



Distributed Computing Lab
Department of Computer Science
Waseda University

A Document based Framework for User Centric Smart Object Systems

Fahim Kawsar

A Document based Framework for User Centric Smart Object Systems

A Dissertation submitted to the
Graduate School of Science and Engineering of Waseda University
in partial fulfillment of the requirements for the Degree
of
Doctor of Engineering in Computer Science

Fahim Kawsar
February 2009

Copyright © 2009 by Fahim Kawsar

"To my parents for giving me the faith to believe in myself."

Abstract

The proliferation of awareness technologies - sensors, actuators and perception algorithms - has created novel design opportunities for everyday objects, and has enabled designers to re-innovate the role of these well known artefacts with new affordances. Attaching awareness technologies to everyday objects makes them perceptive of their operational and situational context, which in turn enables them to provide value added services per se or via external applications. A smart object system encompasses the synergy between these computationally augmented smart objects and external applications and has emerged as one of the principal technologies to embrace the recent paradigm of people centric computing.

This research has focused on the development of a software framework for building smart object systems. The dissertation provides a theoretical foundation for smart object systems and presents a set of requirements and component abstractions for a supporting infrastructure. A document based framework is developed where applications' requirements and smart objects' services are objectified through structured documents. A runtime infrastructure provides the spontaneous federation between smart objects and applications through structural type matching of these documents. This allows externalizing smart object management and addressing heterogeneity issues away from the applications. The framework along with identified design processes reduces the complexity of building extensible smart object systems.

In addition, the dissertation at hand has also looked at what aspects of software architecture can manifest themselves as a part of the user experience. Particularly this work argues that by involving end-users in the deployment and administration of smart object systems, it is possible to elevate end-user experiences. Accordingly, this research exposes user centric architectural qualities by slightly shifting framework's focus onto end-users. Furthermore, the thesis reports a couple of usability studies where end-users were actively involved in deployment, configuration and management of several smart object systems. The implications of these studies illustrate the design considerations of building software architectures for human-centric smart object systems.

Acknowledgements

Tatsuo Nakajima has given me a unique opportunity in my academic career. First by accepting me in his laboratory and then by providing me with continuous supervision, adequate research facilities and the liberty to work independently. I would like to express my sincere gratitude to him. Thank you very much.

I owe many thanks to the rest of my thesis committee for their support and suggestions. Yoshiaki Fukazawa and Shigeki Goto have offered outstanding reviewing and advice on this work.

Perhaps this research would never be at this stage without Kaori Fujinami. He has guided me from the very beginning of this work by sharing interesting ideas, useful comments and valuable suggestions. I am really grateful to him for all his supports. Simo Hosio was actively involved in the later part of this work and was instrumental in reducing my work load significantly. Thank you very much Simo.

This work has been vastly improved due to constructive criticisms, and helpful suggestions from many different people. Especially, Liviu Iftode, Michael Beigl, Gerd Kortuem, Jukka Riekk, Achilles Kameas, Susanna Pirttikangas and Jin Nakazawa have guided me substantially to shape up the work presented in this thesis. Special thanks to Jong Hyuk Park for his continuous support and encouragement.

I would like to express my sincere appreciation to all members (past and present) of DCL, specially to Eiji Tokunaga, Hiroo Ishikawa, Tetsuo Yamabe and Kimura Hiroaki. Living in a different country alone is very difficult. My living in Japan has become much easier because of the friendly environment of DCL. Every time I had a problem, I found a helping hand in DCL, which I really appreciate and will cherish. Thanks to all of you wonderful people.

I have also been very fortunate to have a great group of friends at Waseda. This includes my Tatsumi friends, TIEC mates, and many others, too numerous to name. Thanks to all of you guys for dragging me away from work to have fun and to raise your hands always when I needed support. Special thanks to Wonny Tjon for helping me get past all the self-doubting that inevitably crops up in the course of a Ph.D.

Finally, I would like to dedicate this work to my parents. I consider myself extremely lucky for the family I have. My parents have provided me with immense liberty from the very early stage of my life. Perhaps this has always been my thrust to do best for the justification of my own decisions. Today, at this point I just want to say that it would be never possible for me to come here without your endless support and love from childhood to now. Thank you very much. Thanks to Liza Apa and Nahid Bhaia for their continuous support throughout my life. You people are really special to me. Thanks to my two little nieces, Maisha and Ramisha. It will take a while for you girls to read and understand this thesis but you two have been my inspiration always.

Contents

1	Introduction	1
1.1	Research Motivation	2
1.2	Research Focus	5
1.3	Projected Contributions	6
1.4	Dissertation Roadmap	7
2	Background: Smart Object Systems	9
2.1	The Vision: Ubiquitous Computing	9
2.1.1	Context and Context-Awareness	11
2.2	Smart Objects	12
2.2.1	Definition	12
2.2.2	Properties of Smart Objects	15
2.3	Exploration of Research Projects on Smart Objects	16
2.3.1	Digital Everyday Smart Objects	16
2.3.2	Non-Digital Everyday Smart Objects	17
2.3.2.1	Augmented Household Objects	18
2.3.2.2	Augmented Room and Building Structure	19
2.3.2.3	Augmented Objects in the Workplace	19
2.3.2.4	Electronic Tag Augmented Objects	20
2.3.2.5	Augmented Wearable Objects	21
2.3.3	Summary of the Existing Research	21
2.4	Classification of Smart Object Systems	22
2.5	Sensor Network and Smart Objects	24
2.6	Chapter Summary	25
3	A Framework for Smart Object Systems	27
3.1	Design Issues for Smart Objects	27
3.1.1	Design Requirements for a Smart Object Model	28
3.1.2	Related Work on Smart Object Model	31
3.1.3	A Core-Cloud Theoretical Model for Smart Objects	32
3.2	Design Issues for a Smart Object System Infrastructure	34
3.2.1	Design Requirement for a Smart Object System Infrastructure	35
3.2.2	Existing Support for a Smart Object System Infrastructure	37
3.2.2.1	Three Models of Architecture	37
3.2.2.2	Distributed Component and Device Integration Infrastructures	38
	DCOM and CORBA	38
	UPnP and Jini	39

SpeakEasy	39
XWeb	39
PatchPanel	40
InterPlay	40
3.2.2.3 Pervasive Computing Middlewares	40
Schilit's System Architecture	40
Context Toolkit	41
Technology for Enabling Awareness	41
Gaia	41
Aura	42
iROS	42
Java Context Aware Framework	42
Sentient Computing	43
HP CoolTown	43
Easy Living	43
Stick-e Note	43
Context Fabric	44
3.2.3 Drawbacks of Current Approaches	44
3.2.4 A Document Based Solution Framework	45
3.3 Framework Support for End-Users	48
3.3.1 Related Work on Supporting Tools for End-Users	50
3.4 Chapter Summary	51
4 Implementation of the Framework	53
4.1 Smart Object Wrapper	53
4.1.1 3-Step Design Methodology for Smart Object Augmentation	54
4.1.1.1 Illustration: Design of A Smart Mirror	57
4.1.2 Augmentation Presentation: Implementation of Core-Cloud Model	58
4.1.2.1 Core Component	59
4.1.2.2 Profile	60
4.1.3 Programming Model	60
4.1.4 Representative Documents	63
4.1.5 Location Modalities of Smart Object Wrapper	65
4.1.6 Smart Object Life Cycle	66
4.2 Application Development Process	67
4.2.1 3-Step Application Development Process	67
4.2.2 Programming Model	69
4.3 FedNet Infrastructure	71
4.3.1 Logical Architecture of FedNet	71
4.3.1.1 Smart Object Repository	72
4.3.1.2 Application Repository	72
4.3.1.3 FedNet Core	72
4.3.1.4 Access Point	73
4.3.2 Physical Architecture of FedNet: Distributed Management	73
4.3.3 Specific Features of FedNet	75
4.4 Framework Support for End-Users	76
4.4.1 End-User Interaction Tools	77

4.4.1.1	Graphical User Interface Interaction Tool	77
4.4.1.2	Tangible User Interface Interaction Tool	79
	Hardware	79
	Interaction Mechanism	79
4.5	Chapter Summary	81
5	Evaluation	83
5.1	Evaluation of the Framework	83
5.2	Quantitative Evaluation of the Framework	84
5.2.1	A Prototype Home Entertainment Smart Object System	84
5.2.1.1	A Scenario	84
5.2.1.2	Description of the Smart Object System	85
5.2.1.3	Quantitative Measurements	86
5.2.1.4	Summary of the Quantitative Evaluation	89
5.3	Qualitative Evaluation of the Framework	89
5.3.1	Revisiting the Smart Object Design Factors	89
5.3.2	Revisiting the Infrastructure Design Factors	91
5.4	Evaluation of End-User Aspects through User Study	93
5.4.1	Two Sample Smart Object Systems	94
5.4.1.1	A Scenario	94
5.4.1.2	Descriptions of the Smart Object Systems	94
5.4.2	Study Methodology	96
5.4.2.1	Participants	97
5.4.2.2	Study Sessions	97
5.4.3	Study Results	98
5.4.3.1	System Performance	98
5.4.3.2	End-Users' Performance	99
5.4.4	Implications of the User Study	101
5.5	Chapter Summary	104
6	Discussion	105
6.1	Cross Domain Applications	105
6.1.1	Distributed Component Systems	106
6.1.2	Peer to Peer Computing	106
6.1.3	Service Oriented Computing	106
6.2	Further Look at Design Aspects	107
6.2.1	High Level Abstractions	107
6.2.2	Separation of Concerns	108
6.2.3	Interface and Protocol	108
6.2.4	Simplicity and Features	109
6.2.5	Some Notes on Evaluation	109
6.3	Further Look at End-User Aspects	110
6.4	Reactive or Proactive	111
6.5	Chapter Summary	112
7	Conclusions	113
7.1	Research Summary	113

7.2	Future Research Directions	115
7.2.1	Specification and Description of Smart Object Services	115
7.2.2	Integration of Location Information	116
7.2.3	Incorporating Security Aspect	116
7.2.4	Architectural Qualities for Improving User Experience	117
7.2.5	End-User Tools	117
7.3	Concluding Remark	117
A	Document Type Definition (DTD)	119
A.1	Document Type Definition for Smart Object's Documents	119
A.1.1	Document Type Definition for Smart Object Description Document	119
A.1.2	Document Type Definition for Profile Description Documents	119
A.2	Document Type Definition for Application's Document	121
B	Specification of Programming Interface	123
B.1	Language Interface for Smart Object Development	123
	Profile Class	124
B.2	Language Interface for Application Development	125
	Task Class	125
	AccessPoint Class	125
	XmlProcessor Class	126
C	User Study Material	127
C.1	Post-study Questionnaire	127
C.2	Post-study Interview Questions	129
	Bibliography	131

List of Figures

1.1	Evolution of Personal Computers and Mobile Phones	3
1.2	End-user Role in Personal Computers	3
1.3	Ideal Smart Object System Model	4
2.1	Classification of Smart Object Systems	23
3.1	Affordances of Physical Artefacts	29
3.2	Example of Smart Objects	30
3.3	Core-Cloud Model for Smart Objects	33
3.4	A Conceptual Document based Framework	46
3.5	A Conceptual Document based Framework with End-User Tool	50
4.1	3-Step Design Methodology for Augmentation Role Selection	55
4.2	Construction of a Smart Mirror	57
4.3	Smart Object Wrapper Architecture	58
4.4	Sample Code implementing Sensor Type Profile	61
4.5	Sample Code implementing Actuator Type Profile	62
4.6	Smart Object Description Document	63
4.7	Profile Description Document - For Sensors	64
4.8	Profile Description Document - For Actuators	65
4.9	Location Modalities of Smart Object Wrapper	65
4.10	Life Cycle of a Smart Object	66
4.11	3-Step Application Development Process	67
4.12	Task Description Document	69
4.13	Sample Task based Application Code	70
4.14	FedNet Architecture	72
4.15	Internal Structure of Smart Object Repository	74
4.16	Snapshots of Graphical User Interface Interaction Tool	78
4.17	Tangible User Interface Interaction Tool	80
5.1	Bootstrapping and Access Point Formation Time	87
5.2	Communication Latency	88
5.3	Sample Application Code	91
5.4	Apparatus for the User Studies	95
5.5	User Study Sessions	98
5.6	User Study Results with Graphical User Interface Tool	100
5.7	User Study Results with Tangible User Interface Tool	101
5.8	Subjective Responses	101

A.1	Document Type Definition for Smart Object Description Document	120
A.2	Document Type Definition for Profile Description Document	120
A.3	Document Type Definition of Task Description Document	121
B.1	Structure of a Profile	124

Chapter 1

Introduction

Chapter 1

Introduction

The relation between humans and everyday objects is long standing. Historically, everyday objects have established purposes and have played vital roles in our everyday lives. The recent progression of computing technologies affords us to rethink the roles of these everyday objects and in fact poses us with interesting research and design opportunities to instrument them in a way that dramatically enhances their well established features.

The notion of "Ubiquitous Computing" was introduced by Weiser [Weiser, 1991] to describe a scenario in which, literally, computing is everywhere. This should not be taken in the narrow-minded sense of "*a computer on every desk*", but in the rather subtler one of computers becoming embedded in everyday objects, e.g., a chair, a mirror, a desk, a lamp, a cup etc. This embedding has two direct implications. Firstly, it enables computing devices to pervade in our everyday life in such a way that people do not notice them anymore as separate entities as they become a part of our environment. This blending allows them to operate in the periphery of human, enabling people to focus on the primary task at hand [Weiser and Brown, 1997]. Secondly, it creates novel design opportunities for everyday objects. Instrumenting everyday objects with computing enable designers to re-innovate the purpose of these well known objects and to extract new affordances from them to make them "*smarter*". Part of this vision is already becoming a reality - as the convergence of pervasive technologies (e.g. miniaturization of the micro processors and proliferation of wireless internet, short-range radio connectivity, real time localization, sensor networking etc.) results in the integration of processors and tiny sensors into everyday objects and fabric of the environment. We now interact with our clothes to exchange emotion [Baurley et al., 2007], with our furniture for personalized information services [Tokuda et al., 2004, Fujinami et al., 2005], with our umbrellas for weather forecast¹, etc. Information technology rich everyday objects and spaces are radically changing the style of how we learn, think, interact and behave as social human beings.

At the other end of the spectrum, the emergence of these instrumented smart objects challenges us in multiple aspects. As these everyday objects are appearing as computer aided

¹<http://www.ambientdevices.com/products/umbrella.html>

smart objects, issues such as design methodology and usability seek a new rational approach. It is important to understand the design rationales of smart objects: what augmentation is suitable for a traditional everyday object, what are the ways to instrument them, what are the ways to describe the instrumented features and how to interact with them. Everyday objects' form and interaction factors have evolved over the years and have intrinsic perceptual properties. Thus, it is necessary to keep the balance between their contemporary roles and newly augmented features. Simultaneously, down the stream issues like development model, abstraction, integration and universal interoperability pose significant research challenges. It is essential to understand how to build applications for smart objects, how to integrate multiple smart objects into one or more applications and how to form synergy among them.

The dissertation at hand looked at these system issues for building smart object systems. Primarily, this research provides a theoretical foundation and discusses the design processes and programming models for developing smart object systems. A document centric framework is designed for building reusable and extensible smart object systems with necessary infrastructure. A number of smart objects and applications are prototyped to evaluate the feasibility and applicability of the approach. The dissertation also discusses what aspects of the framework could be part of the user experience. Specifically, this work argues that by involving end-users in the administration of smart object systems, it is possible to elevate user experiences. A couple of user studies are reported to expose how can we involve end-users in the deployment, configuration and management of smart object systems in a generic manner.

1.1 Research Motivation

After the formal proposition by Weiser [Weiser, 1991] in early 90's, the ubiquitous computing has progressed significantly from the technology perspectives. Tagging everyday objects with sensors, actuators and building an instrumented environment are recent practices in industry and academia. In fact, the smart object (i.e., everyday objects augmented with computing to provide value added services.) domain has matured over the years. The combination of Internet and technologies like near field communications, real time localization, sensor networking etc. are bringing smart objects into commercial use. Several successful prototypes and applications have already demonstrated and deployed. However, the lack of commonality among the design principles and the underlying infrastructures of these projects are hindering the exciting future of smart object systems. The primary reason behind this phenomenon is missing rationales for the design and integration of smart objects. Although the processor and communication technology bases are quite solidified in this domain, software architectural aspects like development process, infrastructure platform, programming methodology are still widely open issues. The prime motivation of this research work is to look at these software architecture aspects for developing smart objects and applications for them.

Historically, the evolution of computing in the last few decades could be seen as the prime indicator for laying out the software architectural research challenges for smart object systems.

In the early 80's when the personal computer was first introduced, it was a stand-alone device with a few bundled software. Those machines offered little extension supports and a very few third party applications were available to them. However, with the advent of general purpose operating systems, the explosion of Internet and the proliferation of low cost hardwares, more companies are now involved in developing innovative softwares and manufacturing personal computers and peripheral accessories (Figure 1.1).



FIGURE 1.1: Evolution of Personal Computers and Mobile Phones (Image Courtesy to Google)

The end-users' role has also changed in parallel. Now average computer users can buy a software or download an application from Internet and can install it in their personal computers easily. Furthermore, they can attach peripherals (e.g., a video camera, a microphone, etc.) seamlessly to get the fullest functionalities from their desired applications as depicted in Figure 1.2.

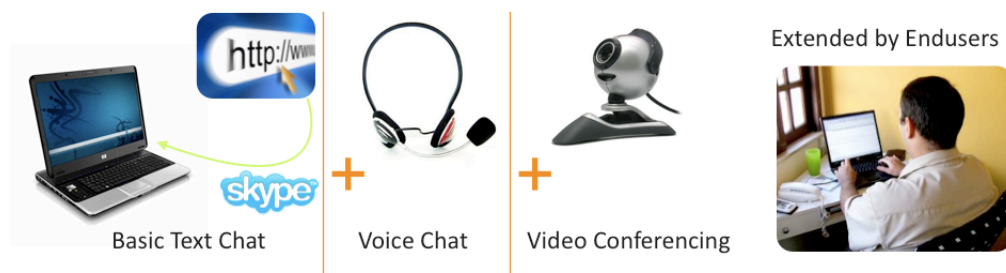


FIGURE 1.2: End-user Role in Personal Computers(Image Courtesy to Google). By incrementally adding new peripherals in a plug and play manner end-users can extend the basic functionalities of an application.

We are seeing the same phenomenon in mobile computing too. In the early 90's when the mobile phone was introduced, it was a mere cordless phone. However, as the network infrastructures improved and market became saturated, mobile phones have undergone a metamorphosis. It has become a fashion accessory, a mobile office, a music player, a camera, etc. With the release of various open development toolkit, we are experiencing thousands of innovative third party applications for mobile phones (Figure 1.1). Most importantly the end-users

are actually involved in transferring their mobile phone into a multipurpose device: they can now download and install their desired applications with a single touch and can extend the basic functionalities of their mobile phones.

This research argues that to mature the notion of smart objects and to bring them out of the laboratory, smart object systems have to go through the same metamorphosis as personal computer and mobile phone did. It is imperative to have common design and development methodologies to develop reusable and extensible smart objects and applications for them independently following the application model of personal computers and mobile phones. The methodologies should enable a smart object to be instrumented in multiple ways for serving multiple purposes depending on the instrumentation variations and applications that run on it. Similarly applications should be developed in such a way that they could be installed in any compatible smart objects.

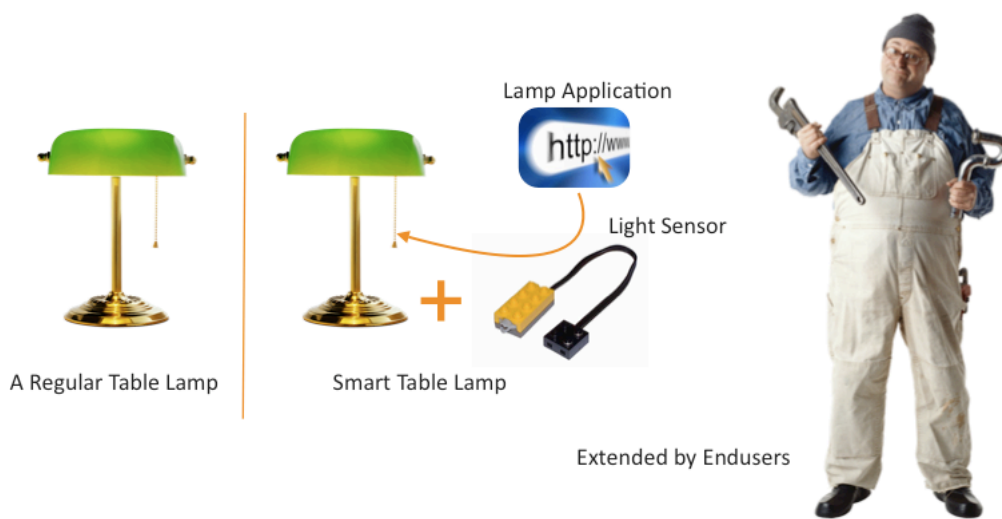


FIGURE 1.3: Ideal Smart Object System Model. Any suitable application can be added to a compatible smart object at any time; Smart object's features can be extended by adding new peripherals; These installations and extensions are carried out by end-users.

Considering smart objects are parts of our environment, they should retain their physical properties and interaction metaphor. These objects should be easy to setup, adaptive to users' needs, and interchangeable with new models. Ideally, smart objects should be identical to existing home appliances e.g., a table lamp, a dish washer, a TV etc. A user may buy one or multiple physical smart objects and applications for them and should be able to install them very easily just like other home appliances. In addition, the user should be able to incrementally enhance the smart object functionalities by upgrading its features or installing new applications. Consider a hypothetical table lamp application that proactively turns the lamp on and adjusts its brightness adapting ambient room lighting. A user can initially buy a regular lamp, and few weeks later he/she can buy a light sensor, attach it to the lamp and download this application into the lamp to make it proactive (Figure 1.3). There are several advantages

in involving end users to build smart object systems in this fashion as observed by Beckmann and his colleagues including less cost, greater user-centric control, more acceptance, better personalization and frequent upgrade support [Beckmann et al., 2004]. Furthermore, learning from the evolution of the personal computers and mobile phones, it could be easily concurred that such end-user involvement is a crucial success factor for smart object systems.

Enabling end-users to be involved in the deployment, extension and maintenances processes of smart object systems can essentially elevate user experiences as they become more intimate with the system. Thus it is important to provide simple, easy, useful and usable interaction tools that can assist end-users in performing smart object administration tasks. However, such involvement also requires specific design methodologies (e.g., plug and play smart objects, loose coupling between the applications and smart objects, etc.) to be addressed at the development phase of the smart object system. Thus, a software framework for supporting the development of smart object systems has to be carefully designed so that it ensures the system level design requirements as well as requirements for supporting end-user involvement in a balanced way. These requirements motivate this research work to focus on the development of common design methodologies with a suitable infrastructure and end-user tools for building user-centric smart object systems.

1.2 Research Focus

There are a large number of research problems in the context of smart object systems. This work specifically looked at a subset of these problems and explored the system design issues related to smart object: how an effective infrastructure for smart object system can be designed, and how it will appear to developers through an effective programming model. One intriguing factor of this research work is bringing end-user aspects in the design of the framework, i.e., what aspects of software architecture can manifest them as a part of the user experience.

The research questions raised above have lead to the following thesis statement:

"Providing a software framework with design methodologies and programming abstractions for building user-centric extensible smart object systems"

Consequently, this work is centered around three aspects:

- Development of a theoretical model for smart objects considering the identified design requirements.
- Development of a framework for building reusable and extensible smart object systems with effective programming model.

- Investigation of specific quality attributes of a software framework that can elevate user experiences with smart object systems.

Some previous research has addressed certain aspects in isolation or in relevant domains, e.g., a range of pervasive middleware infrastructures are available in the field. This work is the first to develop a theoretical model for smart objects and to address an infrastructure specifically designed considering the requirements imposed by the nature of smart object systems. In addition, this is the first work that looks at the end-user deployment and administration aspects both from architectural and interaction tool perspectives and exposes several design factors for building human-centric smart object systems.

The work presented in this thesis has followed a bottom up iterative approach. A prototype framework initiated the research, which was redesigned thrice during the course of this research period.

1.3 Projected Contributions

This research contributes to the field in a number of ways. The expected contributions of this thesis are

1. An in-depth investigation of a range of smart object systems to formalize their design rationales resulting in the development of theoretical model for a smart object.
2. Identification of the requirements to support the development and evolution of smart object systems and consequent development of a document centric framework along with design processes for building reusable and extensible smart objects systems.
3. The concrete implementation of the framework and demonstration of its utilization in a variety of smart object systems.
4. Introduction of the novel notion of end-user centric qualities for system infrastructure and their implications in the design processes.

The first and general contribution is an investigation of existing works in the the domain of smart object systems in an attempt to generalize their characteristics and properties. This work explores the design methodologies and requirements for the construction of reusable, extensible and plug and play smart objects and their integrations in generic context-aware applications. These exploration and formalization of the design processes are applied to develop a theoretical model for smart objects.

The second and third contributions and indeed the main contributions of this work are an in-depth investigation of a suitable framework for building user-centric smart object systems. The goal is to provide a suitable system model and programing abstractions using which reusable,

extensible and plug and play smart object systems can be built. Thus the research focused on three aspects of the framework: an appropriate system model for representing smart objects, a suitable infrastructure for the application developers that enable them to integrate smart objects in their systems, and an effective programming model for both the smart object and application developers. The combination of these aspects will result in faster and rapid development of smart object systems. Furthermore, the programming abstractions built on top the framework will allow the developers to design their systems in a highly structured manner.

The final contribution is a novel one and arguably the first in the field that tries to look what aspects of a system framework could elevate end-user experiences. Two end-user interaction tools are developed as framework services that allow end-users to involve in the deployment, management, administration and extension of smart object systems. The usability studies on these tools and the entire process of end-user involvement expose several significant issues both for systems designers and interface designers for future smart object systems including the evidence of the fact that the end-users might be involved in deploying and administrating future smart object systems if appropriate interaction tools and supporting infrastructures are provided.

1.4 Dissertation Roadmap

The rest of this dissertation is organized as follows.

Chapter 2 provides the detail background, classification and characteristics of smart object systems. This chapter provides an in-depth discussion on existing works to extract the characteristics across variety of smart object systems. This discussion provides the foundation for developing a theoretical model of a smart object and for identifying the architectural requirements to build smart object systems.

Chapter 3 introduces the design issues for building user centric smart object systems. A theoretical core-cloud model for smart object is presented. This is followed by the discussion on the architectural requirements for building smart object systems. Consequently, a document based framework is presented explaining basic design decisions and by highlighting the limitations of existing infrastructures. The chapter then discusses the architectural requirements for supporting end-user involvement in the construction and administration of smart objects systems and explains in detail how the proposed framework design supports those requirements.

Chapter 4 presents the implementation detail of the proposed framework. Each of the components of the framework is discussed with concrete illustrations. The chapter ends by discussing multiple end-user interaction tools built as framework services to engage end-users in the construction and administration of smart object systems.

Chapter 5 provides the evaluation of the proposed framework from quantitative and qualitative aspects. The performance of the framework and quality of service issues are discussed. After that the chapter moves to the qualitative evaluation of the framework by revisiting the design requirements. Finally, the end-user interaction tools and the entire end-user involvement process are assessed through a couple of user trials involving end-users in building and enhancing smart object systems.

Chapter 6 puts forth few issues for further discussion including the design decisions that are considered in the current framework. The applicability of the proposed approach in other domains, and the design implications of such human-centric approach towards system research are also discussed.

Finally, chapter 7 summarizes the research of this dissertation. This chapter presents a set of conclusions and research contributions of the presented work. The chapter ends by outlining some potentially future research work.

Chapter 2

Background: Smart Object Systems

Chapter 2

Background: Smart Object Systems

The design of a new software framework for building a smart object system naturally leverages off the characteristics of a smart object system and its constituents. To have a formidable understanding, this chapter discusses the background, notion and properties of smart object systems by exploring the contemporary research. A range of projects will be introduced to rationalize the features that are common and useful across smart object systems. This discussion will provide the foundation to extract the design requirements for a smart object system and a supporting framework.

2.1 The Vision: Ubiquitous Computing

Mark Weiser put forth the vision of Ubiquitous Computing in his article "The Computer for the 21st Century" published in 1991 [Weiser, 1991] .

"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."

The above statement instigated a paradigm shift in computing by opening a door to a new way of thinking about computing technology. The vision is that - ubiquitous computing will change the way we currently perceive technology. There will be no more desktop computers but computing will be all around us to match our social behavior and will guide our everyday activities. Computers will slowly disappear from our perception and will operate at the periphery of human as a part of natural environment much like the information presented today by clock, poster, street sign, etc. This will enable us to focus on our primary tasks, and to switch our attention to computing only when it is appropriate. The central issue here is that how can we make computer disappear from our perception? Weiser pointed out:

"Such a disappearance is a fundamental consequence not of technology, but of human psychology. Whenever people learn something sufficiently well, they cease to be aware of it." [Weiser, 1991].

This leads to the implication that, making computers disappear is about the deployment of technology in everyday environment in a way that humans do not realize them as a separate entities anymore. One obvious way to realize this is by making computers smaller in such a scale that they could be easily embedded into everyday objects that form the natural human environment. The proliferation of micro electronics in fact enables us to do that. Moore's Law [Moore, 1965], drawn up in 1965 which states that the power of microprocessors doubles about every 18 months, has held true with astonishing accuracy and consistency. Similar convergence has been observed in storage capacity and communications bandwidth resulting in the computer to be much more powerful, smaller, and cheaper. A direct consequence of this technology trend is the integration of computing in the fabric of the environment and the emergence of instrumented everyday objects (so called "*smart objects*") to form intelligent home, intelligent office, intelligent bus stop, etc. This instrumentation emerged as a natural choice to realize the vision of ubiquitous computing. Instead of deploying new devices of varying size and scale in the environment, it is more cost-effective, useful and innovative to integrate computing into the already available artefacts of our surroundings, thus making our connection with digital world a lot closer. Beigl, Gellersen and Schmidt put this in a classical way:

"Computers are becoming ubiquitous in our everyday lives but not as the computers that we know. The computer that we know is a primary artefact, explicitly perceived and used as computer. Instead, the computers that will proliferate further into our everyday lives will mostly be secondary artefacts embedded in primary artefacts that have their own established appearance, purpose and use in everyday experience." [Beigl et al., 2001].

Don Norman looked at this paradigm shift from design and human interface perspectives in his theme of "*Invisible Computer*" and induced the notion of human-centric "*Information Appliances*" integrated into the environment [Norman, 1998].

"The proper way, I argue is through the user-centered, human-centered, humane technology of appliances where the technology of the computer disappears behind the scenes into task-specific devices that maintain all the power without difficulties." [Norman, 1998].

Although he emphasized primarily on the home appliance designed and built to support a specific task, his observations raised interesting design challenges for the emergence of instrumented everyday objects. The human-centric approach is indeed a fundamental design

challenge of ubiquitous computing. As micro-scale computers are being embedded into the fabric of the environment, it is essential to ensure that this embedding is discreet, unobtrusive and matches the social behavior of human. Henceforth, significant research efforts focused on finding the proper balance among ease of interaction, perceptual complexity, computational overload and social acceptability to make sure that computing services are provided in an appropriate way, i.e. considering proper timing, location, identity, intuitiveness and other contextual attributes. Context-awareness is used to denote this property of computing and plays the key role in designing the human-centric ubiquitous computing systems.

2.1.1 Context and Context-Awareness

Context has been in the spot light of the research community with the emergence of ubiquitous computing paradigm, primarily because it summarizes the rudimentary postulates of human-centric invisible computing. There are dedicated works, for instance the Ph.D. thesis in the field [Schilit, 1995, Dey, 2000, Pascoe, 2001, Schmidt, 2002] that defined and explained the term "context" in the viewpoint of ubiquitous computing. All of these definitions are related and correct in their own contexts. However for the sake of discussion, the definition of context and context-awareness given by Anind Dey [Dey, 2000] is highlighted here. He defined 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 application themselves." [Dey, 2000].

The key point of this application oriented definition is that context is always bound to an entity and any information could be context as long as it is relevant to that entity. He goes on defining context-awareness as:

"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task." [Dey, 2000].

This definition says that any system can be context aware if it offers services utilizing the context relevant to the user of the system. This is a very crucial attribute of human-centric systems and plays a vital role in realizing the vision of ubiquitous computing, i.e., collecting context information unobtrusively from the intelligent environment and in turn utilizing those information to provide value added services to guide our everyday life. The next section provides a deeper insight into the building blocks of this instrumented intelligent environment, i.e., the smart objects that form the foundation for the proliferation of context-aware computing.

2.2 Smart Objects

The *Oxford American Dictionary* defines the terms *Smart* as "Having intelligence" and *Object* as "A material thing that can be seen and touched". Combining them, it can be inferred that a smart object literally means a tangible physical object with some sort of intelligence where the intelligence comes from explicit augmentation of computing device. Although the tangible feature of a smart object is well agreed, the degree of its intelligence is still an open topic for argument among the researchers. Some have seen this intelligence in physical object being aware of its situation and capable of sharing its awareness, where others have seen it in objects being autonomous and capable of taking self directed actions. As a result of this most researchers have a general idea of what a smart object is and what are its properties. However, due to this vague notion of smart object, there is no specific design guidelines for building a smart object, or reusing the design process, or using smart objects in applications in a unified way. These in turn hinder the development of a supporting framework for smart objects and their applications. A better understanding of smart object will enable designers and developers to overcome these difficulties. The following sections first introduce the definition of smart object by considering several contemporary definitions and then discuss the properties that a typical smart object exhibits.

2.2.1 Definition

In the work that first introduce non-digital smart object, Beigl et al. defined smart object as: *"An everyday artefact augmented with computing and communication, enabling it to establish and exchange information about itself with other digital artefacts and/or computer applications."* [Beigl et al., 2001]. The key point of their definition is that the artefacts are augmented to collect and share context information instead of utilizing those context to make autonomous decisions. Mattern concurred with these features but stressed that - *"Smart objects might be able not only to communicate with people and other smart objects, but also to discover where they are, which other objects are in their vicinity, and what has happened to them in the past."* [Mattern, 2003]. The prime point of his definition is the stateful feature of the objects capable of tracking their interaction history on top of their self awareness and sharing capabilities. A similar definition was proposed by Kortuem et al. by defining smart objects as: *"Objects of our everyday lives, augmented with information technology and equipped with sensing, computation and communication capabilities, that are able to perceive and interact with their environment and with other smart objects."* [Kortuem et al., 2007]. These definitions primarily emphasize on capturing and communicating context and letting other entities to exploit those contexts (e.g. a context-aware application) while retaining physical object's original use and appearance.

Interestingly, these definitions largely focus on smart objects' capabilities in sensing and sharing only without considering their role in reflecting action back into the real world, i.e., presenting output, taking autonomous decisions to provide proactive services or adapting its

behavior. One obvious issue is the form-factor of these everyday objects for such actuation services. Everyday objects evolved over the years to reach in their current shape and size. Thus, actuation technology has to be very carefully designed so that the augmentation does not modify their original appearance and interaction metaphor. However, smart objects not necessarily mean only the non-digital everyday objects. Information appliances and traditional computing devices, such as mobile phone, PDA, mobile music player, etc. could also be augmented with awareness technologies - sensors, perception algorithm, actuators - to provide value added services. As Norman pointed out while expressing his information appliance model:

"Making a proper information appliance has two requirements: the tool must fit the task and there must be a universal communication and sharing." [Norman, 1998].

So, it is essential for smart objects to consider the actuation technology as a part of the augmentation where applicable. In fact numerous research projects have augmented mobile devices for context-awareness [Rekimoto, 1996, Hinckley et al., 2000, Hinckley et al., 2000]. Siegemund considered this reflective nature in his definition as: *"A smart object can perceive its environment through sensors and communicates wirelessly with other objects in its vicinity. Given these capabilities, smart objects can collaboratively determine the situational context of nearby users and adapt application behavior accordingly."* [Siegemund, 2004]. Unlike the other artefact-centric definitions, Siegemund also emphasized the role of collaboration among multiple smart objects.

