeting resource limited devices such as UbiCollab, because it supports in-memory tables which results in fast data access and also gives more predictable performance. It also has JDBC driver and supports almost full ANSI-92 SQL [23].

Also, HSQLDB is being in many open source project as data storage such as Place Lab[1], OpenOffice.org Base[2] and others

[1] Place Lab: http://www.placelab.org
[2] OpenOffice.org Base: http://www.openoffice.org/product/base.html

5.1.5 Eclipse

Eclipse is free and open source development environment and is widely being used among open source software developers. It provides integrated development environment (IDE) which can be used to develop variety of applications ranging from console applications to fully fledged desktop applications and web applications to web services. Eclipse is not limited to develop only applications written in Java but by downloading different plug-ins software in other languages can also be written.

5.2 Location Service Manager

Today there are many different kinds of technologies available that helps in sensing the location user is in, some of them work outdoors and some of them work indoors. In the context of UbiCollab we cannot restrict ours self to any particular kind of location sensing technology because user can use his UbiNode anywhere and can create different spaces outdoors and indoors. To solve this problem of heterogeneity and to support almost any location sensing technology we proposes Location Service Manager, which provides interfaces and classes to abstract away the hardware differences and to transform low-level location sensor information into high-level, readable and standard format.

Coming sections describe implementation of Location Service Manager in greater detail and also provide how-to-use guideline for the developers.

5.2.1 Components of Location Service Manager

In this section we will describe the components of Location Service Manager in greater detail, which are Sensors, Sensor Reading and Coordinate.

5.2.1.1 Sensor

In the context of our project, sensor or location sensor (we will use location sensor instead of sensor to avoid any ambiguity) is a device with the ability to sense physical location and returns location information in readable format. Some of them represent location in form of geographical or logical coordinates and the location information they provide is sufficient enough to represent any physical location such as Global Positioning System (GPS) sensor, some of them provide a way to tag locations and then differentiate locations based on the tag value such as RFID tag readers and some of them are such kind of sensors that don’t return anything about location but the information they return can be used to calculate their position and this position can then be used to represent the physical location, devices that emits radio signal lies in this category and strength of signal measurement to triangulate position technique is used to calculate the location.

Since there are many different kind of location sensors are available, they work in different ways and provides location information in different formats. The basic idea here is to provide a common and standard interface for the hardware components that are used for sensing locations. Our work provides couple of interface definitions that helps in abstracting away how sensor works and provides location information.

To fully utilize the nature difference of location sensors we have divided them in different classes. Figure 5-1 shows the UML class diagram of those interfaces and classes.

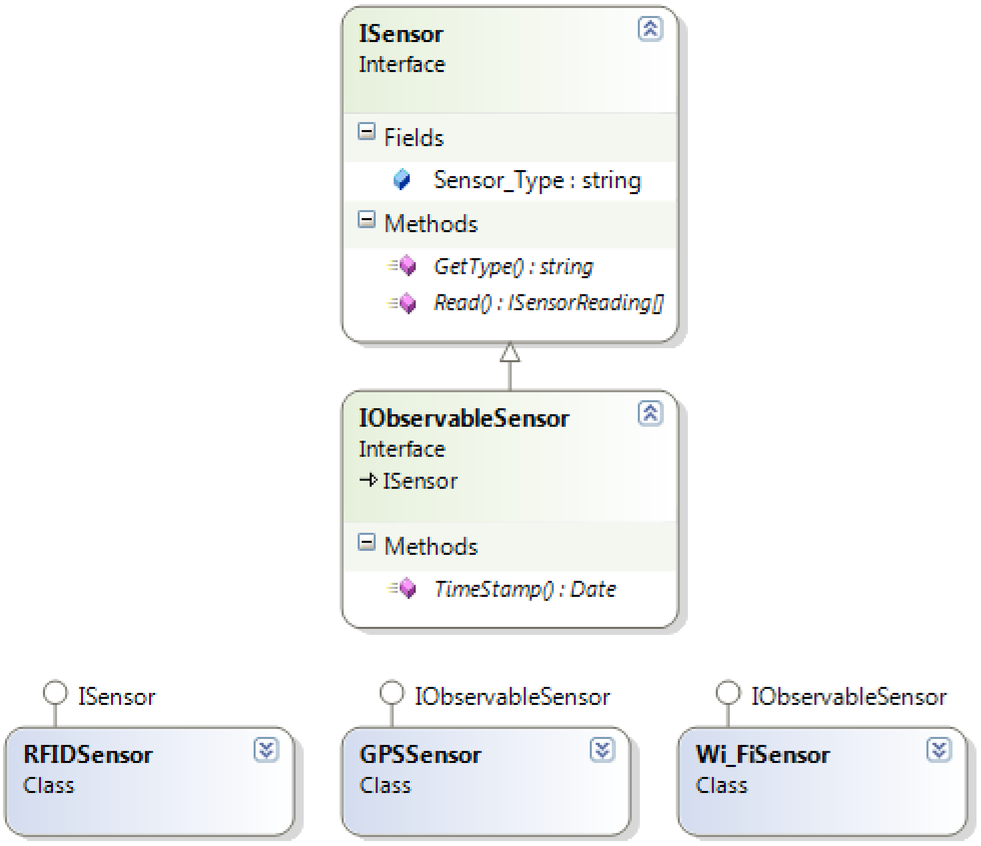


Figure 5-1 Location Sensor interfaces and sample classes

Figure 5-1 above shows the location sensor interfaces and classes in hierarchical manner, *ISensor* interface being on the top of hierarchy provide the generic definition of location sensor and it works as a base interface. This interface also exposes the generic methods that every location sensor must implements in order to be used as a location sensor. *IObserveableSensor* interface extends the base interface *ISensor* to provide the generic definition for all such kind of location sensors that can be read with or without user's interaction. For example Wi-Fi or GPS location sensors that can be read without user interaction, as a part of our work we have implemented Wi-Fi location sensor whose implementation will be described later in this chapter.

RFID Sensor represents the concrete class that implements *ISensor* interface and provides a way to use RFID tags as a location reference. *Wi-Fi Sensor* and *GPS Sensor* classes implement *IObservableSensor* interface make it possible to use Wi-Fi access point signals and GPS coordinates for sensing location respectively.

Given below are the details about each interface and class defined above.

ISensor

Interface for abstracting away the hardware differences of location sensors. This interface also provides API for querying information about location sensor itself and information about location it senses.

Members

`public static final String Sensor_Type`
Character string that represents the type of location sensor.

Methods

`public String GetType()`
Gets the type of location sensor. The return value must be one of them ***WiFi, GPS, RFID, Bluetooth*** *or* ***PositionCoordinate***.

`public ISensorReading[] Read()`
Gets the actual sensor reading. Every location sensor must use this method to returns the location information in form of collection of **ISensorReading**.

IObservableSensor

Extends **ISensor** interface. Interface for those location sensors that can be read with or without user interaction.

Methods

`public String GetType()`
Gets the type of location sensor. The return value must be one of them ***WiFi, GPS, RFID, Bluetooth*** *or* ***PositionCoordinate***.

`public ISensorReading[] Read()`
Gets the actual sensor reading. Every location sensor must use this method to returns the location information in form of collection of **ISensorReading**.

`public Date TimeStamp()`
Get the date and time when location sensor reading is being taken.

5.2.1.2 Sensor Reading

Sensor reading can be defined as location information that any sensor returns, it can be GPS coordinates, RFID tag value or signal strength value. After abstracting away the location sensing hardware using sensor interfaces and classes, the next step is to provide an interface for standardizing the location information that different location sensors return. Without converting the information that different location sensors return to a common format makes it a requirement to write separate piece of code for different type of location information, killing the basic idea of code reusability of Location Service Manager and as minimum coding as possible of UbiCollab. To transform the location information into common format a UML class diagram of interface and a class that has been implemented are shown in Figure 5-2

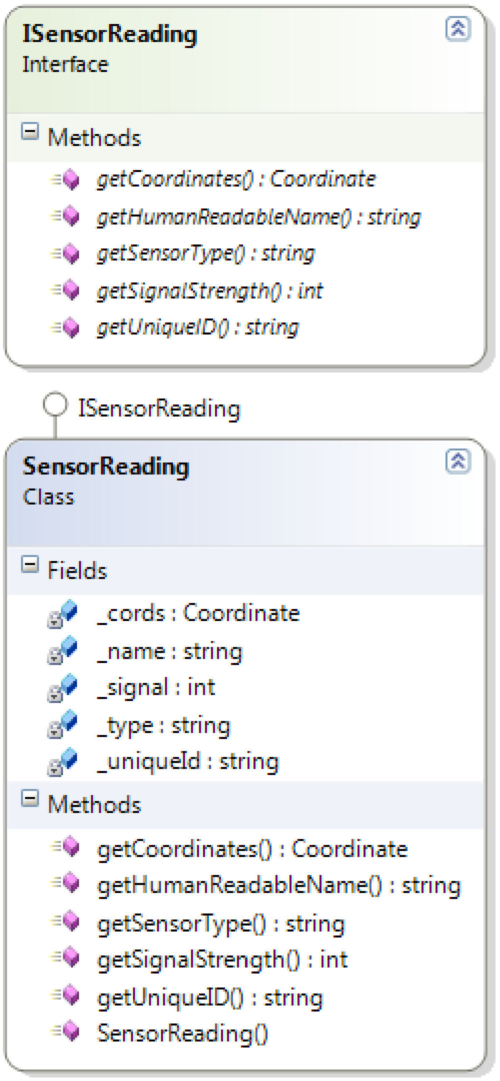


Figure 5-2 Interface and class for standardizing sensor reading

ISensorReading interface provides basic and abstract definition of sensor reading that every location sensor must implement for the reading that it gives. Abstract methods exposed by this interface provide a way to use sensor reading.

SensorReading is a concrete class that implements *ISensorReading* interface and is used to convert the object that implements *ISensorReading* interface to more concrete form. By converting it to more concrete form it makes it possible to be treated as a normal object.

Given below are the details about the *ISensorReading* interface and *SensorReading* class members and methods.

ISensorReading

Provide a way to convert location information from different sensor reading to standard format.

Methods

`public String getHumanReadableName()`
Gets the character string that represents the human readable name of location sensor.

`public String getUniqueID()`
Gets the unique identifier that distinguish location sensor from other location sensors.

`public int getSignalStrength()`
Gets the actual sensor reading for such location sensors that works on radio signals.

`public String getSensorType()`
Gets the type of location sensor in character string format. The value returns must be one of them ***WiFi, GPS, RFID, Bluetooth*** *or* ***PositionCoordinate***.

`public Coordinate getCoordinates()`
Gets the object of `Coordinate` class. Every sensor that returns location information in form of coordinate system must instantiate the object of `Coordinate` class using the coordinate information it has. For all other location sensors this method returns `null`.

SensorReading

Provide a concrete class for defining the location sensor reading. It is used to convert the object of **`ISensorReading`** interface to more concrete form.

Constructor

```
public SensorReading(String UniqueId, String Name, int Signal,
Coordinate Cord, String SensorType)
```
Initialize a new instance of **`SensorReading`** class by using the information about sensor reading specified as arguments.

Methods

`public String getHumanReadableName()`
Gets the character string that represents the human readable name of location sensor.

`public String getUniqueID()`
Gets the unique identifier that distinguish location sensor from other location sensors.

`public int getSignalStrength()`
Gets the actual sensor reading for such location sensors that works on radio signals.

`public String getSensorType()`
Gets the type of location sensor in character string format. The value returns must be one of them ***WiFi, GPS, RFID, Bluetooth*** *or* ***PositionCoordinate***.

`public Coordinate getCoordinates()`
Gets the object of `Coordinate` class. Every sensor that returns location information in form of coordinate system must instantiate the object of `Coordinate` class using the coordinate information it has. For all other location sensors this method returns `null`.

5.2.1.3 Coordinate

Coordinate class is a concrete class that provides the definition for a class that represents 2D coordinates. Any location sensor that provides location information inform of coordinate such as GPS Sensor must returns the object of *Coordinate* class.

Given below are the details about method of **`Coordinate`** class.

Coordinate

Represents a point in a two dimensional coordinate system.

Constructors

`public Coordinate()`
Initializes a new instance of **`Coordinate`** class by using the default values.

`public Coordinate(double X, double Y)`
Initializes a new instance of **`Coordinate`** class by using the values of X and Y coordinates specified as arguments.

Methods

`public double getX()`
Gets the value of X coordinate.

`public double getY()`
Gets the value of Y coordinate.

```
public String toString()
```
Gets both X and Y coordinate in X,Y format.

5.2.2 Wi-Fi Sensor Plug-in

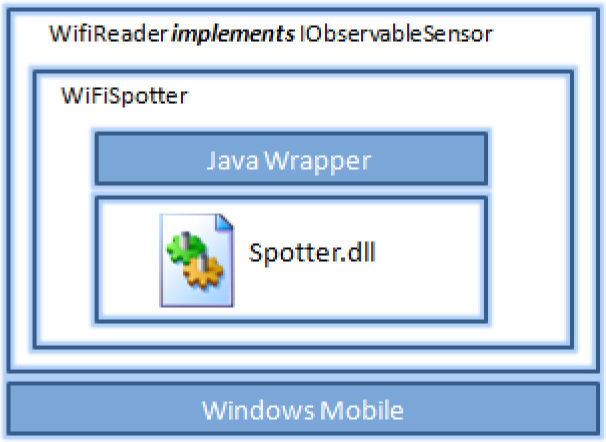
As a part of our project work we have also implemented Wi-Fi Sensor Plug-in that is based on the interfaces and classes provided by Location Service Manager and were described in detail in previous section.

The aim of this work is to proof the concept presented as Location Service Manager and also to show how the interfaces and methods can be used.

In order to get details about all the Wi-Fi access points that are visible to an UbiNode, we need to access system level information. On Windows Mobile currently there is no API available for Java to access this information easily and JNI is the only way to wrap the system level method and make them accessible inside Java code.

Instead of reinventing the wheel and write code from scratch, we have downloaded the spotter.dll, that is written in C++, and its java wrapper from Place lab project's website, the downloaded code is released as an open source and available for download from the Place lab CVS repository[3]. The original spotter.dll was supposed to work when there is no more than 32 access points are available, since we don't require such a restriction we have modified the spotter.dll and now it can work for virtually any number of access points.

Figure 5-3 below shows the overall architecture of Wi-Fi sensor plug-in. Spotter.dll contains all the native code which query for all visible access points and gather information such as name of network (SSID), MAC address of access point (BSSID) and signal strength (RSSI). WiFiSpotter is a Java wrapper for Spotter.dll and is responsible for converting all the information that Spotter.dll returns to the java data structure.



[3] Place Lab CVS repository: http://www.placelab.org/toolkit/doc/cvsdeveloper.php

Figure 5-3 Overall Architecture of Wi-Fi Sensor Plug-in

First of all, in order to use Wi-Fi access point information as location information we need to relate it with the standard format of location information that we have defined in previous chapter. Table below relates the Wi-Fi access point specific information to the location sensor information and shows how the specific information of location sensor can be converted to common format that can then be used inside Space Manager.

Wi-Fi access point information	Description
BSSID	Stands for Basic Service Set Identifier and represents MAC address of access point. No two access point can have similar MAC addresses. This field is equivalent to the Unique ID of location sensor.
SSID	Stands for Service Set Identifier and represents the name of wireless local area network. This field is equivalent to the Human Readable field of location sensor.
RSSI	Stands for Received Signal Strength Identification and represents the power of radio signal. This field is equivalent to the Received Signal Strength of location Sensor.