Streitz et. al. looked at the smart object from two distinctive perspectives: one model is *system-oriented, importunate smartness* where smart objects can take certain self-directed actions based on previously collected information and the other model is *people-oriented, empowering smartness* where smart objects empowers users to make decisions and take mature and responsible actions [Streitz et al., 2005]. The former model could be aligned with Siegemund's definition, however the latter model could be interpreted as a design rationale that aims to keep the user engaged and in control whenever possible, so autonomy is not a desired feature for this class of smart objects.

Each of the above definitions have covered several quality attributes that define smart objects. However, none of these definitions have touched two aspects of smart objects: first is the granularity of the smartness and second is the augmentation of the human perception.

Let us look at the granularity issue first. Want et al. augmented physical objects with passive RFID tags so that they can be uniquely identified and information related to them can be presented to their users [Want et al., 1999]. Similarly RFID tagged objects have been used in supply chain management and other enterprise applications [Konomi and Roussos, 2006]. Typically for these objects, intelligence such as perception, reasoning and decision-making are allocated at the infrastructure with appropriate tracking mechanism. So, can we consider

these objects as "smart" and self-aware? Of course in more sophisticated smart objects perception is integrated into the object itself. The point here is that even though these tagged objects do not have any self-awareness per se, they could still give the impression of being smart through some intermediary. For example, a pack of cookies in the super market could be embedded with a RFID tag containing a specific Internet address as digital information. If a user can read this tag using his RFID reader equipped mobile phone by touching the cookie pack, the mobile phone can independently access and display the associated information from the Internet. The user will have the impression that the cookie pack itself has transmitted the information, although in fact it has been supplied by the mobile phone via internet. This work argues that *it is not necessary for a smart object to be equipped with active awareness technology capable of sensing and actuating per se, rather the notion of smart object covers much lower granularity of computational unit, i.e., digital data*. As long as digital data can be associated with a tangible object which augments its original purpose, it could be said that the object is smart. In such cases, the self-awareness and sociality of the smart object could be achieved passively via the utilization of a secondary infrastructure.

The second issue is more related to our understanding of everyday objects. Simply speaking, would we be considering an augmented everyday object (i.e., a smart chair, a smart table, a smart coffee cup etc.) to be smart if from the very first day of its existence in our life it were capable of playing those "smart" roles. Probably no. It would be called a regular everyday object and the computation in it will never be visible to us, just like the regular electronic appliances in our home, e.g., a microwave oven, a washing machine etc. Ideally, a smart object should be conceptualized by humans in terms of the familiar everyday objects they are based on and the finest smart object design would make us forget about its computational role completely thus letting it to operate at our periphery. But, at the same time it is important to understand and perceptually aware of the capabilities of a smart object. This is particularly important for failing situations. Consider a regular chair, if one of its legs is broken, it is easily perceived by us because we are completely aware of its functionalities. What if the chair is augmented with some sensors (so that it can understand someone is sitting on it) and one of the sensors is not working thus effecting its functional output. Without an explicit understanding of the augmented capability of the chair, it is impossible to consciously perceive this failing situation. It is thus imperative to have perceptual awareness of the augmented features of smart objects. This work argues that *for any everyday object to be considered as smart it is essential that it augments human perception to some scale*. The success of a smart object design depends on minimizing this increase in the human perceptual complexity while retaining the physical appearance and interaction metaphor as close as possible to those of the original one.

With these rationales and considering the previous definitions, this work puts forth the following definition of a smart object:

"A computationally instrumented tangible object with an established purpose that augments human perception, and is aware of its operational situations and capable of providing supplementary services without compromising its original appearance and interaction metaphor

significantly. Supplementary services typically include sharing object's situational awareness and state of use; supporting proactive and reactive information delivery, actuation and adaptive state transition."

This definition is chosen with three primary attributes: *perceptual augmentation*, *device-centric situational-awareness* and *supplementary services*. The *perceptual augmentation* is directly proportional to the degree of instrumentation that change the original appearance. *Situational-awareness* is centered around the object itself, i.e., the context associated with the smart object. Finally, a generic term, *supplementary services* is chosen to encapsulate a broad range of smart object services that can be afforded through corresponding augmentations.

2.2.2 Properties of Smart Objects

Considering the definition proposed above, it is expected that a typical smart object should possess the combination of the following properties.

- **Unique ID:** Each smart object is considered to have a digital presence, so it is essential to uniquely identify a smart object in the digital world. Hence, each smart object must have a unique ID. This ID could be the network interface address or other application specific high level naming with appropriate resolution scheme.
- **Self Awareness:** Typically a smart object is augmented with awareness technology, thus it is expected that a smart object should be capable of knowing its operational and situational states and should be able to describe itself. For some smart objects (e.g., RFID tagged objects), this self-awareness might be provided by a secondary infrastructure.
- **Sociality:** A smart object should be capable of communicating with other smart objects and computing entities (e.g., a context-aware application) to share its self-awareness. This sociality could be passive where computing entities can track the smart objects to collect their awareness information.
- **Autonomy:** A smart object can have the capability to take certain actions. These proactive actions could be as simple as changing its operational state (e.g., switching to off state from on state) or as complex as adapting its behavior by autonomous decision making, actions plans for self healing, self organizing and self sustainable. The degree of autonomy depends on the type of smart object and its underlying environment.
- **State-fulness:** A smart object can maintain local memory to manage its states. This memory can also contain static object models that provide general description of the object, dynamic annotations added by the user or an application, and historic information about an object's former states or uses [Schneider, 2007]. However, the granularity and locality of this memory varies with the type of augmentations.

From human perception perspective, a smart object is expected to design with the following properties.

- **Preservation of Original Appearance and Functionalities:** A smart object must retain its original functionalities and appearances as much as possible. Physical objects evolved over the years in physical appearance and acceptances by the end-users. So, augmentation should extend their physical usages and it is mandatory to decouple the augmented features of smart objects from their original ones. A smart object must support its original functions even if the augmented electronic parts are dead. For example: a smart mirror is a mirror in the first case, we can augment the mirror with a display to show personalized information. This display functionality is within the scope of its primary role of reflecting peoples image and even if the display functionality is failed the mirror is capable of reflection.
- **Natural Interaction for Unobtrusive Realization:** The required interaction with a smart object should be identical to that of the original object. Humans develop a mental model regarding the usage of everyday objects. It is essential to adhere this model while augmenting a physical objects to ensure that the instrumentation is implicit and does not require additional interaction. The interaction should be natural and should activate the cognitive and cybernetic dynamics that people commonly experience in real life, thus persuading them that they are not dealing with abstract, digital objects but with physical real objects. This results in a reduction of the cognitive load, thus increasing the amount of attention on content.

In the next section, a number of research projects that carried out interesting works on smart objects are discussed in an attempt to rationalize the type of smart objects and their applications.

2.3 Exploration of Research Projects on Smart Objects

In a wide range of projects, everyday objects have been computationally augmented to provide supplementary smart services. Considering there are both digital and not digital objects that from the everyday human environment, the augmentation could be observed from two abstract perspectives: augmented digital everyday objects and augmented non-digital everyday objects.

2.3.1 Digital Everyday Smart Objects

The TEA Project from TecO investigated Technologies for Enabling Awareness and their application in mobile telephony to make personal mobile device smarter [Gellersen et al., 2000].

Their hypothesis was that the more a device know about its user, its environment and the situation in which it operates the better in can provide assistance. Their initial prototype augmented a cellphone with an awareness add-on device with a range of on-board sensors. A three layer perception algorithm was applied to generate meaningful context about the user out of raw sensor data (i.e., "in a meeting", "user is walking", etc.) to provide proactive services to the mobile users. They showed that embedding such sensor boards in everyday objects with a suitable perception algorithm could be used to collect rich context information about users which in turn could be used to make the device smarter.

In handheld and mobile computing, several researchers have augmented personal mobile devices with various kind of sensing technology for value added features. Rekimoto added tilt sensors to a handheld to obtain context regarding the handling of the device [Rekimoto, 1996]. Hinckley et al. augmented a PDA with tilt, touch and proximity sensors to obtain operational context and accordingly changing the orientation of the screen, activating the voice recorder and other proactive services [Hinckley et al., 2000]. Schmidt et. al. explored the augmentation of orientation sensors in the handheld devices [Schmidt et al., 1999b]. Similarly, Yamabe et al. showed their multiple sensors augmented PDA prototype "Muffin" capable of providing user-centric contexts autonomously [Yamabe et al., 2005]. A direct implication of these research projects could be seen in the latest consumer hand held devices. However, the contexts obtained from the augmented sensors in these projects are mostly used for user interface extension and in turn to provide better user experience. As a result of this, one interesting aspect of these projects is that all these sensors data were processed locally in the device without involving any secondary infrastructure. Such local processing of sensor data for value addition is also observed in Information Appliances the are stand alone self-contained devices with specific functionalities. The integration of awareness technology enable these appliances to provide value added services in a more user-centric way. Example of these appliances include Internet Fridge from LG Electronics, Visual Answering Machine [Banks et al., 2007], Smart Coffee Machine [Aitenbichler et al., 2007] etc. This defines one end point of the line where smart objects are *stand-alone* and equipped with *awareness-technology* and *self contained local perception algorithm* to provide more user-centric smart services.

2.3.2 Non-Digital Everyday Smart Objects

Over the years researcher have augmented various kinds of everyday objects ranging from static and mobile artefacts of our everyday life to structural material of our environment with awareness technology for providing value added proactive services. For the clarity of discussion, in the following these objects are grouped into five different categories.

2.3.2.1 Augmented Household Objects

The Mediacup project from TecO was one of the earliest efforts of augmenting non-digital smart objects where they studied the capture and communication of context in the environment using everyday object [Beigl et al., 2001]. Their goal was to collect context information by augmented physical object in a transparent way i.e., without changing the function and use of the object. This approach assumed a distributed system in which some objects are augmented to collect context, while other objects are computationally augmented to exploit that context. There were two distinctive features associated with the smart objects in their approach: i) awareness of the real world environment and ii) ad-hoc sharing of that awareness. In the first prototype, a regular coffee cup was instrumented with a computational unit embedded in its base. The unit was composed of sensors, processor and communication components and allowed the cup to collect its operational context (i.e., "cup is moving", "cup is stationary", "cooled off", etc.) and to transmit it to a secondary infrastructure so that other interested entities can utilize the transmitted context to provide value added services. Several applications were built on top of this model using Mediacup as primary source of context. The Mediacup project put forth two interesting issues for further exploration. First, the challenge of transparent instrumentation of awareness technology in everyday objects without compromising their original uses. Second, the instigation of a paradigm shift regarding how sensor enhanced applications should be built. By moving the perception algorithm and processing of sensor data at the source of data (i.e., smart objects), the Mediacup project gave an insight into how future smart object systems could be built.

Several contemporary research projects concurred with these design guidelines and augmented various household objects. Fujinami et al. constructed a mirror augmented with a regular display and a few sensors to present personalized information services in a contextual manner [Fujinami et al., 2005]. Similarly, several other research groups have augmented mirror with awareness technology for using it as a family portrait display [Terrenghi et al., 2008], as a lifestyle feedback system [Nakajima et al., 2008], as an ambient display [Lashina, 2004, Hitachi, 2008], and as an assistive makeup tool [Iwabuchi and Siio, 2008]. MIT's "Things That Think" initiative¹ investigated some novel applications with instrumented objects like kitchen utensils, drawing brush, bottles, etc. Kameas et al. built several everyday objects including chairs, alarm clocks, desks, lamps etc. for providing various proactive services using two kinds of interaction supports: artefact to artefact and user to environment [Kameas et al., 2004]. They also showed a two-step process for building instrumented objects: attachment of the hardware into the physical object followed by the installation of the software to determine the functionalities of the instrumentation. A similar approach was taken by Kawsar et al. to instrument chairs, stand lights and coffee jars and pots to create a smart workspace. These smart objects collected context information to take autonomous decisions for supporting their users in a pleasant way [Kawsar et al., 2005]. Digital Decor project had taken a user-centric approach and augmented traditional drawer and coffee pots to use as a smart storage and a

¹<http://ttd.media.mit.edu/>

media for informal communication respectively where users were explicitly required to interact with the system for such services [Siio et al., 2003]. In these projects, smart objects were the constituents of one or multiple applications that integrate those objects to provide value added services. Another interesting observation is the variety of usage of a same physical object with different instrumentations for different purposes.

2.3.2.2 Augmented Room and Building Structure

Various research groups have investigated the augmentation of awareness technology into the building structures. One advantage of using building structure is the flat surface that enable embedding display of various sizes for providing ambient feedback on the basis of the sensed context. In the Roomware project Streitz and his Ambiente-Team constructed computer-augmented objects resulting from the integration of room elements; walls, doors and furniture with computer-based information devices [Streitz et al., 1998]. Tokuda and his group introduced u-Textures to build custom furniture [Tokuda et al., 2004, Kohtake et al., 2005]. They proposed a new form of material with sensing, actuating, display, computing, and communicating capabilities that can be assembled to build wall, floor, partitions or various types of Smart Furniture. (e.g., chairs, tables, shelves). Similarly other research groups have built smart floor [Orr and Abowd, 2000, Suutala et al., 2004] and room walls with embedded displays [Streitz et al., 2005] in an attempt to make context aware environment. One broad presumption in this class of smart object is that these augmented objects form part of the fabric of the space and usually are not modified by the dwellers. Like, augmented household objects, these smart structural components are constituents of one or multiple larger proactive applications.

2.3.2.3 Augmented Objects in the Workplace

The Mediacup team continued to explore the augmentation of non-computational objects and introduced Smart-its [Gellersen et al., 2004] as an awareness platform that can be attached to everyday objects. One of the projects that used their platform was The "Cooperative Artefacts" where Smart-Its were used to build prototypes of augmented chemical containers that can detect and alert potentially hazardous situations. The embedded Smart-Its are configured with sensors to measure the proximity of containers, and are programmed to share and evaluate knowledge across containers [Strohbach et al., 2004]. This project highlighted cooperative model of smart object based applications where a group of smart objects create a synergy among them by sharing their situational awareness to each other without having a dedicated applications for providing centralized control.

Kortuem et. al. investigated the application of smart objects in the safety critical industrial workplaces [Kortuem et al., 2007]. They utilized sensor augmented construction drill machines to monitor the usage history and to alert the workers accordingly whenever the safety

thresholds are crossed. Along with the organizational aspects, they emphasized on the architectural issues related to this class of application and showed a smart object based approach for industrial workplace monitoring provides better solutions over sensor network approach.

One of the constraints of the instrumentation of everyday objects is attaching the output capability since embedding display might change an object's appearance. Some researchers have taken a proxy approach to overcome this limitation by utilizing secondary display to redirect the outputs of smart object. For example, Molyneux et al. utilized low-cost, small projectors for augmenting chemical containers with non-invasive displays in a workplace environment [Molyneux et al., 2007].

2.3.2.4 Electronic Tag Augmented Objects

Another common approach to augment non-digital physical object for some computational service is to tag them with electronic labels so that they can be referenced by other applications and devices. A well-suited technology for such tagging is RFID tags as they are cheap, small in size, and can be attached to any physical objects in an unobtrusive way. Roy Want and his colleagues augmented books, documents, photo-cubes and wristwatches with RFID tags in an attempt to provide value added digital information relevant to those objects in nearby portable computers unobtrusively [Want et al., 1999]. Their primary motivation was to investigate such tagging mechanism for bridging the gap between the physical and digital world. A similar approach was taken in HP CollTown project where physical objects were tagged with RFID for linking them with their virtual presence in the digital world [Deborah and Debaty, 2000]. Lampe et al. showed the potential of RFID in the movable asset management, where assets were tagged with RFID tags and a secondary monitoring tool was used to track them in an inventory applications [Lampe and Strassner, 2003]. Similarly, Konomi and Roussos have augmented high value apparel products with RFID tags at the Mitsukoshi department stores in Tokyo for item-level tracking across their apparel sales floors for efficient material handling processes [Konomi and Roussos, 2006]. Visual tags have also been used to create smart objects, however mostly in the augmented reality applications. Rekimoto et al. used low cost visual tagged physical objects for associating digital information with them and for indoor navigation [Rekimoto and Ayatsuka, 2000]. Some other researchers have also shown that even simpler technology like barcode can be used to augment physical objects for value added services [Ljungstrand et al., 2000]. For these kinds of tagged smart objects perception, reasoning and decision-making are allocated at the infrastructure. So, in a strict sense the smartness of this physical objects lies in the digital data attached to them that they can share autonomously with the help of a secondary infrastructure. These objects do not have any local processing units except the electronic tags, and do not possess any intelligence per se. However, they obtain passive smartness once they become a part of an application.

2.3.2.5 Augmented Wearable Objects

Although in the wearable computing domain explicit dedicated computing devices (e.g., Head Mounted Display, Camera, Palmtop etc.) are used to support their users in an improved and proactive way, a few projects have investigated augmenting wearable belongings, e.g., shoes, textiles etc. for providing contextual services. For example, Paradiso instrumented a footwear with a range of sensors to collect context information and built various applications utilizing those contexts [Paradiso et al., 2000]. His motivations was to expose the shoe as an expressive user interface. This exploration directly showed the potential of such wearable belongings for value added services. Kawsar et al. constructed a wearable teddy RoonRoon with multiple built-in sensors to provide proactive notifications [Kawsar et al., 2007b]. They were primarily concerned with instrumenting wearable fashion accessories to collect context information which could be used by other portable devices, e.g., mobile phone. Another work that is worth of mentioning is the COMRIS (Co-Habited Mixed Reality Information Spaces) parrot by Van de Velde where a physical embodiment interface in the form of a parrot acted as a wearable advisor [Velde, 1997]. Its focus domain is large public gathering like conference, exhibition etc. It could suggest the wearer about the interesting events. Although, both these projects introduced dedicated artefacts for proactive services, their idea could be easily transported to textile or other wearable objects (e.g., bracelets, ear-ring, spectacles, etc.). A few researcher also investigated augmented textiles. For example Baurley et al. showed how regular textile could be augmented with awareness technology for exchanging emotional expressions using mobile phone messaging [Baurley et al., 2007]. These projects highlighted the fact that appropriate applications with supporting technology could allow embedding perceptual technologies into wearable human belongings to make them smart.

2.3.3 Summary of the Existing Research

The research projects described in the earlier sections highlighted some premier works on smart objects and showed various application areas where smart objects have been used. There are several design commonalities that could be extracted from these projects. In the following these key characteristics are listed.

- The augmentation scopes for smart objects are not well defined. There are various instrumentation choices for a smart object to afford a variety of services.
- Typically smart objects embedded with awareness technology are used to collect context information in a transparent fashion, i.e., without compromising their physical properties and interaction metaphor.
- Most of the smart objects provide context information by locally processing sensor data, i.e., shifting the perception algorithm to the source of data.

- For smart objects instrumented with an electronic tag, it is mandatory to have a secondary infrastructure that provides the communication foundation for smart objects to share their situational awareness.
- Context information collected by smart objects are usually consumed by secondary applications and/or other smart objects. Some smart objects can consume their perceived context information and can provide proactive actuation in a stand-alone fashion.
- The applications that run on smart objects are typically context-aware.
- Considering the spatial distribution of smart objects and applications that integrate them, the system level characteristics (e.g., distribution, transparency, etc.) of a smart object systems are similar to the philosophies of a typical distributed system.

Some smart objects are stand-alone where some are constituents of a larger system. Considering the modality of the smart objects, we can categorize these systems into some groups for a better understanding of the requirement of a supportive framework. The next section presents a classification of smart object systems.

2.4 Classification of Smart Object Systems

Smart objects may operate individually, or collectively to attain specific purposes. When working collectively a network of smart objects is formed which is often referred to as a smart object system². Typically, in a smart object system, context-aware applications run atop a single or multiple smart objects embedded with awareness technologies (sensors, actuators and perception algorithms) where applications uses these objects to collect context information or to perform some services that cause changes in the real world (e.g., adjusting the air-conditioner based on sensed temperature). For some smart objects these applications could be integrated into them so that they can run in isolation without requiring any infrastructure support. On the other hand, for application that use spatially distributed multiple smart objects for their context-aware services an infrastructure is generally required. Henceforth, smart objects systems could be classified into following three categories as shown in Figure 2.1.

1. **Stand-alone Smart Objects:** These are self contained smart objects independent of any infrastructure and are capable of perception, reasoning and decision making autonomously as shown in the category 1 of the Figure 2.1. The awareness technology along with context-aware services are typically embedded into the object per se. Examples are Smart Coffee Machine [Aitenbichler et al., 2007], commercial smart objects from Ambient Device³, etc.

²Often the terms "smart object" and "smart object system" are used interchangeably.

³<http://www.ambientdevices.com>

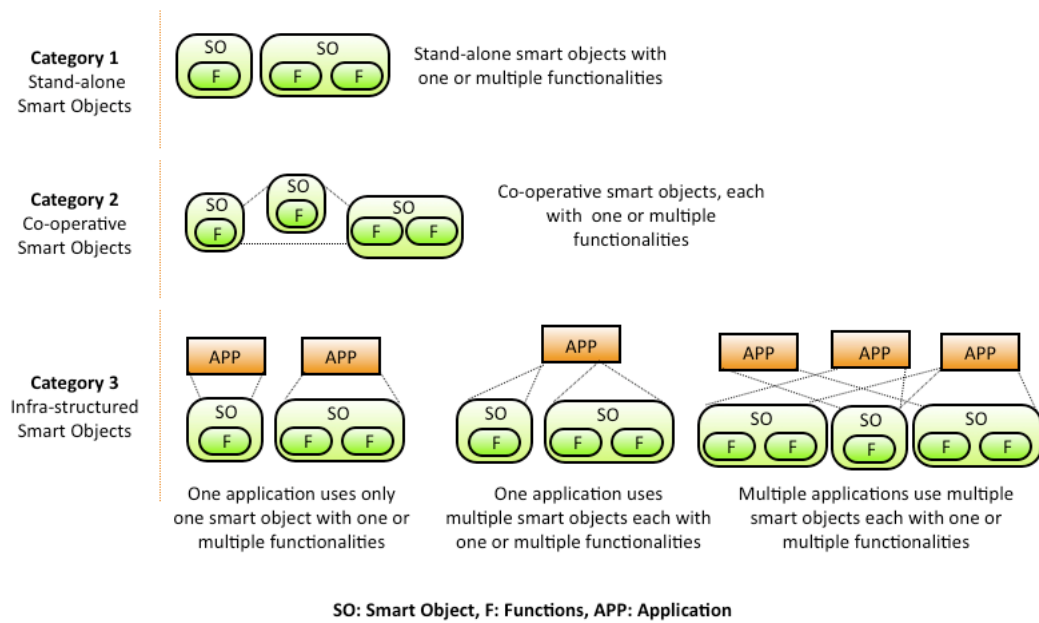


FIGURE 2.1: Classification of Smart Objects Systems

2. **Co-operative Smart Objects:** Smart objects that are capable of communicating with peers to share their self awareness for taking autonomous actions collectively thus creating a co-operative ecology of smart objects. A secondary infrastructure is often used in these types of systems. Co-operative smart objects are shown in the category 2 of Figure 2.1. Cooperative artefacts [Strohbach et al., 2004] in the industrial work place is an example of this class of smart objects.
3. **Infra-structured Smart Objects:** These smart objects are constituents of a larger system and might not be able to work individually. Typically, one or multiple context-aware applications integrate these objects into a proactive system utilizing a secondary infrastructure. There could be different use cases for infra-structured smart objects as shown in the category 3 of the Figure 2.1.
 - One application could use one smart object equipped with one or multiple augmented functionalities. This type of smart object system is identical to stand-alone smart objects except the fact that the application is independent of the smart object and an infrastructure is used to snap the application into the smart object. AwareMirror [Fujinami et al., 2005] is an example of this category of smart object system.
 - One application could use multiple smart objects equipped with one or multiple augmented functionalities. Usually, applications are independent of the smart objects and any suitable smart objects that could satisfy application requirements could be used with the help of a suitable infrastructure. Examples of this class

of system include Ambient Gaming Project [Nakajima et al., 2008], Pleasurable Smart Workspace [Kawsar et al., 2005], etc.

- Finally multiple applications could use multiple smart objects equipped with one or multiple augmented functionalities. Similar to the previous category, in this class of system applications are independent of the smart objects and any suitable smart objects that meet application requirements are used utilizing a suitable infrastructure. Examples include intelligent office [Andreoli et al., 2003], intelligent home [Helal et al., 2005], etc. equipped with multiple smart objects and applications.

This classification shows that majority of the smart object systems usually require a secondary infrastructure for creating a synergy among its constituents, i.e., smart objects and applications.

2.5 Sensor Network and Smart Objects

Before moving to the next chapter, It is imperative to discuss the relation between smart objects and sensor networks. Wireless Sensor Networks (WSN) or shortly Sensor Networks are composed of low-power, distributed, tiny embedded sensor nodes that are capable of collecting and disseminating observations across large and remote physical environments and are connected through self-healing, self-forming wireless networks with flexible topologies. Sensor nodes are distributed through out the target environment either at random or as an embodiment of physical objects. Due to limited resource, each sensor node is capable of only a limited amount of computation. Usually there are a small number of gateway nodes in a WSN that are responsible for streaming the sensor data off the network or for interrogating the network state.

Considering smart objects are constructed with awareness technology (sensor, actuator, perception algorithms), it could be easily concurred that the technology foundation for smart objects and sensor networks is very identical. In fact, often smart objects are build by embedding sensor nodes in everyday objects or even forming a small network within the object itself depending on the complexity of the augmentation. However, there are four distinct differences between smart objects and sensor networks. These are:

1. Each of the smart objects has an original established purpose and it retains that role even after the augmentation. The instrumented feature of the object (e.g., sensing or actuating) is its value addition. On the other hand, sensor network is dedicatedly built for observing physical phenomena without any additional purposes.
2. Smart objects retain their original appearance and interaction metaphor. They are conceptualized and manipulated by humans in terms of the object they are based on. A

sensor network, whereas, is not designed to be interacted with thus it is practically invisible to humans.

3. Typically each of the nodes in a sensor network is identical to other nodes and in most of the cases they all have same properties. Sensor nodes in the smart objects could be similar but not identical and change from smart object to smart object significantly.
4. Finally, from the architectural perspective, a sensor network is dependent on its infrastructure completely for routing and data diffusion. However, as pointed out in the earlier sections, a smart object could be stand-alone or co-operative in nature, thus a infrastructure is not compulsory for a smart object. Even for the cases when smart objects become a component of an infrastructure, the emphasis is on the network discovery, and exchange of services instead of routing and data diffusion.

2.6 Chapter Summary

This chapter introduced the notion of smart objects, its background and properties. Considering a number of contemporary definitions and design rationales, the definition of a smart object was given with three cardinals: perceptual augmentation, device-centric situational awareness and supplementary services. A few key properties of smart objects were highlighted in the light of the proposed definition. Then, a range of smart object projects in different application domains were discussed to extract the design commonalities across smart object systems. These projects have given a deeper insight into different types of smart object systems and their specific features. A number of issues have been learnt through these projects including the characteristics of smart objects in different types of applications, infrastructure requirements, and modality of systems. Essentially, we have seen that the typical applications that run atop smart objects have the context-aware property. Furthermore, the co-operative and infra-structured systems are very much identical to a typical distributed system. These issues are very crucial for extracting the design requirements for building an infrastructure for smart object systems.

Chapter 3

A Framework for Smart Object Systems

Chapter 3

A Framework for Smart Object Systems

The last chapter introduced the notion of smart object systems and discussed the specific characteristics that they exhibit. A range of smart object systems were discussed in an attempt to understand what features are common and useful across smart objects systems.

There are multiple design factors that need to be addressed while designing a framework for smart object system. Each of the constituents of a smart object system (e.g., smart objects, supporting infrastructure, and application) has its own design requirements. This chapter looks at these design factors in detail. For the clarity of discussion these design issues are divided into three parts. First, the chapter will discuss what are the specific design factors that we need to consider while building a smart object and how those design considerations can be reflected in the system through a suitable theoretical model. This will be followed by the design requirements of a supporting infrastructure to build applications that leverage smart objects to provide context aware services. Existing architectures in the literature will be introduced to highlight their limitations in supporting smart object systems. Then the design decisions for the conceptual framework will be presented. Finally, the chapter will look at the required framework qualities to build user-centric systems and how these qualities are accommodated in the proposed framework.

3.1 Design Issues for Smart Objects

The last chapter introduced a range of smart objects systems and their constituents smart objects. A number of design considerations and classification of smart object system have also been illustrated for providing a formidable understanding of the nature of smart object systems. Considering the characteristics of smart objects, it could be implicated that the smart objects pose several design challenges for constructing a theoretical model and design methodologies that can be shared across multiple smart object systems. These design issues are very crucial for the development of a system model that can be used to represent smart

objects. The following subsections will discuss these design issues and will follow up by presenting the design decisions that are taken in this work.

3.1.1 Design Requirements for a Smart Object Model

The nature of smart object systems, deployed environment, and potential user base pose the following five design requirements that need to be addressed for designing a theoretical model for building generic smart objects.

1. **Decoupling Smart Features:** Any physical object - in whatever shape or size - has certain affordances that affect how people use it. These affordances allow people to intuitively come up with new ideas to augment the same object to provide different value added services. This trend is observed in the object augmentation history by different research groups as illustrated in the last chapter, e.g., the augmentation scopes of the smart objects are not well defined. In fact it is hard to confine the augmentation scope. Consider Figure 3.1, depicting two ideal situations, a) a single everyday object capable of playing multiple functional roles and b) multiple objects sharing an identical functional role. In Figure 3.1(a) we have a smart table providing two supplementary functions: an ambient display and a proximity detector. In Figure 3.1(b) we have a mirror whose display functionality can be triggered by any of the three smart objects, e.g., a toothbrush, a comb or a razor. The suitable augmentation of these objects depends on the underpinned scenario, regardless of the multiple functionalities that can be afforded. Thus an important design challenge is to model smart object in a way that enables these smart objects to play different roles while retaining their original usage. Smart objects should be designed in a generic manner such that the smart features are independent of the physical object. It should be possible to apply the same feature in multiple physical objects.
2. **Service Unification - Sensing and Actuating:** In the last chapter we have seen that typically smart objects are used to collect context information by embedding suitable sensors. In most cases smart objects contain the processing unit to run the perception algorithm to generate context information out of raw sensor data. Also, many smart objects are used to push the environment states via actuation either utilizing their own sensed context information or by external applications that integrate them. So smart objects usually can do both: sense the physical phenomenon (pull) and actuate to cause the phenomenon (push). This is a crucial design factor for developing a theoretical model for smart objects and corresponding programming abstractions for the developers. From a system perspective, it is imperative to provide a unified notion of smart object features that can represent both sensing and actuating capabilities.

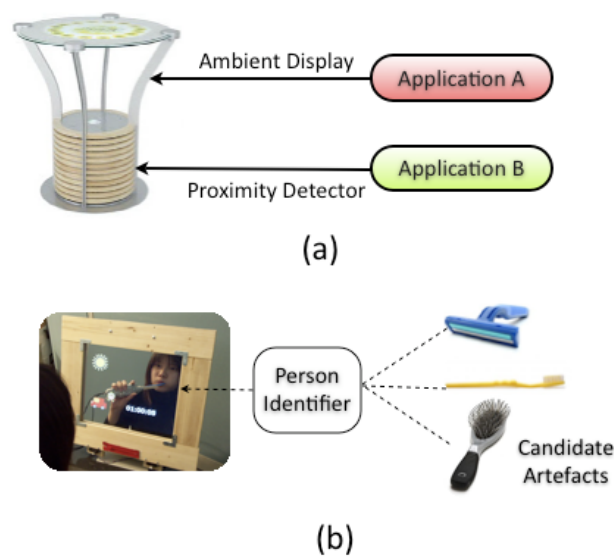


FIGURE 3.1: A single artefact with multiple roles and multiple artefacts with similar roles

3. **Reusability:** One of the reasons that limits smart object research with in laboratory prototypes is the ad-hoc design process that designers apply to build their smart objects. As a result, it is very difficult to reproduce the same smart object in a different setup. Consider Figure 3.2 that depicts some of the smart objects we have seen in the literature like Mediacup [Beigl et al., 2001], Ambient Umbrella¹, Music Bottles [Ishii., 2004], AwareMirror [Fujinami et al., 2005], Intelligent Spoon², Smart Furniture [Tokuda et al., 2004] etc. Unfortunately, none of these smart objects are capable of interacting with each other on the go, and cannot be integrated into one common application without special care. The design practices of one project are rarely carried over to the next. Thus one fundamental design challenge is to provide a design methodology along with a system model that could be used by various research projects to construct smart objects that are reusable in other projects. This has to be both from augmentation and system perspectives.
4. **Plug and Play:** Smart objects are the augmented versions of physical objects and these physical objects are typically ready to use from the outset. It is important to ensure that smart objects are developed in the similar fashion. Ideally, a user should buy a smart object, bring it home, power it up (if necessary) and it should immediately serve him/her. In the last chapter, we have seen three kinds of smart objects, while stand-alone smart object could be designed to exhibit such plug and play capability, it poses significant engineering challenges to enable co-operative and infra-structured smart objects to support plug and play feature.

¹<http://www.ambientdevices.com>

²<http://www.media.mit.edu/ci/projects/intelligentspoon.html>



FIGURE 3.2: Prototype smart objects from various research projects

5. **Incremental Deployment and Extension:** Smart objects are basically part of our environment. Typically we incrementally organize our physical environment with furniture and appliances according to our preferences and styles. Previous studies have shown how end-users continuously reconfigure their homes and technologies within it to meet their demands [O'Brien et al., 1999, Rodden and Benford, 2003]. To support the evolutionary nature of physical spaces it is essential that smart objects can be deployed and extended incrementally. As argued in the first chapter, ideally this deployment and extension tasks should be carried out by the end-users. Thus it is very important to consider the design issues for smart objects so that they can support such incremental deployment and extension. This challenge is further extrapolated if we consider the Do-It-Yourself (DIY) approach that are popular to a specific user base. Recently, many users are engaged in constructing their furniture by themselves. Commercial vendors like IKEA³ actively support this practice. Some researchers have also investigated how such end-user involvement in furniture construction can be supported [Antifakos et al., 2002] by providing proactive guidelines. It can be concurred that in the future, end-users will be involved in constructing smart objects in the similar fashion. So, it is imperative to develop system model and design methodologies for smart objects taking deployment and evolution aspects into account.

Based on the above discussion, it can be said that a theoretical model is needed to provide foundation that enables developing -

- Smart objects that are capable of hosting multiple smart features while decoupling the augmented smart features from its original purpose.
- Smart objects that can both sense to pull and actuate to push environment phenomena.

³<http://www.ikea.com>

- Smart objects that are reusable.
- Smart objects that are plug and play.
- Smart objects that are incrementally deployable and extensible.

3.1.2 Related Work on Smart Object Model

Chapter 2 introduced a range of smart object systems existing in the literature. As discussed all of these works have taken an application oriented approach and have constructed the constituent smart objects in an ad-hoc fashion without focusing on building a common model for representing smart objects. For example, in the Mediacup [Beigl et al., 2001] project, a regular coffee cup was instrumented to provide the state of the cup as context information. Although the Mediacup project and its succeeding SmartIts [Gellersen et al., 2004] provided solid insight into the augmentation of physical artefacts with sensing and processing, they did not provide any generic representation model that can enable smart object to be usable with any general purpose applications. Furthermore, they did not consider the requirements presented above. MIT's "Things That Think" initiative⁴ similarly prototyped several applications with instrumented objects but unfortunately their smart objects are tightly coupled with specific applications, making them hard to reuse in other applications. Tokuda and his group introduced Smart Furniture and u-Textures to build custom furniture [Tokuda et al., 2004], however their approach is also closed and tightly coupled with their underlying scenarios. The same is true for other projects in this area where various objects are augmented for providing value added functionalities [Strohbach et al., 2004]. These objects work fine in a specific scenario, however this assumption of scenario specific objects leads to a close development model, thus inhibiting these instrumented objects to be reused in other applications in a plug and play fashion. Interestingly, there has been very little work in the literature that addresses the deployment and extension issues for smart objects.