Table 5-1 Wi-Fi access point information

WifiReading class captures all this information about every single Wi-Fi access point our Wi-Fi Sensor Plug-in sees. This class not only implements all the required methods of *ISensorReading* interface but also exposes some Wi-Fi Sensor specific methods such as methods for reading MAC address, network name and signal strength. Figure 5-4 shows the UML class diagram of *WifiReading* along with the other classes of Wi-Fi Sensor Plug-in.

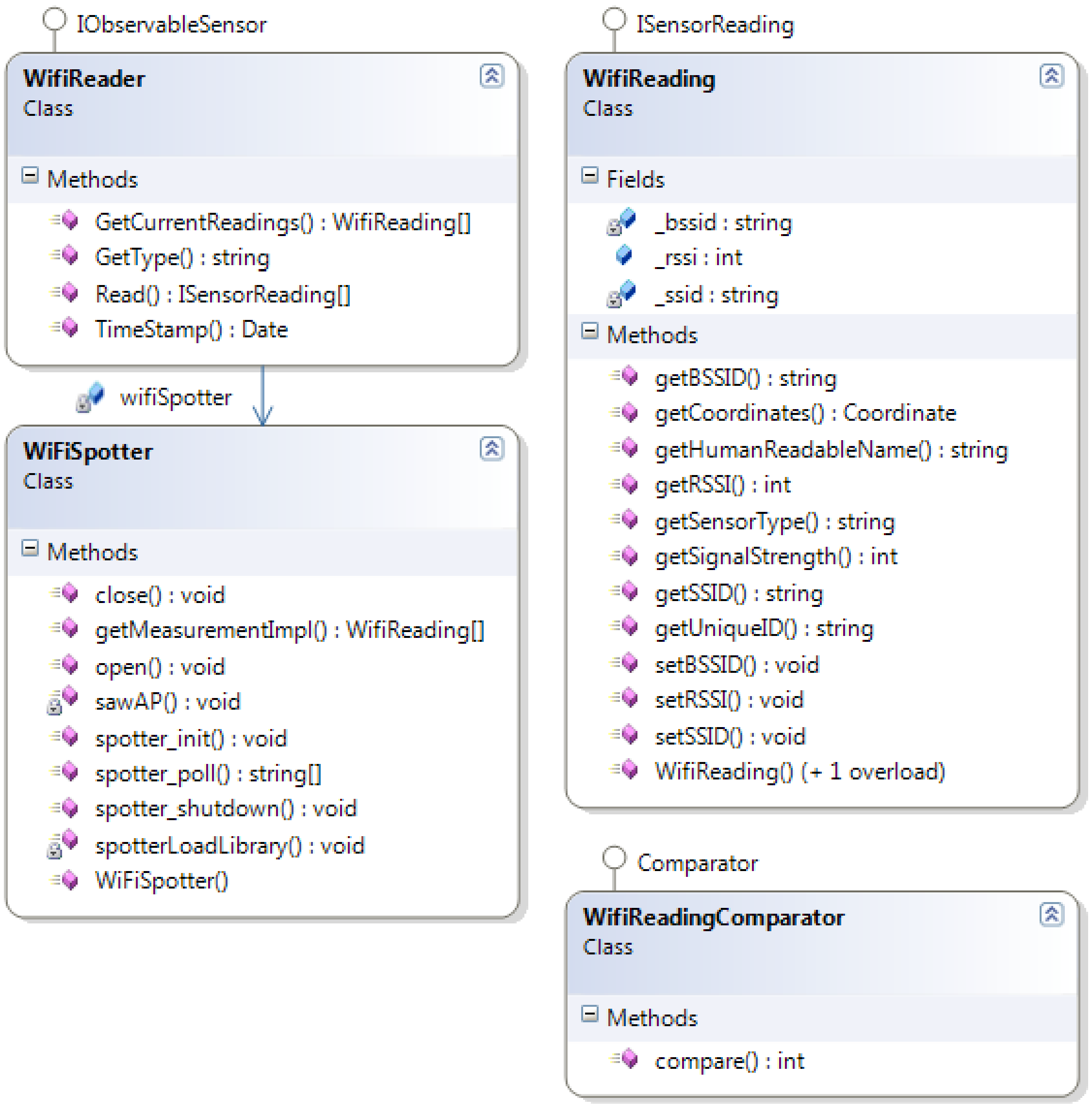


Figure 5-4 UML class diagram of Wi-Fi Sensor Plug-in

WifiReader is a class of type IObservableSensor which contains an object of *WiFiSpotter* class and exposes the methods that every class of type *IObservableSensor* must do. Read is the main method that is being called from any UbiCollab component who wants to get information about current visible access points. Whenever the Read method of *WifiReader* is invoked, it calls the *GetCurrentReadings* which in turn calls the *getMeasurementIMpl* method of *WiFiSpotter* and returns the collection of all *WiFiReading* it sees. Figure 5-5 shows the sequence diagram of *Read* method.

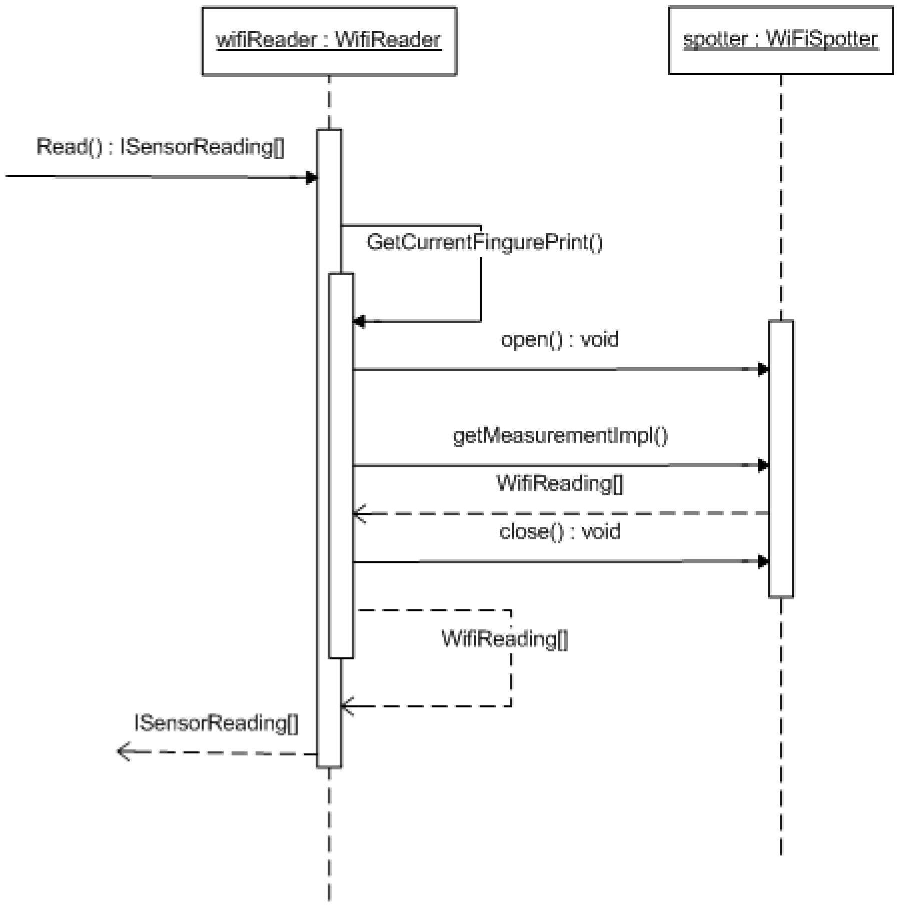


Figure 5-5 Sequence diagram of Wi-Fi Sensor Plug-in Read method

getMeasurementImpl method of *WiFiSpotter* class returns information about all visible access point in form of *WifiReading* collection to *GetCurrentFingurePrint* method of WifiReader. For Wi-Fi figure print we need readings from at most three access points, *GetCurrentFingurePrint* method filter out first three readings according to the signal strength and sort them in ascending order with respect to the signal strength using *WifiReadingComparator* class. This collection of *WifiReading* is then returned to Read method and subsequently to the caller.

5.2.3 Any resource can work as location sensor

Another benefit of our proposed Location Service Manager is that any resource that can be used in UbiCollab can also be used as a location sensor without any extensive coding or effort. Implementing the *ISensor* and *ISensorReading* interfaces by the service proxies of those resources is the only requirement. To make this clearer, we will take two examples into account: an example of printer and of thermometer.

5.2.3.1 Printer as location sensor

Suppose there is a printer in the lab that can be used from UbiCollab either by punching the number or by reading the associated tag. Beside printing user wants to use it more than as a printing device; he wants UbiCollab to determine the space he has created for the lab when he uses printer, in other term he wants to use printer as positioning reference.

It can be done using Location Service Manager if printer somehow knows the x and y coordinates of the place or room it is situated in. For the sake of simplicity, here we assume that the time when user installs the proxy service for printer by either punching the number or reading the tag associated to printer, it also calculates the x and y coordinate of printer and saves it in place where it can be accessed using proxy service class. Let say we have *GetX* and *GetY* methods exposed by proxy service for the printer. The code in Listing 5-1 shows a sample service proxy class.

```java
public class PrinterServiceProxy {
      //proxy service member....
      private String printerName;
      //......
      ...

      private int x;
      private int y;

      //Overloaded constructor
      public PrinterServiceProxy(int X, int Y){
            this.x = X;
            this.y = Y;
      }

      public int GetX(){
            return this.x;
      }

      public int GetY(){
            return this.y;
      }

      //proxy service methods...
      //......
      ...
}
```

Listing 5-1 Sample Proxy Service Class for Printer

Now as we can access the coordinate information associated to printer, we can use it as a positioning reference by using printer as a location sensor and this can then be used inside Space Manager to associate printer's coordinate to the user created space. Code in Listing 5-2 shows how easily and effortlessly it can be done.

```java
public class PrinterServiceProxy implements ISensor {
      //service proxy specific member....
      private String printerName;
      //......
      ...

      private int x;
      private int y;

```

```
    //Overloaded constructor
    public PrinterServiceProxy(int X, int Y){
        this.x = X;
        this.y = Y;
    }

    public int GetX(){
        return this.x;
    }

    public int GetY(){
        return this.y;
    }

    public String GetType() {
        return SensorType.PositionCoordinate;
    }

    public ISensorReading[] Read() {
        class PrinterCoordinateReading implements ISensorReading {

            @Override
            public Coordinate getCoordinates() {
                return new Coordinate(GetX(), GetY());
            }

            @Override
            public String getHumanReadableName() {
                return printerName;
            }

            @Override
            public String getSensorType() {
                return SensorType.PositionCoordinate;
            }

            @Override
            public int getSignalStrength() {
                return 0;
            }

            @Override
            public String getUniqueID() {
                return printerName;
            }
        }

        PrinterCoordinateReading[] reading =
            new PrinterCoordinateReading[1];
        reading[0] = new PrinterCoordinateReading();

        return reading;
    }

    //service proxy methods...
    //......
    ...
```

```
67.  }
```

Listing 5-2 Extended Service Proxy Class for Printer

Code give in Listing 5-2 shows how service proxy of printer can be extended to treat printer as a location sensor by implementing *ISensor* and *ISensorReading* interfaces.

Since printer working as location sensor returns location information in form of two dimensional coordinates, the value returned by *GetType* method of ISensor interface and *getSensorType* method of *ISensorReading* interface must be *SensorType.PositionCoordinate* so that Space Manager knows how to treat the information retrieved from such location sensors.

Read is the method that returns the location sensor reading and is of main interest for the Space Manager, it returns the collection of objects of type *ISensorReading*. In our case x and y coordinate comprises the reading of location sensor, so we need to wrap this information in *ISensorReading* interface. For location sensors that represent location information in form of two dimensional coordinate, Location Service Manager has implemented a class called *Coordinate* and is a part of every *ISensorReading* object. Since service proxy of printer already stores the x and y coordinate, we just have to override the *getCoordinate* method of *ISensorReading* by instantiating a new *Coordinate* object, fill it with x and y coordinate of printer's location and then use it as a return value, line number 32 to 34 of code Listing 5-2 shows how to do this. Line number 29 to 55 shows how to create a class on fly that conforms to *ISensorReading* and wraps the specific location information in standard format.

The next step is to register this extended service proxy class as a location sensor plug-in which is just another name of registering it as an OSGi service, so that it can be used by Space Manager or any other component of UbiCollab. Listing 5-3 shows the code for interacting with OSGi framework.

```
public class Activator implements BundleActivator {
      public void start(BundleContext context) throws Exception {
            ISensor PrinterSensor = new PrinterServiceProxy();

            Properties props = new Properties();
            props.put(ISensor.Sensor_Type,
                          SensorType.PositionCoordinate);

            context.registerService(ISensor.class.getName(),
                        PrinterSensor, props);

            System.out.println("" +
                        "Printer location sensor plugin has started");
      }

      public void stop(BundleContext context) throws Exception {
            String filter = "(&(sensorType="
                  + SensorType.PositionCoordinate + "))";

            ServiceReference[] refs =
                  context.getServiceReferences(
                              ISensor.class.getName(), filter);

            if( refs != null ){
```

```
25.                         for(int i=0; i<refs.length; i++){
26.                               PrinterServiceProxy printerSensor =
27.                                     (PrinterServiceProxy) context.getService(
28.                                                 refs[i]);
29.
30.                               if( PrinterServiceProxy != null ){
31.                                     context.ungetService(refs[i]);
32.                               }
33.                         }
34.                   }
35.             }
36.   }
```

Listing 5-3 Printer Location Sensor Plug-in Activator

Line number 9 of code given in Listing 5-3 shows the actual code that registers *PrinterSensor* as a service by calling *registerService* method, which takes three arguments: the name of interface under which the service is to be registered, also the same name will be used to find this service. In our case it should always be the name of ISensor interface, this makes consumers of Location Service Manager to find this location sensor plug-in without worrying about the type of location sensor, they only need to have information about ISensor interface. Second argument is the actual service object and this object must implements the same interface whose name was passed as a first argument. Here we passed an object of *PrinterServiceProxy* class which was instantiated on line number 3. Third argument is to associate additional metadata with the service we are registering. Since there can be more than one location sensor plug-in available and require to treat them accordingly, it would be good to associate some additional details to tell clients about each location sensor plug-in. Here we passed the object of Properties class, which is a part of Java framework, with only one property entry “type of location sensor” with value SensorType.PositionCoordinate. When Space Manager will retrieve this location sensor plug-in, the value of sensor type property will help in telling how to treat the location information this location sensor gives.

In the code Listing 5-3 stop method shows how to make Printer location sensor unavailable by calling ungetService method and passing an instance of ServiceReference class which contains the service reference of Printer location sensor plug-in. The code from line number 17 till 22 also demonstrates how to use the additional metadata information attached with registered service.