From an application's component point of view, there are several context-aware system infrastructures that tries to model smart objects as context provider or actuator. We will discuss these system infrastructures in detail in the later part of this chapter. At this point it can be emphasized that none of those architectures (as we will see) considered all five design requirements presented above. These infrastructures focus on representing smart objects as application components and model those objects in their domain specific way, regardless of the characteristics and properties of smart objects [Dey et al., 2001, Sousa and Garlan, 2002, Roman et al., 2002, Brumitt et al., 2000, Fox et al., 2000]. For example Context Toolkit provide a widget notion to encapsulate smart objects [Dey et al., 2001]. Although widgets are reusable, they are not capable of hosting augmented features in a plug and play manner. Adding a new feature to an existing smart object requires regeneration of the widget. Thus it can not satisfy the design requirements presented in the previous section. Nevertheless,

⁴<http://ttt.media.mit.edu/>

these infrastructures are not predominantly written for smart object systems, thus it would be fair to say that the specific requirements for a common smart object model were not considered in these systems.

3.1.3 A Core-Cloud Theoretical Model for Smart Objects

Section 3.1.1 introduced five design requirements that are essential to build smart objects and to integrate smart objects in applications. All these requirements are system related and a suitable model that can reflect these features must be generic enough to be ported across a range of systems. The design requirements essentially signify that objects' intelligence can not be confined strictly and smart features must be decoupled from the underlying physical object. Additionally, these features should be unified in a way that can be reproduced and reused with other semantically identical smart objects. Furthermore, these features should be plug and play, i.e., should enable one to attach a feature to an object freely.

Looking at the properties of smart objects (introduced in Section 2.2.2 of Chapter 2), it can be concurred that all objects do share some common features. A suitable smart object should decouple distinctive smart features from these common features. Accordingly, in this work, a *core-cloud* model is developed where the common features of smart objects are combined together in a core that acts as a runtime for supporting the smart features of the objects. These smart features are designed in way that can be freely added to the core without considering the syntactic or semantic heterogeneity. Each smart feature is called a *service profile* in this model. Recalling the definition of smart objects from Chapter 2, these service profiles typically include capability of sharing an object's situational awareness and state of use; supporting proactive and reactive information delivery, actuation and adaptive state transition. A single or a group of service profiles are combined together to form a cloud around the core.

To make a smart object plug and play, it is essential that the service profiles can be plugged in to the core spontaneously. Furthermore, this attachment of profiles into the core should be time independent, i.e., a profile can be plugged into the core of a smart object at anytime. This is very crucial to support the extensible design requirement of a smart object. Furthermore, if a service profile can be built in this fashion, then it can be reused with other smart objects that are suitable for the profile. Since there is no dedicated dependency between the cloud and the core, this design allows to decouple smart features (service profiles) from the underlying objects completely.

Consequently, the core-cloud model in context follows a plug-in architecture, whereas the core is a generic runtime binary present in every smart object and any cloud (one or multiple service profiles) can be plugged into the core. A service profile is also packaged as a generic binary and can be attached to a core dynamically. Figure 3.3, depicts the conceptual core-cloud model for a smart object.

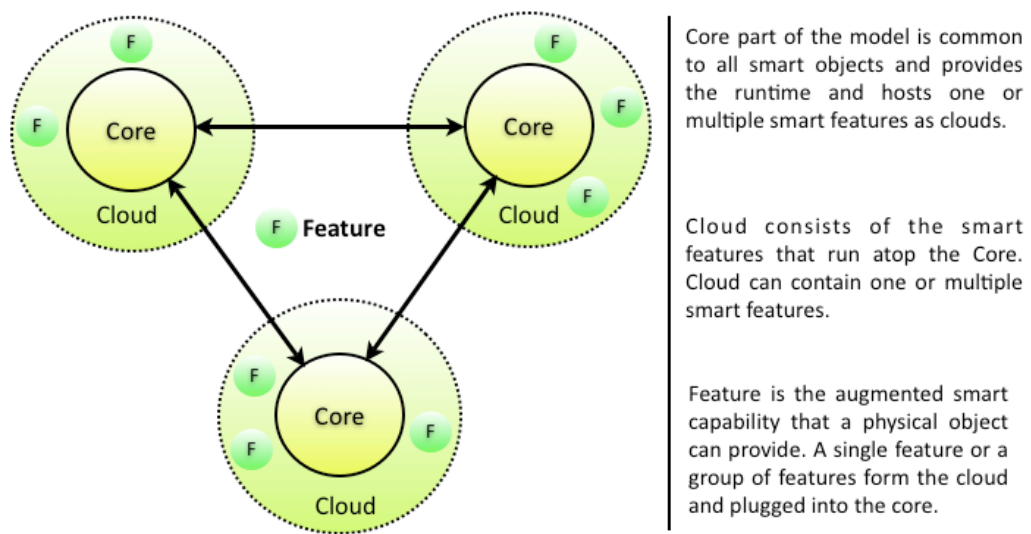


FIGURE 3.3: Core-Cloud Model for Smart Objects

The core contains the basic features that are generally common among the smart objects including

- **Communication Module:** It enables a smart object to communicate with the external world (peers and applications). Each smart object implements a communication protocol, e.g., IEEE 802.11x, Bluetooth etc.
- **Notification Module:** It enables a smart object to provide perceptual feedback of its internal states.
- **Static Memory:** A local memory that is available to support stateful smart objects.
- **Client Handler:** A common service interface for external entities interested in smart objects' services. It delegates the requests to appropriate profiles.
- **Profile Repository:** The plug-in runtime that hosts the service profiles as clouds.

Cloud is a combination of the multiple features (services) that can be afforded from an object. From a designer's point of view, this cloud based approach gives us the liberty to make the smart object design independent of its perceived affordances.

This simple design approach to model smart objects elegantly satisfies the five design requirements presented in section 3.1.1. By decoupling service profiles from the core, this model allows the development of smart objects that can play different role at different application context. Two identical physical objects can provide different functionalities if they are built with different service profiles. On the other hand two different physical objects can provide identical functionalities if they are built with same profiles. In addition, the cloud notion allows

a smart object to provide multiple features by hosting multiple profiles. This design solves the absence of design rationale and reusability issues i.e., the core can be shared among multiple smart objects where as the clouds are scenario dependent thus allowing developers to build specialized smart objects. Each service profile is independent of the runtime core, and packaged as a generic binary. This enables reusing the same profile implementation in multiple smart objects. The generic binary attribute also makes a smart object plug and play. Since there is no dedicated dependency, a smart object can be deployed independent of its service profiles. Furthermore, service profiles can be added to the smart object incrementally due to the dynamic snapping feature of this model. A by product of these design decisions is the support for incremental deployment and extension. A smart object can be initially deployed with or without smart features and can later be extended by adding suitable profiles. This model also gives us a cleaner programming abstraction for building stand-alone, co-operative or infra-structured smart objects as we will see in the later sections. Please, note that, this model can be applied to all class of the smart object systems that have been introduced in section 2.4. Chapter 4 will discuss the detail implementation issues of this core-cloud model and will explain how this model fits elegantly with the rest of the framework in context and allows building various classes of smart object systems.

3.2 Design Issues for a Smart Object System Infrastructure

Typically applications that run on the smart objects are proactive, and are similar to the philosophies of context-aware applications [Dey et al., 2001]. Thereby, these applications run atop distributed smart objects embedded with awareness technologies (sensors, actuators and perception algorithms) where applications use these objects to collect context information or to perform some services that cause changes in the real world (e.g., adjusting the air-conditioner based on sensed temperature) including adaptation of their own behaviors. For stand-alone and co-operative smart objects, context-aware applications are usually integrated into the object per se where as for infra-structured smart objects, context-aware applications utilize spatially distributed smart objects.

A context-aware application that runs atop or within smart objects usually consists of the following components:

- **Basic Application Component:** This component takes care of the basic application behavior, e.g., application logic, interfaces, etc.
- **Communication Component:** This component manages the access to smart objects and utilization of their services. Typical functions performed by this component include locating, managing, configuring and interacting with the smart objects. Different smart objects may have different communication and data protocols. It is the communication component that handles this heterogeneity and provides application with a unified access.

- **Perception and Adaptation Component:** This is where application handles the context data derived from the smart objects by applying application specific context reasoning and modeling policies. Applications use this context information to adapt application behavior accordingly and to further interact with the smart objects via communication component to actuate smart object services.

Although the first component is entirely application dependent, the last two components are recurrent and usually supported by a suitable infrastructure. Typically, such infrastructure handles the access issues by providing a discovery mechanism and provides Application Programming Interface (API) to interact with the smart objects transparently, and there by tries to separate the application from the underlying components (smart objects in this case). Such interactions typically include but not limited to:

- **Data Dissemination, Aggregation and Interpretation:** Infrastructure provides applications with appropriate support for accessing smart objects' data, aggregate and interpret them following application logic in either or both asynchronous (Event based subscription) and synchronous (Polling) communication modes.
- **Service Actuation:** Infrastructure provides applications with appropriate support for executing smart objects' services.
- **Configuration:** Infrastructure is expected to provide applications with the support of configuring smart objects.

For supporting these interactions a typical smart object system infrastructure provides suitable programming model through appropriate abstractions. Considering the constituents of a smart object system (e.g., smart objects and applications) are distributed in a physical space, the characteristics and utilization pattern of a smart object system is similar to the philosophies of encapsulation and reuse behind component based frameworks, e.g., Component Object Model (COM) [Iseminger, 2000], Java Beans [Englander, 1997] as well as more network friendly descendants like DCOM [Krieger and Adler, 1998], CORBA [Mowbray and Zahavi, 1995] or Jini/EJB [Waldo, 1999, Monson-Haefel, 2001]. Thus many architectural issues related to distributed component systems are applicable to smart object systems. In addition the highly dynamic, heterogeneous and fluid nature of smart environments put significant challenges for designing a suitable framework.

3.2.1 Design Requirement for a Smart Object System Infrastructure

The characteristics of smart object systems raise a number of important architectural questions: how can we build application and smart objects for such dynamic environment? how can we build context-aware applications to manipulate smart objects with no prior knowledge? how can we manage those unknown smart objects? In the following these design challenges are summarized.

1. **Heterogeneity:** Each of the smart objects might have different interfaces and might implement different protocols, even semantically same smart objects (e.g. two smart chairs from two different manufacturers) might be heterogenous from implementation point of view. It is obvious that to proliferate smart object systems, application has to be written independently without considering which smart object from which manufacturer will be used in the application. We can not expect an application to be written with prior knowledge of all of the myriad sort of smart objects of different types that it may encounter. The range of possibilities is simply too large and it is impossible to consider all smart objects during the development period.
2. **Augmentation Variation of Smart Objects:** Section 3.1.1 discussed that augmentation of smart objects greatly varies. A single everyday object can provide multiple services and multiple smart objects can provide identical services with different granularities. Thus it is not possible to classify smart objects by object type. This is particularly important as this emphasizes that defining standard interfaces for smart object is not a feasible solution.
3. **Management of Smart Object:** In a conventional component framework applications are responsible for managing (locating/spawning/etc.) the components locally, i.e., applications needs to know the access and configuration semantics. Keeping such functionalities at the level of application complicates the development process. Furthermore, for a dynamic environment like the one where a typical smart object system runs, it makes very difficult for application to adapt to the smart objects that change for mobility purpose or fail.
4. **Evolution of Smart Object System:** Unlike the conventional distributed component systems where applications typically reside in the digital world, smart object systems are deployed in the real world, i.e., our living spaces. An essential property of our living space is its evolutionary nature and receptibility to continual change. To support the evolutionary nature of the real world it is essential that smart object systems support incremental evolution. It is necessary to have suitable programming abstraction that will allow developers to extend smart object systems' functionalities in an incremental fashion, ideally involving end-users.
5. **Effective Programming Model with Suitable Abstraction:** In addition to the above fundamental design challenges, a suitable infrastructure for smart object system should provide application developers with appropriate support for interacting with the smart objects, i.e., collecting context information and/or executing services. A typical infrastructure provides this support through appropriate programming model with effective abstractions that hides heterogeneity, provides transparent communication and allows portability of the applications. Furthermore, the abstraction should allow the developers to extend the applications when needed.

3.2.2 Existing Support for a Smart Object System Infrastructure

The previous section introduced several design requirements of a supporting infrastructure for building smart object systems. This section will put the spotlight on existing system infrastructures intended for context-aware computing, home computing, or other relevant domains. It is natural that many of the features of the framework presented in this work are leveraged off the architectures that preceded it. However, these infrastructures are not designed explicitly for supporting smart object systems. Consequently, their support for building smart objects systems are not adequate. Those limitations of existing infrastructures will be discussed in the context of smart object systems which in turn provides the basis for presenting a conceptual framework for building smart objects system adhering the design requirements presented in the earlier sections. For the sake of clarity, the section will begin by looking at the three basic architectural models as categorized by Winograd [Winograd, 2001]: a widget model, a client-server model and a blackboard model. Most of the existing architectures followed one of these models as their primary design principle. After discussing these models and their tradeoffs, existing infrastructures will be presented.

3.2.2.1 Three Models of Architecture

A number of architectural models have been proposed in the literature for integrating distributed and inter-operating components in the context of context-aware computing, home computing, collaborative systems and relevant domains. Winograd [Winograd, 2001] categorized these models into three distinctive groups: a widget model, a client-server model and a blackboard model. In the following these models are presented.

1. **Widget Model:** The widget model is adapted from the architecture of Graphical User Interface (GUI) and identical to the notion of a device driver. It encapsulates a specific component and provides high level abstraction using which an application can utilize the resource through a central widget manager. A typical example of such widget is a scroll bar of GUI that provides one dimensional position information to the application along with some control signals abstracting the real hardware (e.g., a mouse). The interaction among the applications and the underlying components are implemented in terms of messages and callbacks. In this model, the applications are responsible for component management (e.g., locating, spawning etc.) locally. Although such tight coupling is efficient for a smaller scale application, it leads to complex configuration and less robustness as the system component grows. The Context Toolkit by Dey et al. [Dey et al., 2001] adopts this model in the context of context-aware computing.
2. **Client-Server Model:** The client-server model is more loosely structured where the high-level service components are independent communicating entities. This model is the backbone of the development of Internet-based softwares, where the client and

server reside at different network locations and communicates through Internet protocols. There is no central manager in this model and each component independently manages its own connections, messages, failures, and so forth. This introduces the need for resource discovery and uniform access mechanism, i.e. a client needs to find the location of the appropriate service and the appropriate mechanism to access the service. Although the cost of discovery and independent access mechanism makes this model complex, it is more robust and easier to configure as each component is independent. In addition this model has a number of advantages over widget model [Hong and Landay, 2001]. First, the platform independency of this model allows a wide variety of clients to access a wide variety of service components. Second, the isolation and independency of the components allow them to evolve gradually and maintenance become easier. Third, the client-server approach enables sharing of component services across multiple applications and devices.

3. **Blackboard Model:** The blackboard model takes a data-centric approach and provides a common shared message board for the distributed components that are part of the system to post messages and to receive the same using suitable callbacks. The nucleus of blackboard model is the pattern matching technique and it varies from system to system. All communication in a blackboard system goes through a centralized server that applies pattern matching algorithms to route messages to appropriate subscribers. Blackboard model has a long history in the artificial intelligence research. One of the common approaches used by various systems relevant to smart objects systems is tuple based blackboard that was introduced in Linda language [Gelernter, 1985]. The primary advantages of the blackboard model are its simplicity and unified access mechanism. The loose coupling among the components of a blackboard model enables the addition and removal of component without any side effect. However, this model forces the participating components to use a generic message structure regardless of their underlying protocols and also suffers from communication efficiency.

There are a number of frameworks in the literature that provides support for building applications for pervasive environment. Considering the design requirements presented earlier in this chapter, these infrastructures will be discussed from two perspectives: infrastructures that were meant for device integration and infrastructures for pervasive computing in general.

3.2.2.2 Distributed Component and Device Integration Infrastructures

A range of systems have been proposed in the literature that provides distributed component and device integration mechanisms. Some significant ones are discussed here.

DCOM and CORBA The challenge of heterogeneity is typically handled by existing frameworks using interface standardization [Krieger and Adler, 1998, Mowbray and Zahavi, 1995].

A programmer writes a small software to interact with a specific component / device, e.g., a networked printer. Any application can use the printer using this small software, as long as both the components (printer component, and application) agree beforehand on exactly how components will communicate with each other and the application manages this interaction locally. If the application functionality is extended to use another device, or the same device is replaced by a new one then the application must be rewritten to interact with the new device component.

UPnP and Jini On a more lower granularity level, UPnP⁵ defines a standard set of protocols for specific device types (e.g., audio/video devices) for interoperability where as Jini [Waldo, 1999] describes devices using interface description and language APIs allowing applications to utilize those interfaces. However, application that leverages these devices' services still needs to know the interfaces, and any change at the device end causes the application to fail. This is further complicated considering the nature of smart objects as mentioned earlier. It is very difficult to standardize the protocols for smart objects considering their diversity. Furthermore, these infrastructures focus primarily on the developers rather than the eventual users, consequently their support for user-centric system evolution is limited. For example, it is hard to add features in an existing smart objects and using that feature immediately in the application with these infrastructures.

SpeakEasy SpeakEasy [Edwards et al., 2002] utilizes mobile code to dynamically download the heterogeneous component interfaces at application ends. In this approach, mobile codes (typed data streams and services) are exchanged among heterogeneous devices to create an interoperable environment. End-users are assumed to have the knowledge about composition and considered to be the final arbitrator. The semantic mapping of devices is done by the user considering the real world constraints and context. Devices and services must provide enough information about themselves in a human understandable form to let the users connect the devices together using the infrastructure. Generic applications are still need to be written with the awareness of the interfaces. SpeakEasy's primary focus is on service composition, so its design does not address issues like representation of smart objects, collection of context, unified access, incremental extension of services etc. Moreover, such mobile code based approaches are impractical considering the nature of the smart object systems, for every new smart object an application encounters, it would need to download new codes, even for components that are semantically same.

XWeb XWeb [Olsen et al., 2001] is one of the architectures that influenced this work significantly. It uses a simple protocol and data format for connecting arbitrary devices. Every device exposes an XML file to describe itself. These device states are accessible to other devices that can speak XWeb protocol, and accordingly they can request XWeb to perform operations on

⁵Universal Plug and Play - <http://www.upnp.org>

each other. From an abstract point of view, XWeb can be seen as a logical extension of HTTP for devices. The major advantage of XWeb is its simplicity. However, XWeb does not specify how the XML files and the underlying physical resource are connected, and how the entire system works together. In addition they did not consider the smart object system specific issues as discussed in the earlier sections.

PatchPanel Patch Panel [Ballagas et al., 2004] is a programming tool that provides a generic set of mechanisms for translating incoming events to outgoing events using EventHeap communication platform [Johanson et al., 2002]. It allows new applications to leverage the services of existing components. As we will see the proposed approach is close to Patch Panel as the framework in context also seeks to support incremental integration. However, the proposed framework exploits a document-based approach that enables incremental addition of features to both smart objects and applications.

InterPlay InterPlay [Messer et al., 2006] is a home A/V device composition middleware and uses pseudo sentences to capture user intent, which is converted into a higher level description of user tasks. These tasks are mapped to underlying devices that are expressed using device description. InterPlay's overall design philosophy is very similar to the framework in context from the spontaneous federation point of view. However, it does not address the smart object representation and extension issues. Also, the application development support is not clearly mentioned.

3.2.2.3 Pervasive Computing Middlewares

A range of middlewares for pervasive systems and context-aware computing are investigated in the literature. In the following these systems are introduced and their supports for smart object systems are discussed.

Schilit's System Architecture Schilit presented a system architecture In his Ph.D. thesis for supporting the development of context-aware mobile computing application [Schilit, 1995]. This is one of pioneer works that stimulated context aware application development. Schilit and his colleagues have developed numerous context aware applications in Xerox PARC that inspired the community as whole to focus on context aware applications. Schilit's system deals with the context awareness by Device Agents that maintain the status and the capabilities of the devices, User Agents that maintain the user policies and Active Maps that maintain the location information of the devices and the users. However, the system does not consider device (smart object in the current framework context) representation issues, i.e., that is how a smart object can be constructed to provide a specific features. Also, because of the tight coupling between context and device agents in the architecture, it is difficult to add new devices with new capability limiting the extensibility of a system.

Context Toolkit Context Toolkit [Dey et al., 2001] focuses on the component abstraction by providing the notion of Context Widget and Context Aggregator. Discoverer manages these components and additionally there is a Context Interpreter component that performs the task of context interpretation. Context Toolkit provides very low-level abstraction. The developer needs to provide the details about the context source like location, port etc. Moreover, the application is inherently dependent on the framework as the application is tightly coupled with the architecture components like interpreters, aggregators etc. The primary problem of Context Toolkit in the light of smart object system is the scope of Context Widget that follows a one-to-one mapping, thus if a smart object provides multiple functionalities, for each functions we need a new widget. On the other hand, objects that can actuate are represented by a service model. Thus for a smart object that can both sense and actuate, we need two different programming abstractions, widget and service. Furthermore, such widgets are not capable of hosting augmented features in a plug and play manner. Adding a new feature to an existing smart object requires regeneration of the widget. This limits the extension of smart object services or leveraging new services using existing abstractions. Also, the management of widgets is handled at the application level in the Context Toolkit which complicates the development process and hinders the extension of applications and/or smart objects.

Technology for Enabling Awareness The Technology for Enabling Awareness (TEA) project looks at how to provide context awareness to personal mobile devices [Schmidt et al., 1999a]. It follows a 3-layer architecture composed of a sensor layer - responsible for managing the raw sensors (both physical and logical), a cue layer- responsible for providing abstraction from raw sensor data (i.e., features) and finally a context layer - responsible for representing context by fusing multiple cues. Applications can interact with the context layer to gather context information to provide proactive services or to adapt their own behaviors. Although, this simple architecture provides clear separation of concerns, there is limited support for application developers to specify their requirements. In addition the developers need to manage the interactions with TEA infrastructure locally at the application level which makes application development and extension complicated.

Gaia Gaia is [Roman et al., 2002] meta operating system and its design philosophy is centered around the concept of Active Space that is capable of knowing all the available resources and providing services to users in a contextual manner. Gaia is composed of five components. An Event Manager that tracks all the events of the system, e.g., applications startup, users entering etc. A Context Service based on first order logic that represents contexts and runs always to capture the context of the environment. A Presence Service that can identify the presence of an entity, i.e., human and objects using tag based recognition. A Space Repository that is the registry of all the available devices in the environment. Finally a Context File System that stores data with context (e.g., location, identity, etc.) based index. Gaia also specifies its own application framework based on MVC pattern [Krasner and Pope, 1998], where each component is mapped to one or multiple underlying devices. A scripting language called

LuaOrb is provided to express the application requirement. Although, Gaia has covered a lot of design requirements of pervasive systems, its primary shortcoming in the context of smart object system is its device representation mechanism that is not suitable for wrapping a smart object. In addition, the tight coupling among the applications and the underlying framework makes it very difficult to port and extend applications or underlying components.

Aura The Aura architecture is built on the concept of user centric task to support mobile users i.e., a task with all its resources and files can follow a user [Sousa and Garlan, 2002]. Environment services can be dynamically loaded based on task requirement and can be adapted to dynamic changes e.g., sudden bandwidth drop. The architecture is composed of four primary components. A Task Manager called Prism is the repository of all user tasks and provides definition of tasks. These definitions are used by the other components to follow a user in context. A Context Observer component is responsible for detecting context related to user and physical environment. An Environment Manager is responsible for the registration of services, for handling the task migration and for binding resources to a task dynamically. Finally the Service Suppliers that are wrapped using predefined APIs to provide services of different types. Example of services are text editing service, e.g., word, emacs etc. Like Gaia project mentioned above, Aura also approached to provide a generic platform for pervasive computing. However, many technical aspects are not addressed in the project, e.g., how to handle smart objects, how to handle heterogeneity of devices, how to build generic portable applications, etc.

iROS Stanford's Interactive Room Operating Systems (iROS) adopted a blackboard architecture to integrate multiple devices for multiple users in a shared physical space with defined scope [Fox et al., 2000, Johanson et al., 2002]. The iROS is composed of three sub-systems: Data Heap, iCrafter and Event Heap. Data Heap is responsible for moving data, and allows any application to place data associated with the local environment. Event Heap is the underlying communication infrastructure for applications within the interactive workspace. It also provides the dynamic application co-ordination. Event Heap uses T-Spaces and provides faster distribution of event tuples. iCrafter provides a system for service advertisement and invocation, along with a user interface generator for services. The design philosophy of iROS project, specially the two layers of blackboard as Event Heap and Data Heap are very close to the design decision taken in this work. As explained in the earlier part of the chapter that such blackboard approach provides a significant advantage from application development perspective due to the access and utilization unification. However, the iROS project did not address the device representation issues that is very crucial for building smart object systems.

Java Context Aware Framework JCAF [Bardram, 2005] is Java based architecture following the pattern of J2EE for context aware domain. It is mainly composed of two components, context service and context client. Context service is analogous to J2EE server where context

client is analogous to J2EE servlet notion. Each context service encapsulates context information while providing interfaces for its clients. It also has a security component to protect the services. The architecture does not address the smart object specific design concerns, e.g., providing a suitable representation model for reusable and extensible smart objects etc. Also, there is no specification about how to handle the heterogeneity of the underlying context sources while providing unified abstraction.

Sentient Computing The Sentient Computing project [Addlesee et al., 2001] utilizes Active Bat location system to provide an architectural base for indoor applications exploiting a world model. All location information is stored in a central database, which is used as the base for providing generic services for application developers. However, they have only used location predominantly as context information in their system. The major goal of this project was to provide a virtual world model of the underlying physical world manipulating the location information. Because of this specific focus many design concerns related to the smart object systems are not addressed in this project.

HP CoolTown HP Cool Town [Deborah and Debaty, 2000] encapsulates the world by providing web presence of places, people and things. It enables interaction with these entities primarily exploiting RF technology. Each entity is represented by its own website which is updated automatically based on changes in the physical world. However, CoolTown is not developed for supporting general context aware applications thus limiting its applicability in smart object systems.

Easy Living The Easy living [Brumitt et al., 2000] research project is initiated to investigate a suitable architecture for smart spaces that will support coherent user experiences as users interact with a variety of the devices. The project utilizes a middleware called "InConcert" that encapsulates the underlying devices and represents it in the application. In the process of representation, it models the underlying real world thus associating the users with nearby available devices for just-in-time service provision. There is partial support for context specification in the application space. Easy Living does not address reusable and plug and play smart object representation, programming abstractions for application developers, unified access mechanisms etc.

Stick-e Note The Stick-e Notes system [Brown, 1996] is an interesting work that provides simple semantics for writing rules that specify what action to perform based on the acquired context, mainly focusing on non-programmers to author context aware services. These rules specify what actions to take when a particular combination of context is realized. However, it is not clear how the context information is extracted or what sort of abstraction is provided to hide the detail of context sources, i.e., smart objects.

Context Fabric Context Fabric [Hong and Landay, 2004] is an architecture that primarily focuses on privacy protection. Its objective is to assist the application developers to manage the privacy of context information. Mainly location privacy is depicted in this work although the approach can be extended to cover any context information that are privacy sensitive. The architecture is composed of three data model - InfoSpace Server, InfoSpace and Context Tuple analogous to Web Server, Website and Webpage notion of Internet technology. Each user has his/her own InfoSpace where his/her personal information is collected and located. This information is only shared with other InfoSpaces that are approved by the user. This sharing follows a hierarchical manner, i.e., the user can explicitly specify the level up to which he/she wants to share his/her information for contextual service provision. This work is the first architecture that primarily focuses on client end security and data protection. However, the detail of several system related issues like device representation, access mechanism, distribution etc. are not clear. So it cannot be said what system related functionalities it provides beyond managing client end security.

3.2.3 Drawbacks of Current Approaches

The last section looked at several middlewares that provide infrastructure support for various aspects of mobile and ubiquitous computing. However, this work argues that these platforms can not fully accommodate the required features for smart object systems. In the following the drawbacks of these systems are summarized in the context of a smart object system:

1. **Tightly Coupled Presentation of Smart Object:** Many existing infrastructures provide widgets or device proxies to encapsulate smart object features. These component representations are not capable of hosting multiple augmented features or do not allow incremental addition of features to a smart object. Adding a new feature to an existing object requires generation of a new device proxy or widget. This solution is inadequate and impractical because for one physical object, we might end up in multiple component representation, one for each augmented features. In addition they do not consider the five design requirements essential for representing a smart object as discussed earlier in this chapter. Thus it is not possible to develop reusable, extensible, plug and play smart objects using these infrastructures.
2. **Smart Object Management:** Existing infrastructures typically specify their application development processes strictly. These middlewares usually provide end-to-end support for application developers. A range of API specification is provided to the developers to perform the smart object management (e.g., locating, spawning, configuring, etc.) and thereby to deal with the heterogeneity at the application level i.e., smart objects are encapsulated into wrappers and an array of APIs is provided to the applications to manipulate them. Handling these issues at the application scope complicates the application development process and makes the applications and smart objects virtually incompatible in other environments.

3. **Abstruse Programming Abstraction:** Programming abstraction in the existing middlewares [Sousa and Garlan, 2002, Roman et al., 2002, Fox et al., 2000] is oriented to context [Dey, 2001] predominantly. Actuation functions are often presented as actions of infrastructure services. Thus to represent a smart objects that can provide context and as well as perform some actions requires multiple programming abstractions (e.g., context and service). This causes confusion in building applications with smart objects.
4. **Inadequate Infrastructure Support:** In chapter 2, three classes of smart object systems were presented. Although the infra-structured smart object systems could be supported by existing infrastructures discussed in the last section by providing a wrapper that is tightly glued with the rest of the infrastructure, they have no clean support for stand-alone or co-operative smart object systems with a single or multiple built-in functions. Smart objects can not be accommodated natively as stand-alone objects and/or co-operative objects in these infrastructure environments without special care.
5. **Inadequate Support for Smart Object System Evolution:** A byproduct of the above deficiencies resulted in minimal support for evolution of smart object systems by existing infrastructures. Due to the tight coupling among the system components it is very difficult to add/remove components from the system and replace application and/or smart objects with updated versions over time. Consequently, it is very difficult to build plug and play and extensible smart object systems using existing infrastructures.

3.2.4 A Document Based Solution Framework

In the previous section, a range of infrastructures and their limitations in supporting the development of smart object systems were discussed. Primarily these infrastructures have limited support for building smart object systems considering the design requirement imposed by smart object systems. Nevertheless, most of these infrastructures were not built for supporting smart object systems, so it is expected that specific design requirements are not addressed by them. In this section, a solution framework that addresses smart object specific features is introduced with explanations of its design rationales.

As we discussed, two major design requirements for a smart object system framework are adequate support for applications to handle heterogeneity and management of smart objects, i.e., discovering, spawning, configuring etc. One way to address these issues is if we look at the functional aspects at the application end only and leave the protocol heterogeneity issues at the infrastructure end while enabling applications to use a generic access mechanism to manage the smart objects regardless of their types. Accordingly, a data-centric approach is taken in the work to handle these issues utilizing documents. The proposed framework forces an application to expose its functional tasks that need the service of a smart object (i.e., a component) in a document without addressing how to access that smart object service. These tasks are atomic actions that represent smart objects' services, e.g., *“sense current light sensitivity”*, *“turn on the lamp”*, etc. Similarly, a smart object is forced to expose its service features

via documents. A secondary infrastructure then connects the application to the smart objects by matching the documents. However, applications and smart objects are not directly connected. Instead they communicate to the intermediary infrastructure to delegate their service requests and service responses respectively. This underlying infrastructure can provide the technical building blocks to allow applications to use arbitrary number of smart objects as long as they provide the functionalities that are expected by the application.

The infrastructure takes the management of smart objects away from the applications, so applications do not need to care for access, configuration or management issues. To facilitate this, both the application and smart objects are forced to implement a standard communication protocol. The basic idea here is to combine the client-server and blackboard models of architecture that we discussed earlier in this chapter. While the loosely coupled client-server model ensures that the heterogeneity is handled away from the application, blackboard model ensures the unified access and delivery of smart objects services appropriately.

In the earlier section, the core-cloud model for smart objects was introduced. The design challenge imposed by the augmentation variation of smart objects is thus handled by this model. The core of a smart object is a generic runtime that can host any number of smart features as plug-ins. This design allows developers to decouple smart features of a smart object and applying same features in multiple smart objects. Thus a smart objects might be used in multiple ways under different circumstances for different applications/purposes. In addition, features can be incrementally added to a smart object to extend its initial functionalities. Simultaneously, application's functionalities can also be extended by introducing new smart objects or updated smart objects that allows some of the application tasks to leverage the features of newly added or updated smart objects. The combination of these approaches, i.e., document based framework for connecting applications with smart objects, and core-cloud model for representing smart objects support the evolution and extension requirements for smart object systems.

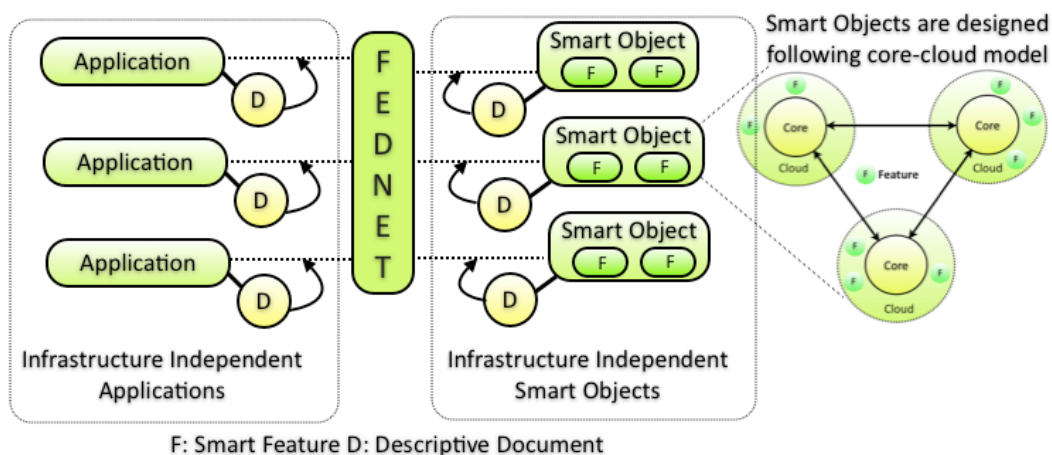


FIGURE 3.4: A Conceptual Document based Framework

The conceptual document based framework is depicted in Figure 3.4. The framework consists of a *Smart Object Wrapper* that implements the core-cloud model for smart objects, an *Application Development Model* and a *Runtime Intermediary Infrastructure* called FedNet. Smart object wrapper represents a smart object by encapsulating its augmented functionalities (e.g., proactivity of the table lamp) in one or multiple **Profiles** (cloud) atop a runtime (core) and allows additions of profiles to be added incrementally. Applications are represented as a collection of implementation independent functional **Tasks**. These tasks are atomic actions that require smart objects' services. An infrastructure component FedNet, manages these applications and smart objects and maps the task specifications of the applications to the underlying smart objects' services by matching respective documents (that express the applications and the smart objects) thus externalizing smart object management and addressing heterogeneity issues away from the applications allowing developers to focus on the application functionalities only. Primarily these two abstractions **Profile** and **Task** are used in the framework and realized by corresponding documents. Thus, the proposed framework provides a very simple programming model that allows application developers:

- To develop smart objects and profile through consistent abstraction with structured documents without concerning the target application requirement.
- To externalize application's requirements and utilize smart objects without concerning interfaces and the management of smart objects.
- To extend both applications and smart objects using primary abstractions.