Extended printer proxy service has been registered as location sensor plug-in, now the next logical step is to show how this location sensor plug-in can be used to retrieve to location information. This can be done in two ways, one when we already know about the location sensor we are going to use, which rarely be the case because it will create dependency on the consumer to import the java package PrinterServiceProxy class is part of. Second, we don’t have any specific information the implementation of location sensor plug-in and who is providing the location information, all we know is the type of location sensor i.e. *SensorType.PositionCoordinate* and how it represents location. Listing 5-4 shows the code when we have information about the *PrinterServiceProxy* class and shows the code when we don’t know anything about *PrinterServiceProxy* class, it also shows how generic our approach is.

```
01.   public void ReadPrinterCoordinates() {
02.               String filter = "(&(sensorType="
                  + SensorType.PositionCoordinate + "))";

        ServiceReference[] refs =
              context.getServiceReferences(ISensor.class.getName(),
                    filter);

        if( refs != null ){
              for(int i=0; i<refs.length; i++){
                    PrinterServiceProxy printerSensor =
                          (PrinterServiceProxy ) context.getService(
                                refs[i]);

                    ISensorReading[] reading = printerSensor.Read();
                    if( reading.length > 0 ){
                          //Since we know that this sensor is
                          //of type SensorType.PositionCoordinate
                          //it returns location information in form
                          //of two dimensional coordinates

                          double x = 0;
                          double y = 0;

                          Coordinate c = reading[0].getCoordinates();
                          x = c.getX();
                          y = c.getY();
                    }
              }
        }
  }
```

Listing 5-4 Using Printer location sensor plug-in using PrinterServiceProxy class

```
  public void ReadPrinterCoordinates() {
        String filter = "(&(sensorType="
              + SensorType.PositionCoordinate + "))";

        ServiceReference[] refs =
              context.getServiceReferences(ISensor.class.getName(),
                    filter);

        if( refs != null ){
              for(int i=0; i<refs.length; i++){
                    ISensor unkownCoordinateSensor =
                          (ISensor) context.getService(refs[i]);

                    ISensorReading[] reading =
                          unkownCoordinateSensor.Read();
              if( reading.length > 0 ){
                    //Although we don’t know anything about this sensor
                    //except it’s type i.e. SensorType.PositionCoordinate
                    //Which means it returns location information in form
                    //of two dimensional coordinates

                    double x = 0;
                    double y = 0;

```

```
                        Coordinate c = reading[0].getCoordinates();
                        x = c.getX();
                        y = c.getY();
                 }
          }
   }
```

Listing 5-5 Generic code for using Printer location sensor plug-in

Code given in Listing 5-5 shows the generic behavior of Location Service Manager, we can use any location sensor plug-in without worrying about how the location sensor plug-in was implemented. The only thing which we as a consumer of location sensor are interested in is the information about location and how it is being represented i.e. in form or radio signal strengths, geographical positioning coordinate or simple two dimensional coordinate. We can either apply filter on *sensorType* property of registered location sensor plug-ins while retrieving available location sensor from OSGi service repository or we can check the type of location sensor by either using *GetType* of *ISensor* interface or *getSensorType* method of *ISensorReading* interface. Listing 5-5 shows how to retrieve all such location sensors that represent location information in form or two dimensional coordinate and how to read the actual sensor information using the *Read* method.

5.2.3.2 Thermometer as location sensor

In previous section we demonstrated how printer can also work as location sensor if it knows the x and y coordinates. In this section we will we will take an example of thermometer that we want to use as a location sensor and would like to identify different spaces with different temperature readings. For example if the thermometer reading reads 20° C then the space user is in is My Room. In order to achieve this goal, the proxy service installed for thermometer must implement *ISensor* interface and thermometer reading that it returns must be of form *ISensorReading*, just like the way we did in previous section for *PrinterServiceProxy* class. The code shown in Listing 5-6 shows the sample code for Thermometer service proxy class, which is later then extended to be able to register as location sensor plug-in in code presented in Listing 5-7.

```
public class ThermometerServiceProxy {
       //proxy service specific member....
       private int temperature;
       private String thermometerId;
       //......
       ...

       //proxy service methods...
       public int ReadTemperature(){
              return temperature;
       }
       //......
       ...
}
```

Listing 5-6 Sample Proxy Service Class for Thermometer

```
public class ThermometerServiceProxy implements IObservableSensor {
        //proxy service specific member....
        private int temperature;
        private String thermometerId;
        //......
        ...

        //proxy service methods...
        public int ReadTemperature(){
              return temperature;
        }
        //......
        ...

        public ISensorReading[] Read(){
              class TemperatureReading implements ISensorReading{

                    public String getHumanReadableName() {
                          return null;
                    }

                    public String getSensorType() {
                          return "Thermometer";
                    }

                    public int getSignalStrength() {
                          return ReadTemperature();
                    }

                    public String getUniqueID() {
                          return thermometerId;
                    }

                    public Coordinate getCoordinates() {
                          return null;
                    }
              }

              TemperatureReading[] readings = new TemperatureReading[0];
              readings[0] = new TemperatureReading();

              return readings;
        }

        public String GetType() {
              return "Thermometer";
        }

        public Date TimeStamp() {
              return new Date();
        }
  }
```

Listing 5-7 Extended Proxy Service Class for Thermometer

Listing 5-7 shows how Thermometer proxy service can easily be extended to be used as location sensor, the only need is to implement *Read*, *GetType* and *TimeStamp* methods of *ISensor*. The same code also demonstrates the usage of *ISensorReading* interface and how to wrap temperature reading in standard format.

getSignalStrength is the method that returns the location sensor reading for any sensor and in our case it returns current temperature that thermometer reads, *getSignalStrength* does nothing other than calling *ReadTemperature* method which is the method exposed by proxy service for thermometer. The other thing you would have noticed that we used *IObservableSensor* interface instead of *ISensor*, this is because we assumed that thermometer's reading can be read without any interaction from the user and *IObservableSensor* is just an extension of *ISensor* for such kind of sensors that can be read with or without user's interaction.

After wrapping thermometer proxy service in *ISensor* interface and it's reading in *ISensorReading* the next and final step is to register it as an OSGi service so that Space Manager can access it and treat it as location sensor plug-in. Code given in **Error! Reference source not found.** shows the code snippet for egistering and unregistering thermometer service proxy as location sensor plug-in.

```
public class Activator implements BundleActivator {
      public void start(BundleContext context) throws Exception {
            IObservableSensor thermometerSensor = new
                ThermometerServiceProxy();

            Properties props = new Properties();
            props.put(ISensor.Sensor_Type, "Thermometer");

            context.registerService(ISensor.class.getName(),
                        thermometerSensor, props);

            System.out.println("" +
                        "Thermomemter location sensor plugin has
started");
      }

      public void stop(BundleContext context) throws Exception {
            String filter = "(&(sensorType=Thermometer))";
            ServiceReference[] refs =
                  context.getServiceReferences(ISensor.class.getName(),
                        filter);

            if( refs != null ){
                  for(int i=0; i<refs.length; i++){
                        ThermometerServiceProxy temperatureReader =
                              (Thermometer) context.getService(refs[i]);

                        if( temperatureReader != null ){
                              context.ungetService(refs[i]);

                        }
                  }
            }
      }
}
```

Listing 5-8 Temperature Location Sensor Plug-in Activator

Now the thermometer service proxy has been registered as location sensor plug-in, user can use it as a reference to location from Space Manager. The code given in Listing 5-9 below shows how this thermometer location sensor plug-in can be retrieved and used.

```java
public void ReadTemperature() {
            String filter = "(&(sensorType=Thermometer))";
            ServiceReference[] refs =
                  context.getServiceReferences(ISensor.class.getName(),
                              filter);

            if( refs != null ){
                  for(int i=0; i<refs.length; i++){
                        IObservableSensor temperatureReader =
                              (IObservableSensor)
context.getService(refs[i]);

                        ISensorReading[] reading =
temperatureReader.Read();
                        if( reading.length > 0 ){
                              int currentTemperature =
reading[0].getSignalStrength();
                        }
                  }
            }
      }
```

Listing 5-9 Generic code for using Thermometer location sensor plug-in

5.3 Space Manager

Space Manager is the major concern of this project work, everything about Location Service Manager we have discussed so far is to help Space Manager in positioning user, so that information about physical location can be associated to the space. In this section, we will describe implementation of Space Manager in greater detail and show how it fulfills the user requirements related to spaces.

In section 4.2.3 we presented the API that Space Manager will provide to handle space related queries, each API method represents the intrinsic operation required for using spaces such as create new space, set current space or get space.

5.3.1 Space

Before going into the details of these methods, it is important to show how each Space get represented in Space Manager. Figure 5-6 below shows class diagram representing Space. Every member field of Space class represents the relevant field of Space structure defined in previous chapter. Methods with *get* prefix return the value of member fields e.g. *getName* method returns the value of Name field, which actually represents the name of space. Similarly methods prefixed with *set* assign the value to respective field such as *setParentSpaceId* assigns the value passed as an argument to field ParentSpaceId field.

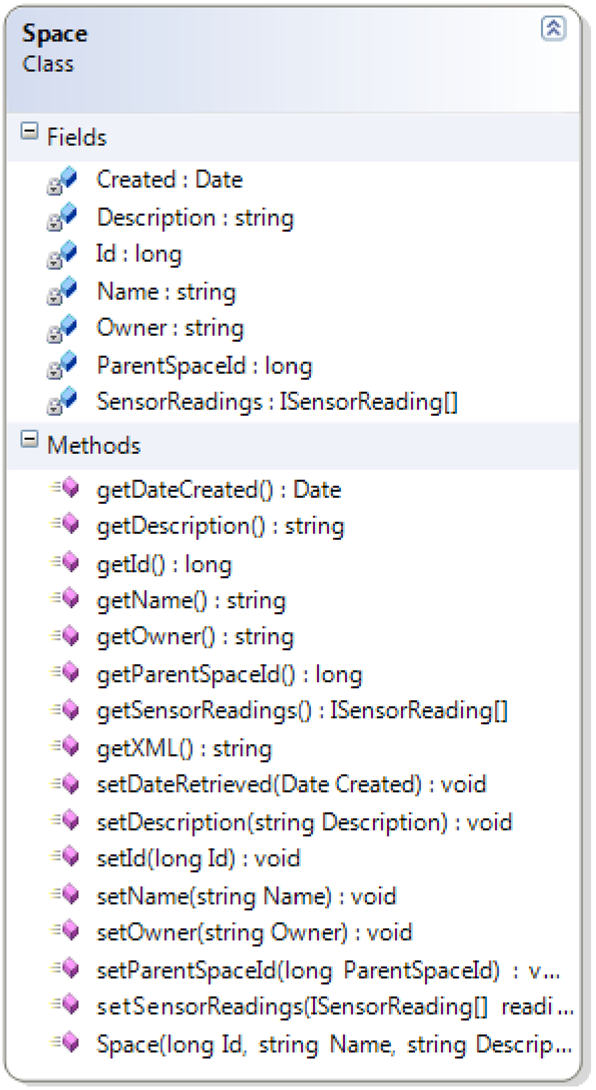

Space
Class
Fields
Created : Date
Description : string
Id : long
Name : string
Owner : string
ParentSpaceId : long
SensorReadings : ISensorReading[]
Methods
getDateCreated() : Date
getDescription() : string
getId() : long
getName() : string
getOwner() : string
getParentSpaceId() : long
getSensorReadings() : ISensorReading[]
getXML() : string
setDateRetrieved(Date Created) : void
setDescription(string Description) : void
setId(long Id) : void
setName(string Name) : void
setOwner(string Owner) : void
setParentSpaceId(long ParentSpaceId) : v...
setSensorReadings(ISensorReading[] readi...
Space(long Id, string Name, string Descrip...

Figure 5-6 UML Class diagram of Space class

Within Space Manager, instance of Space class is used to represent each individual space and to perform space functions, for outside world space is represented as an XML format shown in Listing 5-10.

```
<?xml version="1.0" encoding="utf-8" ?>
<Space>
  <Id>Space Id</Id>
  <Name>Name of Space</Name>
  <Description>Description</Description>
  <ParentSpaceId>Parent Space Id</ParentSpaceId>
  <Sensors>
    <Sensor>
```

```
            <UniqueId>Sensor Id</UniqueId>
            <Name>Sensor Name</Name>
            <Reading>Sensor Reading</Reading>
            <SensorType>Sensor Type</SensorType>
        </Sensor>
    </Sensors>
</Space>
```

Listing 5-10 XML representing Space

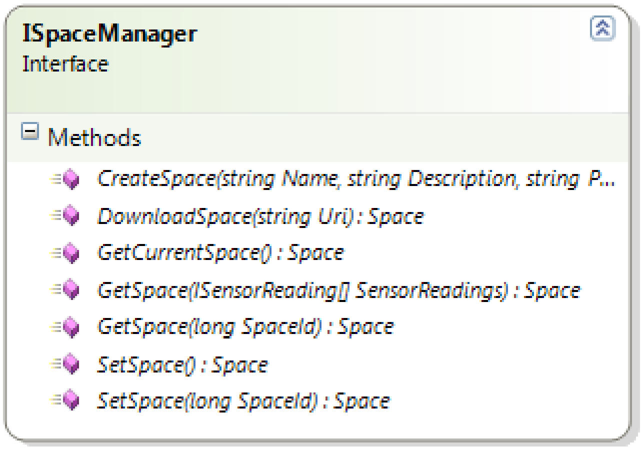
All the XML tags are self explanatory:

- Id tags contains unique identifier of space
- Name of space is placed in Name tag
- Text describing space goes inside Description tag
- ParentSpaceId tag contains the unique identifier of parent space, it contains -1 if space doesn't have any parent space (which may also means that this space itself a parent space).
- Sensors tag contains list of all location sensor along with their reading associated with the space.

Some benefits of using XML for representing Space are it easy to use and understand XML document, any Java application is capable of processing XML, XML is platform independent and it perfect choice for transmitting data over network. Therefore, any component of UbiCollab can use Space information, as each Space in UbiCollab represents different context of usage, this XML in turns provide information about context in form of Space XML. Also XML makes it easier for sharing and downloading spaces as Space XML file can easily be downloaded over HTTP or can be beamed to other UbiNode.

5.3.2 Space Queries

Space Manager provides set of basic methods to perform "space queries" which are: create new space, identity current space, set any space as current space, get space from space database and download space. Figure 5-7 shows UML class diagram of *ISpaceManager* interface, which exposes relevant methods for space queries.



ISpaceManager
Interface

Methods

- *CreateSpace(string Name, string Description, string P...*
- *DownloadSpace(string Uri) : Space*
- *GetCurrentSpace() : Space*
- *GetSpace(ISensorReading[] SensorReadings) : Space*
- *GetSpace(long SpaceId) : Space*
- *SetSpace() : Space*
- *SetSpace(long SpaceId) : Space*

Figure 5-7 ISpaceManager interface

You would have noticed that for some space query such as for setting current space, there is more than one method available, which means that this space query can be performed in more than one ways. In the coming up sections we describe each of these space queries in more detail.

5.3.2.1 Create new Space

Before using spaces it is necessary and natural to create a space first. As we have already described in previous chapters that space in UbiCollab is more than representing some place or area, spaces not only provides the contextual information about the area they are associated with but can also be used as resource for collaboration.

Therefore, it is quite apparent that to create new space is more than inserting a record in a Space database; it must also capture all aspects necessary to represent context i.e. contextual information. Space Manager exposes *CreateNewSpace* method and it is quite clear from method's name that it create new space. This method takes space name, description about space, purpose of space, unique identifier of parent space if any and collection of sensor readings for associating location information, if available.

Sequence diagram given in Figure 5-8 Sequence Diagram of Space Manager's CreateNewSpace shows how *CreateNewSpace* method works when called from GUI interface provided by Space Manager and presented in 4.4 GUI Mockups. GUI interface for create new space provides different controls to user for entering information about space such space name etc, also user can find his current position by using the Position Me button provided as a part of subjected user interface. Although, all the arguments this method takes are quite straight forward but getting location information from available location sensor plug-ins is bit tricky.

When user makes a request to position him, system searches for all available location sensor plug-ins which are of kind *IObservableSensor*, the reason behind this is to do as much work automatically as possible. If system is able to find such a location sensor plug-in, it then requests plug-in to return the location information and saves it. This information about location is then passed to *CreateNewSpace* method along with other required details.

CreateNewSpace method first calls the *ExecuteUpdate* method of database helper class to insert space information into the space database, this call *ExecuteUpdate* method returns the unique identifier assigned to this newly created space. The next step is about inserting location information in space database, if caller has provided collection of location sensor readings, then for each individual reading it Space Manager first check if sensor this belongs to already exists, get the unique identifier of sensor otherwise add sensor information to database and then returns the unique identifier of sensor. Then insert sensor reading along with the unique identifier of space and sensor to space database. Finally return the unique identifier of newly created space to the caller.