The runtime intermediary, FedNet handles smart object management (Bootstrapping, Discovery, Utilization) and provides mapping between application and smart object services based on structural type matching thus separating the concerns of the application and the middleware. These design decisions enable developers to write applications and to build smart objects in a generic way regardless of the constraints of the target environment. This results in simple and rapid development of smart object systems.

This document centric design has been influenced by two successful approaches existing currently. First one is Internet which is an excellent example of document based system. The Internet is a collection of millions of anonymously authored digital documents that are encoded in a pre defined semantics that enable heterogenous platforms to exchange these documents. The fundamental issue here is the pre negotiation of the semantics. The most widely used protocol for Internet, i.e., HTTP basically acts as the envelope for this documents and provides the negotiation semantics to both the sender and recipient (i.e., servers and client browsers and vice versa) through it's headers for a flawless communication. Henceforth, structured document is the primary resource and HTTP (headers) acts as the connecting glue in the Internet infrastructure. In the proposed approach, applications are considered as the consumers and smart objects are considered as the resources. Thus if both are expressed and amended with

pre negotiated semantics using documents like HTTP headers, we can easily provide a run-time association. The second influencing approach is the commonly used shell scripting to connect arbitrary programs using the UNIX pipe facility where file handles (i.e., stdin, stdout, stderr) are used to differentiate and route data [Leffler et al., 1989]. From an abstract view point we can observe that this capability of semantic mapping by pipe facility is basically the negotiation of input/output structure. Thus, a structured document with pre negotiated semantics can perform the similar piping between application and smart objects. Henceforth, documents can glue an application with underlying smart objects given the fact that they have pre negotiation of their data semantics.

3.3 Framework Support for End-Users

As the technology is becoming mature and reaching end-users, it is essential to build smart objects systems in a more human-centric way, i.e., we need to understand how we can involve end-users in the administration of smart object systems. This work argues that end-user experiences with smart object systems can be elevated substantially if we can involve them in the deployment and administration of smart object systems. Most of the user-centric research on smart objects looked at the interaction paradigms of smart objects systems and focused in building suitable interfaces [Norman, 1998, Ishii., 2004, Norman, 1990]. However, it is also essential to understand how to place and manage smart object systems into the environment. This is particularly important for the home where the dwellers have a greater control. it is important to consider the existing mental models of end-users towards everyday object and information appliances. One essential property of our home is its evolutionary nature and receptibility to continual change [Rodden and Benford, 2003]. We incrementally organize our homes with furniture and appliances according to our preferences and styles. Previous studies have shown how end-users continuously reconfigure their homes and technologies within it to meet their demands [O'Brien et al., 1999, Rodden and Benford, 2003]. Edwards et al. observed that the networked home of the future will emerge in a piecemeal fashion [Edwards and Grinter, 2001]. To support the evolutionary nature of our homes it is essential that smart object systems facilitate the incremental deployment and ideally by the end-users, i.e., the end-users should be capable of incrementally enhancing the smart objects functionalities by upgrading its features or installing new applications. The dwellers have in-depth knowledge of the structure of their home and their activities, resulting in a better understanding of where and which physical artefact and application to deploy. Furthermore, involving end-users in the process leads to higher acceptability and a greater feeling of having control due to their active participations. It also reduces deployment cost as professional assistance is not needed.

The above discussion raises important design challenges for the smart object system developers. Providing such end-user centric supports requires specific architectural qualities from the

framework that is used to build smart object systems. In the following four such requirements are presented that have been identified during this work.

1. **Plug and Play Smart Objects and Application:** To support end-user involvement, it is mandatory that the constituents of a smart object system (i.e., smart objects and applications) are plug and play. Consider, the current practice of deploying home appliances. A user can buy a home appliance (e.g., a microwave, a toaster etc.) and can plug the cable to outlet to make the appliance work. Such simplicity enables end-users in actively deploying these appliances. The same is true for personal computer peripherals (e.g., a microphone, a camera, etc.). A smart object system should be identical. Users should be able to buy one or multiple smart objects and applications for them, and should be able to deploy them in a seamless fashion.
2. **Extensible Smart Objects and Application:** In the above discussions, we have seen that previous studies showed how end-users continuously change their homes by buying new appliances and/or upgrading to new models to meet their needs and styles. Thus, it is very important to enable this evolutionary quality in smart object systems. A user can buy one smart object with specific features and can update the feature over time. Similarly a user should be able to install one application in a smart object and replace it with a new one to match his/her needs. So, smart objects and applications should be developed in a such a fashion that they support this extensibility feature.
3. **Loose Coupling among Smart Objects, Applications and supporting Infrastructure:** To involve end-users in the administration processes it is important that system components are very loosely coupled and do not have any coherent dependency on each other. Each component (i.e., smart objects and applications) should run independently in its own isolated environment. This allows end-users to deploy them easily since there is no architectural dependency that has to be maintained.
4. **Support for a Suitable Interaction Tool for End-users:** Since, majority of the smart object systems have spatially distributed physical components, it is essential to have a suitable process to deploy, configure and administrate the smart object systems. Most of the smart objects might not have any user interfaces per se, e.g., smart objects that are collecting context information. Thus, a proxy based approach is needed that allows interacting with these smart objects through a secondary interface.

As we discussed earlier in this chapter, the proposed conceptual framework satisfy these requirements elegantly. The core-cloud model enables the development of plug and play and extensible smart objects as each service profile is packaged as a generic binary and can be added to the core of a smart object at any time as plug in. Application is also independently written as generic binary and its functionality can be extended over time with the introduction of new smart objects. FedNet acts as the underlying infrastructure that host all the smart

objects and applications that are loosely coupled. Using the documents it creates the spontaneous federation among these components thus eliminating direct dependency. Such framework design with suitable component representation (i.e., independent generic binary that can run in isolation) meets the first three requirements.

The support for end-user interaction tools can be seen as the by product of the loosely coupled design of the proposed framework as shown in Figure 3.5. Since all the installation, federation and run time data association are pushed to the infrastructure end, (i.e., to FedNet) while smart objects and applications run in isolation, it is possible to build secondary tools on top of the framework to administrate and monitor the components that run atop the FedNet infrastructure, i.e., the smart objects and applications.

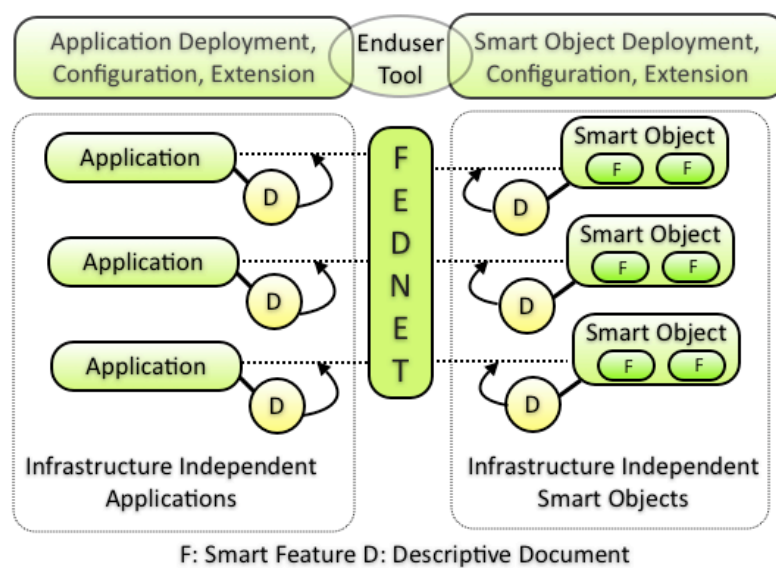


FIGURE 3.5: A Conceptual Document based Framework with End-User Tool

As a result of these architectural design choices and support for suitable interaction tool, the proposed framework enables end-users to engage in the deployment and administration processes of smart object systems. A couple of end-user interaction tools on top of FedNet using different user interfaces are developed in this work and will be presented in chapter 4.

Before moving to the next chapter, let us look at the contemporary works that have investigated end-user involvement in context-aware applications.

3.3.1 Related Work on Supporting Tools for End-Users

Most of the works that approached end-user support in pervasive literature have taken either rule based or recognition based perspective to customize the pro-activity of the application. Rule based tools like iCAP [Dey et al., 2006], Stick-e-notes [Pascoe, 1997], Alfred

[Gajos et al., 2002] provide visual tool, or sound macros to the end-users to define conditional rules based on the context to connect input and output events. Similarly recognition tools, or more formally Programming by Demonstration systems like CAPpella [Dey et al., 2004] uses machine learning techniques to allow end-users to associate personalized rules with real world events. These approaches are valid for rapid prototyping and also to personalize the pro-active behavior of the applications. However, they do not provide any general guideline regarding application or instrumented smart object deployment and extension by the end-users. Moreover, their support are primarily for developers to assist rapid prototyping and are not suitable for casual users with no or minimal technical background. As previous studies have shown, the deployment process has to be very simple with minimal configuration complexities [Beckmann et al., 2004, Antifakos et al., 2002]. Also, the process should resemble the current practices as closely as possible with which the end-users are familiar, e.g., installing a home appliance like a washing machine, a microwave etc.

One notable example is the Jigsaw Editor [Humble et al., 2003] that uses puzzle metaphor and allows non expert users to configure services intuitively by assembling available components (e.g., connecting a doorbell to a camera for taking a photo shot when someone rings the bell). Their study shows that end-users understand the semantic association of devices and can manipulate them in order to meet their changing house hold demands. The proposed design is highly influenced by their findings and the overall approach of providing simple, easy-to-use interaction tool to support deployment and administration is aligned with their projection. However, the proposed approach extrapolates by allowing end-users to introduce new applications or to extend existing smart object services using simple tools regardless of the application or smart object types which simplifies end-users involvement considerably. This in turn is possible because of the architectural qualities of the framework.

3.4 Chapter Summary

This chapter discussed the design issues for a suitable framework for building smart object systems and presented a conceptual document based framework. After providing the design requirements for representing a smart object as system component, the core-cloud model for smart objects was presented. In this model, the functionalities that are common across smart objects are combined together into a core. Specific smart features are then plugged into this core as clouds. This simple model satisfies the basic design requirements, i.e., decoupling smart features, building plug and play, reusable and incrementally extensible smart objects that can do both sensing and actuating. Then, the discussion was switched to the design requirements for a suitable infrastructure for smart object system. Five basic design concerns: handling heterogeneity, augmentation, management and evolution of smart objects and effective programming model were presented. A range of related infrastructures were introduced and their shortcomings were highlighted. This was followed by the introduction of the conceptual document based framework. The framework forces applications'

requirements and smart objects' features to be externalized through documents and employs a secondary infrastructure, FedNet to create a spontaneous federation among the application and smart objects using the corresponding documents. The framework along with the core-clod model for smart objects elegantly satisfy the basic design requirements for a supporting infrastructure to develop smart object systems. The chapter was concluded by looking at the user centric design aspects and argued that involving end-users in the deployment and administration of smart object systems can elevate user experiences. Accordingly, architectural qualities to support this involvement were discussed. In the next chapter the implementation and technical details of this proposed framework will be discussed.

Chapter 4

Implementation of the Framework

Chapter 4

Implementation of the Framework

Chapter 3 discussed the design challenges of building smart object systems. A range of systems were introduced that provide infrastructure support for smart objects systems and similar types of applications, e.g., distributed context aware applications. Leveraging off the previous works and design challenges a document centric framework was presented where smart objects' services and applications' requirements are objectified through external documents and a secondary infrastructure takes care of creating a spontaneous federation among these objects and applications. This chapter looks at the implementation of the proposed framework. Each of the framework components along with associated design methodologies is explained in detail. After that, end-user aspects of the framework is discussed. Two interaction tools built on top of the framework are presented that assist end-users in deployment and administration of smart object systems.

There are three primary components in the current framework: i) Smart Object Wrapper to construct and represent smart objects ii) Application Development Process to write applications for smart objects and iii) A secondary infrastructure FedNet, that provides the runtime association between the applications and spatially distributed smart objects. In the following sections, these three components are discussed in sequence.

4.1 Smart Object Wrapper

In chapter 3, a core-cloud model was presented that provides a theoretical foundation for building smart objects. It was discussed that the core encompasses the common features shared across a variety of smart objects where as the clouds are the specialized smart features that can be applied atop the core. In the proposed document based framework this core-cloud model is used to design smart objects and corresponding documents are used to externalize their services.

The core-cloud model is realized by the smart object wrapper component in the implementation of the framework. This component is used to construct a smart object and to represent it digitally. There are two steps involved in the construction process of a smart object:

1. *Selection of Appropriate Augmentation*: In the context of core-cloud model, clouds are the specialized features of a smart object. It is imperative to select these features in a systematic fashion so that they suit the physical object in context.
2. *Representation of Smart Object*: Once the augmentation role of the physical object is selected, the object can be instrumented accordingly. An instance implementing core-cloud model can represent the smart object digitally along with the corresponding documents.

Accordingly, this section first discusses a 3-Step design methodology for appropriate augmentation of a physical object (i.e., the selection of the clouds) and then discusses the technical details of the layered smart object wrapper component.

4.1.1 3-Step Design Methodology for Smart Object Augmentation

One intriguing factor is the selection of the smart features of a smart object, i.e., what augmentation role is suitable for a physical object? Typically the selection of augmentation role is highly influenced by the designers' intuitions and in most cases the selection process is ad-hoc. This ad-hoc fashion limits designers to repeat the steps in developing smart objects rapidly and consistently. In this section a 3-step design methodology is provided to formalize this role selection method. This design methodology is adopted from Fujinami's 5-step sensor selection framework [Fujinami and Nakajima, 2005]. The key point utilized for appropriate role selection is providing the right balance among the required functionalities, physical properties of a physical object and its interaction metaphor. Several works in the existing literature have looked at the augmentation selection considering only the sensing role of the smart objects. For example, Beigl and his colleagues showed the catalogues of sensors to select the appropriate sensor for fabrication from target phenomena's point of view [Beigl et al., 2004]. However, it lacks analysis of the earlier stage in terms of suitability of the smart object augmentation for the target phenomena. Furthermore, they did not consider the actuation role of a smart object. In the following a 3-Step design methodology (Figure 4.1) is proposed that considered both the sensing and actuating roles of smart objects while ensuring that the augmentations do not alter their physical appearances, properties and interaction metaphor.

1. **Step 1 - Clarify the Required Functionality**: The first step in designing a smart object is to answer the basic question, "What functionalities are required?" As discussed earlier, this functional requirement cannot be confined since it depends completely on designers' intuitions and target scenario in context. However, the core-cloud model

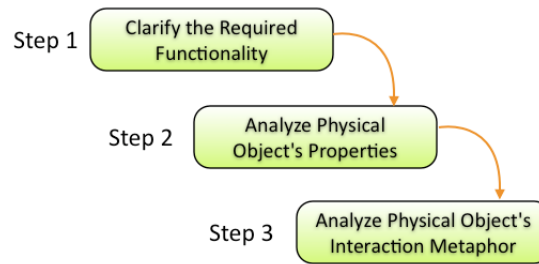


FIGURE 4.1: 3-Step Design Methodology for Augmentation Role Selection

provides an elegant solution to this problem by separating the runtime from the features thus allowing a smart object to support multiple functional roles. Thus in this step a designer can only consider the functionalities that he/she expects the physical object to support. These functionalities can be of two types: sensing and actuating. The former is responsible for observing some real world phenomenon where the latter is responsible for causing some phenomenon in the real world. For example, in AwareMirror [Fujinami et al., 2005] the required functionality is to show some superimposed data/image in a mirror, in Mediacup [Beigl et al., 2001] the required functionalities are to identify the state of use and physical conditions of the cup. So, in the first step a designer has to clarify his/her requirements for the augmentations in this fashion.

2. **Step 2 - Analyze Physical Object's Properties:** Once the required functionalities are determined, the next step is to analyze the target objects physical properties, i.e. size, shape, surface etc. In this step, the designer has to be very careful in ensuring that the augmentation does not alter the basic properties of the object, and that the physical object retains its original purpose intact even after the augmentation. This is imperative to make the augmentation natural as much as possible. For example, a mirror or a window can be augmented to support ambient display functionality, as their form-factors (e.g., flat surface) are appropriate for embedding display feature. This display feature does not conflict with their original purposes and does not alter their appearance significantly. Similarly, providing state-of-use context is suitable for a coffee mug as demonstrated by Mediacup [Beigl et al., 2001]. However it is impractical to augment a mirror for identifying "Being-Carried" context considering its size and shape. So, a designer should assess carefully whether the target smart object can provide the required functionalities naturally while retaining its physical properties. Schmidt has provided an excellent insight into the context patterns that can be observed by augmenting different kinds of real world objects varying in physical properties [Schmidt, 2002] and his observation can be used as a guideline for this phase to analyze the affordability of a smart object for the required functionalities.
3. **Step 3 - Analyze Physical Object's Interaction Metaphor:** The final step is to analyze the interaction metaphor of the target physical object and to identify how the augmentation will affect this interaction. The physical objects have evolved over the years for

their current interaction pattern and human has developed a specific mental model on that. While augmenting a physical object, it is vital to ensure that the original interaction metaphor is intact as much as possible. The augmentation may introduce new ways to interact with the object, but it must not alter and conflict with its original usage techniques. As Norman pointed out, a smart object may not be usable unless it retains its original interaction metaphor [Norman, 1990]. One way to approach this analysis is by first answering the questions: "How to use the object? How it affects the real world?" The result of the analysis classifies the usage into primitives, which include holding, opening, closing, bending, approaching, putting, removing, touching, leaving, pushing, pulling, rotating, shaking, leaving, storing, extracting, etc. For example, in case of sitting on a chair, a user's hip is "put" on the seat with some force, and the back is lean on the back seat i.e., "touching". Each of these primitives relate to a physical phenomena. So, once the original interaction primitives are identified, the designer should consider how the augmented feature will affect these interaction primitives. Ideally, the augmentation should exploit the original interaction primitives as much as possible. For sensing type augmentation, identifying the phenomenon caused by the interaction primitives provides the answer whether instrumentation can provide the right information. For example consider a augmented table, when something is "put" on the table, there might be a physical phenomena like the change of pressure on the surface, the vibration of the surface, noise, the change of temperature on the surface, etc. So, if the required functionality is to capture these phenomena to generate high level context, then the interaction required by the augmentation fits perfectly with the original interaction primitives of the table. Clarifying the interaction and corresponding physical phenomena in such a fashion allows a designer to augment the appropriate functionality to a smart object.

This 3-step design methodology provides a generic guideline regarding how to select the appropriate augmentation role for a smart object. Once the augmentation roles are defined the next phase is to physically augment the object by selecting appropriate sensors and actuators. Selecting the appropriate sensors and actuators for physical augmentation depends on many aspects, e.g. qualities, performance, form factor, cost, power consumption, availability, aesthetics, etc. The trade-off depends on overall requirements of the prototyping or product. Fujinami discussed these issues in detail in his sensor selection framework and his approach can be applied for physical instrumentation of a smart object [Fujinami and Nakajima, 2005]. Please note that, in the context of core-cloud model this 3-step design methodology provides a systematic way of selecting suitable clouds for a smart object. It is discussed that each smart object will have a common core that will provide the communication foundation and the run-time for hosting these clouds.

4.1.1.1 Illustration: Design of A Smart Mirror

In this section, the design of a Smart Mirror is discussed in the light of the 3-step design methodology.

- In the first step, it is determined that a contextual display functionality is required from a mirror. It is desired to augment a mirror in such a way that images and texts can be super imposed on the mirror surface when someone is in front of it.
- Next, the physical properties of the mirror are identified. These properties include flat surface, varying size, typically mounted on the wall or similar flat surfaces, etc. Also, the primary purpose of the mirror is identified, i.e., reflecting image of the entity in front of it. Here, the properties of interest are the flat surface and wall mountable attribute. A regular computer display can easily be attached at the back of a mirror, considering the display also has a flat surface. Also, attaching the display does not change mirror's original purpose of reflecting image. For identifying someone's presence in front of the mirror, a suitable sensing device (e.g., proximity detectors, motion detectors, camera etc.) can easily be attached on the frame of the mirror. This attachment does not conflict or alter its physical properties.
- In the next step, the interaction primitives of a mirror are analyzed. Typically, to use a mirror we just need to stand in front of it. A display has the similar interaction requirement, thus it can easily blend with existing interaction metaphor of a mirror. Most importantly it does not conflict with the original interaction primitive. Similarly the proximity sensing easily fits into a mirror's interaction primitive, since it only needs an entity to be present in front of the mirror.

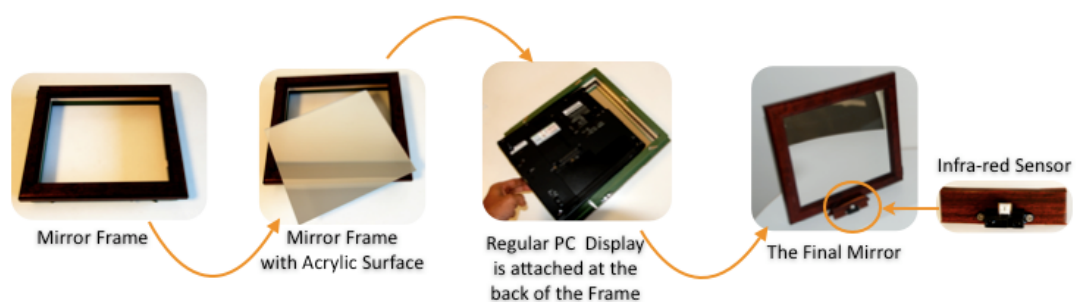


FIGURE 4.2: Construction of a Smart Mirror

Since this contextual display functionality from a mirror satisfies all three steps, we can now go for the actual instrumentation. The primary hurdle here is in finding the suitable mirror surface that allows images and texts to be super imposed simultaneously reflecting the image of the entity in front of it. If such a surface can be used, then the display can easily be attached at the back of the mirror. One such surface is acrylic magic mirror that allows bright colors to

penetrate the surface from behind. Utilizing this feature, a smart mirror is constructed where an ordinary computer display is attached at the back of the acrylic mirror surface. When the display is turned on, bright colored texts and images can penetrate the acrylic mirror surface and become visible from front. At the same time, the acrylic mirror reflects images of the entities in front of it. For enabling contextual display feature, a sensing device needs to be attached. In this construction an infra-red sensor is used because of its good performance, small size, cheap price and well availability. The complete construction process is depicted in Figure 4.2

4.1.2 Augmentation Presentation: Implementation of Core-Cloud Model

Once a smart object's augmentation scope is fixed, it is needed to create a suitable digital representation. The smart object wrapper provides this digital representation and follows the core-cloud model where basic smart object functionalities are combined in a core component. This core primarily encapsulates the communication capability of a smart object and provides a runtime to host the augmented features that can be added as plug-ins. In the last section, we have discussed a 3-Step design methodology that can be used to select these augmented features. In the context of smart object wrapper, each of these augmented features is called a service profile of a smart object. These profiles are physical object independent and represent generic services, For example: sensing room temperature could be one profile, and multiple physical objects (e.g., a window, an air-conditioner, etc.) can be augmented with a thermometer for supporting this profile. Core provides the runtime to host these services profiles and makes them available to external applications. In the current document based framework this externalization of smart object services are done through structured documents. The basic architecture of the smart object wrapper is shown in Figure 4.3. In the following the internals of this architecture is discussed.

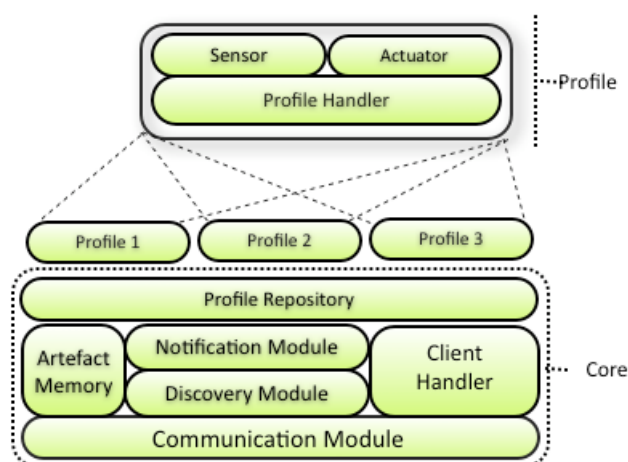


FIGURE 4.3: Smart Object Wrapper Architecture

4.1.2.1 Core Component

The core component of smart object wrapper encapsulates the common features (e.g., communication, static memory, etc.) shared across smart objects and provides a runtime to host the service profiles. The entire core is packaged in an executable binary and runs independently. Inside the core the following modules are pre-packaged.

1. **Communication Module** This module is responsible for external communications of a smart object and encapsulates the transport layer components. Upper layer modules contact with this module to interact with the external world (Internet, peer objects, applications, etc.). In the current implementation, primarily IEEE 802.11x (TCP/IP) and Bluetooth (RFCOMM) are supported in this module. All the communications are XML message based over HTTP where messages are encoded in prescribed document format.
2. **Discovery Module** Each object can advertise their services and can respond to query requests of external applications or peers using this module. After deployment, discovery module listens for broadcast messages and if the message requests match objects functional profiles, it responds to the message by sending the identity of the object for further communication using communication module. As we will see later in this chapter, in the context of current framework this module is used by smart object to interact with the runtime FedNet infrastructure.
3. **Notification Module** This module provides the service profiles with some common utilities to indicate their status. Considering, these notifications largely depend on the physical augmentation (i.e. LED, Text Display, Vibration, Sound, etc.) the current implementation provides libraries for commonly used tools. Further tools can be accommodated easily by extending this module.
4. **Static Memory** This is a shared space utilized by all the components for their data operations. Typically this module contains smart object's property data (color, shape, owner, etc.), profile descriptions, client lists and other temporal data needed by the profile handlers and corresponding profiles. In current implementation, this memory is managed in the form of a XML database.
5. **Client Handler** This component is the request broker for the profiles' services and delegates the external requests to specific profiles. It maintains a cache of client list and the corresponding profile information using the static memory module. It provides two modes of communication for the external clients (peer smart objects, applications). In Synchronous mode, the request is instantly processed and dispatched to clients (poll), where as in asynchronous mode, the information is dispatched to interested clients in an event based manner (subscription). Clients can subscribe to specific profile's service.

6. **Profile Repository** This component hosts the array of profiles and provides the plug-in runtime for service profiles to be attached dynamically. It utilizes a dynamic class loader to load the profiles dynamically when requested. Also, it maintains all the profile descriptions that it hosts in the static memory which is used by the discovery module to advertise the profile services.

4.1.2.2 Profile

A smart object can implement one or more profiles. Each profile implements a specific functionality and has a profile handler which implements the underlying logic of the profile service, i.e. providing context data by analyzing sensors' data that are attached to physical object or actuating an action of the object (e.g., increasing the brightness of a smart lamp, changing the orientation of smart blinds, etc.) The most basic profile is the one that tracks object's states and notifies via the notification module. During the bootstrapping process, profile repository dynamically loads the profile handlers and the profile descriptions which includes a brief overview of the profiles. Though multiple profiles could be part of a single object, the profile handler namespace is different. That means each profile handlers has its own client lists. When client handler receives requests from applications it simply delegates the requests to appropriate profile handlers. The profile handler has an abstraction layer that hides the heterogeneity of the underlying devices. Each profile is packaged as generic binary and get associated with the core dynamically through the plug-in runtime of the core.

4.1.3 Programming Model

As discussed in the previous section, there are two components in the smart object wrapper: i) the core and ii) profiles. The core is a generic component and is disseminated as a pre-packaged ready to use binary. The constituents modules of the core can be extended by the developers if needed. However, the profiles that a smart object can host are needed to be developed independently. In the earlier section, we have seen how a 3-Step design methodology can be used to select the augmentation role for a smart object. Once the roles are selected, it is required to develop the corresponding profile in the semantics of smart object wrapper. Thus a suitable programming model is required for the developers using which they can easily construct the profiles. In the current work, the programming model offered to the developers utilizes a profile based abstraction, i.e, developers can encapsulate the specific functionalities in one or multiple profiles. As shown in Figure 4.3 the profile consists of a profile handler and a device (sensor or actuator) handler. Thus developers are required to perform two tasks:

1. Writing a module to handle the device (i.e., sensors and actuators) specific code (i.e., accessing the sensors, collecting sensor data, etc.). This module should generate the higher level context and should execute the device functions as needed.

2. Connecting the above module with the smart object wrapper component. This component should provide suitable support to represent the generated context in framework specific ways. In addition, it should also interpret framework specific service requests in device specific way.

While the first task is completely device specific, the second task is more recurrent and thus suitable programming support is desirable. In the current implementation, the profile handler provides this support, and exposes a template for the developers to plug in their device code and context calculation/service actuation logic. Once developers attach their device specific code to profile handlers, the entire profile can be packaged as generic binary that can be plugged into the core. A profile implementation needs to inherit a base **Profile** class. This class enables the core to load this profile and to further communicate (forwarding application requests etc.) with it. Following Figure 4.4 and 4.5 show code snippets of how developers can use the profile handler template to plug-in their device specific codes for sensor and actuator type profiles respectively.

```

1. public class ProximityProfile extends Profile {
2.
3.     protected String position,distance;
4.     public ProximityProfile(String path)
5.     {
6.         super(path);
7.         position=""; distance="";
8.         new IRSensor(this); /* Handles the Protocol Heterogeneity */
9.     }
10.
11.    public void setSML() /* Sets the Profile Output in Predefined SML Syntax */
12.    {
13.        this.sml.setOutput("position", this.position);
14.        this.sml.setOutput("proximity", this.distance);
15.        this.notifyAccessPoint();
16.    }
17. }

```

FIGURE 4.4: Sample Code implementing Sensor Type Profile

The code snippets in Figure 4.4 shows the partial implementation of the proximity profile of the smart mirror illustrated in section 4.1.1.1. Here the mirror is instrumented with an infra-red sensor and the class **IRSensor** (line 8) implements the protocol specific code to access the sensor and to collect sensed data. Once the sensor data is analyzed for generating appropriate context information the **setSML()**(line 11) function is invoked that generates the profile

output in framework specific way, i.e., following the syntax of Profile Description Document (discussed in the next section).

```
1. public class DisplayProfile extends Profile {
2.
3.     public DisplayProfile(String path)
4.     {
5.         super(path);
6.         Display display = new Display(this); /* Handles the Protocol Heterogeneity */
7.     }
8.     /* Parameter represents the service requests generically */
9.     public String executeService(Service args)
10.    {
11.        String id = args.getIdentification();
12.        String state = args.getStateName();
13.        Input input = args.getInput(id,state);
14.        return(display.execute(state,input));
15.    }
16.}
```

FIGURE 4.5: Sample Code implementing Actuator Type Profile

The code snippets in Figure 4.5 above shows the partial implementation of the display profile of the smart mirror. As discussed, the mirror was instrumented with a regular display. So here the class **Display** (line 6) implements the protocol specific code to access the display. Once a service request (encoded in Actuator Modeling Language discussed in the next section) is received from the external entities (e.g., applications, peer smart objects, etc.) the base **Profile** class parses the request and corresponding display function **display.execute(state,input)** (line 14) is invoked to execute the display feature.

In chapter 3, we discussed that regardless of the type of services (e.g., sensing and actuating), a unified programming abstraction is needed to develop reusable and extensible smart object systems. Accordingly, in the current framework profile is used as the primary abstraction for encapsulating both sensing and actuating. A profile is packaged as generic binary that allows it to be snapped with the generic core at runtime. This generic binary and profile based unification have two direct side effects: i) it enables a profile to be reused since a suitable profile once developed can be used in multiple smart objects. ii) it enables a smart object to be extensible as profiles can be added at anytime to a core of a smart object.

4.1.4 Representative Documents

In the earlier section, we have seen how profiles can be constructed that are dynamically associated with the core. This association is done through documents, the inherent part of the current framework. In addition, documents are also used to externalize the services of smart objects that can be discovered and exploited by external applications and peer smart objects. There are primarily two documents utilized for smart object wrapper, these are:

1. **Smart Object Description Document (SODD):** This document is the generic description of the smart object in context as shown in Figure 4.6. It provides the meta information regarding the smart object. However the most important part of this document is the **<profiles>** node. The smart object core parses this node to dynamically load the profiles. The node contains links to the generic binaries of the respective profiles. This document is also used by the secondary infrastructure FedNet (as we will see in the later part of this chapter) to discover the services of the smart objects and associate smart objects with applications.

```
<?xml version="1.0"?>
<artefact>
  <name>Mirror</name>
  <vendor></vendor>
  <profiles>
    <profile name="Proximity">
      <codebase>ArtefactSpace/Mirror/ProximityProfile/ProximityIRProfile.jar</codebase>
    </profile>
  </profiles>
</artefact>
```

FIGURE 4.6: Smart Object Description Document for a Mirror with *Proximity Profile*

2. **Profile Description Document (PDD):** Each smart object contains one or multiple profiles. These profiles are either sensor type or actuator type and usually have different input and output specifications. However, to create synergies between smart objects and applications, it is needed to have a pre-negotiations on the format of the moveable data. In the current framework context, this issue is addressed by forcing each profile to publish its input/output data format in structured documents so that the external applications and peer smart objects know how to interact with the profile's service. This document is also used by profile handler to encode implementation output and to decode service request inputs. Each PDD contains either a **<detector>** or an **<actuator>** node based on the profile type. It also contains a quality of service (QoS) block which specifies profile's quality. Furthermore, PDD contains an **<installation-instruction>** block that provides hardware installation guidelines for end-users. The later part of this chapter will discuss how this installation guideline is used to assist end-users in the deployment tasks.

The sensor type profile's description follows the specification of the customized Sensor Modeling Language (SML) adopted from SensorML¹. It specifies the output format of the profile with parameters as shown in Figure 4.7. The primary reasons of adopting SensorML are its soft typed attribute, reference frame and parameters, with which the semantics of different sensor data platforms can easily be understood and interchanged.



FIGURE 4.7: Profile Description Document for a sample *Proximity Profile*, Customized SensorML is used in the *detector* node.

For an actuator profile custom designed *Actuator Modeling Language (AML)* is used as shown in Figure 4.8. The **<state>** node is used to abstract the operational states of a smart object's service. It contains the input parameters to change the states along with required data type. Please note that the protocol to handle the underlying device is implemented in the profile implementation.

¹OpenGIS Sensor Modeling Language (SensorML) Specification: <http://www.opengeospatial.org>



Profile Description Document for actuator type profiles.
Actuator Modeling Language is used in the <actuator> node.

FIGURE 4.8: Profile Description Document for a sample *Display Profile*. Custom designed Actuator Modeling Language is used in the *actuator* node.

4.1.5 Location Modalities of Smart Object Wrapper

The smart object wrapper essentially is the digital identity of a smart object. So an obvious issue is the location of this digital part. There are two choices as shown in Figure 4.9: i) At-the-

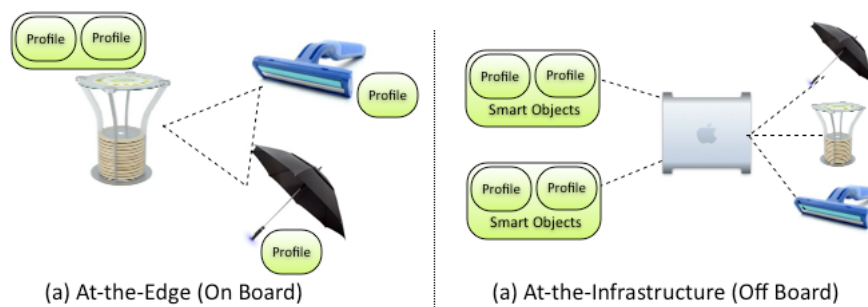


FIGURE 4.9: Location Modalities of Smart Object Wrapper

Edge (On-Board) and ii) At-the-Infrastructure (Off-Board). At-the-Edge means that a smart object itself has a processing unit that hosts its digital representation (i.e., smart object wrapper) whereas At-the-Infrastructure means that a proxy, running in a separate location represents a smart object and communicates with the smart object to retrieve sensor data or to actuate smart object's functions using some communication protocol, e.g., Bluetooth, IEEE 802.11x, etc. Both choices have pros and cons. While At-the-Edge approach provides pre-configurable, self sustainable and aesthetic smart objects, it has minimal support for rapid prototyping, DIY

(Do-It-Yourself) approach and prone to limited capability. On the other hand, although At-the-Infrastructure approach requires manual configuration and maintenance, the primary advantages are the interoperability (through secondary infrastructures) and DIY support. Also, it enables rapid prototyping.