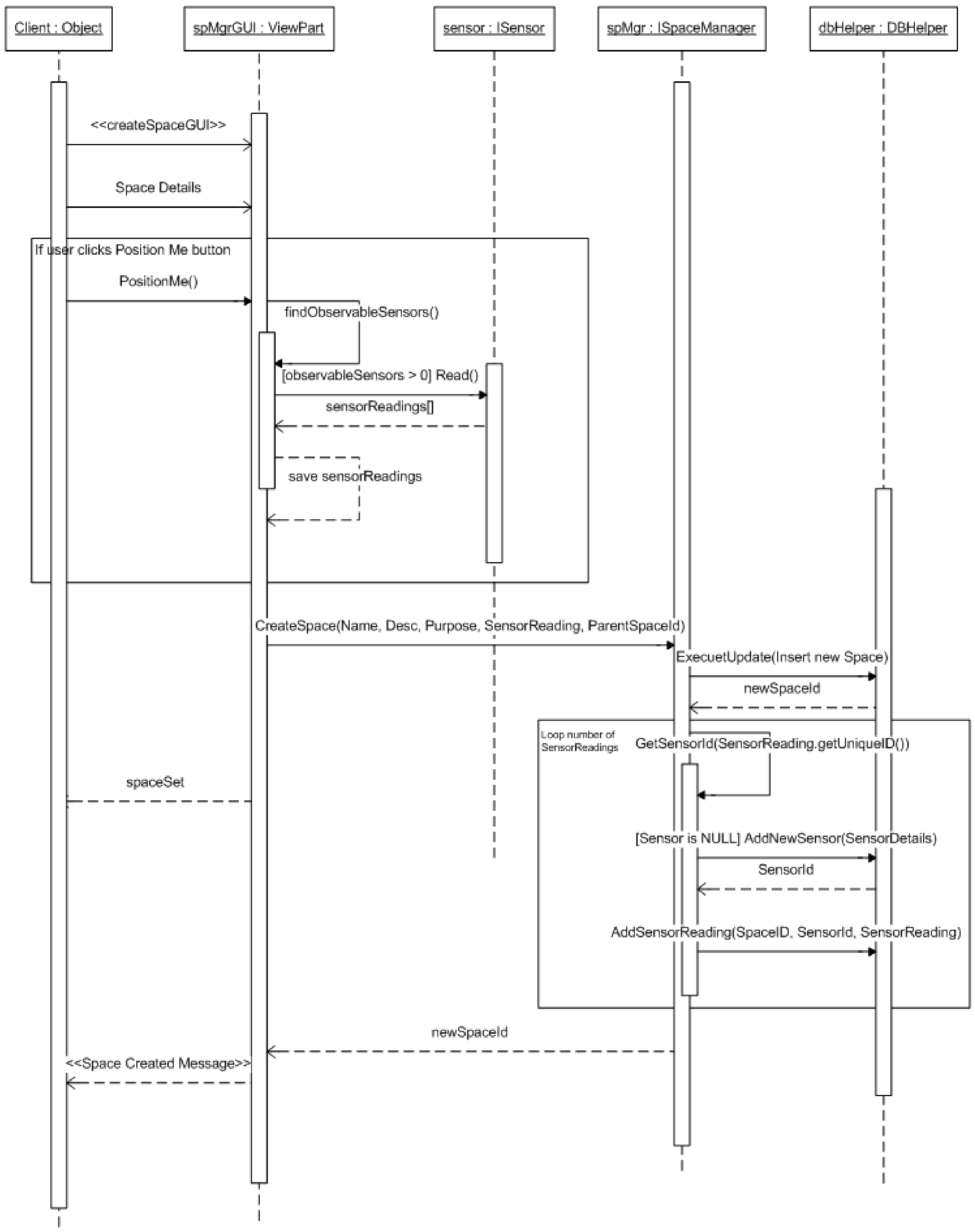

Client : Object
spMgrGUI : ViewPart
sensor : ISensor
spMgr : ISpaceManager
dbHelper : DBHelper
<<createSpaceGUI>>
Space Details
If user clicks Position Me button
PositionMe()
findObservableSensors()
[observableSensors > 0] Read()
sensorReadings[]
save sensorReadings
CreateSpace(Name, Desc, Purpose, SensorReading, ParentSpaceId)
ExecuetUpdate(Insert new Space)
newSpaceId
Loop number of SensorReadings
GetSensorId(SensorReading.getUniqueID())
spaceSet
[Sensor is NULL] AddNewSensor(SensorDetails)
SensorId
AddSensorReading(SpaceID, SensorId, SensorReading)
newSpaceId
<<Space Created Message>>

Figure 5-8 Sequence Diagram of Space Manager's CreateNewSpace

5.3.2.2 Set Space as Current Space

In order to set current space, Space Manager provides two methods, *SetSpace* which doesn't take any argument and other which takes unique identifier of space as argument. *SetSpace* which doesn't take any argument is for setting current space automatically using the collection of *ISensorReading*. When caller invokes this method, Space Manager first look for all available sensor plug-ins of kind *IObservableSensor*, if found Space Manager requests sensor plug-in for readings and then calls *GetSpace(ISesorReading)* method, which in turn query space database to find matching space with respect to the sensor reading. *GetSpace* method is described in more detail in section 5.3.2.3 Get Space below. Figure 5-9 shows sequence diagram of setting current space automatically.

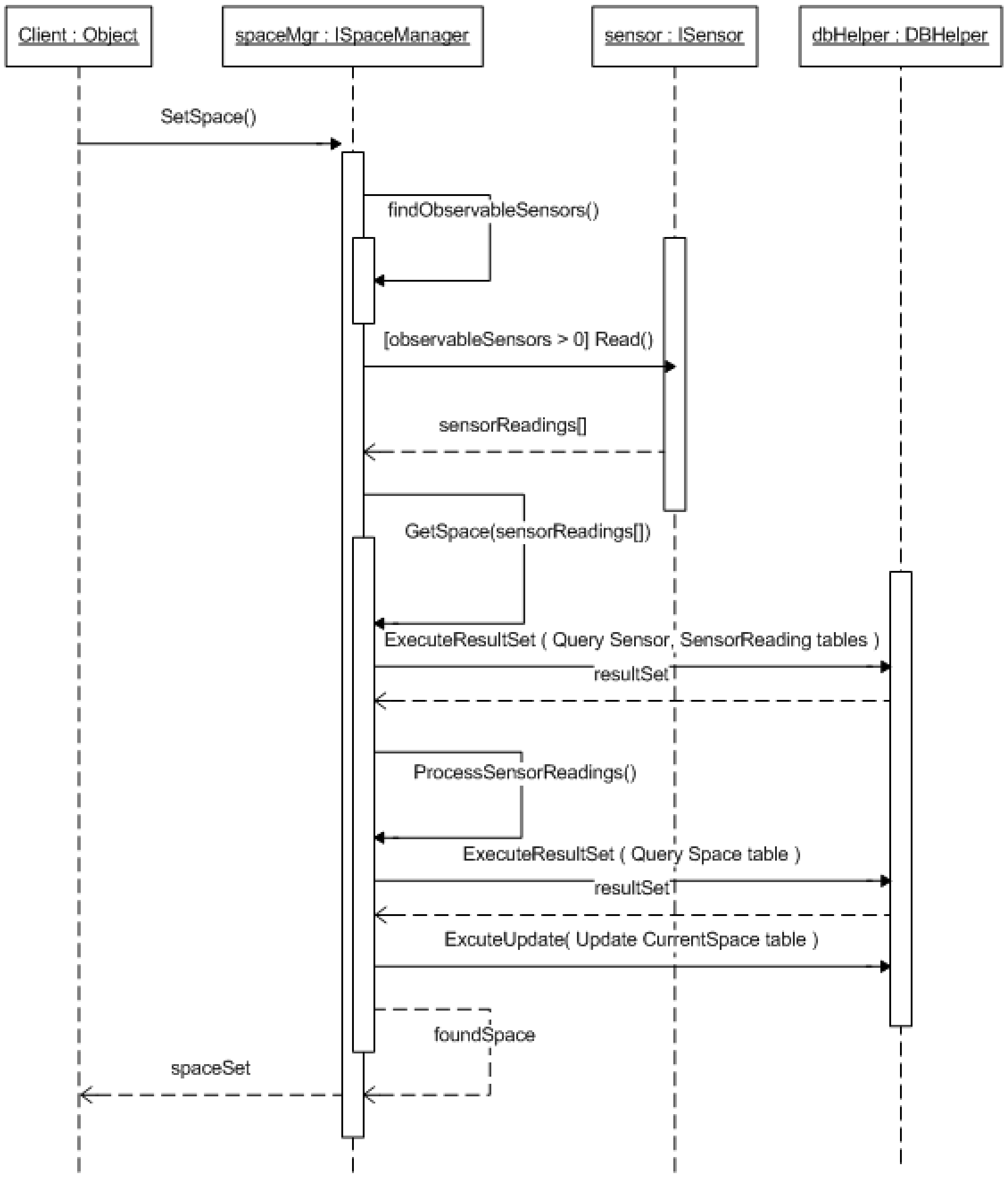

Client : Object
spaceMgr : ISpaceManager
sensor : ISensor
dbHelper : DBHelper
SetSpace()
findObservableSensors()
[observableSensors > 0] Read()
sensorReadings[]
GetSpace(sensorReadings[])
ExecuteResultSet (Query Sensor, SensorReading tables)
resultSet
ProcessSensorReadings()
ExecuteResultSet (Query Space table)
resultSet
ExcuteUpdate(Update CurrentSpace table)
foundSpace
spaceSet

Figure 5-9 Sequence Diagram of Space Manager's SetSpace method

The other overloaded *SetSpace* is used for setting current space manually; it takes unique identifier of space as an argument. Figure 5-10 shows setting space manually using the GUI provided by Space Manager, first client requests all the spaces he owns, which are actually saved in the space database. Space Manager shows all user owned spaces in selectable list format using which user can select any space he wants and set as current space. Space Manager then request database helper class to update CurrentSpace table.

Any other component of UbiCollab can set space manually by invoking *SetSpace* method with the unique identifier of space as an argument.

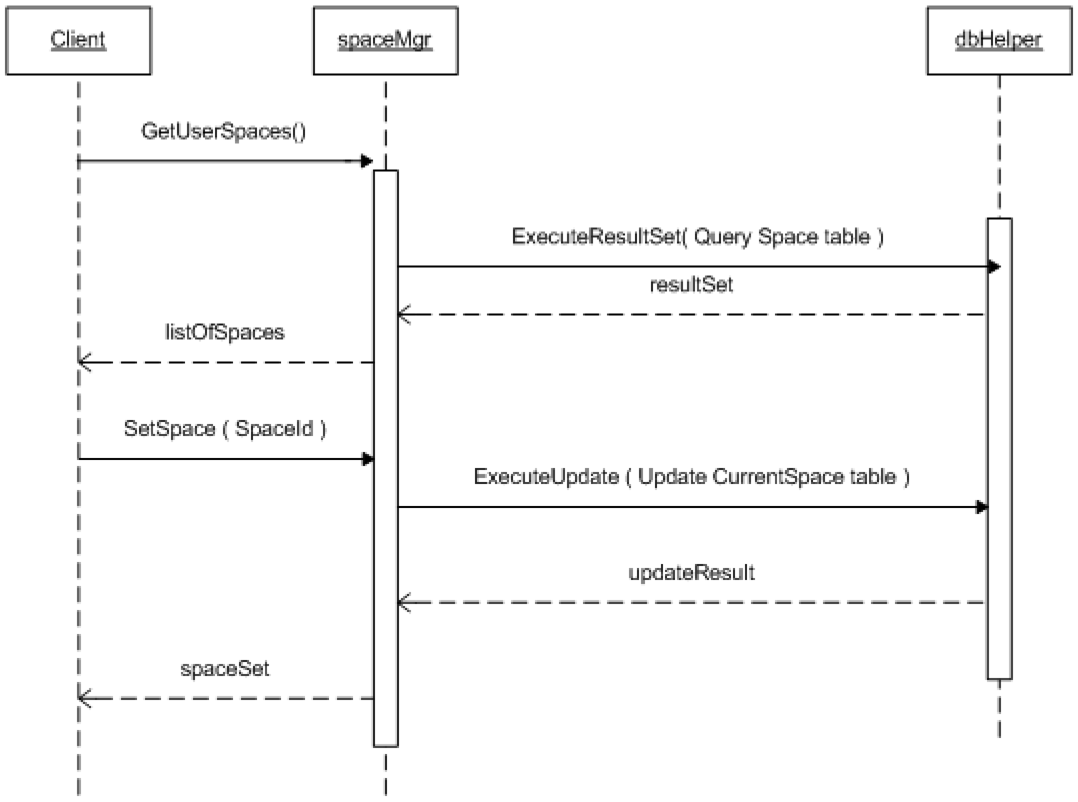


Figure 5-10 Sequence Diagram of Space Manager's overloaded SetSpace method

5.3.2.3 Get Space

Space database contains all the spaces user owns, in order to work user owned spaces Space Manager exposes overloaded *GetSpace* method. From Space database, spaces can be retrieved using two different ways, either by using unique identifier of space, in this case space identifier must be known and only that specific space will be returned, or through sensor readings. In later case all spaces that match sensor reading will be returned, for example there is a space called ‘Home’ which contains a sub space ‘Kitchen’, if sensor reading matches of those saved for space ‘Kitchen’, then both ‘Home’ and ‘Kitchen’ will be returned in form of collection of space.; ‘Kitchen’ being on the top of collection. Figure 5-11 shows the sequence diagram of *GetSpace* when collection of *ISensorReading* is passed as argument.

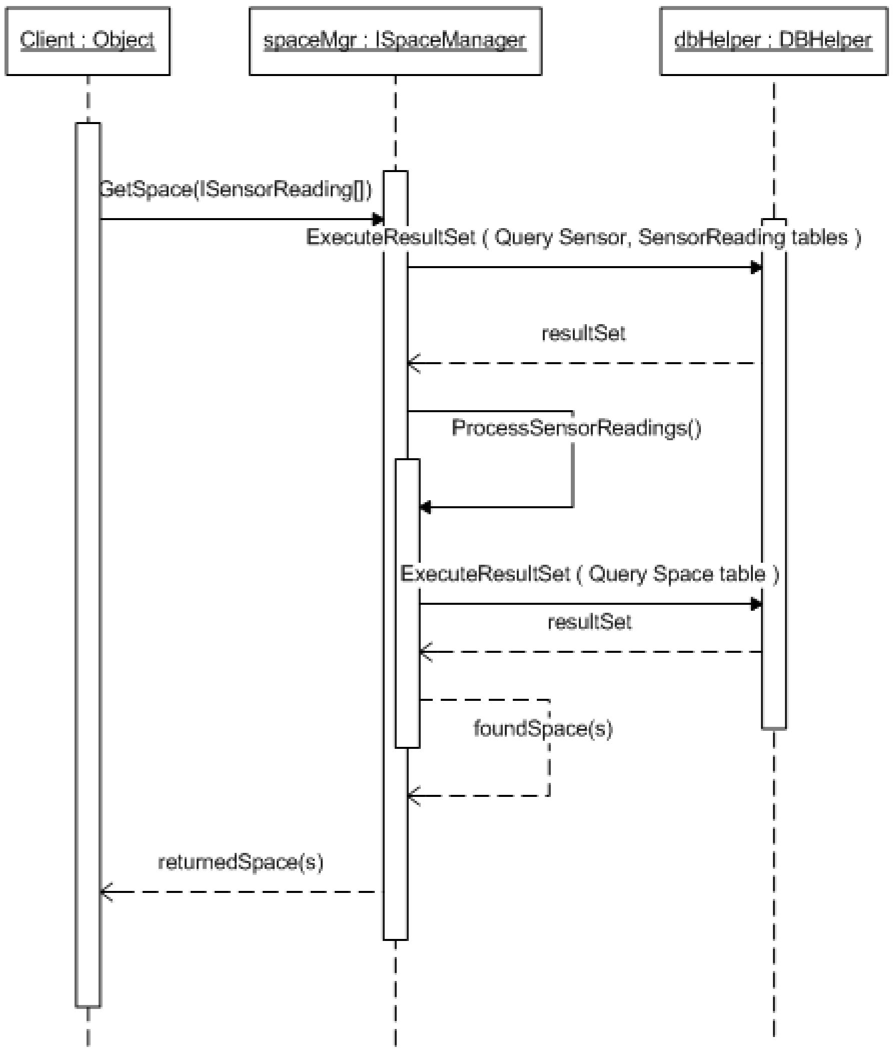


Figure 5-11 Sequence Diagram of Space Manager's GetSpace method

This same method is being called when user wants to set current space automatically and was described in greater details in 5.3.2.2 Set Space as Current Space.