4.1.6 Smart Object Life Cycle

In the earlier sections, the layered architecture of smart object wrapper that implements the core-cloud model is discussed along with the programming model and expressive documents. Both the core and profiles are packaged as generic binaries, and dynamically snapped with each other at runtime. The entire life cycle of a smart object in the context of current framework is shown Figure 4.10.

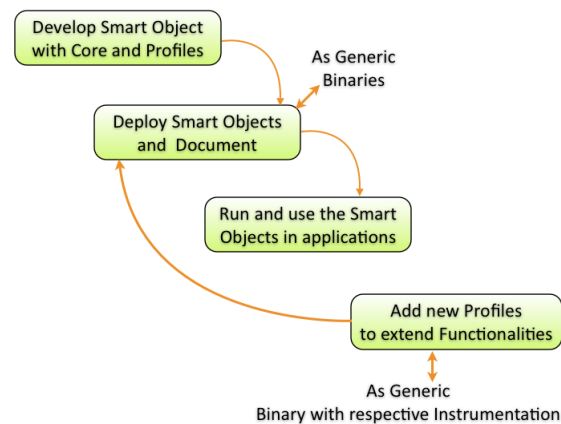


FIGURE 4.10: Life Cycle of a Smart Object

So, once a smart object is constructed with a core and suitable profiles it can be deployed with associated documents that externalize the profile services. These documents are used by the secondary infrastructure to let applications use the smart objects. Furthermore, the initial functionalities of a smart object can be extended afterwards. From a developer's perspective, the construction and extension of a smart object thus consist of the following three activities:

- Instrumenting the physical object by selecting appropriate roles applying the 3-Step design methodology.
- Implementing the profile following the prescribed programming model.
- Externalizing the profile specification in Profile Description Documents (PDD) and writing the generic Smart Object Description Document (SODD) with appropriate **<profiles>** nodes to let the smart object core dynamically load the attached profiles.

In the next section, we will look at the document based application development process that enable applications to utilize these smart objects.

4.2 Application Development Process

In chapter 2, the classification of smart object systems showed that a majority of smart object systems composed of one or multiple applications that integrate one or multiple smart objects to provide proactive services. It was also discussed that these applications are typically context aware in nature and composed of three components: i) basic application component, ii) communication component, and iii) perception and adaptation component. The first and third components are basically application specific and usually differ from application to application. However, the second component, i.e., the communication component is recurrent across applications as it is responsible for communication and interaction with the external environment, i.e., smart objects. Typically these interactions include accessing and managing smart objects, aggregating context data, actuating services etc. Thus a supporting infrastructure that provides simple mechanisms to take care of these recurring tasks can dramatically reduce the complexities of application development.

In the proposed document-based framework these recurrent actions are supported through indirection using documents, i.e., application specifies their requirements in a high level structured document without considering how to attain those requirements. A secondary structure FedNet uses these documents along with smart objects' documents (explained earlier) to create a runtime association between the application and the smart objects. The infrastructure provides an access point to applications with unified access mechanism for interacting with the underlying smart objects. However, there is a 3-Step development process that an application developer has to follow to enable an application to work in the current framework context. In the following this design process is explained.

4.2.1 3-Step Application Development Process

An application developer can follow any library and implementation language to code the execution logic of the application and to construct the application parts. To enable the application to interact with the underlying environments, however, a developer has to follow the semantics of the current framework. Essentially, there are three steps as shown in Figure 4.11 that a developer should follow:

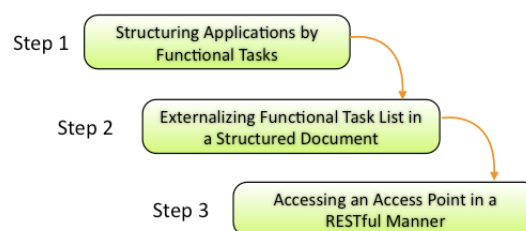


FIGURE 4.11: 3-Step Application Development Process

1. **Step 1 - Structuring Applications by Functional Tasks:** An application is composed of several functional tasks, i.e., atomic actions. In smart object applications, these atomic actions may be: *"turn the air-conditioner on"*, *"sense the proximity of an object"* etc. The first step of the application development in the current framework context is to structure the application in a collection of functional tasks that require the services of smart object profiles (e.g., context data, service actuation etc.)
2. **Step 2 - Externalizing Functional Task List in a Structured Document:** The next step is to externalize application's requirements as a collection of functional tasks in a structured Task Description Document (TDD). Each task specifies the respective profiles it needs to accomplish its goal. Figure 4.12 shows part of the Task Description Document for a smart display application. Each task also contains Quality of Service (QoS) requirements for the target profiles. This document is used by the secondary infrastructure FedNet to identify the required smart objects and to provide a spontaneous federation between the application and smart objects. There are a couple of things that need further discussion at this point. As we see in the Figure 4.12, the application does not state a specific smart object, rather it only puts forward its profile requirements. This enables any suitable smart object to be used in the application as long as the object provides the required service profile. The next point of interest is the description of a task. The task can describe not only the specific service but also the required communication mode, i.e., synchronous (poll) and asynchronous (subscription). These modes are specifically important for aggregating context data from smart objects. The final point of interest is the meta information regarding the application. This meta data contains the access point identity and will be explained in the next step.
3. **Step 3 - Accessing an Access Point in a RESTful Manner:** The final step in application development is enabling an application to interact with smart objects in a unified manner. Typically this interaction includes discovering and managing smart objects and accessing their services. However, in the current framework context the discovery and management processes are eliminated from the application scope. By externalizing smart object service requests through Task Description Document, applications allow the secondary infrastructure FedNet to perform discovery of smart objects. However, application still needs to access smart object services. Addressing this access issue at the application level makes application development very complicated. In current approach, this access mechanism is simplified by enabling an application to access a common point for interacting with underlying smart objects, regardless of their types and protocol heterogeneity. When an application is deployed, the secondary infrastructure assigns an access point to the application, and the identity of that access point is injected into the meta data block of the Task Description Document as shown in Figure 4.12. An application needs to access this point to send requests and to receive responses from the underlying smart objects. During the application's instantiation time, the required physical smart objects data semantics (*<detector>* and *<actuator>* nodes of the Profile Description Document) are provided to the application from this access point by the

```

<?xml version="1.0" encoding="UTF-8"?>
<application>
  <name>Smart Display Application</name>
  <app-purpose>Providing Personalized Information with Situational Awareness</app-purpose>
  <binaryPath>ApplicationSpace/SmartDisplay/SmartDisplayApp.jar</binaryPath>
  <accesspoint>
    <IP>10.0.1.3</IP>
    <port>9824</port>
  </accesspoint>
  <task-list>
    <task>
      <id>T1</id>
      <purpose>Measuring Proximity</purpose>
      <required-profile-type>Sensor</required-profile-type>
      <profile-name>Proximity</profile-name>
      <communication-mode>asynchronous</communication-mode>
      <profile-QoS-attribute>
        <qos>
          <name>latency</name>
          <datatype>int</datatype>
          <measurement-unit>millisecond</measurement-unit>
          <high-threshold>70</high-threshold>
          <low-threshold>60</low-threshold>
        </qos>
      </profile-QoS-attribute>
    </task>
    ----- More Tasks -----
  </task-list>
</application>

```

FIGURE 4.12: Task Description Document (partly)for a Smart Display Application

secondary infrastructure FedNet, to let the application prepare for the moveable data accordingly. Considering the simplicity and proliferation of web technologies, the access mechanism and data exchange protocol between the application and access point are based on HTTP and XML following the Representational State Transfer (REST) approach [Fielding, 2000]. For continuous polling (i.e., subscription), auto discoverable RSS feeds are used.

4.2.2 Programming Model

In the last section, a 3-Step development process is discussed for building applications integrating smart objects in the context of current framework. Application developers can follow any language and implementation platform for developing these applications as the interaction protocols are based on generic web technologies. However, in the current implementation, a simple library in Java comprised of a REST features (simple HTTP/XML) and Auto Discoverable RSS Parser is provided to the application developer that they can use to further simplify the application development process.

One important aspect of the application development is the programming abstraction. As discussed in the earlier section, the primary abstraction offered to the developers is *Task*. Applications are structured based on the functional tasks (atomic actions) that require smart object services. The offered library makes this task based application structuring simple. The following code snippets in the Figure 4.13 illustrates this task based application development.

```
1. Enumeration<Task> vector = this.taskList.elements(); /* Retrieve all the tasks */
2. while(vector.hasMoreElements()){
3.     Task task=(Task)vector.nextElement();
4.     if(task.getID().equalsIgnoreCase("T1") && task.getProfileStatus()){
5.         /* Task is executable, profile is available */
6.         AccessPoint.sendTaskRequest(xmlProc.generateOutgoingMessage(
7.             Constant.TASKREQUEST,task.getID()));
8.         task.subscribe(this,"proximityListener");
9.     }
10. }
11. /* Listener for Sensor type Profile, Parameter is structured
12.    following Profile Description Document Semantics */
13. public void proximityListener(String profileSML)
14. {
15.     DetectorData data = xmlProc.parseIncomingMessage(profileSML);
16.     /* perform actions based on sensed data */
17. }
```

FIGURE 4.13: Sample Task based Application Code

As we discussed once the developers identify the functional tasks, they need to externalize them in a Task Description Document as illustrated in Figure 4.12. Then developers can use the task reference accordingly in the application by using language interfaces provided in the simple library. For example, in the above code snippets application retrieves all the tasks that are externalized in the document (line 1) and then based on the availability of the smart object that supports the profile requirements of the respective tasks, application can request specific profile's service (line 4-8). Since the task request for sensor data in asynchronous mode, i.e., subscription, a corresponding callback function is stated (line 8). This function is invoked by the underlying library using reflection mechanism to update the application with sensor data encoded in prescribed format. Please note that applications only use the task reference here without considering discovery, access mechanism, service actuation policies etc. All these are moved away from the application scope and performed by the secondary infrastructure FedNet. Applications are only responsible for utilizing the profile service. Such high level abstraction completely hides the heterogeneity and distribution aspects making application development very simple and rapid. Furthermore, there is no dedicated link between the applications and smart objects, enabling a smart object to be replaced by another that provides

similar profile service. This is very important in a dynamic environment where a smart object typically runs, since smart objects may move away from the application scope or fail. In addition, this task based abstraction also enables applications to extend their functionalities over time. There may not be any suitable smart objects for some tasks at the early stage of the application deployment. However with the introduction of new smart objects those tasks can be supported afterwards, thus allowing the extension of applications' functionalities. Such extension of application features using existing abstraction (e.g., task) is very crucial for smart object systems as discussed in chapter 3.

The final point of discussion here is the binary structure of the applications. Every application in the current framework context is disseminated as generic binary. This is possible because there is no hard dependencies between the application and the underlying infrastructure. Applications use generic web technologies in a RESTful manner to access the infrastructure (i.e., access point) and thereby smart objects. This allows applications to be packaged independently. In the meta data part of the Figure 4.12, the **<binaryPath>** node specifies the path of application binary. This binary path is utilized by the infrastructure components to enable end-user activities. This will be further discussed in the later part of this chapter.

The next section will bring the spotlight on the secondary infrastructure that plays the key role in providing the runtime association between the applications and smart objects in the context of current document based framework.

4.3 FedNet Infrastructure

In the current document based approach both the applications and smart objects are infrastructure independent and expressed in high level descriptive documents. Thus to create a runtime association between the applications and underlying smart objects, an intermediary is needed that can connect the applications with respective smart objects. This intermediation is done by FedNet in the current framework. FedNet provides this spontaneous federation by utilizing the descriptive documents of the smart objects and applications that are discussed earlier. FedNet can contact the communicator module of the smart object core using the semantics described in the smart objects' documents for mapping applications' tasks. Similarly application can contact FedNet using generic web access mechanisms in a RESRful manner. In the following the internal architecture of FedNet is explained to illustrate how it provides these supports to smart objects and applications.

4.3.1 Logical Architecture of FedNet

FedNet itself is packaged in a generic binary and composed of four components as shown in Figure 4.14.

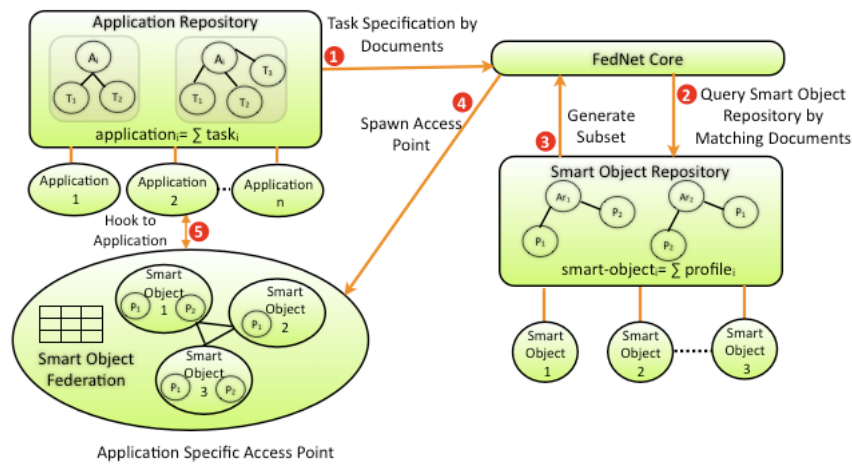


FIGURE 4.14: Logical Architecture of FedNet

4.3.1.1 Smart Object Repository

Smart Object Repository manages all the smart objects running in FedNet environment. During smart objects' deployment, the executable binary implementing the smart object wrapper and the Smart Object Description Document (SODD) are submitted to this repository. When a profile is added to a smart object, the profile information is dynamically injected into SODD and the respective profile is attached to the smart object. Thus this repository has the information of all the profiles available in the environment at any time.

4.3.1.2 Application Repository

Application Repository hosts all the applications that run on FedNet environment. During an application's deployment, the binary executable and the Task Description Document (TDD) are submitted to this repository. FedNet Core generates an access point for the application and updates the respective TDD by dynamically injecting the identity of the corresponding access point.

4.3.1.3 FedNet Core

FedNet Core provides the foundation for the runtime federation. When an application is deployed the task specification is extracted from the application repository by the FedNet Core. It analyzes the task list by querying the smart object repository using structural type matching of documents and generates an appropriate template of the federation and attaches it into a generic access point component for that application. This type matching basically involves extracting the profile requirement from the Task Description Document and then searching the smart object repository to find the appropriate profile using the Smart Object Description

Document and its constituent Profile Description Document. Since all these files are written using XML, this searching is performed by using XQuery². When an application is launched, the access point is instantiated and the respective template is filled by the actual smart objects available in the environment right at that moment thus forming a spontaneous federation.

4.3.1.4 Access Point

Access Point represents the physical environment needed by an application. FedNet assigns a unique access point for each application rather than providing a common access point for all applications. FedNet takes this approach considering the following two principles:

- Every application has unique runtime requirements for smart objects.
- Even though there are multiple applications deployed in the environment, there might be cases when some applications are not running all the time. Thus maintaining a common access point involving multiple smart objects that are not used by the running applications leads to high runtime cost (e.g., unnecessary resource consumption, management complexities etc.).

This means multiple federations of smart objects can co-exist in the environment. Simultaneously, each smart object can participate in multiple federations. When an application is launched, the access point sends the federated smart objects' data semantics, (i.e., Sensor Modeling Language and Actuator Modeling Language) to the application. This allows an application to know the semantics of movable data in advance. From then on, the application delegates all its requests to the access point which in turn forwards them to the specific smart objects. The smart objects response to these requests by providing their profile outputs either by pushing the environment state (actuation) or pulling the environment states (sensing) back to the access point that are fed to the application.

4.3.2 Physical Architecture of FedNet: Distributed Management

The last section provided the explanation of the functional roles of the primary components of FedNet infrastructure. From physical implementation point of view all these components could be distributed, i.e., instrumented smart objects can run in their own nodes, applications can run on the smart object nodes, or in separate nodes integrating multiple smart objects. Similarly, FedNet components can run in a common node or can be distributed over a number of nodes that collectively manage the environment.

The primary point of interest here is the structure of the smart object repository. It is discussed that all the interaction between applications and smart objects are mediated via FedNet utilizing the notion of application specific access point. This interaction is done over

²<http://www.w3.org/TR/xquery/>

generic web protocols (HTTP/XML) in a RESTful manner. Thus application does not need to worry about the protocol heterogeneity at the smart object end. However, as we discussed in section 4.1 that a smart object can have different transport protocols, and the specific instrumentation for supporting profiles may also introduce heterogeneous interfaces. For example, consider a smart umbrella augmented with a temperature sensor that has RS232 serial interface, and the umbrella itself has Bluetooth interface to connect to external entities to provide the sensed temperature. Thus, smart object repository can host smart objects that might have different transport protocols and data semantics. Earlier we have discussed, how the programming model of smart object wrapper hides the sensor/actuator level heterogeneity by using profile handlers. So, in this example, a suitable profile implementation will be able to handle this RS232 interface, and to generate the sensor output in specific format following the Profile Description Document (PDD). However, the smart object itself will have a communication module that implements Bluetooth interface, where as the entire FedNet communication (i.e, between applications and access point, between access point and smart objects) is HTTP/XML based. Thus to enable unified communication among all these components, it is needed to handle the protocol heterogeneity (e.g., discovery, data semantics, communication etc.) introduced by smart objects.

The smart object repository handles this by using an IP based bridging mechanism. The structure of the smart object repository is shown in Figure 4.15. It is composed of five components that are explained in the following.

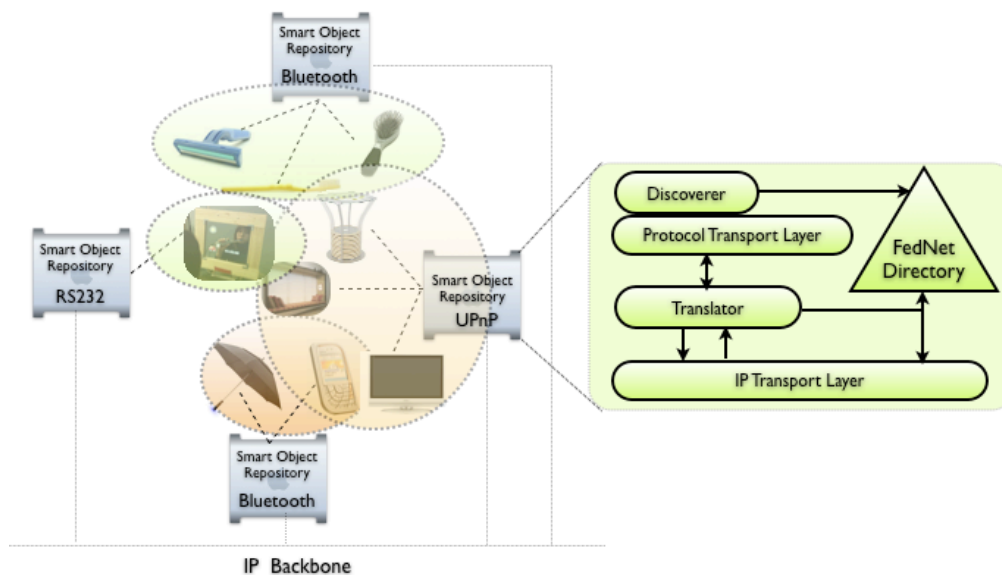


FIGURE 4.15: Internal Structure of Smart Object Repository

1. **Discoverer:** This component is responsible for implementing protocol specific discovery mechanisms. For example, in case of Bluetooth it implements Service Discovery Protocol (SDP).

2. **Protocol Transport Layer:** This component handles the device (sensor/actuator) specific transport protocols. For example, in case of Bluetooth nodes it implements Radio Frequency Communications (RFCOMM) protocol.
3. **Translator:** Each protocol has its own specific data format. The translator component translates this data format into the format of Profile Description Document and vice versa.
4. **IP Transport Layer:** This layer is the FedNet specific interface and enables access points to interact with the smart objects through IEEE 801.11x interface.
5. **FedNet Directory:** This is a static memory of smart object repository where it maintains service information of all the available smart objects that it handles with in its scope. This directory is updated periodically to reflect the availability of the smart objects. This information is used by FedNet to map applications' tasks to smart objects' services.

In essence, smart object repository is composed of a collection of distributed nodes, where each node is responsible for a specific native protocol (e.g., Bluetooth, UPnP, IEEE 802.11x, RS232 etc.). All these nodes are connected to an IP backbone. However this is the case for a large scale system; for a simpler system, one node can accommodate multiple native protocol components.

4.3.3 Specific Features of FedNet

FedNet provides the runtime association between applications and smart objects. The earlier sections illustrated how FedNet enables this association by using the descriptive documents. There are a couple of issues that are hidden in this runtime association process. This section brings those two issues under spotlight.

1. **Discovery Service:** In FedNet there is no dedicated discovery service for the applications. Application exports their task lists through Task Description Documents. When a smart object joins the environment it notifies the FedNet about its profiles through its respective Smart Object Description Document. In FedNet environment, both the applications and smart objects are explicitly deployed along with their documents. FedNet uses these documents and perform structural type matching to map application tasks to respective profiles to enable the spontaneous federation. Because of this indirection, an explicit need for discovery service is eliminated from the application perspective.
2. **On-Demand Resource Allocation:** The environment resources are only allocated to an application at runtime in FedNet, i.e., when needed by the application execution logic. This on-demand federation enables a smart object to participate in multiple federations, since there is no hard-leasing method from the application instantiation time. Simultaneously, multiple applications can use the same smart object. However, in the

case of conflicts among multiple applications for a single actuator type smart object (e.g., a mirror display), a simple FIFO (First In, First Out) basis resolution logic is used to break the tie.

4.4 Framework Support for End-Users

The earlier part of this chapter discussed the primary components of the proposed document based framework. This sections brings the focus on end-users. Particularly, it will discuss how the proposed framework exposes a few architectural qualities that elevate end-user experiences with smart object systems built atop the framework.

In chapter 3, it was discussed that one way to elevate end-user experiences with smart object system is to involve them actively in the deployment and administration tasks in a similar fashion that mimic end-users current practice with furniture and other home appliances. In chapter 1, we also discussed that how the existing personal computers and mobile phones enable end-users to involve in the deployment and extension tasks. These supports eventually help the proliferation of these platforms. This work argues that, smart object systems should have identical support for end-users. Enabling end-users in the deployment and administration tasks requires suitable support from the infrastructure that is used to construct the smart object systems. Recall from chapter 3 that there are four primary architectural requirements in providing end-users with the deployment and administration supports:

- Plug and play smart objects and applications.
- Extensible smart objects and applications.
- Loose coupling among the smart objects, application and the supporting infrastructure.
- Support for a suitable end-user interaction tool to assist the deployment and administration tasks.

The earlier part of this chapter discussed how the first three requirements are fulfilled by the current document-based framework. Specifically, the smart object wrapper implementing core-cloud model enables development of plug and play smart objects. Also, profiles can be added to a smart object at anytime thus enabling a smart object to be extensible. Similarly, it was discussed that applications are developed independently with task based structuring which allows applications to be extended over time with the introduction of new smart objects. Furthermore, both the applications and smart objects are disseminated as generic binaries along with respective documents without having any dedicated dependencies on the underlying infrastructure. This in turn enables the secondary infrastructure FedNet to provide the runtime association between smart objects and applications exploiting the documents. A

combination of these aspects enable the deployment and extension of smart objects and their applications without any side effects.

Simultaneously, since each of these components (i.e., smart object, application, infrastructure) is a generic binary, it is possible to construct tools to manipulate them. In the context of current work, these manipulations include deployment, configuration and extension of smart object systems. So, if such tools can be constructed that enables these manipulations in a seamless fashion than end-users can easily be involved in these tasks. Typically, this tool should be identical to the "*installer*" concept of personal computer platform that enables seamless installation of application in the personal computers. Accordingly, a couple of end-user interaction tools on top of the current framework are designed in this work to support end-users in the deployment, configuration and extension of smart object systems. In the following these interaction tools are discussed.

4.4.1 End-User Interaction Tools

The end-user interaction tool is needed to install the smart objects and applications into the corresponding environments. In the language of the proposed framework the tool should enable end-users to install the smart objects and applications into corresponding smart object repository and application repository of FedNet and to add profile plug-ins into the smart objects. Furthermore, this tool should enable the end-users to configure and control these smart objects and applications. There are two fundamental design requirements that need to be fulfilled while designing tools for these tasks.

1. **Contextual Feedback and Guidelines:** It is necessary to provide feedback of users' actions and to guide the users with proactive suggestions [Antifakos et al., 2002].
2. **Semantic Mapping of User Actions:** To support distributed deployment with a singular tool we need to identify which action is for which entity (smart object, application and profiles) i.e., we need a selection phase followed by an action phase.

Following these design guidelines, two deployment tools have been constructed, the first is using a graphical user interface and the second is using a tangible user interface.

4.4.1.1 Graphical User Interface Interaction Tool

The first tool supports end-users in the deployment process by providing a graphical user interface. A Web 2.0 application is built atop the framework that allows end-users to deploy smart objects and applications in the environment. This tool acts as an interface for end-users to interact with FedNet and all the actions performed by the user are transformed into specific

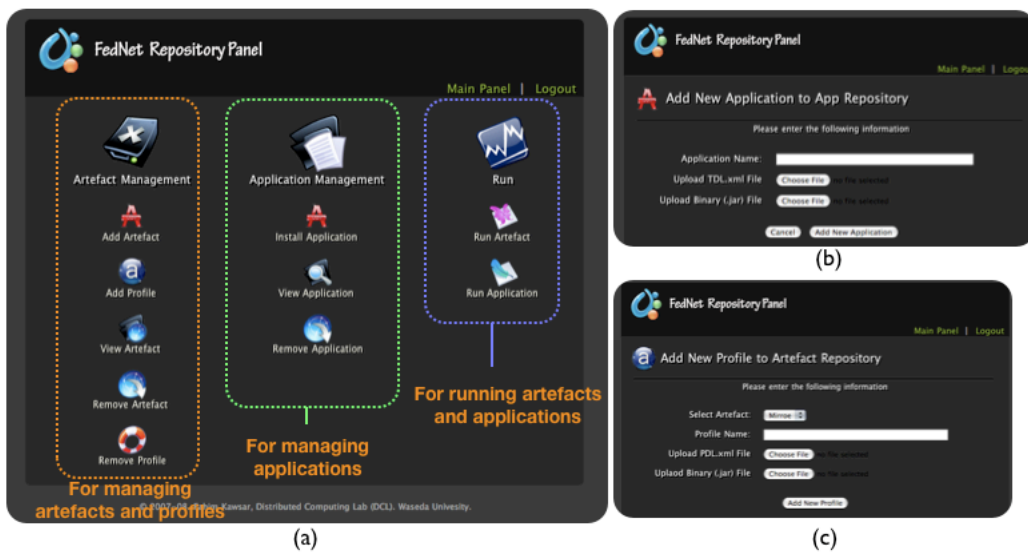


FIGURE 4.16: Snapshots of Web based Graphical User Interface Interaction tool (a) Main Panel, (b) Installing Application (c) Adding Profile

actions of FedNet as discussed in the earlier sections. The snapshots of this tool are shown in Figure 4.16.

There are three parts dedicated for three categories of tasks involved in the deployment and administration of smart objects systems. These are:

- **Deploy and Extend Smart Objects:** These functions are grouped under the Smart Object panel. Each smart object and its profiles is disseminated as generic binary that can be downloaded from a pre-designated address in the Internet or can be supplied in a portable media (e.g., a CD, a Flash Memory Stick, etc). End-users can deploy a smart object or extend a smart object service by attaching appropriate profile using functions available in the tool.
- **Deploy Application:** These functions are grouped under the Application panel. Each application is packaged as generic binary that can be downloaded from a pre-designated address in the Internet. Alternatively, it can be supplied in a portable media (e.g., a CD, a Flash Memory Stick, etc). End-users can deploy an application and associate it with a specific smart objects or a group of smart objects using functions available in the tool.
- **Configuration and Control:** These functions are grouped under the Configuration panel and allows end-users to configure a specific smart object or application. Furthermore, end-users can start and stop an application or a smart object using these functions.

Each of the functions of this tool is guided by suitable textual and graphical user guidelines and each user action is responded with proper feedback. For installing profiles, it is also needed

to physically instrument the smart object. The installation steps are provided in the Profile Description Document (PDD) as shown in Figure 4.7 and 4.8. These steps are provided to the user through textual feedback. If users make mistakes, a proper feedback is instantaneously generated and appropriate steps are suggested. By using GUI based selection of applications and smart objects, this tool can associate users' actions to respective components.

4.4.1.2 Tangible User Interface Interaction Tool

A pilot user study revealed several usability problems of the Graphical User Interface tool as it contradicts end-users' conceptual model of installing home appliances. Although, the deployments tasks were identical to regular desktop computing, e.g., software installation, end-users found it difficult to comprehend and suggested that the process has to be more mechanical and tangible. Also, the GUI suffered from fragmentation of attention as users had to switch back and forth from GUI to physical artefact. Considering the implications a tangible interface is designed for the deployment purpose. RFID (representing smart objects, application and profiles) and RFID Reader with touch buttons are provided for end users' interaction. Following the design guidelines, the tool provides visual (blinking LEDs) and sound feedback along with speech guidelines. These are contextual, i.e. the possible next steps are provided based on the user's latest activity and the state of the system. By using RFID identification for selecting applications and smart objects, the tool can associate users' actions into corresponding components.

Hardware The deployment tool is built with a RFID reader, 3 touch buttons and 3 LEDs as shown in Figure 4.17. The touch buttons and LEDs are connected to an interface kit. The whole unit has a USB interface and can be connected to a regular PC/laptop that provides the power, audio output and controls the unit.

Interaction Mechanism Each application, smart object and profile is disseminated as an executable binary. Here the assumption is that each of these is packaged with a RFID card that embeds a remote URL from where the binaries can be collected. There are six functions (organized in 3 groups) that the end-users can perform using these cards and the deployment tool.

- *Deploy a Smart Object or Application:* These functions can be performed by placing the RFID card of the smart object/application and then pressing the green button as shown in Figure 4.17-(a). During installation, the binary is downloaded from the remote URL embedded in the RFID. Current implementation assumes that RFID contains a unique number which is resolved by consulting a secondary file to extract the binary URL.
- *Administrating a Smart Object or Application:* These are performed by placing the RFID card of the smart object/application and then pressing the red button as shown in Figure

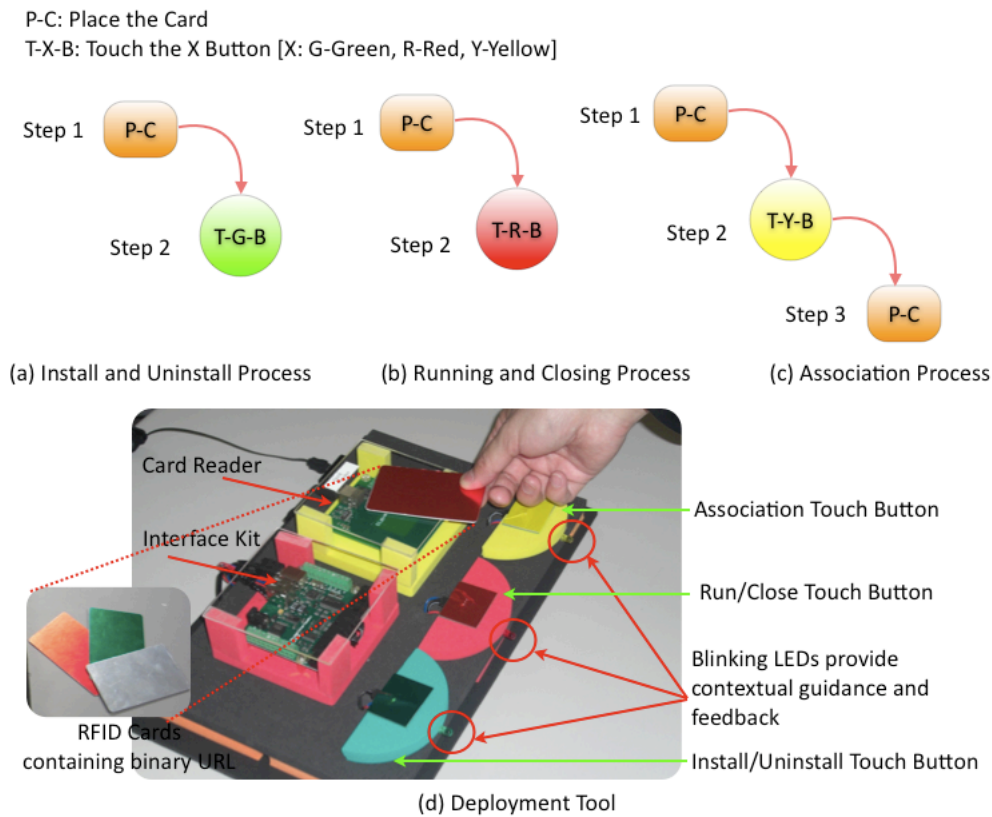


FIGURE 4.17: Tangible User Interface Interaction Tool

4.17-(b). Although most of the smart object applications are assumed to run continuously, these explicit controlling functionalities are provided to the end-users since there is no secondary user interface in the system. Thus if a user needs to stop an application or smart object's functionalities for some reason (e.g., going for a vacation), they can do so using these functions.

- *Associating/Removing a Profile/Application*: These functions are needed to associate a profile or an application to a specific smart object. Note that the applications that are not associated with any specific smart object can be installed following Figure 4.17-(a). However, for the applications and profiles that are specific to one smart object an association phase is needed. This is done by placing the target smart object card first into the reader, touching the yellow button and then placing the newly installable profile's or application's card into the reader as shown in Figure 4.17-(c). If a profile description file contains hardware installation instructions, the tool provides audio guidelines to support the installation.

In the next chapter, a usability study of these tools are presented where end-users are actively involved in deployment, configuration and extension of a couple of smart object systems.

4.5 Chapter Summary

This chapter provides the implementation and technical details of the proposed framework. Each of the component is discussed in detail with appropriate illustrations. The early part of the chapter discussed how a physical object can be augmented by following 3-Step design methodology to select the augmentation role and then using smart object wrapper to provide its digital counter part. Then the task based application development process and corresponding programming model were explained highlighting the library support for the developers. After that FedNet, the infrastructure that provides the backbone of this framework was discussed in detail with illustrations of how it provides the runtime federation among the smart objects and applications. Then the chapter focused on end-user aspects, and discussed how the proposed framework provides support to involve end-users in the deployment and administration of smart object systems. Two end-user interaction tools built on top of the framework were also presented. In the next chapter the evaluation of the framework will be discussed.

Chapter 5

Evaluation

Chapter 5

Evaluation

The last chapter had given a deep insight into the technical detail of the proposed framework. Primary framework components were discussed with appropriate illustrations to show their applicability in the domain in context. Furthermore, infrastructure support for the end-users was also explored. Two end-user interaction tools were presented that allow end-users to deploy, extend and configure smart object systems. This chapter brings the spotlight on evaluation aspects. Specifically, quantitative and qualitative evaluations of the framework will be presented through deployed smart object system setup. Also a couple of user studies that involve end-users in the deployment and extension of smart object systems will be reported to show the effectiveness of the entire end-user involvement process and the usability of the interaction tools. The chapter will end by discussing the implications that emerged from the post analysis of the study results.