Other overloaded version of *GetSpace* method is quite straight forward, it takes space identifier of space that user want to retrieved from database query database and returns space details encapsulating them as an instance of Space class. Figure 5-12 shows the sequence diagram of *GetSpace* method which takes space identifier as an argument.

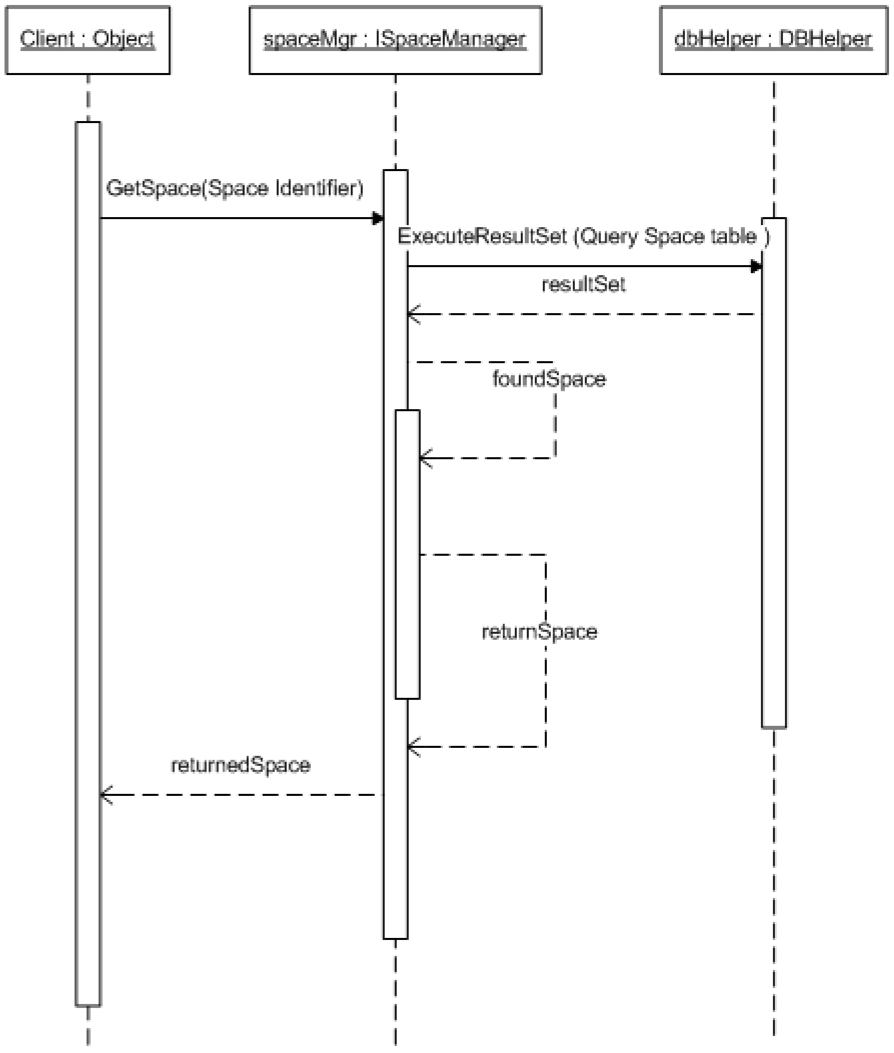


Figure 5-12 Sequence Diagram of GetSpace method exposes by Space Manager

5.3.2.4 Get Current Space

As the name of this section suggests, here we will discuss about the method Space Method exposes to get current space. In chapter sections 5.3.2.2 Set Space as Current Space and 5.3.2.3 Get Space we described how Space Manager gets users owned spaces from Space database and how any space can be set as current space either manually or automatically. Once any space is marked as current space, its reference is saved in CurrentSpace [33] table of Space database. When *GetCurrentSpace* is being called, Space Manager requests database helper to retrieve the unique identifier of current space, if there is any. Otherwise Space Manager calls its own method *SetSpace* to detect current space based sensor reading, where possible. If found, Space Manager saves the unique identifier of space to *CurrentSpace* table; finally Space Manager returns space details in form of Space class instance. Figure 5-13 below shows above described functionality in form of UML sequence diagram.

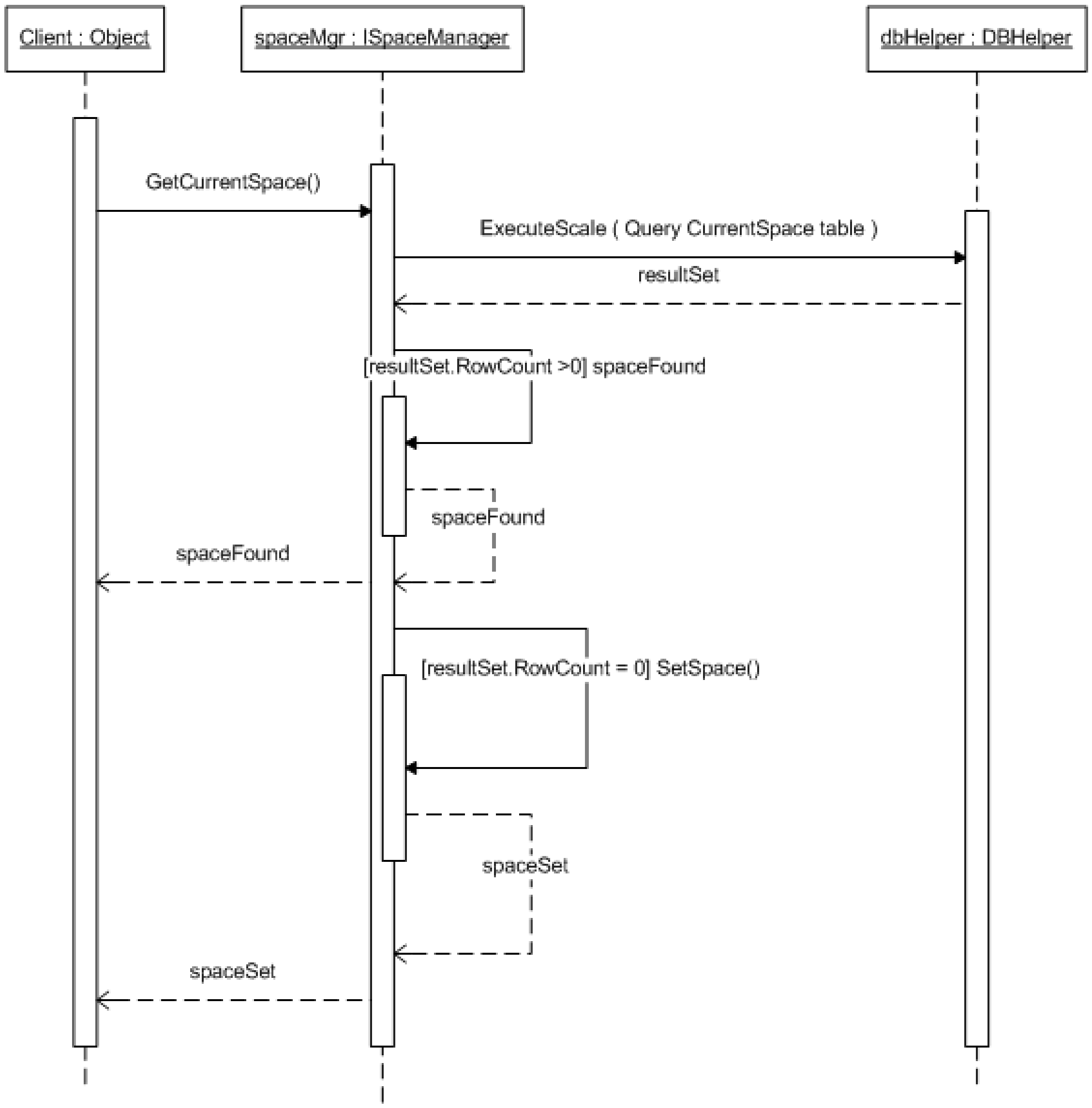


Figure 5-13 Sequence Diagram of GetCurrentSpace

5.3.2.4 Download Space

UbiCollab users can easily share spaces in form of structured XML file shown in Listing 5-10. Space Manager exposes *DownloadSpace* method which takes URI (Uniform Resource Identifier) to the space xml file, when caller calls *DownloadSpace* method, Space Manager verify URI before downloading into the UbiNode. Next Space Manager checks if downloaded file is actually an XML file with known structure. Then Space Manager goes through each space field and then calls *CreateNewSpace* method [65] to save space into Space database. Figure x shows sequence diagram of *DownloadSpace* method when being called using the GUI provided by Space Manager and presented in 4.4 GUI Mockups.

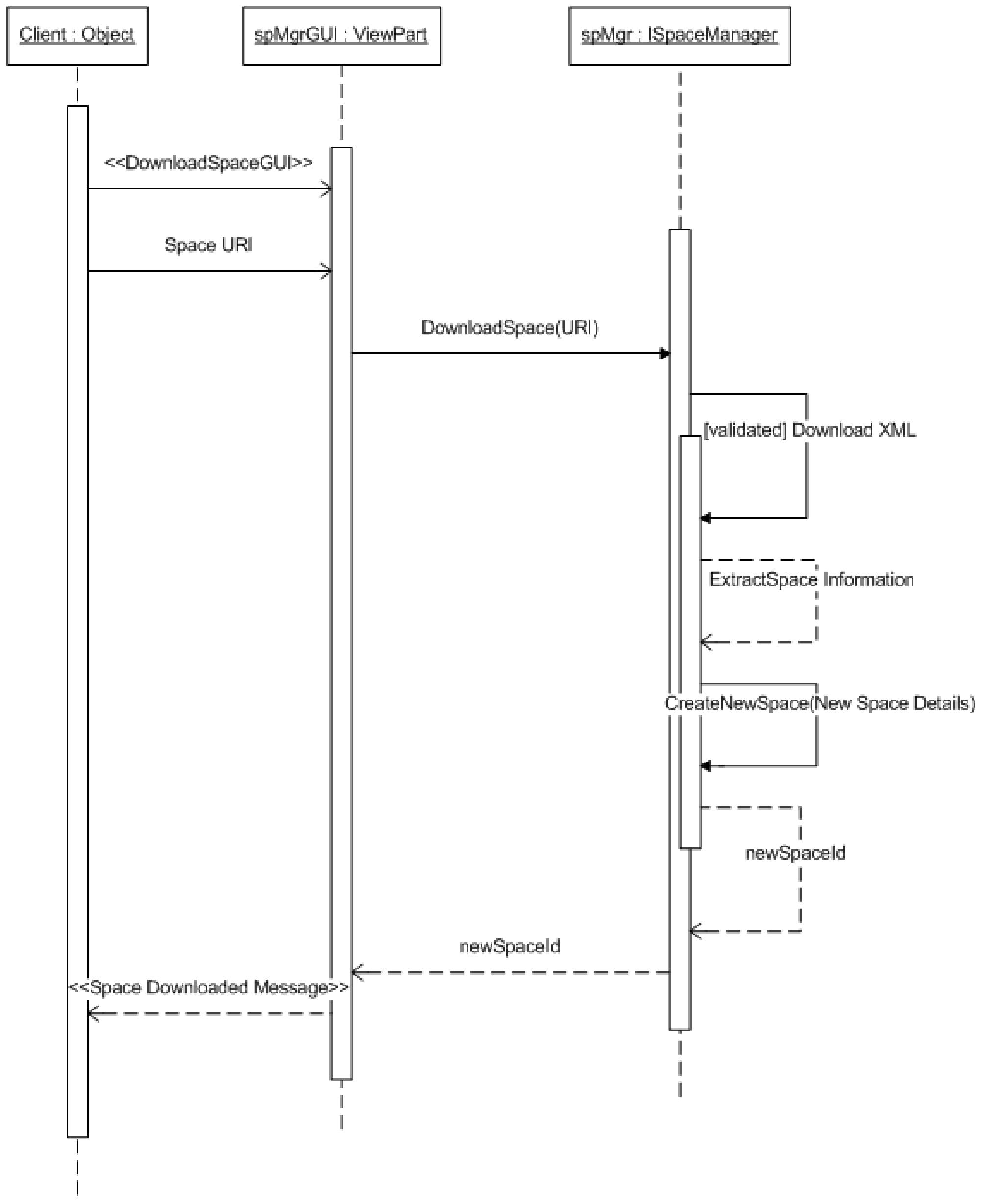
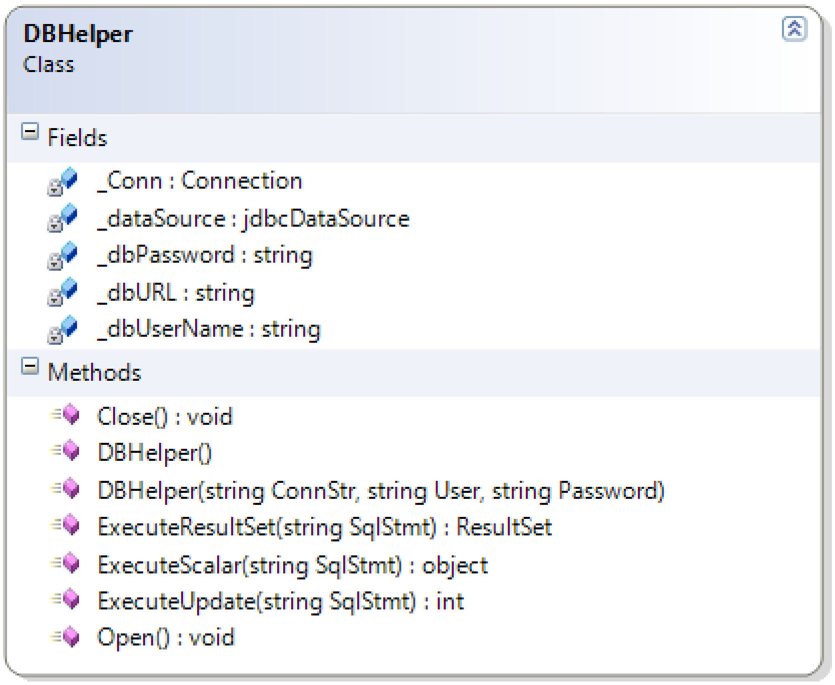


Figure 5-14 Sequence Diagram of Space Manager's DownloadSpace method

As space has been downloaded and saved to user's Space database, user can use this space as other spaces he created by himself. If downloaded space also contains information about location in form of location sensor readings, Space Manager can also identify if user comes to this space using sensor readings. Appendix shows two different example of space xml file, one that contains information about space including location information and other that represents space which doesn't have any location information associated with it.

5.3.3 Space Database Helper

This section briefly describes implementation of Space Database Helper package which comprises of database helper class to interact with space database, this database helper class exposes some methods that can be used to query database. Figure 5-15 shows the UML class diagram of *DBHelper* class.



```
DBHelper
Class

Fields
  _Conn : Connection
  _dataSource : jdbcDataSource
  _dbPassword : string
  _dbURL : string
  _dbUserName : string
Methods
  Close() : void
  DBHelper()
  DBHelper(string ConnStr, string User, string Password)
  ExecuteResultSet(string SqlStmt) : ResultSet
  ExecuteScalar(string SqlStmt) : object
  ExecuteUpdate(string SqlStmt) : int
  Open() : void
```

Figure 5-15 UML class diagram of DBHelper class

Space Manager uses an object of *DBHelper* class in order to query space database, *DBHelper* class provides two different constructors, a default constructor and an overloaded one. While creating a new instance of *DBHelper* class overloaded constructor can be used to connect to database whose connection string and credentials are being passed as argument, otherwise *DBHelper* uses predefined settings to connect with default database.

After instantiating an object of *BDHelper* class, *Open* method should be called before calling any method that results in database interaction. *Open* method tries to connect with the database either to default of specified one, returns an object of *Connection* class which is a part of Java framework, throws exception otherwise. This instance of *Connection* class is then used every time any interaction with database happens.

Close method on the other hand call the *close* method of *Connection* class which results in releasing all resources being used by database connection.

ExecuteScalar method takes SQL query in form of character string as an argument and returns the result of SQL statement as an output. As name suggest this method only returns single value and in our case it returns the value of first column of first row of result set returned by SQL query.

ExecuteResultSet on the other hand returns an object of *ResultSet* class which is a part of Java Framework, which contains all the rows that is returned by the SQL statement passed as an argument.

ExecuteUpdate method is use to perform INSERT, UPDATE and DELETE SQL queries. This method returns the number of rows affected by the SQL query this is passed as an argument.

Chapter 6 Evaluation

This chapter is dedicated towards the evaluation of work done as a part of this project. The implemented solution is evaluated using the scenario described in 2.3.1 Scenario and using the SpaceTwitt application we built as a part of, which updates Twitter[4] status with the name of space user is in.

We start this chapter by going through scenario evaluation and then continue toward evaluation using application, we end this chapter with satisfaction status of requirements that were presented in 2.4 Requirement Specification.

6.1 Evaluation based on Scenario

6.1.1 Scenario

Markus uses UbiCollab not only to collaborate with his subordinates and colleagues but also to remain in contact with distant friends and family members. However, he doesn't want to be disturbed by his family or friends, whenever he is in his office, only, collaboration from the people within the workplace is allowed. Using the create new space interface of Space Manager, Markus creates a space in UbiCollab giving it a name 'Markus' Office' and associates it with the physical location of his office. This physical location-to-space mapping is done through location sensor plug-in available in his office, Wi-Fi in this case.

Now whenever he enters in his office, he runs an UbiCollab application which basically sets instant messenger profile based on the space information. This application calls Space Manager for current space using the API Space Manager exposes, as a result Space Manager automatically senses the location as 'Markus' Office', sets this space as current space and returns it to the application. In turn application changes his status to 'Not available or Busy' to friends and family and 'Available' to colleagues.

Being a head of IT services, giving presentations at different locations in different departments is his routine job. It is usual for him to visit the same meeting room he was before in, so instead of discovering resources available in the room using UbiCollab Resource Discovery Manager every time and then configuring them for usage, he has created separate spaces using Space Manager for most common room. Now whenever he visits any meeting room which as an associated space, UbiCollab automatically detects the current space, by using information about current space he can browse through the list of available resources using Resource Discovery Manager.

For creating new space Markus starts create new space interface of Space Manager, while preparing the create new space interface, Space Manager also loads all the spaces Markus owns and display them in form of selectable list. Using this interface he creates a new space by providing the name 'Markus' Room – Home', other details and selects 'Home' as a parent space.

[4] Twitter: http://www.twitter.com

6.1.2 Scenario Walkthrough

Before starting this scenario walkthrough, there are some assumptions to make:

- User already has UbiCollab installed on his UbiNode with all latest packages.
- Wi-Fi location sensor plug-in is already downloaded and installed on UbiNode.

Markus is the name of our UbiCollab user and this name will be used throughout this section when we our intention is to show end user action.

1. Markus starts UbiCollab on his UbiNode as shown in Figure 6-1.

2. Then from the list he taps on Space Manage, although this screen shows Space Manager only but in real it will show all core components that can be used to make changes in setting of UbiCollab; such as Space Manager, Service Discovery Manager. As soon as user taps Space Manager, screen with available Space Manager options comes up as shown in Figure 6-2.

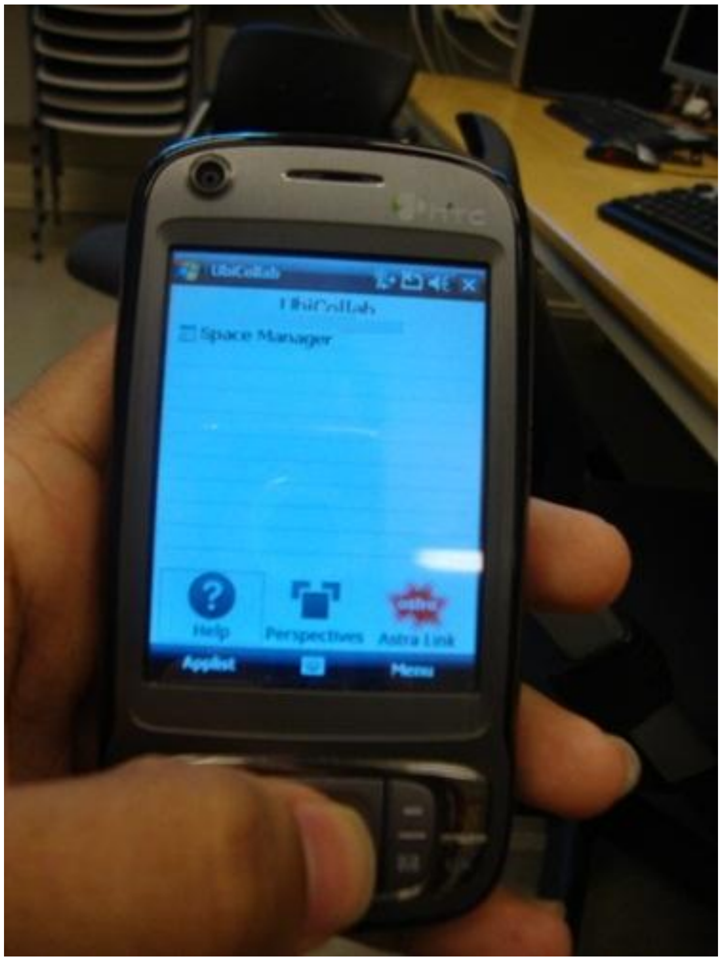


Figure 6-1 User starts UbiCollab on his UbiNode

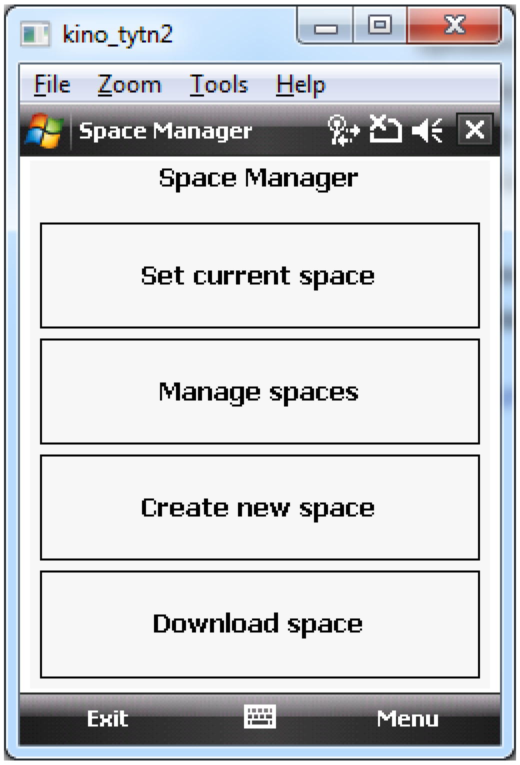


Figure 6-2 Space Manager GUI

3. As Markus wants to create a new space, he taps on Create new space option which brings up a screen shown in Figure 6-3.

4. Now Markus enters details such as name of space, description or purpose as shown in Figure 6-4. Figure 6-5 shows how Markus can make this new space he is creating as a sub space of another, by selecting the parent space from the list of all space he owned.

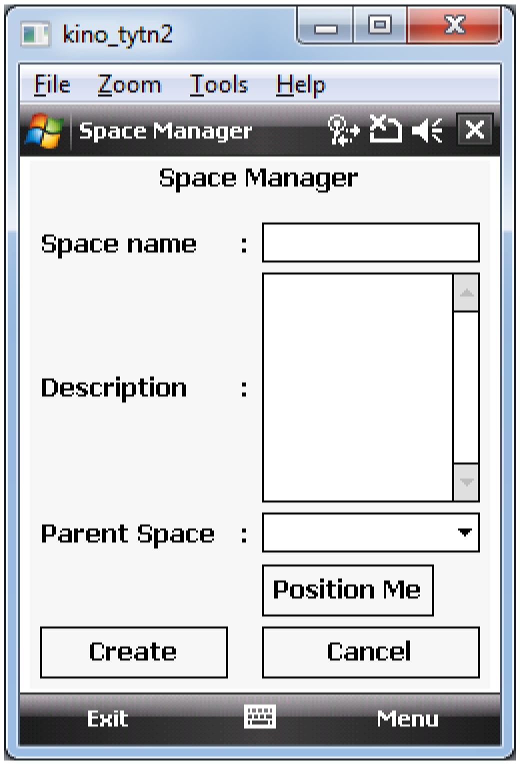
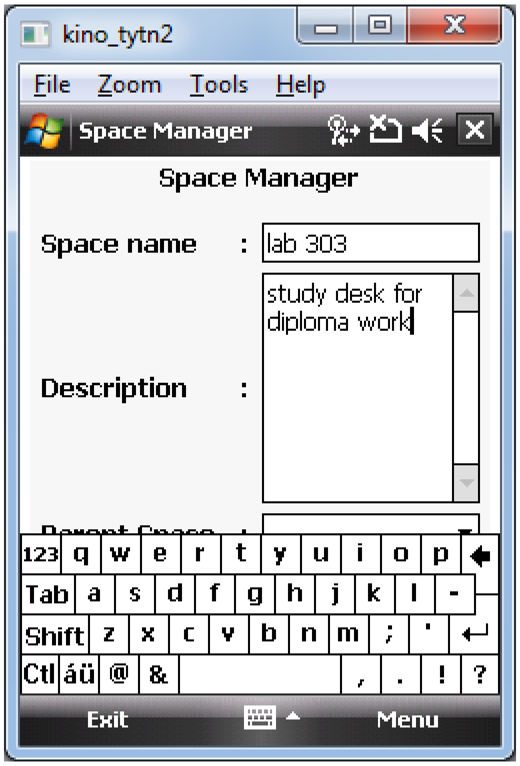


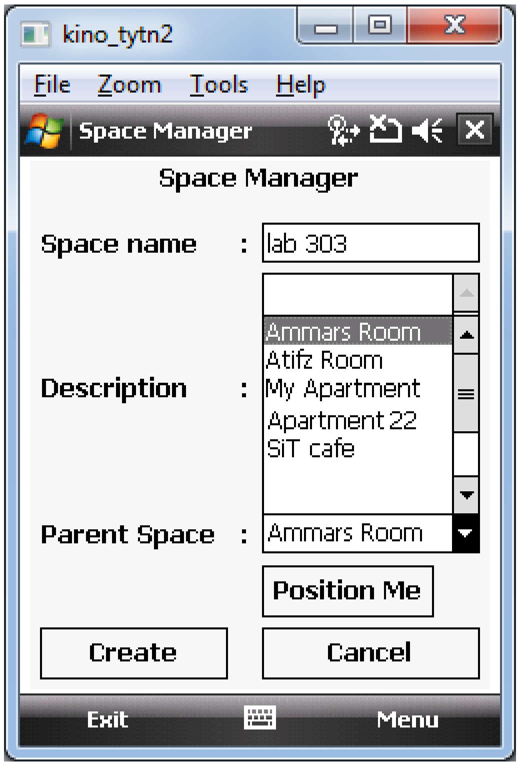
Figure 6-3 User starts create new space screen



Space name : lab 303

Description : study desk for diploma work

Figure 6-4 User inputs details about new space



Space Manager

Space name : lab 303

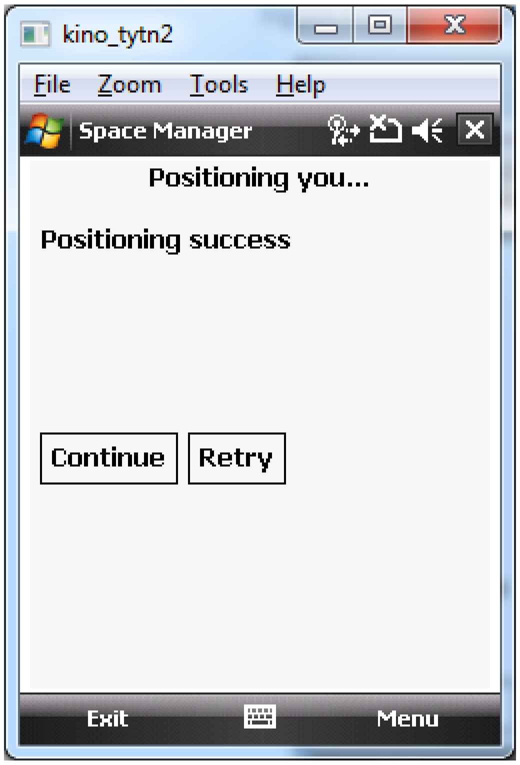
Description :

Ammars Room
Atifz Room
My Apartment
Apartment 22
SiT cafe

Parent Space : Ammars Room

Figure 6-5 Creating Sub Space within Space

5. In order to add location information Markus taps on Position Me button which is shown in Figure 6-4, Space Manager tried to position Markus automatically by using any available location sensor plug-in. As Wi-Fi location sensor plug-in has already been installed, Space Manager associated location information gathered from plug-in with the new space and displays a confirmation message. See Figure 6-6 below.



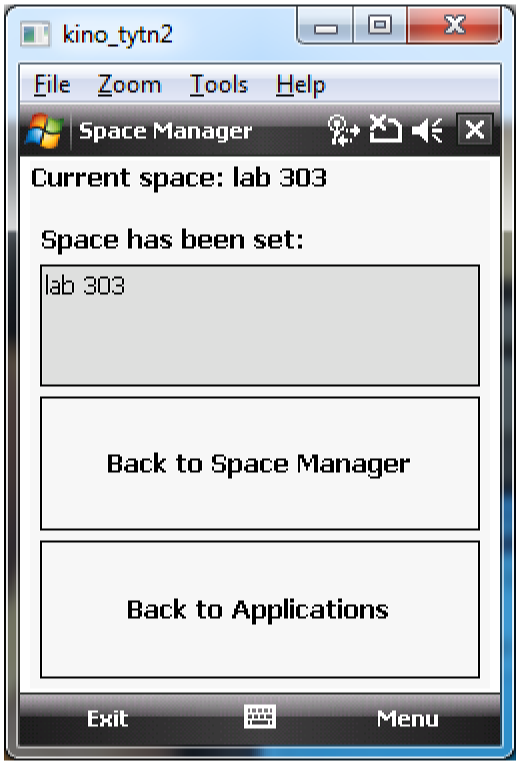
Positioning you...