5.1 Evaluation of the Framework

Evaluating a framework developed for smart object systems is always tricky. This is because the inherent quality of service requirement of a software framework for a smart object system is not identical to a typical distributed system, even though both systems are identical in nature from an architectural perspective. Chapter 3 discussed the design requirements of a smart object system framework from three perspectives: smart object, supporting infrastructure for application development and end-user aspects. Each of these requires specific quality metrics for validating proposed framework's approach to meet the design requirements. A least common denominator among these metrics is to develop real life *proof-of-concept* lightweight and effective prototypes to expose the core features of the framework and to evaluate how those features meet the design goals [Edwards et al., 2003]. In the current framework context, these core features are the framework support for developing user-centric smart objects, i.e., how efficiently smart objects and applications for them can be designed by using proposed Smart Object Wrapper, Application Development Process and FedNet respectively

and how effectively the framework exhibits user-centric qualities to elevate user experiences with smart object systems. While the prime concern here is to revisit the design requirements and evaluate how the framework in context meets those requirements, a complementary aspect is to look at some quantitative performances of the framework from system perspective. In the following sections, the assessment of the framework will be discussed from two perspectives: quantitative evaluation and qualitative evaluation. The assessment of user-centric aspect of the framework requires involvement of actual users. So, a series of user studies will be presented to show the effectiveness of the framework approach.

5.2 Quantitative Evaluation of the Framework

The proposed Framework is composed of three primary components: Smart Object Wrapper that implements the Core-Cloud model for smart objects, a task-based application development process and a document centric runtime infrastructure FedNet that creates a spontaneous association among the smart objects and applications. From quantitative evaluation point of view, the point of interests are the performance of FedNet in forming the runtime association and the overhead associated with the communication across the entire framework. It has been discussed in Chapter 4 that both the applications and smart objects are deployed in the environment as generic binary executables per se with no inherent dependency on FedNet unless the applications require smart objects services. Thus, it is imperative to look at how FedNet supports this runtime association and how the performance of the application integrating smart objects are affected by this association. To look at these aspects, a prototype home entertainment smart object system integrating a range of smart objects are developed. In the following this prototype system and its constituents are explained followed by the depiction of the runtime performance of the framework.

5.2.1 A Prototype Home Entertainment Smart Object System

In this section, first a hypothetical scenario is presented to illustrate the primary workflow and capabilities of the prototype system. Then the implementation of the system is explained.

5.2.1.1 A Scenario

Alice recently got a UPnP HDTV and a Bluetooth Headset from her parents. Today when she was telling her colleague Bob about the superb picture quality of her TV, Bob introduced her an application that she can buy from the internet to make her entertainment room exciting and smarter. It can automatically control room temperature and ambient room-light level, can pause, restart a TV program, can increase-decrease TV volume while a conversation is going on, and can redirect the audio stream of HDTV to external speakers or headsets. The

application requires some other smart objects (Chairs, Air-conditioner, Lights, Table, Window) that are expected to be available in the room. She decided to buy the application. In the application website, the roles(profile) of each smart objects are mentioned. However these roles can be played by other compatible smart objects too. For example, the application assumes that the window is augmented with a small weather box that can provide current room temperature, humidity etc, that the application uses to control the air-conditioner. This weather box can be installed under the coffee table or under any other artefact with a flat surface. Alice already has a smart couch, that can identify if someone is sitting on it and a UPnP enabled Air Conditioner. After the office, Alice went to the Tokyo Hand Creative store and bought one stand lamp that matches the required profiles. She decided not to buy any other smart objects as she thought she could use her currently available smart objects for rest of the required profiles. Instead she bought only the required service profiles for the rest of the smart objects. Each profile comes with Do-It-Yourself hardware installable into smart objects and a RFID that contains the software for that profile. Later that night, Alice came home, reorganized his new entertainment room with the lamp(ambient light profile), augmented the window with room-temperature profile and coffee table with ambient-noise profile and installed all these profiles and the application using the RFIDs and her Tangible Interaction Tool in her home's FedNet system. Alice could not wait to start her new entertainment center so she activated the system. The application perceived the room condition and accordingly set the room temperature and lamp brightness. Since Alice turned on the TV, the application fed the TV's audio stream into Alice's new shiny Bluetooth Headset as she started watching a baseball game.

5.2.1.2 Description of the Smart Object System

The above scenario is realized by a few smart objects and an application that integrates these objects. The entire system is composed of the following:

- *Smart Couch*: A regular couch is augmented with state-of-use profile. The profile comes with 6 force sensors and three photo sensors. All sensors are connected to a Phidget Interface Kit [Greenberg and Fitchett, 2001] which in turn is connected to a Gumstix¹ platform. The Gumstix runs linux and has Bluetooth and IEEE 802.11b interfaces. The Gumstix contains the core and the newly attached profile binary. The couch once augmented with state-of-use profile can identify when someone is sitting on it.
- *Smart Lamp*: A regular X10 stand-lamp that is augmented with ambient-light profile. The profile comes with 2 Phidget photo sensors, an interface kit and a Gumstix platform. The lamp with ambient-light profile can provide the brightness of the room.

¹<http://www.gumstix.com>

- *Smart Window* A regular window that is augmented with room-temperature profile that can track the room temperature by analyzing indoor and outdoor environment temperature. The profile comes with a Cooike sensor node [Hanaoka et al., 2006] that is connected to a Gumstix platform over Bluetooth.
- *Smart Coffee Table*: A regular coffee table that is augmented with ambient-noise profile that can track ambient noise of the room. The profile comes with a small microphone unit that is connected to Gumstix platform over Bluetooth.
- *UPnP TV and Air Conditioner*: Due to the unavailability of the real device, simulated UPnP TV and Air Conditioner were utilized using CyberLink² implementation of UPnP.
- *Bluetooth Headphone*: A regular bluetooth headphone with Advanced Audio Distribution Profile (A2DP)³ profile support. The headphone was also connected to a Gumstix platform.
- *JukeBox Friend Application*: This application basically implements the scenario described above. By perceiving ambient light level and room temperature via smart lamp and smart window it can proactively adjust the room ambient brightness and room temperature. If the TV is turned on, the application tries to sense the ambient noise level by contacting the coffee table and accordingly sets the volume. Furthermore, if a Bluetooth headphone is found, the application redirects the TV audio stream into Bluetooth headphone. Please note that, here the application contacts the respective profiles regardless of the smart objects that host the profiles. Furthermore, for the purpose of measurement the applications task requests were manually controlled through a secondary interface.

FedNet Infrastructure and the deployment Tool: The FedNet infrastructure runs in a laptop computer (Apple MacBook Pro, 2.4 GHz, 4 GB RAM, Mac OSX 10.4, with IEEE 802.11b interface). This machine is the host of the FedNet Core, Smart Object Repository, Application Repository and the Access Point for the JukeBox Friend application. The deployment tools are connected to this machine.

5.2.1.3 Quantitative Measurements

The whole purpose of building this smart object system is to evaluate the runtime performance of FedNet. Intentionally, heavy-weight protocols like UPnP and Bluetooth were chosen so that the approximate worst-case overhead of bootstrapping process, access point formation, communication among application components can be recorded. The application task requests were manually controlled to see the performance of FedNet. In the following the results are reported.

²<http://cgupnpjava.sourceforge.net/>

³<http://www.a2dp.info>

Bootstrapping Time: The first concern is the bootstrapping time of FedNet, i.e., Smart Object Repository, Application Repository and FedNet Core. In the current experimental setup, the Smart Object Repository needs to contact six Gumstix platforms that represent the smart objects and their profiles to set the repository with the availability of the profiles and to populate the FedNet directory. Since other components (e.g. Application Repository and FedNet Core) reside in the primary node, their initialization time does not add any overhead to the overall system. Figure 5.1(a) shows the performance of this bootstrapping process for the environment having six smart objects with gradual formation of the repository in a parallel fashion. Please note that, here the native protocols are implemented in the smart object, thus it is assumed that smart objects are already setup, i.e., Bluetooth discovery and Piconet formation are already performed for smart window and Bluetooth headphone. As shown in the Figure 5.1(a) on an average 5.32 seconds is required to create a FedNet environment with six smart objects of varying native transport protocols for profiles and bridged to FedNet via IEEE 802.11b. The variation of 180 milliseconds (approximately) were observed due to the communication latency and occasional packet losses.

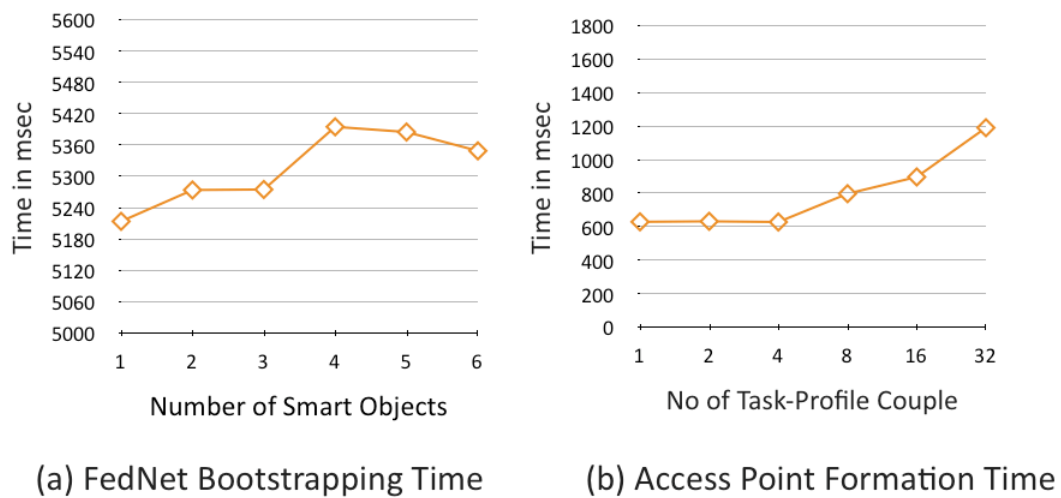


FIGURE 5.1: Bootstrapping and Access Point Formation Time

Realtime Access Point Formation Time: The next concern is the Access Point formation time by FedNet for specific application. Recall from Chapter 4 that an Access Point template is attached to an application by FedNet Core during application deployment time. This template is filled by actual smart objects (e.g., association between the application and the smart objects by mapping application tasks to smart object service profiles.) when the application starts up. Thus the Access Point formation time is directly proportional to the number of tasks that require service profiles of different smart objects. The Access Point formation process requires FedNet to contact the respective smart object by consulting its FedNet directory and application task specification and to create a link between the application tasks and smart object profiles. The following Figure 5.1 (b) shows the Access Point formation time for 32 task-profile

couple with 6 six different smart objects. For analysis purpose, half of the task were push type (actuate) and the rest half is pull type (sense). Approximately 1.2 seconds were required to setup the 32 task-profile couple, i.e., establishing six communication channels between the application and smart objects via Access Point. The initial 0.6 seconds can be considered as the setup overhead for the FedNet Core to parse the application tasks and to map the tasks to respective profiles by consulting its FedNet Directory.

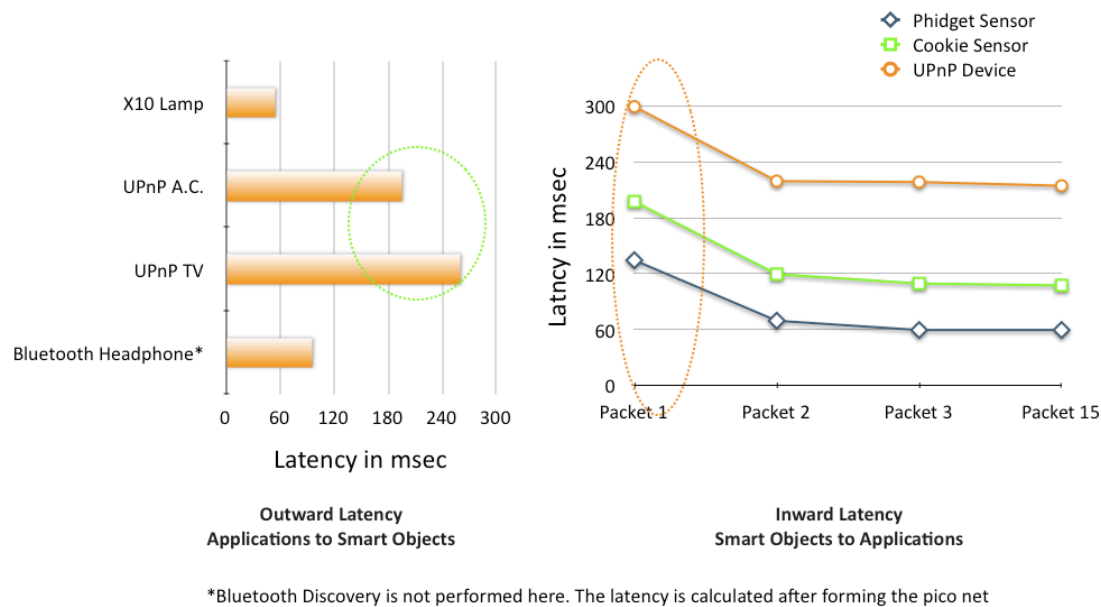


FIGURE 5.2: Communication Latency

Communication Overhead: There are basically two types of communication links in the entire framework. Outward link from application to smart objects through Access Point when application requests the service of a smart object and Inward link from smart object to application when smart object service profiles push service data to the application. The outward link is synchronous where an acknowledge is being pushed to the application from smart objects. The inward link is asynchronous where application only receives subscribed smart objects service data without providing an acknowledgement. In both type of communication, there might involve device level translation of data packets into FedNet defined data packets if the smart objects native protocol is different than the primary transport protocol used in FedNet. Figure 5.2 shows the communication latency both outward and inward for the smart objects with different transport protocol. As shown, for X10 and Bluetooth headphones outward communication latency is in the range 0-60 milliseconds. However, UPnP devices were slow to response due to the fact of larger conversation time of the UPnP packets into FedNet packets. For inward communication, there is always latency in the delivery of the first packet. This is due to the activation of the communication link that was established during Access Point formation period. However, after that the delivery latency of the packets was

reduced. In inward communication cases too, UPnP packets took longer time to be delivered than Bluetooth or Phidget Sensor based packets due to the packet translation time of UPnP.

Overall System Performance: From the overall functionality perspective, FedNet and the target system provided a stable performance. Application and all the smart object profiles were externalized through the Task Description Document and Profile Description Document. FedNet provided the runtime association by structured type matching and consequent formation of the Access Point to provide the foundation for inward and outward communications.

5.2.1.4 Summary of the Quantitative Evaluation

In the above section, the runtime performance of the FedNet in the light of a home entertainment smart object system were discussed. The performance of FedNet can be considered satisfactory taking into account that the target smart object system successfully operated without any significant delay or failure during the experimental period. However, this experiment was not meant to evaluate the functional aspects of the framework, rather to quantify the performance of movable components of the framework. From that perspective, this quantification adds little value to the evaluation of the framework. The next section will look at the qualitative aspect of the framework and will assess the functional features of the framework by revisiting the design requirements.

5.3 Qualitative Evaluation of the Framework

One effective metric of evaluating the functional aspect of a framework is to revisit the design requirements and to justify how the implementation of the framework meets those requirements. This section revisits the design requirements for building smart objects and for developing a supporting infrastructure that was introduced in chapter 3 and will validate how implementation of the proposed framework, i.e., smart object wrapper, application development process and FedNet runtime infrastructure meet those requirements.

5.3.1 Revisiting the Smart Object Design Factors

In chapter 3, five design requirements were presented for developing smart objects. A core-cloud based theoretical model was proposed that meets these requirements. In the current framework, smart object wrapper implements the core-cloud model. This section discusses how smart object wrapper meets these design requirements and allows building reusable, extensible and plug and play smart objects.

1. **Decoupling the Smart Features:** The core-cloud model essentially meets this design goal elegantly where core provides the runtime to host the clouds. In the context of the

proposed framework, smart object wrapper implements the core-cloud model, and the core contains plug-in runtime that allows any augmented features (developed following the framework's profile semantics) to be plugged in. These plug-in structure allows a smart object to contain multiple features (service profiles) as well as same profiles can be applied to multiple compatible smart objects.

2. **Service Unification (Sensing and Actuating):** The notion of service profile in the context of current framework is independent of any smart object and can implement any functionality, i.e., a profile can be push (actuating)or pull (sensing) type. A smart object wrapper handles all the profiles in an identical fashion regardless of their types. Through this profile based abstraction, service unification is achieved.
3. **Reusability:** It is the core-cloud model that provides the foundation for the reusable smart objects. Core is the minimal runtime of any smart object, and in the current framework context it is assumed that every smart object will be equipped with this core. As long as this core is present, a smart object can host any service profiles through the plug-in runtime. Similarly, profiles are built independently, thus can be applied to multiple smart objects. For example, a proximity profile capable of identifying the presence of an obstacle/entity in front of it's host, can be applied to any suitable smart object that needs this kind of functionality, e.g., a table, a door, a mirror, a drill, etc. Since there is no inherent dependency between the profile and the core of a smart object, it is very easy to reuse a smart feature to augment multiple smart objects.
4. **Plug and Play:** The implementation of the core-cloud model, i.e. the smart object wrapper is structured in a highly modular fashion and it's components are disseminated as generic binaries, i.e. the core of a smart object wrapper is a binary executable that can associate with any profiles at runtime, where each profile is also exposed as generic binary executable. These binary representations allow a smart object wrapper to be plug and play.
5. **Incremental Deployment and Extension:** A by product of the above design factors contributes to satisfy this design requirement. Due to the highly modular, binary representation and plug and play nature of smart object wrapper and profiles, it is reasonable to claim that the the proposed core-cloud model and its implementation (smart object wrapper) enable the incremental deployment and extension of smart objects. Consider a smart lamp, initially it would come only with the core of a smart object wrapper. A couple of weeks later a suitable profile can be added to this core to extend the functionalities of the lamp. A few weeks later another profile can be added or the previous profile can be updated. Since, each profile is free from any inherent dependency on the core, the smart object wrapper naturally supports incremental deployment and extension.

In the later part of this chapter, two smart object systems integrating multiple smart objects will be presented which will further demonstrate these issues and will validate the claims of meeting the above five design factors by the proposed framework.

5.3.2 Revisiting the Infrastructure Design Factors

In chapter 3, five design requirements for a supporting infrastructure were presented. Above section explained how the core-cloud model support the requirement of addressing augmentation variation of smart objects. In the following, the rest four design requirements are discussed in the light of proposed document based framework.

1. **Heterogeneity:** One fundamental requirement for a infrastructure supporting distributed systems is hiding heterogeneity and providing transparent distributed communication from applications perspective. Typically the heterogeneity issue arises considering the protocol variations of underlying components that the application uses. In the proposed document based framework, applications are forced to externalize their runtime requirements through document and to structure their application based on functional tasks. It is FedNet that takes care of protocol heterogeneity issue while providing transparent communication to the application. Consider the following code snippets (Figure 5.3) showing how an application interacts with the underlying component. As it is visible, application does not need to consider the underlying protocols of the smart object that supports this task. At the same time, the distributed communication is completely transparent to the application. From a device perspective the Profile Handler layer handles the instrumentation specific heterogeneity and provides uniform communication scheme through smart object core. This in turn is overlaid by Smart Object Repository of FedNet to make sure all communications follow a uniform protocol, i.e., HTTP/XML. Thus, the framework in context can elegantly handles the heterogeneity requirement.

```

1. Enumeration<Task> vector = this.taskList.elements(); /* Retrieve all the tasks */
2. while(vector.hasMoreElements()){
3.     Task task=(Task)vector.nextElement();
4.     if(task.getID().equalsIgnoreCase("T1") && task.getProfileFound()){
5.         /* Task is executable, profile is available */
6.         AccessPoint.sendTaskRequest(xmlProc.generateOutgoingMessage(
7.             Constant.TASKREQUEST,task.getID()));
8.     }
9. }

```

FIGURE 5.3: Sample Application Code

2. **Management of Smart Objects:** In a typical distributed system, application needs to handle the underlying component management issues (e.g., locating, accessing etc.). A

useful infrastructure usually takes these issues away from the application and provides simple mechanism to accomplish these tasks to simplify the application development process. In the current framework, FedNet takes away these tasks completely from applications and provide applications with task based abstraction only. For example, in the code snippets in Figure 5.3, we can see that the application only requests the respective tasks to Access Point that it wants to accomplish without considering how to locate the smart object that can support this task. This discovery process is completely eliminated from the application's scope. Furthermore, the interaction of application with underlying environment has a unified access mechanism (i.e., accessing the access point assigned to the application by FedNet using a simple HTTP/XML based communication scheme) regardless of the type of smart objects. This further simplifies application development.

3. **Evolution of Smart Objects Systems:** As discussed in chapter 2, a smart object system can be stand-alone, co-operative or infra-structured while majority of the smart object systems are composed of one or multiple applications that integrate one or multiple smart objects. Thus the evolution of a smart object system depends how easily and efficiently, the application and smart objects can be extended. We have already discussed how the core-cloud model enables the extensibility of smart objects by providing generic binary core that can host any number of profiles as binary plug-ins over time. From application's point of view, this evolution of smart objects has two side effects: i) an application can provide basic services in the beginning and can incrementally provide richer services as new smart objects are introduced. ii) an application can provide basic functional features in the beginning and can provide richer services over time as new profiles are plugged into the core. Furthermore, since an application is developed in isolation without any dependencies on the underlying infrastructure, the entire application and corresponding document can be replaced with an updated one to incrementally evolve a smart object system. Because of the fair loose coupling among the components of the system and core-cloud model of smart objects, the current framework supports the evolution of a smart object system very effectively.
4. **Suitable Programming Abstraction:** We discussed in chapter 3 that an effective programming model is needed for developers to build smart object systems. From application developer's point of view, this abstraction should hide heterogeneity, distributed communication, and should allow unified access mechanism. In the current framework context, developers are offered a *Task* based abstraction for application development. Applications are required to be structured based on atomic actions that require underlying components' (i.e, smart object) services. As shown in the code snippets of Figure 5.3, this task based abstraction can hide the heterogeneity and distributed communication aspects elegantly. Furthermore, by adding new tasks to an application, it is possible to leverage off new functional features from an application using existing abstraction, i.e., *Task*. From a smart object's perspective, the abstraction provided to the developers is *Profile*. Regardless of the type of services that the smart objects support (e.g.,

sensing, actuating, etc), every augmented feature is represented as a *Profile* and the corresponding programming model allows developers to implement the feature using this *Profile* based abstraction. Such abstraction enables a smart object to represent its services in a unified manner, and which in turn is exploited by *Task* based applications.

The combination of these quality features also satisfy the architectural qualities required by an infrastructure to enable end-users in the deployment and extension of smart objects, i.e., plug and play and extensible smart object systems with loose coupling among system components. Next section reports the usability studies on the entire end-user involvement process along with the interaction tools.

5.4 Evaluation of End-User Aspects through User Study

The final aspect of the evaluation is validating the claim that the proposed framework elevates the end-user experiences with smart object system by considering user-centric issues in the architectural design. Chapter 3, discussed that one way of increasing the framework quality is by considering how end-users will use the system that leverages off the framework and how that experience can be enhanced. This work argued that to enhance user-experience it is imperative to involve end-users in the deployment, extension and administration of smart object systems. This is particularly important for smart object systems as they will be a part of our environment where end-users have the ultimate control. Consequently, in the proposed framework smart object systems are structured in a highly modular fashion, where components are independent and are glued together via a runtime infrastructure FedNet. This modular design allows the framework to provide foundation for involving end-users in the management process of smart objects systems and enables design opportunities for building novel interaction tools atop the framework. Thus an integral evaluation aspect of this work is this end-user involvement and supporting tools. One way to evaluate the usability of the end-user involvement process and the interaction tools is by inviting ordinary individuals to perform the deployment and administration tasks related to smart objects systems. Consequently, a series of user studies are designed and the detail of these studies are presented in this section.

Following the guidelines of Edwards et al. [Edwards et al., 2003] for evaluating ubicomp systems, a couple of *proof-of-concept* smart objects systems that include multiple smart objects, profiles and application are re-developed following the approach of the framework in context and are provided to end-users for real time deployment, extension and administration in two separate studies. In the first study the smart object management task was assisted by the GUI interaction tool, whereas in the second study TUI interaction tool was provided to the participant for the management tasks. In this section, first the two *proof-of-concept* systems are presented followed by the report of two study sessions.

5.4.1 Two Sample Smart Object Systems

In this section, first a scenario is presented to illustrate the concepts that form the basis of the study. This will be followed by two systems that implement the scenario.

5.4.1.1 A Scenario

Alice recently moved into a new home and bought a new mirror augmented with a display for her wash room. She found and downloaded an interesting application on the internet that can show some information (e.g. weather, stock quote, movie listing etc.) in the mirror display and installed it on the mirror. While reading the application manual, she realized that the application has some advanced features that can be enabled by adding some add-ons in the mirror. For example, if the mirror is augmented with a sensor that can recognize someone's presence in front of it, the application can show the information only at that time, for the remainder of the time it will switch to the power-save mode. Similarly if the mirror is augmented with an input device, the application allows the user to interact with the application, e.g. to know more detail about weather information. Alice decided to enable all these features one by one. A week later she bought an infra red sensor and placed it in front of the mirror, now the application automatically goes to the power-save mode when no one is in front of it. After a few weeks, she bought a touch button and attached it to the mirror, so that she can interact with the application more closely. Now, her mirror application is running in full fledged mode just as she liked. A few months later she found a new application for her mirror that can monitor her toothbrushing activities and can give feedback using the mirror display. The feedback is conveyed allegorically through a virtual aquarium. She bought the application and installed it in her mirror. She also bought a compatible toothbrush that work with her fancy application. Now her toothbrushing becomes fun as well as more healthier than ever.

The above scenario is implemented using two smart object systems. In the following these two systems and their constituents are described.

5.4.1.2 Descriptions of the Smart Object Systems

The first system is composed of the following

- *Smart Mirror*: A regular display is augmented with an acrylic mirror panel (Figure 5.4(a,f)). The acrylic panel is attached in front of the display, and only bright colors from the display can penetrate the panel. The mirror display has an extension board for attaching sensors. A Smart Object Wrapper instance represents the mirror.
- *AwareMirror Application*: This application runs in a mirror and displays some up-to-date information [Fujinami et al., 2005]. The application's default functionality can be

enhanced if the mirror is augmented with *Proximity* and *Bi-state Interaction* profile. The former enables the application to show information only when someone is in front of it and the latter enables the users to interact with it, e.g., to know detail information. This application adheres FedNet semantics, e.g., expresses tasks in a description file and access the smart objects in a RESTful manner. (Figure 5.4(g)).

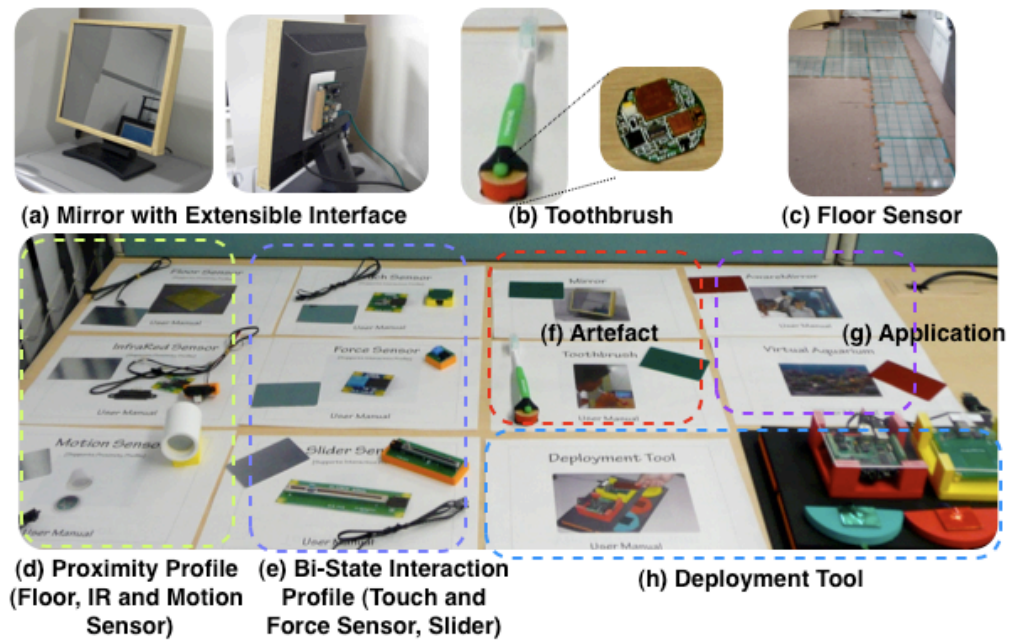


FIGURE 5.4: Smart Objects, Profiles and Applications, Manuals and Interaction Tools

The second system is composed of the following:

- *Smart Mirror*: Same as the above system.
- *Smart Toothbrush*: A toothbrush (Figure 5.4(b,f)) is augmented with a wireless 3D accelerometer sensor [Hanaoka et al., 2006]. It can provide its state of use information and is represented by an instance of the Smart Object Wrapper with a *state-of-use* profile. The sole purpose of the *state-of-use* profile is to provide the usage state, e.g., toothbrush is in use, etc. profile plugged into it.
- *Virtual Aquarium Application*: This application has the objective of improving users' dental hygiene by promoting correct toothbrushing practices [Nakajima et al., 2008]. The application turns the mirror artefact into a simulated aquarium. Fish living in the aquarium are affected by the users' toothbrushing activity if a toothbrush is available. The application's default functionality can be enhanced if the mirror is augmented with *Proximity* profile that enables the application to show the aquarium only when someone is in front of it. This application adheres FedNet semantics. (Figure 5.4(g)).

Profiles: Both systems' functionality can be enhanced by adding one or multiple profiles into the mirror artefact. These are:

- **Proximity Profile:** This profile's purpose is to recognize the presence of an entity in front of its host smart object. This functionality can be achieved in multiple ways, i.e., using an infra-red sensor, a motion sensor, a camera, etc. Three implementations for this profile (Figure 5.4(d)) were provided using infra-red sensor, floor sensor (Figure 5.4(c)) and motion sensor respectively.
- **Bi-State Interaction Profile:** This profile enables a user to interact with its host smart object. It provides a simple two-state input facility suitable for applications that needs binary input. There are multiple instrument choices for the profile implementation and three implementations (Figure 5.4(e)) were provided for the study: one with a touch sensor, one with a force sensor and the last one with a slider.

FedNet Infrastructure and the deployment Tool: The FedNet infrastructure runs in a laptop computer. The web based tool runs on this machine and the tangible deployment tool is connected to this machine.

5.4.2 Study Methodology

Each of these applications, smart objects and profiles is developed following proposed framework semantics. Also, the same profile is built with multiple sensors. The successful deployment and incremental integration of these components by the end-users will highlight the core features of the system and will reveal the usability of the overall process. To support end-users in these tasks, two interaction tools are developed, one with Graphical User Interface (GUI) and the other with Tangible User Interface (TUI) as illustrated in chapter 4. Thus, involving end-users in a user trial will also reveal the usability of these two interaction tools and will put forth future design issues. Accordingly, two user studies are conducted to understand the feasibility of the proposed approach. Both the study were identical from the methodology point of view except the fact that in the first study the end-users were assisted with the GUI interaction tool where as in the second study users were assisted with the TUI interaction tool. It was decided to go for two completely different study sessions for the following reasons:

1. To ensure that the participants do not loose focus or motivation considering the tasks involved in the study sessions might take long time.
2. To ensure that there is no learnability effect on the outcome of the study considering the consecutive repetitions of the same tasks by two different tools might result in outliers.

5.4.2.1 Participants

50 individuals (28 Male, 22 Female, age range 19-42) with moderate computing skills (familiar with web, email, and basic office applications) were invited through an open invitation in a social networking site. 25 (12 Male, 13 Female, age range 19-42) of them participated in the first study and rest 25 (14 Male, 11 Female, age range 22-39) of them participated in the second study. Majority of the participants (94%) did not have any engineering background. Users were screened such that their professions were fairly disperse (e.g., law students, house wives, office workers, etc.) to balance the skill level. Furthermore, it was ensured that the participants in both the studies have similar background.

5.4.2.2 Study Sessions

Each study session was held for 90 minutes for each participant and included four phases. In phase one, the overall concept of smart object system and a tutorial on the respective deployment tool were presented. In phase two, the subject was given 10 minutes to get familiar with the tool. Next, in phase three, the participant was given the following four tasks to complete using the deployment tools and the respective smart objects, profiles and applications.

- **Task 1:** Deploying a smart mirror, i.e. addition of a smart object in the environment.
- **Task 2:** Installing either AwareMirror or Virtual Aquarium application on the smart mirror and running the application.
- **Task 3:** Closing the application and the smart mirror. Then adding the Proximity Profile⁴ into the mirror by selecting one of the three implementations if the user selected AwareMirror application or adding the smart object toothbrush if the user selected Virtual Aquarium application. This task ends with running the smart object and the application again.
- **Task 4:** Closing the application and the smart mirror. Then adding the bi-state interaction profile into the mirror by selecting one of the three implementations if the user selected AwareMirror application or adding the Proximity Profile into the mirror by selecting one of the three implementations if the user selected Virtual Aquarium application. This task also ends with running the smart object and the application again.

To complete these task end-users utilized the respective GUI interaction tool and TUI interaction tool and followed the interaction mechanism as discussed in chapter 4 (section 4.4).

Finally in phase four, participant was requested to attend a questionnaire followed by an in-depth interview session. The questionnaire contained 14 statements structured with a 5-item

⁴ The profile hardware installation requires attaching the sensor to the mirror using double sided adhesive tape and connecting the sensor cable to the interface board located in the backside of the mirror. For the floor sensor, hardware installation was not needed except for placing the floor mat.

Likert scale to indicate their level of agreement or disagreement. Question 1-10 were designed following the System Usability Scale (SUS) [Brooke, 1996] and the remaining four questions regarding the complexities of each tasks ⁵. Following the questionnaire, participants were interviewed to gain further insight into their assessments. Each session was video taped for later analysis. Figure 5.5 shows some snapshots from the experiment sessions.



FIGURE 5.5: Participants consulting manuals, deploying smart objects, installing applications, adding profiles, etc.

5.4.3 Study Results

For the clarity of the discussion and comparison of the study results, in the following the combined results of the study are presented.

5.4.3.1 System Performance

FedNet and the target systems provided a stable performance in all the sessions and the end-users' activities were properly converted into system events accordingly. The flawless deployment and the successful utilization of the two smart objects systems can be considered as an qualitative evaluation of the system aspects of the framework in context. In the following two specific system related concerns are highlighted.

1. *Plug and Play Extensible Smart Object*: The smart mirror was deployed by the participants and its functionality was extended by attaching a couple of profiles (with multiple

⁵Likert scale was normalized to complexity levels.

sensor choices) to enhance applications features. Regardless of the sensor type, profiles were seamlessly added into the smart object wrapper. Furthermore, the order of profile addition had no effect on the deployment process. So participants picked whatever profile they felt like adding. This highlights the capacity of the smart object wrapper for hosting multiple profiles implementing different device interfaces. The combination of these are important for enabling the *Do-It-Yourself (DIY)* support for the end-users.

2. *Infrastructure Independency and Spontaneous Federation*: Both the smart objects and applications were expressed in high level descriptive documents and disseminated as executable binaries independent of the FedNet infrastructure. This allowed the end-users to install them easily. FedNet provided the runtime association enabling applications to use the artefacts and to switch to respective advanced modes when new profiles were added. For the end-users, these mechanisms were completely transparent as they could only see the effect of their actions. This highlights the simplicity and power of the proposed approach to involve end-users in the deployment process.

5.4.3.2 End-Users' Performance

For the clarity of the discussions, the variety of actions that are performed by the end-users during the task session (phase three) as mentioned in section 5.4.2.2 are structured into 4 categories. These are:

- Smart object addition.
- Application installation.
- Profile addition, thereby extending a smart object's features and the applications running on it (when applicable).
- Smart object systems administration, i.e., associating an application with a smart object, controlling (start/stop) an application and respective smart objects, etc.

In the following the end-user performance in study sessions are discussed.