Positioning success

Figure 6-6 Positioning User successfully

6. Markus taps on continue to return back to Figure 6-4, where he taps create button to create a new space and save it in space database.

7. As new space with name lab 303 has been created, Space Manager is now able to detect this space with the help of Wi-Fi location sensor plug-in and if Markus installs any new resource to his UbiNode, Resource Discovery Manager can query Space Manager to get information about current space and can associated that resource with the current space. This way whenever Markus comes to this space again, he can find all the resources available within the space. Figure 6-7 demonstrates what happens when Markus taps on Set current space option.



Current space: lab 303

Space has been set:

lab 303

Figure 6-7 Space Manager to find and set current space

8. In this case Space Manager has already detected the current space correctly as lab 303, if it would not be the case, Space Manager displays list of all available space Markus owns and giving an option to set any space as current space manually.

9. Using Space Manager Markus can share the space with his friends or family members; he can either beam or provide WWW URI to space xml to share spaces he owned with other users or he can either receive space via beam or download from URI. Download Space interface provides an easy way to download space from WWW by providing the URI to space xml file, Figure 6-8 and Figure 6-9 show Download Space interface and Markus entering URI to download space, respectively.

10. Space Manager downloads space xml file from provided URI, parse xml, displays space information to user and saves in space database as shown in Figure 6-10.

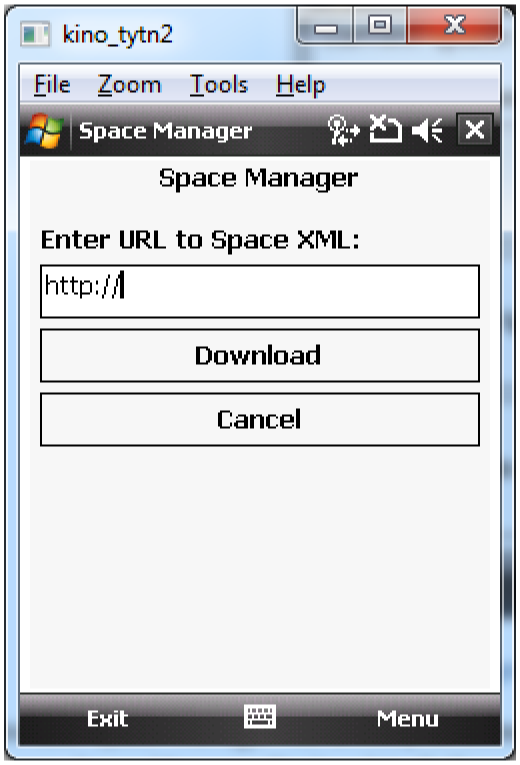


Figure 6-8 Download Space interface of Space Manager

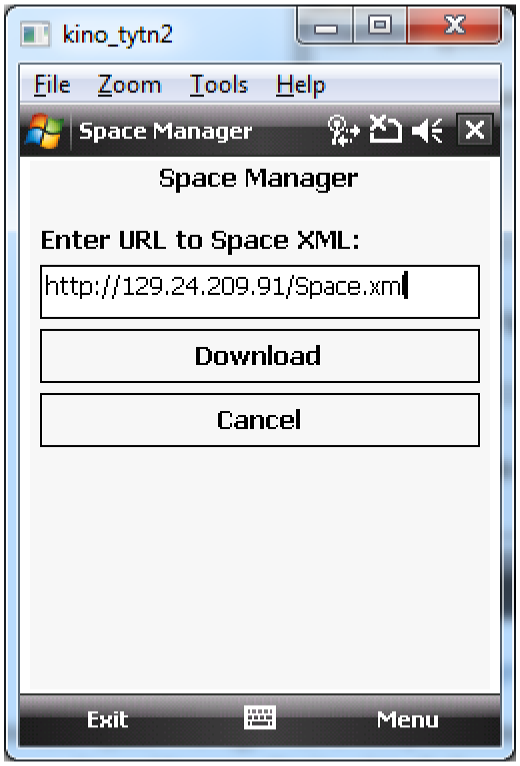
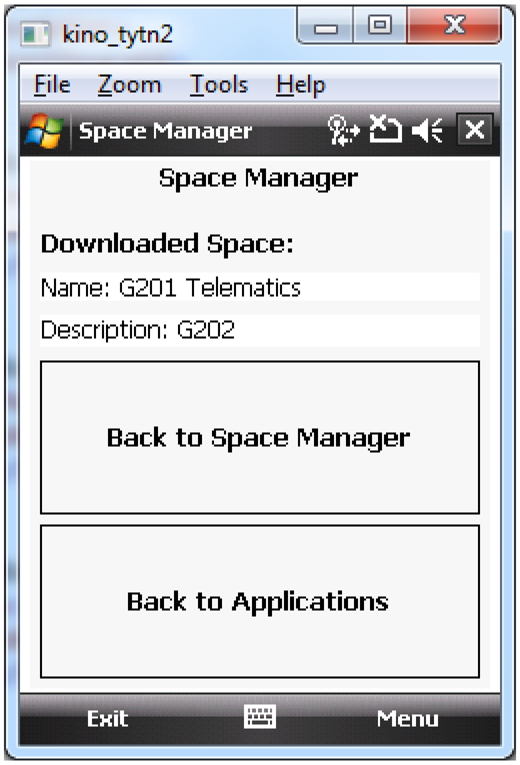


Figure 6-9 User to download space from www using URI to space



Space Manager

Downloaded Space:

Name: G201 Telematics

Description: G202

Figure 6-10 Space Manager downloads space, displays to user and save to space database

6.1.2 Evaluation using application

This section is dedicated towards evaluating our work with the perspective of an application that queries Space Manager.

As a part of our work we have developed an application called ‘SpaceTwitt’ which queries Space Manager API for current space of user and if found updates user’s twitter space, for this application we have used an open source API called Twitter API ME[5] to in order to interact with twitter. This application also shows how applications can user space information and can adapt them according to the context use.

1. User starts UbiCollab on his UbiNode and switch to Applications.

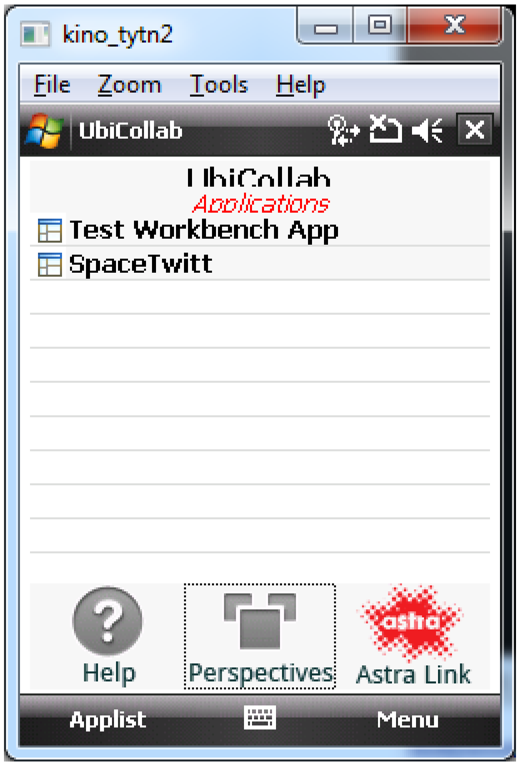


Figure 6-11 Starting SpaceTwitt Application

2. User taps on SpaceTwitt option, which results in displaying following screen, shown Figure 6-12.

3. Now user taps on Set Space which results in querying Space Manager for current space user is in. Figure 6-13 and Figure 6-14 shows how Space Manager correctly finds the current space and returns to application. Our SpaceTwitt application on other hand displays the name of space and also updates the twitter space of user.

[5] Twitter API ME: http://kenai.com/projects/twitterapime/pages/Home

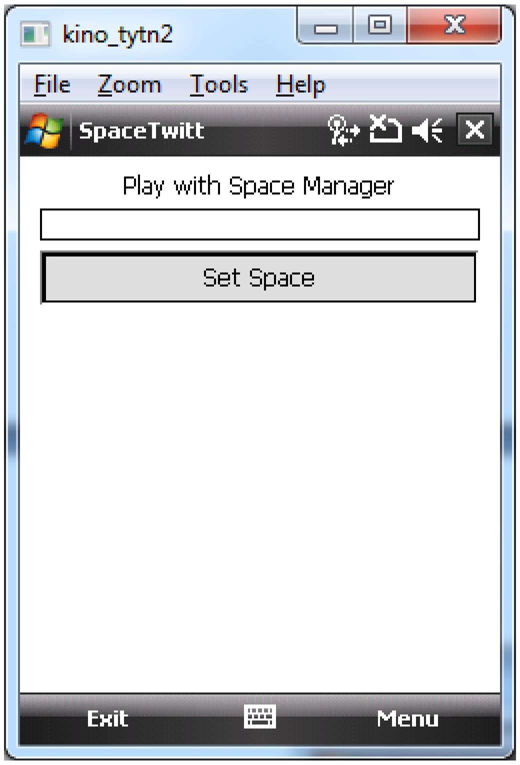
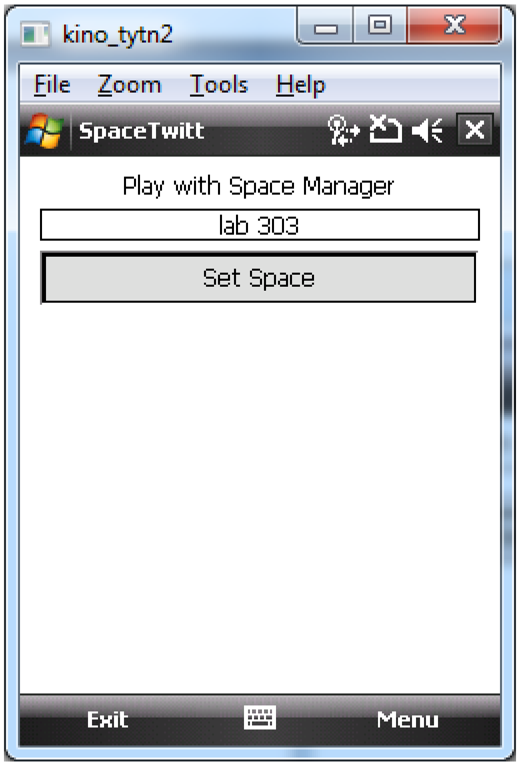
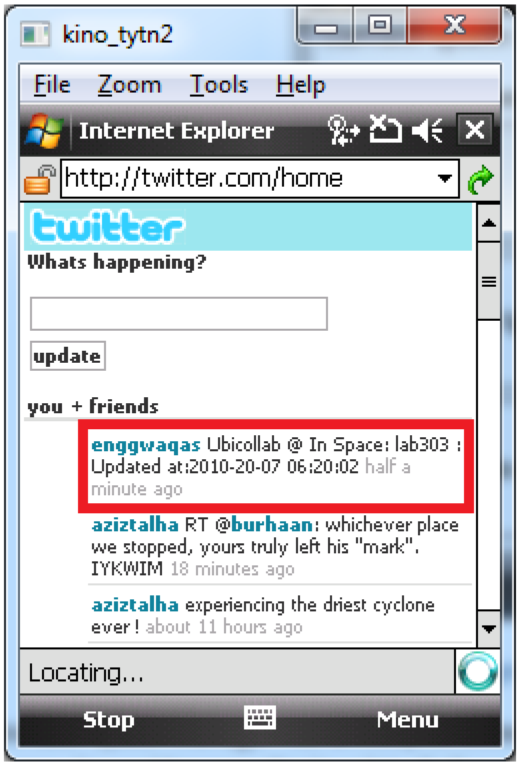


Figure 6-12 SpaceTwitt application started



lab 303

Figure 6-13 SpaceTwitt interacting with Space Manager via API



enggwaqas Ubicollab @ In Space: lab303 : Updated at:2010-20-07 06:20:02 half a minute ago

aziztalha RT @burhaan: whichever place we stopped, yours truly left his "mark". IYKWIM 18 minutes ago

aziztalha experiencing the driest cyclone ever ! about 11 hours ago

Figure 6-14 Twitter status updated with name of current space

6.2 Requirement Satisfaction Status

This section briefly discusses the status of satisfaction of both functional and non-functional requirements.

6.2.1 Requirement Satisfaction Status of Functional Requirement

Requirement ID	Status	Comments
FR-SM 1	Partially Satisfied	User can create new spaces but part for modifying existing spaces is not yet implemented.
FR-SM 2	Fully satisfied	While creating new space user can input all relevant information.
FR-SM 3	Fully satisfied	With the help of Location Service Manager, Space Manager can associate location information while creating a new space.

FR-SM 4	Fully satisfied	Space Manager is able to identify current space based on sensor readings.
FR-SM 5	Fully satisfied	Space Manager provides both way automatic and manual for setting any space as current space.
FR-SM 6	Fully satisfied	User can browse all the space he owns
FR-SM 7	Partially Satisfied	Right now it is only limited to download spaces using space xml via world wide web.
FR-SM 8	Partially Satisfied	Although users can create sub spaces with space but still some research is needed to be done in identifying and distinguishing them.
FR-SM 9	Fully satisfied	With the help of Location Service Manager, Space Manager can deal with location information given in different formats.

Table 6-1 Satisfaction status of functional requirements

6.2.1 Requirement Satisfaction Status of Non-Functional Requirement

Requirement ID	**Status**	**Comments**
NFR-G 1	Fully Satisfied	Solution implemented during the course of this project targets resource limited devices and runs on Java mobile edition.
NFR-G2, NFR-G3, NFR-G4	Fully satisfied	Implemented solutions conform to all technical constraints put by UbiCollab. Space Manager and Location Service Manager both run on UbiNode and don’t require any changes in current platform.
NFR-G5, NFR-G6	Fully satisfied	Implemented solutions run on CDC Java Virtual Machine running OSGi framework.
NFR-SM 1, NFR-SM 2	Fully satisfied	Beside API for other components for internal interaction Space Manager also provide graphical user interface for easy user interaction.
NFR-SM 3, NFR-	Fully satisfied	

SM 4		
NFR-LSM 1, NFR-SM 2, NFR-LSM 3, NFR-LSM 4	Fully satisfied	Location Service Manager provides set of interfaces and classes that helps in abstracting away hard differences, converting low-level sensor data into high-level information and standardizing this information with sensor information also to treat information according.
NFR-LSM 5	Partially Satisfied	User can install and uninstall any location sensor plug-in, right now only manually. Naïve user can't perform this action now as it requires dealing with console and commands.

Table 6-2 Satisfaction status of non-functional requirements

Chapter 7 Conclusion and Future work

This chapter is dedicated towards concluding this project report by presenting the contributions that have been made and some suggestions for the possible future work.

7.1 Contributions

The main focus of this project work is in the domain context-aware ubiquitous computing that what constitute context in ubiquitous environment, how contextual environment can be gathered and used; and then using the outcomes of this research on existing UbiCollab system to extend its functionalities.

Below are the contributions that will be made during the course of this project:

1. Research: Context-model for context data representation. Context-awareness has become a very important aspect of ubiquitous computing and a lot of research is going on in the domain of making ubiquitous system able to adapt them according to environment. But defining what information constitute context and how to represent contextual information is still a research problem. In this work of our we have tried to answer these research questions by using the definition of context from earlier work on UbiCollab and proposing simple yet effective model to represent contextual information.

2. Research: Algorithm for extracting information from low-level location sensors and transforming them into common and understandable format. Location-awareness is an important aspect of context awareness; thanks to the advancements in technologies nowadays there are many different sensors available that can be used to sense the location. But due to heterogeneous nature of these sensors it’s difficult to use them and there is a huge gap lies in converting low-level sensor data into high-level location information. By proposing and implementing Location Service Manager for this project we tried to bridge this gap.

3. Implementation: Developing Space Manager module for creating and managing spaces

4. Implementation: Creating GUI for Space Manager using the previously developed GUI framework for UbiCollab

5. Implementation: Generic database interface for storing and retrieving space information. Context information that system collects either from user input or location sensors need to stored in some flexible data store which also makes sure not to lose any key information during structuring data according the format of data store. By doing research in this area and taking help from previously done work we proposed a database which has a very generic structure, very flexible and reliable that no important information will be lost during saving or retrieving contextual information.

6. Implementation: Wi-Fi Sensor Plug-in
7. Implementation: Fingerprinting algorithm for Wi-Fi based location detection

7.2 Conclusions

Being any other ubiquitous platform UbiCollab also needs to be aware of the context it is being used in and if possible posses the ability to adapt itself according to environment. Beside resources for collaboration concept of spaces in UbiCollab also represents the context of use. Moreover information about space can be used to identify different context. Our solution here was to user space and context information interchangeably which also means that all relevant information should be collected when creating a space. Out proposed Space Manager proved be successful in collecting this information providing users the ability to input only specific but important information easily. On the other hand Location Service Manager facilitates Space Manager to collect location information without worrying about sensing hardware. Although Location Service Manager needs some more testing but still it is currently able to support any kind of sensing hardware, the only need is to write sensor plug-in according to the guidelines Location Service Manager provides. Not limited to Space Manager, Location Service Manager can later be extended to be used as a standalone user tracking system.

7.3 Future work

This section is dedicated towards ideas for possible future work.

Behavior Adaptation: After being able to create and manage spaces and capture context information, one of the possible usages is behavior adaptation. Applications in UbiCollab can change their setting and/or behavior according the information about current space user is in. Applications can also be extended in a way to take decisions on the behalf of user, if contextual information is being used with the information about user's acts which he usually do in particular context; this can be done by maintain some kind of where about diary.

Privileges in Services and Resource Discovery: By tagging services and resources with the space they are in we can make them context aware. Although this way we can find the available resources in proximity but there still remains an issue of privacy. What if user has downloaded space information from another user who has access to all the resources and there some resource which can only be used by users with certain credentials?

References

[1]. Farshchian, B.A. & Divitini, M., “UbiCollab Architecture White Paper”, 2007

[2]. Mora, S., 2009. "A mobile extensible architecture for implementing ubiquitous discovery gestures based on object tagging". Master's Thesis, NTNU

[3]. Farshchian, B. A. & Divitini, M., 2005. “UbiCollab: Improving collaboration with location services”. Conference Paper. In IEEE International conference on pervasive services (ICPS), IEEE, Santorini, Greece, p.417 – 420

[4]. Weiser, M., 1999. “The computer for twenty-first century”. In ACM SIGMOBILE Computing and Communication Review, ACM, Vol. 3, Issue 3 (Ed. Victor Bahl), pp. 3 – 11

[5]. Abowd, D.G., Dey, A.K., Brown, P.J., Davies, N., Smith, M. & Steggles, P., 1999. “Toward a better understanding of context and context-awareness”. In Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, Springer-Verlag, pp. 304 – 307

[6]. Dix, A., Rodden, T., Davies, N., Trevor, J., Friday, A. & Palfreyman, K, 2000. “Exploiting space and location as a design framework for interactive mobile systems”. In ACM Transactions on Computer-Human Interaction (TOCHI), ACM, pp. 285 – 321

[7]. Schilit, B., Adams, N. & Want, R., 1994. "Context-aware computing applications". Conference Paper. In IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, USA, pp.85 – 90

[8]. Hevner, A., March, R., Park, J. & Ram, S., 2004. “Design Science in Information System Research”, 2004. In Management Information Systems Quarterly, Vol. 28, No.1 (Ed. Ron Weber), pp. 75 – 105

[9]. Oates, J., 2006. “Researching Information Systems and Computing”. London: SAGE Publications Ltd.

[10]. Rogers, Y., Sharp, H. & Preece, J., 2002. “Interaction Design: Beyond Human-Computer Interaction”, Wiley

[11]. Castelli, G., Mamei, M., Rosi, A. & Zambonelli, F., 2009. “Extracting High-Level Information from Location Data: W4 Diary Example”. In Mobile Networks and Applications, Vol. 14, Issue 1, Kluwer Academic Publishers, pp. 107 – 119

[12]. Bartlett, N., 2009. “OSGi In Practice”. Open Book. Creative Commons. Retrieved from: http://s3.amazonaws.com/neilbartlett.name/osgibook_preview_20090110.pdf on Oct, 15th 2009.

[13]. “Eclipse RCP website”. http://www.eclipse.org/ercp/

[14]. “Place Lab CVS Repository”. http://www.placelab.org/toolkit/doc/cvsdeveloper.php

[15]. Helal, S., Winkler, B., Lee, C., Kaddourah, Y., Ran, L., Giraldo, C. & Mann, W., 2003. “Enabling Location-Aware Pervasive Computing Applications for the Edlerly”. In PERCOM Proceedings of the

First IEEE International Conference on Pervasive Computing and Communications, IEEE Computer Society, pp. 531

[16]. Sohn, T., Griswold, W.G., Scott, J., LaMarca, A., Chawathe, Y., Smith, I. & Chen, M.Y., 2006. “Experiences with place lab: an open source toolkit for location-aware computing”. In Proceedings of the 28th international conference on Software engineering, Shanghai, China, ACM, pp. 462 – 471

[17]. Hightower, J., LaMarca, A. & Smith, I.E., 2006. “Practical Lessons from Place Lab”. In Pervasive Computing, IEEE, Vol. 5, Issue 3, IEEE Computer Society, pp. 32 – 39

[18]. Dey, A.K., Abowd, G.D. & Salber, D., 2001. “A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications”. In Human-Computer Interaction, vol. 16, Issue 2, L. Erlbaum Associates Inc., Hillsdale, NJ, USA, pp. 97 – 166

[19]. Salber, D., Dey, A.K. & Abowd, G.D., 1999. “The context toolkit: aiding the development of context-enabled applications”. In Conference on Human Factors in Computing Systems Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit, Pittsburgh, Pennsylvania, United States, pp. 434 – 441

[20]. Hong, J.I., 2002. “The context fabric: an infrastructure for context-aware computing”. In Conference on Human Factors in Computing Systems: CHI '02 extended abstracts on Human factors in computing systems, Minneapolis, Minnesota, USA, ACM, pp. 554 – 555

[21]. Want, R., Hopper, A., Falcão, V. & Gibbons, J., 1992. “The active badge location system”. In ACM Transactions on Information Systems (TOIS),Vol. 10, Issue 1, ACM, pp. 91 – 102

[22]. Bahl, P. & Padmanabhan, V.N., 2000. “RADAR: an in-building RF-based user location and tracking system”. In INFOCOM 2000, Proceedings of Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE, Vol. 2, pp. 775 – 784

[23]. “HSQLDB website”. http://www.hsqldb.org

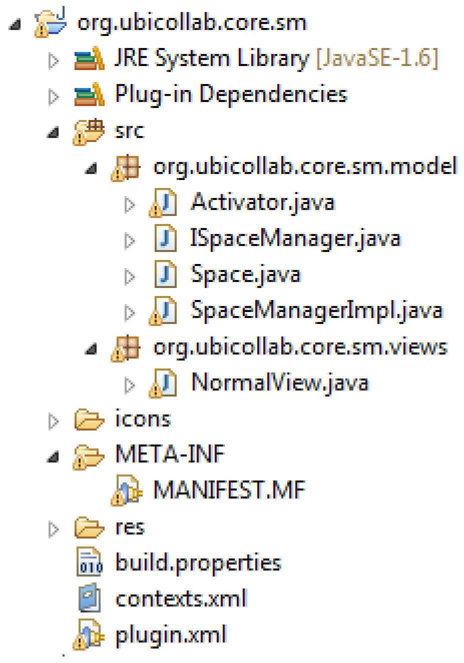
Appendix A. Acronyms

Acronym	Name
ANSI	American National Standards Institute
API	Application Programming Interface
BSSID	Basic Service Set Identifier
CSCW	Computer Supported Cooperative Work
CSL	Context Specification Language
FR	Functional Requirement
G	General
GPS	Global Position System
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
IDE	Integrated Development Environment
NFR	Non-functional Requirement
RCP	Rich Client Platform
SM	Space Manager
SOA	Service-Oriented Architecture
SQL	Structured Query Language
SSID	Service Set Identifier
UML	Unified Modeling Language
URI	Uniform Resource Identifier
WWW	World Wide Web

Appendix B. Java Packages

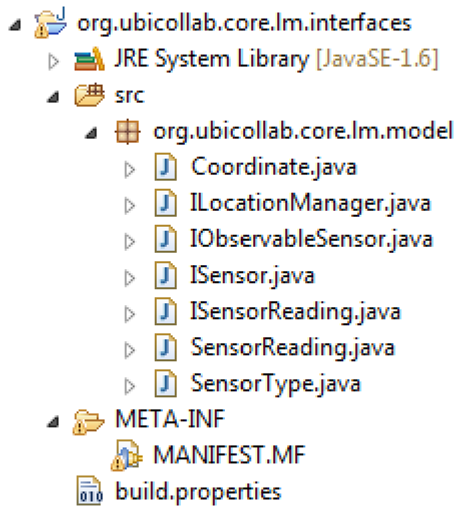
This appendix lists all the packages, interfaces and classes each packages contain, that are developed during the course of this project.

B.1 Space Manager



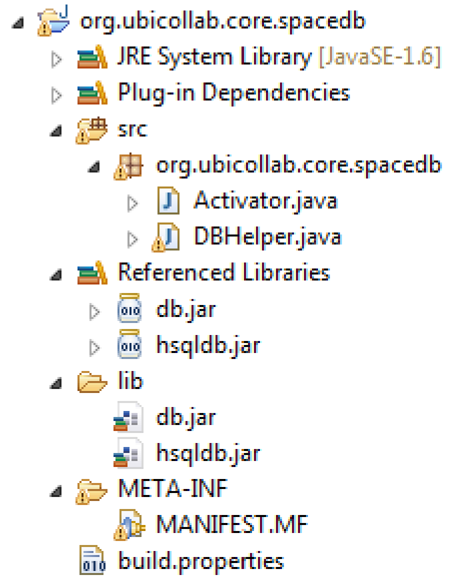
```
org.ubicollab.core.sm
    JRE System Library [JavaSE-1.6]
    Plug-in Dependencies
    src
        org.ubicollab.core.sm.model
            Activator.java
            ISpaceManager.java
            Space.java
            SpaceManagerImpl.java
        org.ubicollab.core.sm.views
            NormalView.java
    icons
    META-INF
        MANIFEST.MF
    res
    build.properties
    contexts.xml
    plugin.xml
```

B.2 Location Service Manager



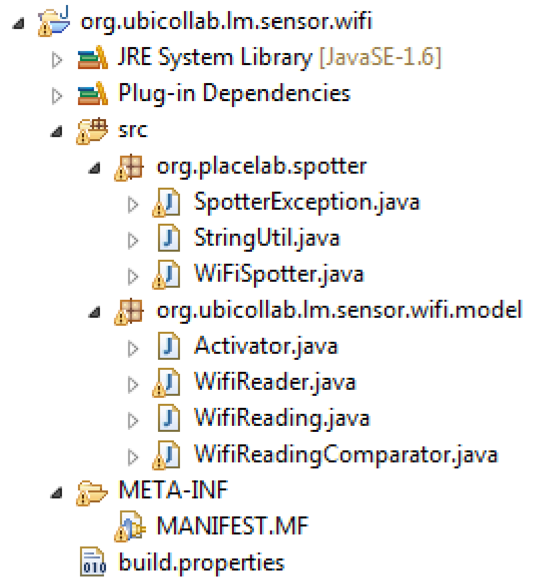
```
org.ubicollab.core.lm.interfaces
    JRE System Library [JavaSE-1.6]
    src
        org.ubicollab.core.lm.model
            Coordinate.java
            ILocationManager.java
            IObservableSensor.java
            ISensor.java
            ISensorReading.java
            SensorReading.java
            SensorType.java
    META-INF
        MANIFEST.MF
    build.properties
```

B.3 Space Database



```
org.ubicollab.core.spacedb
    JRE System Library [JavaSE-1.6]
    Plug-in Dependencies
    src
        org.ubicollab.core.spacedb
            Activator.java
            DBHelper.java
    Referenced Libraries
        db.jar
        hsqldb.jar
    lib
        db.jar
        hsqldb.jar
    META-INF
        MANIFEST.MF
    build.properties
```

B.4 Wi-Fi Location Sensor Plug-in



```
org.ubicollab.lm.sensor.wifi
    JRE System Library [JavaSE-1.6]
    Plug-in Dependencies
    src
        org.placelab.spotter
            SpotterException.java
            StringUtil.java
            WiFiSpotter.java
        org.ubicollab.lm.sensor.wifi.model
            Activator.java
            WifiReader.java
            WifiReading.java
            WifiReadingComparator.java
    META-INF
        MANIFEST.MF
    build.properties
```

Appendix C. Sample Space XML files

Sample Space XML file with location information

```
<?xml version="1.0" encoding="utf-8" ?>
<Space>
  <Id>001</Id>
  <Name>Vicki's Room</Name>
  <Description>G202</Description>
  <ParentSpaceId>-1</ParentSpaceId>
  <Sensors>
    <Sensor>
      <UniqueId>00:1e:e5:64:f1:15</UniqueId>
      <Name>linksys</Name>
      <Reading>-36</Reading>
      <SensorType>WiFi</SensorType>
    </Sensor>
    <Sensor>
      <UniqueId>00:0b:85:89:b9:fd</UniqueId>
      <Name>eduroam</Name>
      <Reading>-62</Reading>
      <SensorType>WiFi</SensorType>
    </Sensor>
    <Sensor>
      <UniqueId>00:0b:85:89:b9:ff</UniqueId>
      <Name>ntnu</Name>
      <Reading>-62</Reading>
      <SensorType>WiFi</SensorType>
    </Sensor>
  </Sensors>
</Space>
```

Sample Space XML file without location information

```
<?xml version="1.0" encoding="utf-8" ?>
<Space>
  <Id>001</Id>
  <Name>Vicki's Room</Name>
  <Description>G202</Description>
  <ParentSpaceId>-1</ParentSpaceId>
  <Sensors />
</Space>
```

Appendix D. Space database SQL Script

Table: Space

```
CREATE TABLE Space(
      Id int,
      Name varchar(20),
      Owner varchar(20),
      Description varchar(500),
      IsShareable bit,
      DateCreated TIMESTAMP
);
```

Table: Sensor

```
CREATE TABLE Sensor(
      SensorId int,
      SensorType varchar(20),
      UniqueIdentifier varchar(20),
      HumanReadableName varchar(20)
);
```

Table: SensorReading

```
CREATE TABLE SensorReading(
      ReadingId int,
      SpaceId int,
      SensorId int,
      Reading int,
      Coordinate varchar(500)
);
```

Table: CurrentSpace

```
CREATE TABLE CurrentSpace(
      SpaceId int,
      UpdatedDate TIMESTAMP
);
```