Study Session with GUI Interaction Tool: There were 100 tasks in total, four for each participant. Each of the participants was successful in finishing the assigned tasks. However, 11 participants needed active support in early stages and 4 of them needed support for all the tasks. On an average 34 minutes were required for the third phase (accomplishing the four tasks). Figure 5.6 shows the time (Figure 5.6 (a)) required for each task and the corresponding complexity (Figure 5.6 (b)) associated with the task. Participants required least time of 2-4 minutes in finishing task type 1, where as they required fairly large time for other tasks as shown in Figure 5.6 (a). They struggled in general in manipulating the GUI. The generic binaries for the smart object, applications and profiles were provided in flash memory stick which

further confused them. For example, while installing application, they thought pressing install button will install the application without realizing the fact that they need to link the binary stored in the supplied memory stick in the installation window. Similar mistakes were seen by several participants in the first few tasks. However, in the later stages they were able to perform the tasks without any mistakes. The profile addition tasks took longest time due to the installation of the hardware. The difficulties of using the GUI interaction tool to accomplish the assigned tasks resulted in quite drastic user response in terms of the complexities. As shown in Figure 5.6 (b) all the participants comprehended that the overall tasks are complicated. However, later interview revealed that, their impression is basically on the interaction tool, not on the entire end-user involvement process. This was further confirmed in their subjective responses (presented later). All participants have shown progress in repeating tasks and on an average they required 37% less time in redundant activities, e.g. when adding the second profile plugin, attaching hardware, or restarting an application, etc.

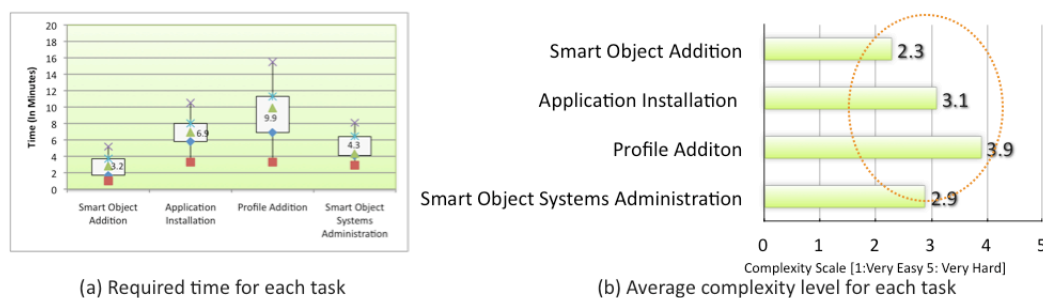


FIGURE 5.6: Average time taken and average complexities for completing experiment tasks by end-users using Graphical User Interface Tool

Study Session with TUI Interaction Tool: There were 100 tasks in total, four for each participant. All participants successfully finished the assigned tasks, though 6 participants needed active support in the early stages, primarily because of the misconception of the deployment process (explained later). On an average 16 minutes were required for the third phase (accomplishing four tasks cumulatively including the intervals between the tasks). Figure 5.7 shows the time (Figure 5.7 (a)) required for each task type and the corresponding complexity (Figure 5.7 (b)). Task type 1, 2, and 4 required fairly little time (1-4 minutes) since they consist of 2 step interaction. However, smart object addition time was slightly higher because of the task order. Since, every participants' first task was to add a smart object, it required a slightly more time. The end-users also found these tasks easy with an average complexity of 1.2 out of 5.0 for the task type of 1, 2 and 4. The profile addition task took maximum time (4-9 Minutes) where a large portion was spent for the hardware installation. Moreover, 8 participants made mistake in the association step e.g., placing the smart object card later than the profile card onto the card reader or not placing the smart object card at all. However, the deployment tool rejected these interactions and suggested the correct steps. This allowed the end-users to accomplish the task without secondary assistance. These factors also impacted the profile addition task's complexity (average: 3) as shown in Figure 5.7 (b). All participants have shown

progress in repeating tasks and on an average they required 28.3% less time in redundant activities, e.g., when adding the second profile, attaching hardware, etc. This indicates the fast learnability of the deployment process.

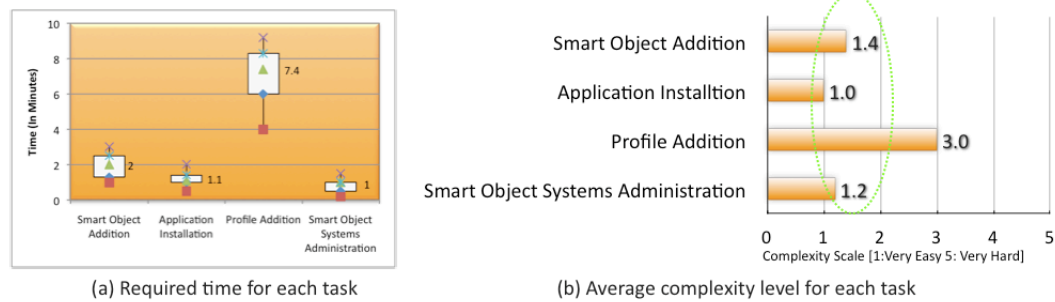


FIGURE 5.7: Average time taken and average complexities for completing experiment tasks by end-users using Tangible User Interface Tool

Combined Subjective Results of End-Users Feedback: The composite SUS score was 76.3 out of 100 (Standard Deviation: 13.2, Max: 91.2, Min: 59.3) regarding the overall usability of the deployment tool and the process. These values can be considered as quite promising. Moreover, the individual frequency of the acceptance statement in SUS: *"I would like to have this system if it were available"* (Strongly Agree: N=37, Somewhat Agree: N=8) suggests users positive response regarding the acceptance of the overall approach (Figure 5.8).

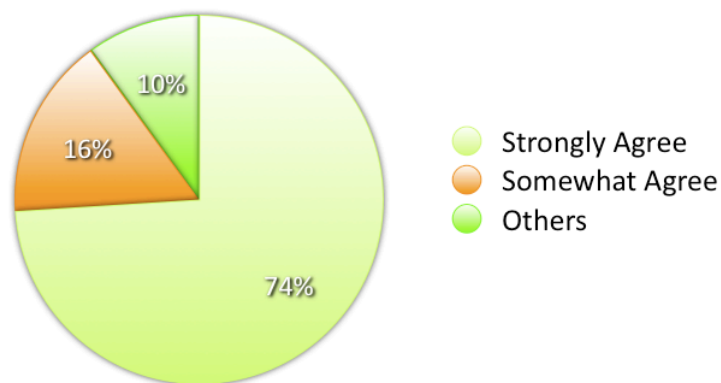


FIGURE 5.8: Subjective responses of the end-users on the overall process

5.4.4 Implications of the User Study

Later interviews with the participants revealed several interesting aspects regarding their understanding and qualitative assessments of the entire process. In the following these aspects are discussed.

1. **Concept is difficult to comprehend:** The notion of profile and application were difficult for the end-users to comprehend and differentiate. For them the smart object profile and the application were the same. One participant commented *"I did not get this profile thingy, is not it an application for the artefact? It's a bit confusing, what is the difference between profile and application?"* Another participant remarked *"I understand that profiles are artefact features, but since the installation process is same, it is hard to differentiate the role of the artefacts, profiles, and applications. May be the profile addition button should be at the other end so that we know it has a different purpose."* They also had difficulties in understanding what a profile is, as they associated the term *profile* with someone's background or record. So, they could not correlate how a physical object could have multiple profiles, which also affected the performance of adding profiles as shown in Figure 5.6 and in Figure 5.7. These facts imply that, the current notions are not self explanatory to end-users and it is needed to provide a more comprehensive way of expressing these concepts. Also, as one of the above quotes pointed out, the orientation and placement of the interaction button for extending smart objects feature should be different than installation buttons.
2. **Installation process was difficult:** Although, all the participants were familiar with the Internet, they found it difficult to use the GUI web interface tool to install the smart object, application and profiles. Later interviews revealed that, it was primarily because of the user interface since they need to switch their attention from physical object to computer frequently to see their actions output. With the TUI interface, participants had better experiences since their interactions were more coherent. The end-users mainly struggled in installing profiles. The end-users suggested that the process of profile deployment has to be plug and play, when attaching the profile hardware the corresponding software should be installed with minimal intervention.
3. **Different packaging is preferred over DIY:** Several participants mentioned that they can buy a product with different functional granularity according to their preferences. one articulated participant pointed out *"Why don't you make different versions of this mirror with installed sensors and applications, then I can just go and buy whatever I need rather than trying to figure out where to put which sensor."* She goes on saying *"I would like to have this kind of cool stuff in my home but perhaps I would ask my boyfriend to install it for me."* 12 participants had similar view points. They concurred that the smart object should be pre packaged, for example: one mirror could be packaged with a proximity profile and another with both profiles, etc. In this case they have the flexibility to buy different packages. Although, they agreed that the DIY approach is fun, interesting and inexpensive, but it limits the acceptability of the product to a mass population. One housewife remarked *"My husband would love to play with your mirror, but I don't know how often I will do this."* These views suggest that, to make smart objects available to a larger user base, packaging with variant options are needed. The incremental DIY approach can further extrapolate the packaging scheme.

4. **Familiarity with sensors is crucial:** Multiple sensor implementations for the same profile were provided in the study. It was found that different participants have picked different sensors. For proximity profile infra-red sensor was picked 32 times, the motion sensor 15 times and the floor sensor 3 times. For the bi-state interaction profile, the touch sensor was picked 41 times, and the slider 5 times and the force sensor 4 times. The end-users pointed out that their familiarities with the infra-red, motion and touch sensors in everyday life (e.g., in washroom faucet, magnetic door, garage, touch screen etc.) influenced their selection, since it was easy for them to understand how the sensors work and how to interact with them. Also domestic concerns were highlighted by a few participants. Most of them rejected the floor sensor since they found it big, and problematic while cleaning the floor. Similarly, they mentioned that the slider's sharp edge might harm their kids. Moreover, since they knew what the sensor does, they revealed that it would be very simple for them to deactivate it. The latter findings actually confirms what Beckmann et al. concurred about domestic concern and greater feelings of control [Beckmann et al., 2004]. These factors imply that the smart objects and the profiles should use the sensors that are common in our everyday life to involve end-users in the deployment process.
5. **Instantaneous feedback is necessary:** It is essential that when a user perform some actions, the system provides instantaneous feedback with appropriate level of information to confirm their actions. For example, when a profile is added to a smart object, the new functionalities should be reflected in the smart object instantly and erroneous installation should be reported immediately to confirm the users' actions. In the current approach, the only way to realize a profile's functionality is by running the application. However, the participants were curious in knowing whether their action was successful instantly after the installation. This missing feature caused frustration among the participants and was reported during the interview. This suggests that smart object systems needs to have an instant feedback facility for confirming users' actions.
6. **Tangible interface has potential:** Figure 5.6 and 5.7. clearly show that the GUI interaction tool failed to attract users. The variation between the overall time taken by the end-users using GUI and TUI tools also highlights this fact, as end-users required 47% less time in completing tasks with TUI tool. End-users could not relate such GUI based deployment process with installing other home appliances. However, most of the users found the tangible tool familiar and user friendly. One user commented *"I like this tool because it gives me the feeling of installing a physical thing, this touching, pressing are something I am familiar with, for example using my TV or washing machine, its simple and more user friendly."* Similar comments were received from other participants too where they emphasized that tangible interaction for household appliances is more familiar and suitable to them. Considering their projections, it can be envisaged that tangible interface might be a potential candidate for deploying ubicomp technologies.

7. **Intuitive hardware interface is needed:** Considering, the participants performances and subjective feedback, it can be concurred that the intuitiveness of the profile hardwares are essential for the success of the DIY approach. For example, in the experiment, except for the floor sensor, all the sensors had one cable that could be attached to the mirror. However, there were two ports in the mirror for the cables and each port was specific to a profile. 13 of the participants made mistakes in picking the right port. Although it was clearly written in the manual, they did not consult it and tried to do it intuitively. Of course, If the smart object is designed without further augmentation such *port or other hardware interfaces* need not be intuitive, however for a DIY approach it is necessary that the hardware installation process is self explanatory. Furthermore, it was noticed that the participants were quite serious about the aesthetics of the mirror while attaching the sensors. Later interviews revealed that it is important for them to make sure that the overall appearance of the physical objects matched their style. These facts suggest that the manual had a minimum role in the DIY approach and that the instrumentation has to be intuitive to the end-users.
8. **Balance with current practice is required:** Participants noted that when they buy furniture or home appliances, they do not need any software installation. Usually they just plug it in and it works. However, in this approach software installation is required. Although, the process is identical to regular desktop computing, it has no similarity with home appliances. Thus the participants found it conceptually hard to think of the mirror as a piece of furniture instead of a computer. This was further extrapolated by the fact that a tablet PC was attached to the mirror which made them perceive the mirror as a regular computer display. They suggested that the software installation process should be absent, and that the hardware installation should be the only task since it needs manual intervention.

5.5 Chapter Summary

In this chapter, the evaluation of the proposed framework was presented. The framework was evaluated from three different perspectives: quantitative evaluation through the analysis of the runtime performance of the framework, qualitative evaluation by revisiting the design requirements and finally the evaluation of the end-user centric qualities of the framework through two user studies. Three different smart object systems integrating a range of smart objects were also presented in the context of these evaluation aspects. A series of user studies were conducted to understand the effectiveness of involving end-users in the smart object management tasks. These studies also exposed the pros and cons of the interaction tools built atop the proposed framework. The result of these studies showed that end-users are quite interested in deploying smart object systems just like other home appliances based on their personal likings and choices as long as appropriate tools and supporting infrastructures are provided.

Chapter 6

Discussion

Chapter 6

Discussion

In the last chapter, the evaluation of the framework was presented from quantitative and qualitative aspects. After discussing the runtime performance of FedNet, the framework was qualitatively assessed by revisiting the design requirements. Also, a couple of usability studies were reported to highlight the effectiveness of the proposed framework in supporting user centric qualities. This chapter discusses a few generic issues that are raised by this work. The chapter will start off by discussing the applicability of the proposed framework in other related domains. This will follow by the discussion on several design aspects of building system infrastructure for ubiquitous environments. The chapter will close by shining the spotlight on a more abstract and philosophical issue of finding appropriate balance between the proactive and reactive approaches towards smart object systems.

6.1 Cross Domain Applications

The proposed document based approach enables applications' interactions with underlying smart objects in an indirect fashion. Unlike typical distributed computing environment where an application is offered unified programming interfaces to exploit distributed components, the proposed approach requires an application to externalize its functional task lists in a structured document. This document is used by the secondary infrastructure to provide application with an access point to interact with underlying smart objects following a unified access mechanism. This indirection allows application developers to focus on the application logic only, whereas the infrastructure takes care of protocol heterogeneity and distribution issues. From a communication perspective, this approach represents every smart objects with a logical HTTP extension, as application can access them using simple HTTP/XML based protocol. This indirection and simple communication scheme enable the document based approach to be applied in other domains. In the following, a few of these domains are discussed.

6.1.1 Distributed Component Systems

As we have discussed earlier in this thesis, a smart object system is identical to a distributed component system and both systems share similar philosophies regarding component encapsulation and reuse. Thus, the proposed document based approach can be easily applied to distributed component systems, i.e., both applications' requirements and components' services can be externalized through structured documents that can be used by the infrastructure to provide runtime association between them.

6.1.2 Peer to Peer Computing

In a Peer to Peer (P2P) network, peers are the computer systems that are connected to each other via the Internet. Files and other computational resources can be shared between systems over the network without the need of a central server. In such a computing environment, the proposed document based approach can be easily applied for different scenarios. For example, peers may externalize meta-data of the files and resources to be shared in a document. Similarly other peers may externalize meta-data of required files and resources to be acquired in a document, and a secondary co-ordinator computer (another peer) can provide the association between these peers. A slight variation of this scenario is actually realized by the BitTorrent¹ file sharing protocol.

6.1.3 Service Oriented Computing

In a service oriented computing environment, systems group functionality around business processes and package these as interoperable services. In this environment, proposed document based approach can fit elegantly. Service specifications can be externalized in documents, similarly business applications' requirements can be exposed as function task list in a document. Since the entire platform is network centric, a secondary infrastructure can provide the runtime association between the applications and services. This approach will simplify the business application development process significantly as developers do not need to consider service interfaces.

From an abstract perspective, the proposed document based approach can be applied to any network systems that are structured around a group of distributed devices/components/services. Of course, each domain will have its own specific assumptions and constraints, but hypothetically this document based approach is generic enough to cover a wide range of application domains.

¹<http://www.bittorrent.com/>

6.2 Further Look at Design Aspects

In chapter 3, we have discussed several design issues that were considered in the proposed framework. This section further exposes these design issues and discusses some generic development aspects.

6.2.1 High Level Abstractions

In a distributed computing environment where applications are composed of several heterogeneous components, abstractions play a vital role in easing the development effort. One of the major challenges in designing middleware infrastructure for pervasive systems is to decide which properties a middleware should hide from application's scope. Suitable abstractions hide the differences among implementations of heterogeneous components, provide transparent distributed communication, and allow portability of the application. However, it is very difficult to provide abstractions that can meet these requirements completely. Each abstraction may have different assumptions and constraints in hiding these properties. It is not easy to provide high level abstractions in a middleware infrastructure generically that can be shared across a range of systems having different characteristics. Thus, we need to carefully consider what abstractions are suitable while designing a system infrastructure. Previous works on middleware infrastructure for supporting synchronous collaboration in an office environment [Tandler, 2003] and middleware infrastructure for building location-aware applications [Harter et al., 1999] suggested that it is useful to design middleware infrastructure and corresponding high level abstractions for supporting a specialised application domain. The infrastructure and the abstractions may be generic enough to support a wide range of applications. In the context of smart object systems, it is expected to have suitable abstractions that enable developers:

- To represent smart object services in a unified manner by hiding implementation heterogeneity.
- To support applications with unified access to smart objects by providing transparent distributed communication.
- To support extensibility of a smart object system, e.g., spontaneous update of an application or a smart object.
- To leverage new services using existing abstractions.

There are primarily two abstractions underneath the documents that have been applied in the current framework. From smart objects' perspective it is the notion of profile that handles the service implementation detail and protocol issues. Since profiles are independently built following a plug-in architecture, a smart object service can be extended anytime by adding

new sensors or actuators and attaching corresponding profile into the smart object's core. Also, if a specific service needs to be updated only the corresponding profile needs to be replaced, not the entire smart object or the applications utilizing them.

The second abstraction is from applications' perspective, i.e., tasks that simply externalize an applications' requirements, so any application can be expressed with this abstraction. This simple abstraction is powerful enough to hide the heterogeneity and distribution issues from the application completely, as we have seen in the illustration of application development in earlier chapters. This abstraction also enables an application to grow incrementally. Not necessarily all tasks of an application can be supported by an existing environments, however with the incremental addition of new smart objects in the environment or porting the application to another environment with richer smart objects might enable the full functional features of an application. In addition an applications functionalities can be updated independently (application binary and the document) without concerning the impact of such update in the middleware or smart objects. In current approach, such flexibilities are provided elegantly by only expressing applications' task specifications in documents and ignoring smart object management issues at the application level.

6.2.2 Separation of Concerns

One of the primary requirements for building distributed middleware is to provide sufficient separation of concerns among the system components. This ensures loose coupling and modularity aspect of the systems. In the context of current framework this separation is achieved by completely isolating application development from the infrastructure and underlying smart objects. This disassociation of applications from the smart objects they reference is identical to the Model-View-Controller (MVC) architecture from Smalltalk [Krasner and Pope, 1998]. In the MVC architecture, data (the model) is separated from the presentation of the data (the view) and events that manipulate the data (the controller). Similarly, documents in the current framework act as the glue that associates smart objects services to applications that manipulate the services. Such separation of concerns (i.e., both the applications and smart objects are independent of FedNet and come as ready-to-run binaries), and data centric approach enable the framework to provide additional services orthogonally. For example, the end-user interaction tools atop FedNet that enable end-users to deploy, configure and manage the applications and smart objects running in the FedNet environment.

6.2.3 Interface and Protocol

Majority of the existing middleware systems provide APIs to application developers for interacting with the underlying environment. This approach is quite suitable in a limited scale but it hinders the portability and reusability of the applications. Also, managing the configuration and access of the distributed components at the application level makes the development

process complicated and lengthy. On the other hand, building systems by adopting standard protocols makes application development faster as access mechanism is unified, and heterogeneity of interfaces is removed. Furthermore, adopting standard protocol makes it easy to integrate commercial products. Smart objects, devices and appliances may change their functionalities and interfaces over time. Thus assuming a consistent interface is not appropriate for application design. This work argues that it is not necessary to fix the interface between the appliances but to determine the data format for communication. Accordingly, in the proposed document based framework, rather than interface standardization, a common communication protocol (HTTP/XML) is chosen along with the document based association mechanism. Documents describing smart objects input/output specifications are delivered to the application ahead of the actual communication by the infrastructure thus letting application to prepare for the movable data. Furthermore, a smart object may provide services in various granularities thus supporting multiple applications requiring services at different scale (i.e., some applications may ignore some service features by ignoring part of the document).

6.2.4 Simplicity and Features

In the earlier prototypes of this work, several secondary features, e.g., security, personalization, etc. were supported. However, through the development of a series of applications. it was realized that these features add little values as most of the applications have their specific needs and defining these features generically at a global scope is very difficult. In fact, for a smart object middleware the primary features i.e., abstracting physical objects, and providing lucid access mechanism to aggregate events in a simplest way by hiding complexities (discovery, marshaling messages, etc.) are the keys for the developers' satisfaction. For example, the current framework hides the discovery process completely from the developers as it utilizes documents to create the runtime association. This transparency is found to be more important to the developers than providing a versatile separate discovery service at an infrastructure. These issues highlight one significant aspect that secondary features have no value unless the primary features of a middleware are complete and adequate.

6.2.5 Some Notes on Evaluation

Ubicomp research is experimental in nature and applications are the whole point of ubiquitous computing [Weiser, 1993]. This makes it difficult to evaluate a middleware of ubicomp systems. Primarily because the performance metrics typically used to benchmark a distributed middleware are not compelling to measure the quality of a ubicomp middleware. For example, efficiency of a smart object middleware is not constrained by faster throughput or minimum latency, in stead support for proper context identification and triggering of proactive service in a timely fashion are more important metrics for defining efficiency. Another example is the system robustness; a smart object system is often physically distributed and provides proactive services contextually. This characteristics suggests that users attentions'

on smart object systems are not coherent. Thus if a particular node fails and restarts silently, it is very likely that users will be unaware of that fact. Of course, in situations where users are actively interacting with the system, or if the level of error is very critical, e.g. entire hardware damage, etc., failures will be visible. However considering the physical nature of these systems, most of the time robustness is hidden from end-users. Generally speaking, a smart object middleware have very little commonalities with traditional distributed system middleware from benchmarking perspectives.

6.3 Further Look at End-User Aspects

In the last chapter the quantitative and qualitative results of the user study designed to evaluate the end-user involvement were presented. These results have two implications. First from a framework perspective, the successful completion of the study tasks validates the framework's support for involving end-users in the deployment, extension and management of smart object system. This validation in turn exposes the architectural qualities that needs to be considered while designing user-centric smart object systems. For example, plug and play and extensibility are the inherent requirements for end-user involvement. For supporting the development of plug and play and extensible distributed systems (e.g., smart object systems), it is imperative to provide loose coupling among the system components, and to provide infrastructure support in an indirect fashion. In the proposed framework this indirection was achieved through the document based association between smart objects and applications. Second implication is more from an end-user perspective. Specifically, the study shows that it is viable to involve end-users in the administration of smart object systems as it provides end-users with higher control, more confidence and better personalization support. The interesting point to highlight here is the fact that even though the participant had no knowledge of smart object system, they were successful in performing the requested tasks. This can be directly compared with end-user involvement in personal computers and other information appliances. As long as the technological requirements for bringing end-users in these activities can be satisfied in a persuasive way by exposing the benefits, it can be concurred that end-users will accept the technology. For example, event though the overall study results are not compelling considering the fairly high average time taken by end-users (25 minutes in an average in both studies) to complete the tasks, the encouraging fact here is their subjective responses that show 74% (Figure 5.8 in chapter 5) of the participants strongly agreed in involving themselves in the deployment, extension and management tasks as long as suitable interaction tools and supporting system infrastructures are provided.

This work is one of earliest efforts in the literature that looked at these aspects of smart object systems from an end-user perspective. There are still a lot of issues to be investigated to further mature this process. One of the immediate avenue of future work is so consider the interaction tool. From the study we have seen that approximately 47% better response time was achieved by replacing the graphical user interface with tangible user interface. However,

this tangible tool is not completely adequate as it was found during the course of the work. For example, it is not possible to deploy spatially distributed smart objects with this tangible tool since it is centralized. At the same, the orientations and positions of the tool components are not optimized as highlighted by the end-users. Thus it is imperative to state here that further case studies are needed to formalize a design guideline for building this kind of interaction tools.

Another interesting aspect is the feedback provision. How to provide appropriate feedback in response to end-users actions. There are multiple choices: i) using an embedded display in the tool, ii) using a secondary display, iii) using multi modal feedback. e.g., speech, etc., iv) embedding the feedback in the smart object itself. The first three options are straight forward, since the feedback is not perceivable physically, whereas the last option is more channelling as we need to consider how can we map such feedback features in smart objects interaction primitives.

From the above discussions it can be concurred that there are various scopes for further investigations on end-user aspects both from system architecture and interaction perspectives.

6.4 Reactive or Proactive

One critical design decision taken in the current framework is to involve end-users in the smart object system. This involvement is not only for the purpose of deployment but also for management and administration. In fact, every deployed application and smart object is required to be launched through explicit interaction in the current framework context.

Numerous ubiquitous computing architectures and meta operating systems, employ a combination of sensor fusion and artificial intelligence techniques to enable an always-on smart environment that can deduce user's need from contextual cues to provide them with just-in-time smart services in a proactive fashion. This is indeed an arguable way of perceiving the requirements of a ubiquitous environment. Do we really need a "smart space" to be smart when there is no one in that space or when the user does not need to use the smart services? This always-on approach suffers from the following problems:

- A centralized aggregator component is typically used for deducing user need from contextual cues applying sensor fusion and artificial intelligence techniques to create a proactive smart spaces. This approach tries to solve very complex problem of artificial intelligence and human psychology to reason users' intention and lacks in reliability and accuracy. Although in limited scale and scenario dependent environment, this approach performs well, it suffers in adapting dynamic environment as the context reasoning logic is predefined.
- Always-on systems consume resources and introduce overhead because of being active continuously.

- This approach typically binds underlying resources with an application at instantiation time limiting other applications to use those resources even when the first application is not in use. In addition, this approach offers limited opportunities to dynamically replace the underlying smart objects as they are tightly coupled with the applications.
- From human factor point of view a completely proactive system often conflicts with humans' mental model. For example, a proactive action from the system might not be always accepted by the users for a variety of reasons, e.g., emotion, social context, etc.

These problems require rethinking the approach of forming a pervasive environment. Reliability and accuracy issues can be resolved if we construct reactive smart spaces and employ end-users to initiate the smart services, i.e. converting a regular environment into a smart one in an on-demand fashion as a reaction to users' actions. In the simplest case this could be done by providing end-users with a simple user interface to indicate their intentions of using pervasive services. An underlying infrastructure can take care of the rest from this point to form a smart space. The argument here is that, if we involve user to initiate proactive services in this fashion, we can achieve better performance with high reliability and precision. This approach also leads to better data and resource management, less overhead, and less administration. Rogers [Rogers, 2006] and Bell [Bell and Dourish, 2007] concurred similar views on human-centric computing.

In the current framework, this reactive approach is chosen and end-users are involved in initiating a smart object system. Once launched, the system can perceive the situational and operational context to change its behaviour or to provide proactive services. From this perspective the current framework approach is to provide "passive smartness". This is because the environment is capable of being smart always but it only acts smart when needed and invoked by users. Thus applications and smart objects are not tied to each other, rather the best composition is made when an application is initiated by a user. This approach is flexible and allows an environment to react differently for different users sharing the space. The current framework follows this principle and utilizes FedNet to form a spontaneous federation among applications and smart objects.

6.5 Chapter Summary

This chapter discussed some generic issues in the light of the current framework. First the applicability of the proposed framework in cross domain applications were discussed. Then, the chapter looked at some design issues of building middleware systems for ubiquitous environment. After that, the discussion was switched to end-user aspects. The chapter was concluded by looking at the design tradeoff between reactive and proactive approaches towards smart object systems.

Chapter 7

Conclusions

Chapter 7

Conclusions

This thesis presents a document centric framework that supports the development of user-centric, reusable, plug and play, and extensible smart object systems. This last chapter briefly summarizes the work presented in this thesis, and puts forth a few research issues for future exploration.

7.1 Research Summary

Chapter 1 introduced the research presented in this thesis and described the motivation behind this work by drawing evidences from the evolution of computing. Then it put forth the primary research question of this work by illustrating the problem space and design challenges. Chapter 2 provided a detail background of smart object systems, their classification and characteristics by discussing a range of existing works on smart objects systems. After carefully observing the characteristics of smart objects and contemporary notions, a definition of smart object was given with three cardinals: perceptual augmentation, device centric situational awareness and supplementary services. A few key properties of smart objects were highlighted including unique ID, self-awareness, sociality, autonomy and stateful-ness. A classification of smart object systems composed of stand-alone, co-operative and infra-structured smart objects was introduced with appropriate illustrations.

With a formidable understanding of the nature of smart object systems, the thesis moved to chapter 3 and discussed the design aspects from three perspectives: a common theoretical model for smart object, an infrastructure support for building smart object systems and user-centric design issues. Considering the design requirements of smart object, a core-cloud model was proposed that consist of:

- A Core that combines the functionalities that are common across a variety of smart objects.

- Clouds are composed of augmented features of smart objects that can be plugged into the core.

This model satisfies the basic design requirements, i.e., decoupling smart features, building plug and play, reusable and incrementally extensible smart objects that can do both sensing and actuating.

After that, the thesis put the spotlight on the framework aspects for building smart object systems and discussed five basic design concerns: handling heterogeneity, augmentation variation and management of smart objects, evolution of smart object systems and effective programming model. A number of existing device integration systems intended for context-aware computing, home computing, or other relevant domains were discussed to form the basis for the necessity of a new framework. Then the proposed document based framework was introduced while explaining the design decisions in respect to the design requirements. The framework forces applications' requirements and smart objects' features to be externalized through documents and employs a secondary infrastructure, FedNet to create a spontaneous federation among the application and smart objects using the corresponding documents. The later part of the chapter looked at the end-user aspects and argued that involving end-users in the deployment and administration of smart object systems can elevate end-user experiences. Accordingly, architectural qualities for a framework to support end-user involvement were discussed.

Chapter 4 presented the implementation of the proposed framework and its constituents along with corresponding design methodologies. The framework is composed of three components:

- **Smart Object Wrapper:** The core-cloud model is realized by smart object wrapper. The core contains communication module, notification module, static memory, client handler, and profile repository. Augmented feature is developed as profile and plugged into the core. The profiles' services along with basic smart objects properties are objectified in external documents.
- **Application Development Process:** The proposed framework forces an application to be structured based on functional tasks that require smart objects' services and externalize these tasks in a document. Furthermore, it is required for an application to implement a basic RESTful communication protocol (HTTP/XML) to access FedNet to interact with the underlying smart objects.
- **FedNet Runtime Infrastructure:** FedNet provides the runtime association among the applications and smart objects by utilizing the documents describing smart objects and applications' runtime requirements. It can contact smart objects using the semantics described in the corresponding documents for mapping application tasks, similarly application can contact FedNet in a RESTful manner for utilizing the underlying smart objects.

Each of these component was discussed in detail with concrete illustrations. After that, two end-user interaction tools built on top of the framework were discussed.

Chapter 5 discussed the evaluation of the proposed framework from three presepectives: quantitative, qualitative and end-user aspects. Three smart object systems were presented that were used in the evaluation process. After discussing the runtime performance of Fed-Net, the chapter revisited the design requirements to assess the framework qualitatively. Finally, a couple of end-user studies were reported that highlighted the usability of the interaction tools and exposed a few design implications for building user-centric smart object systems. The take away message from the study is the fact that the end-users might be involved in deploying future smart object systems if appropriate tools and supporting infrastructure are provided.

In Chapter 6, several issues raised by this research were further discussed. The applicability of the proposed approach in other domains was also highlighted with a few conceptual illustrations.

The major contributions of this thesis are

1. An exploration of a range of smart object systems to formalize their design rationales and consequently the development of a core-cloud theoretical model.
2. Identification of the design requirements of a supporting framework for smart object systems and accordingly the development of a conceptual document centric framework along with design processes for building reusable and extensible smart objects systems.
3. The concrete implementation of the framework with appropriate illustrations.
4. Introduction of the novel notion of end-user centric qualities for system infrastructure to elevate user experiences and the consequent implications of these quality requirements in the design of the framework.

7.2 Future Research Directions

There are, however, a number of issues that are raised by this research but could not be addressed fully. This section investigates some of these issues and points to potentially fruitful areas of future research based on this work.

7.2.1 Specification and Description of Smart Object Services

The smart object wrapper in the presented document based framework implements the core-cloud model and hosts a collection of service profiles. Application typically utilizes the smart

objects by interacting with these profiles. In the proposed framework sensor modeling language and actuator modeling language are used to describe the profiles' services. The profile notion has the potentially serious implication that standard common vocabularies or ontologies will be needed to support general interoperability of profiles and applications. This is one of the major problems that has been identified through the development of this work. However, the profile abstraction does not attempt to define and standardize the schema, instead provides a structure that designers can use to disseminate their implemented data format and glue it with the rest of the infrastructure. Defining the widely accepted and approved description/specification in a standard way is the hardest part of pervasive computing not the encoding. This work does not claim that the proposed framework is providing a solution to that. An important future research direction is to have a common consensus across developers of smart object systems and to define common schemas for describing profiles. One of the fastest and easiest ways is to construct a web thesaurus that can independently be updated and shared across the developers.

7.2.2 Integration of Location Information

One aspect that is currently missing in the current framework is the utilization of explicit location of smart objects unless one of the profiles of a smart object is responsible for providing location information. Location is considered as the primary context for proactive service provision and is particularly important for future ubiquitous environment where various pervasive services will exist to support end-users. A suitable location system that is available for the infrastructure (e.g., FedNet) and the application would enable both to support more sophisticated service provisions. Indoor localization is the most active research topic in the field and till date several location systems are investigated, e.g., location systems using dedicated infrastructures [Want et al., 1992, Priyantha et al., 2000, Beigl et al., 2002], location systems using existing indoor infrastructures [Patel et al., 2006], location systems using smart objects [Kawsar et al., 2007a] etc. An interesting future research direction is to combine a suitable localization technique with the current framework.

7.2.3 Incorporating Security Aspect

One issue that has not been discussed in this work is the security aspect. This is particularly important for smart object systems in context due to the fact that in a smart object system it is expected that environment components will be responsible for monitoring our activities and taking autonomous decisions. Although, this issue was not addressed in this work concretely, the proposed framework design allows incorporating this aspect in smart object systems effectively. For example, one profile could be developed that end-users can configure to state the suitable security and access control scheme for a smart object. In addition, FedNet is modular and plug-gable, making it very easy to associate a global security component for the entire environment in context that can protect smart object services from malicious attacks.

Security and privacy issues are active research topic in mobile and ubiquitous computing research. One immediate avenue of future work is to adopt the outcome of their research in smart object systems.

7.2.4 Architectural Qualities for Improving User Experience

There are a few of design issues that have been identified in the current framework that were essential to expose the system to end-users: i) supporting plug and play smart objects and applications, ii) supporting extensibility, iii) isolating smart objects, applications and infrastructure completely and letting them snap at runtime dynamically and iv) supporting the development of a suitable tool on top of the infrastructure to enable end-users to involve in the deployment, extension and administration processes mimicking their experiences in personal computing environment. Although these issues are well exposed through this work, it is imperative to mention that there could be several other design issues at the architectural layer that could be brought out to further improve end-users experiences with smart object systems. Thus an interesting future research direction is to explore further to identify architectural qualities that can elevate user experiences. Furthermore, the outcome can be applied to find an appropriate balance between system architectures, user interfaces, and usability research.


7.2.5 End-User Tools

One of the unique aspects of the proposed framework is its support for end-user involvement in the deployment, extension and administration processes of smart object systems in a Do-It-Yourself (DIY) manner. This work is one of the earliest efforts in this direction. Currently, a couple of interaction tools were built to support end-users in these activities. However, it is still not clear that what sort of user interfaces and interaction techniques are most suitable for end-users of varying skills. A very important future research direction is to conduct further case studies in order to identify a suitable design guideline for building such end-user tools to support their involvement in the deployment, extension, administration of smart object systems.

7.3 Concluding Remark

The dissertation has focused on three primary issues, i.e., an exploration of smart object systems to formalize the design rationales, an appropriate framework for building smart object systems and involving end-users in the administration process of those systems, which in turn provides several design implications for the proposed framework. The dissertation at hand discusses these aspects and shows that the proposed document centric framework with appropriate abstractions can elegantly meet several challenges of building and extending smart

objects systems in a more humane way. The framework dramatically reduces the complexity of building and extending smart object systems by isolating all access issues, eliminating dedicated discovery process and API dependencies, hiding heterogeneity and providing appropriate balance between transparency and awareness. In addition of approaching these fundamental system level challenges, the framework has also considered human-centric aspects from a system perspective and argued that by involving end-users in the system administration process it is possible to elevate their experiences with smart object systems. The implications of this work are very useful for further research exploration in the ubiquitous computing domain, particularly one that involves smart objects.



Appendices

Appendix A

Document Type Definition (DTD)

This appendix lists the Document Type Definitions (DTDs) for the different types of documents that are used in the framework. Developers can use these definitions to write documents for their respective smart object systems.

A.1 Document Type Definition for Smart Object's Documents

There are basically two types of documents that developers need to create while disseminating smart objects as generic binaries. These two documents are:

- Smart Object Description Document
- Profile Description Document

These documents are written using custom defined XML tags. In the following the Document Type Definitions (DTDs) for these two documents are presented.

A.1.1 Document Type Definition for Smart Object Description Document

This document is the generic description of a smart object and contains references of the profiles that run on the smart object. The Document Type Definition for this document is shown in Figure A.1.

A.1.2 Document Type Definition for Profile Description Documents

The Profile Description Document (PDD) describes the input and output specifications of the profile. Each PDD contains either a **<detector>** or an **<actuator>** node based on the profile type.

```

<!DOCTYPE artefact SYSTEM "sodd.dtd">

<ELEMENT artefact (name?, vendor?, profiles?)>
<ELEMENT profiles (profile*)>
<ELEMENT profile (codebase+)>

<!ATTLIST profile
  name CDATA #REQUIRED>

<ELEMENT name (#PCDATA)>
<ELEMENT vendor (#PCDATA)>
<ELEMENT codebase (#PCDATA)>

```

FIGURE A.1: Document Type Definition for Smart Object Description Document

PDD can also contain a quality of service (QoS) block which specifies profile's quality. Since, each profile requires a physical instrumentation of the smart object, PDD can also provide installation instruction. These instructions are encoded under the **<installation-instruction>** block. The Document Type Definition for this document is shown in Figure A.2.

```

<!DOCTYPE profile SYSTEM "pdd.dtd">

<ELEMENT profile (name+, purpose+, type+, detector?, actuator?, profile-QoS-attribute?, installation-instruction*)>
<ELEMENT detector (identification+, referenceFrame*, inputs?, outputs?)>
<ELEMENT actuator (identification+, states+)>
<ELEMENT states (state+)>
<ELEMENT state (name+, inputs+, outputs+)>

<ELEMENT inputs (input+)>
<ELEMENT input (name+, parameter*)>
<ELEMENT parameter (MIMEdatatype+, value+)>

<ELEMENT outputs (output+)>
<ELEMENT output (name+, datatype+, value+)>

<ELEMENT profile-QoS-attribute (qos+)>
<ELEMENT qos (name+, datatype+, measurement-unit+, high-threshold+, low-threshold+)>

<ELEMENT installation-instruction (instruction+)>
<ELEMENT instruction (stmt+)>

<ELEMENT name (#PCDATA)>
<ELEMENT purpose (#PCDATA)>
<ELEMENT type (#PCDATA)>

<ELEMENT identification (#PCDATA)>
<ELEMENT referenceFrame (#PCDATA)>
<ELEMENT datatype (#PCDATA)>
<ELEMENT MIMEdatatype (#PCDATA)>
<ELEMENT value (#PCDATA)>

<ELEMENT measurement-unit (#PCDATA)>
<ELEMENT high-threshold (#PCDATA)>
<ELEMENT low-threshold (#PCDATA)>
<ELEMENT stmt (#PCDATA)>

```

FIGURE A.2: Document Type Definition for Profile Description Document

A.2 Document Type Definition for Application's Document

The framework in context forces an application to expose its functional task lists in a Task Description Document. The specification of each task expresses its runtime profile requirement. Each task can also specify the Quality of Service (QoS) requirements that should be attained by the respective profile. This document is also written using custom defined XML tags. The Document Type Definition for this document is shown in Figure A.3.

```
<!DOCTYPE application SYSTEM "tdd.dtd">
<!ELEMENT application (name? , app-purpose?, binaryPath+, accesspoint+, task-list*)>

<!ELEMENT accesspoint (IP+, port+)>
<!ELEMENT task-list (task+)>
<!ELEMENT task (id+, purpose?, required-profile-type+, profile-name+, communication-mode?, profile-QoS-attribute?)>

<!ELEMENT profile-QoS-attribute (qos+)>
<!ELEMENT qos (name+, datatype+, measurement-unit+, high-threshold+, low-threshold+)>

<!ELEMENT name (#PCDATA)>
<!ELEMENT app-purpose (#PCDATA)>
<!ELEMENT binaryPath (#PCDATA)>

<!ELEMENT IP (#PCDATA)>
<!ELEMENT port (#PCDATA)>

<!ELEMENT id (#PCDATA)>
<!ELEMENT purpose (#PCDATA)>
<!ELEMENT required-profile-type (#PCDATA)>
<!ELEMENT profile-name (#PCDATA)>
<!ELEMENT communication-mode (#PCDATA)>

<!ELEMENT datatype (#PCDATA)>
<!ELEMENT measurement-unit (#PCDATA)>
<!ELEMENT high-threshold (#PCDATA)>
<!ELEMENT low-threshold (#PCDATA)>
```

FIGURE A.3: Document Type Definition of Task Description Document

Appendix B

Specification of Programming Interface

In the current framework all three components (smart objects, applications, FedNet) are packaged as generic binaries that are dynamically snapped to each other. This dynamic association is achieved through documents and by forcing applications and smart objects to implement a common protocol (HTTP/XML). So, in the current framework the notion of application programming interface is not actively utilized.

Basically, there are two points where explicit development is necessary in this framework taking the nature of generic binary component into account. The first point is the development of profiles that are smart object independent and the second point is the application itself. From a smart object perspective, the core is a generic binary that dynamically load different profiles. These profiles should be developed independently. To ensure that these profiles work with the core at runtime, a development library is offered to developers that enable them to provide device specific implementation. From an application perspective, basically there is no need for development library in a true sense as the external communication uses standard HTTP/XML based protocol in a RESTful manner. However to further ease the development effort, in the current work a simple development library is offered to the developers that abstracts the communication protocol and enables application developers to structure their application code around functional tasks. In the following a few important language interfaces of these libraries are provided.

B.1 Language Interface for Smart Object Development

Each smart object comes with a generic binary core that contains a plug-in runtime to enable different profiles to be plugged-in. These profiles are developed independently and should be implemented in the semantics of current framework so that they can work with the underlying core component. Each profile is structured into two layers as shown in Figure B.1.

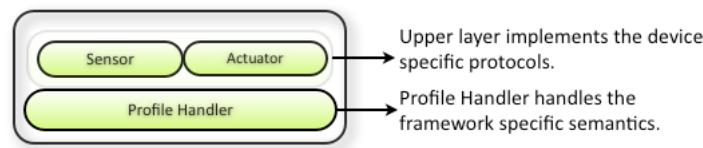


FIGURE B.1: Structure of a Profile

The Profile Handler encapsulates all the functionalities to ensure that the profile works with the smart object core in the context of current framework. These functionalities include

1. Implementing the plug-in structure.
2. Encoding a profile output following the prescribed Sensor Modeling Language of the respective profile as defined in the Profile Description Document.
3. Decoding a profile input encoded in the prescribed Actuator Modeling Language of the respective profile as defined in the Profile Description Document.
4. Providing the runtime communication, i.e., delegating the profile output and input to and from smart object core.
5. Providing a template for the developers that they can use to implement their device specific code.

So, the only development effort needed here is to implement device specific code and connect this code to Profile Handler. To perform this operations, developers are required to manipulate the *Profile* class.

Profile Class Each profile should extend the base *Profile* class and should implement two abstract functions. These are:

```

public void setSML();
public String executeService(Service args);

```

For a sensor type profile, *setSML()* is need to be implemented. Inside the function developers should use the

```

Profile.SML.output(String output, String value);

```

to provide the profile output and then invoke

```

Profile.notifyAccessPoint();

```

to instruct the profile handler to initiate external communication.

For an actuator type profile, `executeService(Service args)` is required to be implemented. This function is automatically invoked by underlying Profile Handler whenever an external request is received. The parameter of this function is an instance of `Service` class that represents the decoded value of the profile input following the syntax of Actuator Modeling Language.

B.2 Language Interface for Application Development

As mentioned earlier, this library is not a mandate for application developers to use for building applications that work in the current framework environment. However, this library is offered to further ease the development effort by providing high level abstractions. There are three classes that need to be discussed here. In the following these classes and their important interfaces are explained.

Task Class This class is provided to the developer to structure application code around functional tasks. Once developers externalize their functional task lists in a Task Description Document, the member functions of this class can be used to operate on the tasks. Each functional task is represented by an instance of `Task` class. The important member functions of this class useful for application development are:

`public String getID();`

This function provides the task ID that can be used to refer the task for subsequent operations.

`public String getTaskStatus();`

This function provides the current execution status of the task.

`public String getProfileStatus();`

This function provides the status of the profile that the task needs.

`public void subscribe(Object source, String callback);`

This function enables subscribing to the profile output that the task represents.

AccessPoint Class This static class acts as the communication point for the application developers to communicate with underlying Access Point component of FedNet and thereby interacting with smart objects. The most important functions of this class are:

`public int sendTaskRequest(String message, String taskID);`

Application uses this function to send task request to Access Point. The underlying implementation of the library takes care of all access issues.

XmlProcessor Class This is a meta class that application developers can use to operate on Task Description Document. Furthermore, this class provides support to encode and decode the profile input and output following the syntax of Profile Description Document of respective profiles. The important member functions of this class are:

public boolean parseTDD();

This function parses the Task Description Document to set up the task list of the applications.

public String generateOutgoingMessage(String taskType, String taskID);

This function generates the outgoing XML message following the syntax of Actuator Modeling Language of the respective profiles.

public DetectorData parseIncomingMessage(String profileSML);

This function parses the incoming profile output specification encoded in the syntax of Sensor Modeling Language to generate an instance of *DetectorData* that represents the decoded value of the profile output.

Appendix C

User Study Material

In this appendix the questionnaires and the interview questions used in the end-user usability studies are presented.

C.1 Post-study Questionnaire

The questionnaires presented to the participants were structured into three parts. The first part was designed to collect the biographical and background data of the participants. These questions are listed below.

1. Age Range -
:: 15-19 :: 20-24 :: 25-29 :: 30-34 :: 35-39 :: 40-44 :: 45-49 :: 50+
2. Gender -
:: Male :: Female
3. Profession (If you are a student please write your major.) -
4. Experience with computers -
:: Less than 1 year :: 1-2 years :: 3-4 years :: 5+ years
5. How would you rate your computer knowledge?
:: Novice :: Intermediate :: Expert

The second part of the questionnaires contained ten statements. These statements were designed following the System Usability Scale (SUS) [Brooke, 1996] The System Usability Scale (SUS) is a simple Likert scale giving a global view of subjective assessments of usability. Ten carefully selected statements are offered to the participants and they can respond by indicating the degree of agreement or disagreement with the statements on a 5 point scale. In the following these statements are listed.

1. I think that I would like to have this system if it were available -
i) 1 [Strongly Disagree] ii) 2 iii) 3 iv) 4 v) 5 [Strongly Agree]
2. I found the system unnecessarily complex -
i) 1 [Strongly Disagree] ii) 2 iii) 3 iv) 4 v) 5 [Strongly Agree]
3. I thought the system was easy to use -
i) 1 [Strongly Disagree] ii) 2 iii) 3 iv) 4 v) 5 [Strongly Agree]
4. I think that I would need the support of a technical person to use this system -
i) 1 [Strongly Disagree] ii) 2 iii) 3 iv) 4 v) 5 [Strongly Agree]
5. I found the various functions in this system were well integrated -
i) 1 [Strongly Disagree] ii) 2 iii) 3 iv) 4 v) 5 [Strongly Agree]
6. I thought there was too much inconsistency in this system -
i) 1 [Strongly Disagree] ii) 2 iii) 3 iv) 4 v) 5 [Strongly Agree]
7. I would imagine that most people would learn to use this system very quickly -
i) 1 [Strongly Disagree] ii) 2 iii) 3 iv) 4 v) 5 [Strongly Agree]
8. I found the system very cumbersome to use -
i) 1 [Strongly Disagree] ii) 2 iii) 3 iv) 4 v) 5 [Strongly Agree]
9. I felt very confident using the system -
i) 1 [Strongly Disagree] ii) 2 iii) 3 iv) 4 v) 5 [Strongly Agree]
10. I needed to learn a lot of things before I could get going with this system -
i) 1 [Strongly Disagree] ii) 2 iii) 3 iv) 4 v) 5 [Strongly Agree]

Finally in the third part four questions were designed to identify the complexity level of the four tasks that were given to the participants. They were asked to rate the answers in the 5 point scale to indicate the perceived complexity level. In the following these questions are presented.

1. How complex was the process of adding a smart object -
i) 1 [Very Hard] ii) 2 iii) 3 iv) 4 v) 5 [Very Easy]
2. How complex was the process of installing an application -
i) 1 [Very Hard] ii) 2 iii) 3 iv) 4 v) 5 [Very Easy]
3. How complex was the process of adding a profile to a smart object -
i) 1 [Very Hard] ii) 2 iii) 3 iv) 4 v) 5 [Very Easy]
4. How complex was the process of administrating the entire system -
i) 1 [Very Hard] ii) 2 iii) 3 iv) 4 v) 5 [Very Easy]

C.2 Post-study Interview Questions

After the questionnaires, each participant was invited for an interview session. On an average each interview took 20 minutes. The interview questions were organised into two parts. In the first part, five questions were asked related to the tasks involved in the study session. These questions were as follows:

1. Which application (AwareMirror and Virtual Aquarium) did you pick first and why?
2. Which profile hardware did you pick for proximity profile and why?
3. Which profile hardware did you pick for Bi-state interaction profile and why?
4. Where did you attach the profile hardware in the mirror and why?
5. Which task was the most complicated one and can you suggest what would make it easy for you?

The second part of the interview was loosely structured, a few questions were put forth for understanding participants' views on the entire end-user involvement process and for identifying potential improvement areas. In the following these questions are listed.

1. Do you find any similarities between today's study tasks and your activities at home for installing devices, appliances, if yes how?
2. Do you think such approach is useful for making a smart home by yourself, if yes why do you think so?
3. Is there any specific things that you noticed or experienced in this test that would discourage you to adopt these kinds of systems in your home?
4. How often do you think you will interact with such systems to upgrade system functions?
5. What other applications you can think of with these kinds of hardware/software available at your home?

Bibliography

- [Addlesee et al., 2001] Addlesee, M., Curwen, R., Hodges, S., Newman, J., Steggels, A. W. P., and Hooper, A. (2001). Implementing a sentient computing system. *Cover Feature in IEEE Computer*, 34(8):50--56.
- [Aitenbichler et al., 2007] Aitenbichler, E., Lyardet, F., Austaller, G., Kangasharju, J., and Mühlhäuser, M. (2007). Engineering intuitive and self-explanatory smart products. In *22nd Annual ACM Symposium on Applied Computing (SAC 2007)*, pages 1632--1637.
- [Andreoli et al., 2003] Andreoli, J.-M., Castellani, S., Grasso, A., Meunier, J.-L., Muehlenbrock, M., O'Neill, J., Ragnet, F., Roulland, F., and Snowdon, D. (2003). Augmenting offices with ubiquitous sensing. In *Smart Objects Conference (SOC 2003)*.
- [Antifakos et al., 2002] Antifakos, S., Michahelles, F., and Schiele, B. (2002). Proactive instructions for furniture assembly. In *4th International Conferenc on Ubiquitous Computing (UbiComp 2002)*, pages 351--360.
- [Ballagas et al., 2004] Ballagas, R., Szybalski, A., and Fox, A. (2004). Patch panel: Enabling control-flow interoperability in ubicomp environments. In *2nd IEEE Annual Conference on Pervasive Computing and Communications (PerCom 2004)*, pages 241-- 252.
- [Banks et al., 2007] Banks, R., Regan, T., Harper, R., and Sellen, A. (2007). Bubbleboard: A visual answering machine. In *10th European Conference on Computer-Supported Cooperative Work (ECSCW 2007)*.
- [Bardram, 2005] Bardram, J. E. (2005). The java context awareness framework - a service infrastructure and programming framework for context-aware applications. In *The 3rd International Conference on Pervasive Computing (Pervasive 2005)*, pages 98--115.
- [Baurley et al., 2007] Baurley, S., Brock, P., Geelhoed, E., and Moore, A. (2007). Communication-wear. In *Workshop on Transitive Materials at 9th international conference on Ubiquitous computing (UbiComp 2007)*.
- [Beckmann et al., 2004] Beckmann, C., Consolvo, S., and LaMarca, A. (2004). Some assembly required: Supporting end-user sensor installation in domestic ubiquitous computing environments. In *6th international conference on Ubiquitous computing (UbiComp 2004)*, pages 107--124.

- [Beigl et al., 2001] Beigl, M., Gellersen, H. W., and Schmidt, A. (2001). Media cups: Experience with design and use of computer augmented everyday objects. *Computer Networks, special Issue on Pervasive Computing*, 35-4:401--409.
- [Beigl et al., 2004] Beigl, M., Krohn, A., Zimmer, T., and Decker, C. (2004). Typical sensors needed in ubiquitous and pervasive computing. In *the First International Workshop on Networked Sensing Systems (INSS 2004)*, pages 153--158.
- [Beigl et al., 2002] Beigl, M., Zimmer, T., and Decker, C. (2002). A location model for communicating and processing of context. *Personal and Ubiquitous Computing*, 6(5-6):341--357.
- [Bell and Dourish, 2007] Bell, G. and Dourish, P. (2007). Yesterday's tomorrows: Notes on ubiquitous computing's dominant vision. In *Personal and Ubiquitous Computing*, volume 11(2), pages 133--143.
- [Brooke, 1996] Brooke, J. (1996). *SUS: A quick and dirty usability scale*, pages 189--194. Usability Evaluation in Industry. Taylor and Francis, London.
- [Brown, 1996] Brown, J. P. (1996). The stick-e document: A framework for creating context aware applications. *Electronic Publishing*, 8(2):259--272.
- [Brumitt et al., 2000] Brumitt, B. L., Meyers, B., Krumm, J., Kern, A., and Shafer, S. (2000). Easyliving: Technologies for intelligent environments. In *2nd International Symposium on Handheld and Ubiquitous Computing (HUC 2000)*, pages 12--29.
- [Deborah and Debaty, 2000] Deborah, C. and Debaty, P. (2000). Creating web representations for places. In *2nd International Symposium on Handheld and Ubiquitous Computing (HUC 2000)*, pages 114 -- 126.
- [Dey, 2000] Dey, A. K. (2000). *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, College of Computing, Georgia Institute of Technology.
- [Dey, 2001] Dey, A. K. (2001). Understanding and using context. *Personal and Ubiquitous Computing Journal*, 5(1):4--7.
- [Dey et al., 2001] Dey, A. K., Abowd, G., and Salber, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2-4):97--166.
- [Dey et al., 2004] Dey, A. K., Hamid, R., Beckmann, C., Li, I., and Hsu, D. (2004). a cappella: Programming by demonstration of context-aware applications. In *ACM Conference on Human Factors in Computing Systems (CHI 2004)*, pages 33 -- 40.
- [Dey et al., 2006] Dey, A. K., Shon, T., Streng, S., and Kodama, J. (2006). icap: Interactive prototyping of context-aware applications. In *4th International Conference on Pervasive Computing (Pervasive 2006)*, pages 254--271.

- [Edwards et al., 2003] Edwards, W. K., Bellotti, V., Dey, A. K., and Newman, M. W. (2003). Stuck in the middle: The challenges of user-centered design and evaluation of infrastructure. In *ACM Conference on Human Factors in Computing Systems (CHI 2003)*, pages 297--304.
- [Edwards and Grinter, 2001] Edwards, W. K. and Grinter, R. (2001). At home with ubiquitous computing: Seven challenges. In *Ubicomp 2001*.
- [Edwards et al., 2002] Edwards, W. K., Newman, M., Sedivy, J., Smith, T., and Izadi, S. (2002). Challenge: Recombinant computing and the speakeasy approach. In *8th Annual International Conference on Mobile Computing and Networking (MobiCom 2002)*, pages 279--286.
- [Englander, 1997] Englander, R. (1997). *Developing Java Beans*. O'Reilly and Associates.
- [Fielding, 2000] Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine.
- [Fox et al., 2000] Fox, A., Johanson, B., Hanrahan, P., and Winograd, T. (2000). Integrating information appliances into an interactive workspace. In *IEEE Computer Graphics and Applications*, pages 54--65.
- [Fujinami et al., 2005] Fujinami, K., Kawsar, F., and Nakajima, T. (2005). Awaremirror: A personalized display using a mirror. In *3rd Third International Conference on Pervasive Computing (Pervasive 2005)*, pages 315--332.
- [Fujinami and Nakajima, 2005] Fujinami, K. and Nakajima, T. (2005). Sentient artefacts: Acquiring user's context through daily object. In *The 2nd International Workshop on Ubiquitous Intelligence and Smart Worlds (UISW2005)*, pages 335--344.
- [Gajos et al., 2002] Gajos, K., Fox, H., and Shrobe, H. (2002). End user empowerment in human centered pervasive computing. In *International Conference on Pervasive Computing (Pervasive 2002)*, pages 1--7.
- [Gelernter, 1985] Gelernter, D. (1985). Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80--112.
- [Gellersen et al., 2004] Gellersen, H., Kortuem, G., Schmidt, A., and Beigl, M. (2004). Physical prototyping with smart-its. *IEEE Pervasive Computing*, 03(3):74--82.
- [Gellersen et al., 2000] Gellersen, H. W., Schmidt, A., and Beigl, M. (2000). Adding some smartness to devices and everyday things. In *Third IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2000)*, pages 3--10.
- [Greenberg and Fitchett, 2001] Greenberg, S. and Fitchett, C. (2001). Phidgets: Easy development of physical interfaces through physical widgets. *14th Annual ACM Symposium on User Interface Software and Technology (UIST 2001)*, pages 209 -- 218.

- [Hanaoka et al., 2006] Hanaoka, K., Takagi, A., and Nakajima, T. (2006). A software infrastructure for wearable sensor networks. In *The 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2006)*, pages 27--35.
- [Harter et al., 1999] Harter, A., Hopper, A., Steggles, P., Ward, A., and Webster, P. (1999). The anatomy of a context-aware application. In *5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 59--68.
- [Helal et al., 2005] Helal, A., Mann, W., Elzabadani, H., King, J., Kaddourah, Y., and Jansen, E. (2005). Gator tech smart house: A programmable pervasive space. *IEEE Computer*, 38(3):50--60.
- [Hinckley et al., 2000] Hinckley, K., Pierce, J., Sinclair, M., and Horvitz, E. (2000). Sensing techniques for mobile interaction. In *13th ACM Symposium on User Interface Software and Technology (UIST 2000)*, pages 91 -- 100.
- [Hitachi, 2008] Hitachi (2008). Miragraphy <http://hhil.hitachi.co.jp/products/miragraphy.html> (in japanese).
- [Hong and Landay, 2001] Hong, J. I. and Landay, J. A. (2001). An infrastructure approach to context aware computing. *Human-Computer Interaction*, 16:287--303.
- [Hong and Landay, 2004] Hong, J. I. and Landay, J. A. (2004). An architecture for privacy-sensitive ubiquitous computing. In *The Second International Conference on Mobile Systems, Applications, and Services (Mobisys 2004)*, pages 177--189.
- [Humble et al., 2003] Humble, J., Crabtree, A., Hemmings, T., Karl-Petter Åkesson, B. K., Rodden, T., and Hansson, P. (2003). Playing with your bits: User composition of ubiquitous domestic environments. In *5th International Conference on Ubiquitous Computing (Ubi-comp 2003)*, pages 256--263.
- [Iseminger, 2000] Iseminger, D. (2000). *Com+ Developer's Reference*. Microsoft Press.
- [Ishii, 2004] Ishii, H. (2004). Bottles: A transparent interface as a tribute to mark weiser. *IEICE Transactions on Information and Systems*, E87-D(6):1299--1311.
- [Iwabuchi and Siio, 2008] Iwabuchi, E. and Siio, I. (2008). Smart makeup mirror: Computer augmented mirror to aid makeup application. In *Adj. Proceedings of 10 International Conference on Ubiquitous Computing (UbiComp 2008)*.
- [Johanson et al., 2002] Johanson, B., Fox, A., and Winograd, T. (2002). The interactive workspaces project: experiences with ubiquitous computing rooms. *IEEE Pervasive Computing*, 1(2):67 -- 74.
- [Kameas et al., 2004] Kameas, A., Mavrommati, I., and Markopoulos, P. (2004). *Computing in tangible: using artifacts as components of ambient intelligence environments*. IOS Press.

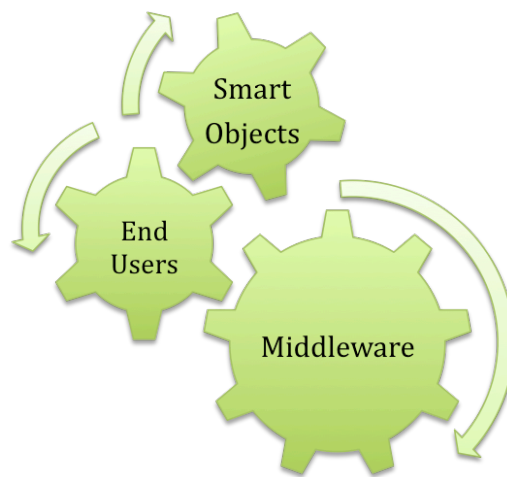
- [Kawsar et al., 2005] Kawsar, F., Fujinami, K., and Nakajima, T. (2005). Augmenting everyday life with sentient artefacts. In *2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies (sOc-EUSAI 2005)*, pages 141--146.
- [Kawsar et al., 2007a] Kawsar, F., Fujinami, K., and Nakajima, T. (2007a). A lightweight indoor location model for sentient artefacts using sentient artefacts. In *The 2007 ACM Symposium on Applied Computing (SAC 2007)*, pages 1624--1631.
- [Kawsar et al., 2007b] Kawsar, F., Fujinami, K., Pirttikangas, S., Hayashi, K., and Nakajima, T. (2007b). Roonroon: A wearable teddy as social interface for contextual notification. In *The International Conference and Exhibition on Next Generation Mobile Applications, Services and Technologies*, pages 76--84.
- [Kohtake et al., 2005] Kohtake, N., Ohsawa, R., Iwai, M., Takashio, K., and Tokuda, H. (2005). u-texture: Self-organizable universal panels for creating smart surroundings. In *7th International Conference on Ubiquitous Computing (Ubicomp 2005)*, pages 19--36.
- [Konomi and Roussos, 2006] Konomi, S. and Roussos, G. (2006). Ubiquitous computing in the real world: Lessons learnt from large scale rfid deployments. *Personal and Ubiquitous Computing*, 11(7):507--521.
- [Kortuem et al., 2007] Kortuem, G., Davies, N., Efstratiou, C., Kinder, K., White, M. I., Hooper, R., Finney, J., Ball, L., Busby, J., and Alford, D. (2007). Sensor networks or smart artifacts? an exploration of organizational issues of an industrial health and safety monitoring system. In *9th International Conference on Ubiquitous Computing (UbiComp 2007)*, pages 465--482.
- [Krasner and Pope, 1998] Krasner, G. and Pope, S. T. (1998). A cookbook for using the model view controller user interface paradigm in smalltalk-80. *Journal of Object-Oriented Programming*, 1(3):26 -- 49.
- [Krieger and Adler, 1998] Krieger, D. and Adler, R. (March, 1998). The emergence of distributed component platforms. *IEEE Computer Magazine*, pages 43--53.
- [Lampe and Strassner, 2003] Lampe, M. and Strassner, M. (2003). The potential of rfid for movable asset management. In *Workshop on Ubiquitous Commerce at 5th International Conference on Ubiquitous Computing (Ubicomp 2003)*.
- [Lashina, 2004] Lashina, T. I. (2004). Intelligent bathroom. In *Ambient Intelligence Technologies for Wellbeing at Home, Workshop on European Symposium on Ambient Intelligence (EUSAI 2004)*.
- [Leffler et al., 1989] Leffler, S. J., McKusick, M. K., Karels, M. J., and Quarterman, J. S. (1989). *The Design and Implementation of the 4.3 BSD UNIX Operating System*. Addison-Wesley, Reading, MA.

- [Ljungstrand et al., 2000] Ljungstrand, P., Redström, J., and Holmquist, L. E. (2000). Webstickers: using physical tokens to access, manage and share bookmarks to the web. In *DARE 2000 (Designing augmented reality environments)*, pages 23 -- 31.
- [Mattern, 2003] Mattern, F. (2003). From smart devices to smart everyday objects. In *Smart Object Conference 2003*.
- [Messer et al., 2006] Messer, A., Kunjithapatham, A., Sheshagiri, M., Song, H., Kumar, P., Nguyen, P., and Yi, K. H. (2006). Interplay: A middleware for seamless device integration and task orchestration in a networked home. In *4th Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2006)*, pages 307--316.
- [Molyneaux et al., 2007] Molyneaux, D., Gellersen, H., Kortuem, G., and Schiele, B. (2007). Cooperative augmentation of smart objects with projector-camera systems. In *9th International Conference on Ubiquitous Computing (UbiComp 2007)*, pages 501--518.
- [Monson-Haefel, 2001] Monson-Haefel, R. (2001). *Enterprise Java Beans*. O'Reilly.
- [Moore, 1965] Moore, G. (1965). Cramming more components onto integrated circuits. *Electronics*, 38(8).
- [Mowbray and Zahavi, 1995] Mowbray, T. J. and Zahavi, R. (1995). *The Essential Corba: System Integration Using Distributed Objects*. John Wiley and Sons.
- [Nakajima et al., 2008] Nakajima, T., Lehdonvirta, V., Tokunaga, E., and Kimura, H. (2008). Reflecting human behavior to motivate desirable lifestyle. In *The Conference on Designing Interactive Systems (DIS 2008)*, pages 405--414.
- [Norman, 1990] Norman, D. (1990). *The Design of Everyday Things*. Broadway Books.
- [Norman, 1998] Norman, D. (1998). *The Invisible Computer*. MIT Press.
- [O'Brien et al., 1999] O'Brien, J., Rodden, T., and Hughes, M. R. J. (1999). At home with the technology: an ethnographic study of a set-top-box trial. *ACM Transactions on Computer-Human Interaction*, 6(3):282 -- 308.
- [Olsen et al., 2001] Olsen, D., Nielsen, T., and Parslow, D. (2001). Join and capture: a model for nomadic interaction. In *4th annual ACM symposium on User interface software and technology (UIST 2001)*, pages 131--140.
- [Orr and Abowd, 2000] Orr, R. J. and Abowd, G. (2000). The smart floor: A mechanism for natural user identification and tracking. In *Conference on Human Factors in Computing Systems (CHI 2000), Extended Abstracts*, pages 275 -- 276.
- [Paradiso et al., 2000] Paradiso, J., Hsiao, K., and Benbasat, A. (2000). Interfacing to the foot: Apparatus and applications,. In *ACM Conference on Human Factors in Computing Systems (CHI 2000) Extended Abstracts*.

- [Pascoe, 1997] Pascoe, J. (1997). The stick-e note architecture: Extending the interface beyond the user. In *2nd international conference on Intelligent user interfaces (IUI 1997)*, pages 261 -- 264.
- [Pascoe, 2001] Pascoe, J. (2001). *Context-Aware Software*. PhD thesis, Computing Laboratory, University of Kent at Canterbury.
- [Patel et al., 2006] Patel, S. N., Truong, K. N., and Abowd, G. (2006). Powerline positioning: A practical sub-room-level indoor location system for domestic use. In *8th International Conference on Ubiquitous Computing (UbiComp 2006)*, pages 441--458.
- [Priyantha et al., 2000] Priyantha, N. B., Chakraborty, A., and Balakrishnan, H. (2000). The cricket location-support system. In *The Sixth Annual ACM International Conference on Mobile Computing and Networking (MOBICOM 2000)*, pages 32--43.
- [Rekimoto, 1996] Rekimoto, J. (1996). Tilting operations for small screen interfaces. In *9th ACM Symposium on User Interface Software and Technology (UIST 1996)*, pages 167 -- 168.
- [Rekimoto and Ayatsuka, 2000] Rekimoto, J. and Ayatsuka, Y. (2000). Cybercode: Designing augmented reality environment with visual tags. In *Designing Augmented Reality Environment (DARE 2000)*, pages 1--10.
- [Rodden and Benford, 2003] Rodden, T. and Benford, S. (2003). The evolution of buildings and implications for the design of ubiquitous domestic environments. In *ACM CHI 2003*.
- [Rogers, 2006] Rogers, Y. (2006). Moving on from weiser's vision of calm computing: Engaging ubicomp experiences. In *The Eighth International Conference on Ubiquitous Computing (UbiComp 2006)*, pages 404--421.
- [Roman et al., 2002] Roman, M., Hess, C. K., Cerqueira, R., Ranganathan, A., Campbell, R. H., and Nahrstedt, K. (2002). A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74--83.
- [Schilit, 1995] Schilit, B. N. (1995). *A Context-Aware System Architecture for Mobile Distributed Computing*. PhD thesis, Columbia University.
- [Schmidt, 2002] Schmidt, A. (2002). *Ubiquitous Computing-Computing in Context*. PhD thesis, Lancaster University.
- [Schmidt et al., 1999a] Schmidt, A., Aidoo, K. A., Takaluoma, A., Tuomela, U., Laerhoven, K. V., and de Velde, W. V. (1999a). Advanced interaction in context. In *1st International Symposium on Handheld and Ubiquitous Computing (HUC '99)*, pages 89--101.
- [Schmidt et al., 1999b] Schmidt, A., Beigl, M., and Gellersen., H. W. (1999b). There is more to context than location. *Computers and Graphics*, 23(6):893--902.
- [Schneider, 2007] Schneider, M. (2007). Towards a general object memory. In *1st International Workshop on Design and Integration Principles for Smart Objects (DIPSO 2007) at 9th International Conference on Ubiquitous Computing (UbiComp 2007)*.

- [Siegemund, 2004] Siegemund, F. (2004). A context-aware communication platform for smart objects. In *Second International Conference on Pervasive Computing (Pervasive 2004)*, pages 69--86.
- [Siio et al., 2003] Siio, I., Rowan, J., Mima, N., and Mynatt, E. D. (2003). Digital decor: Augmented everyday thing. In *Graphics Interface 2003*,, pages 159--166.
- [Sousa and Garlan, 2002] Sousa, J. P. and Garlan, D. (2002). Aura: an architectural framework for user mobility in ubiquitous computing environments. In *3rd Working IEEE/IFIP Conference on Software Architecture*, pages 29--43.
- [Streitz et al., 1998] Streitz, N., Geißler, J., and Holmer, T. (1998). Roomware for cooperative buildings: Integrated design of architectural spaces and information spaces. In *1st International Work- shop on Cooperative Buildings (CoBuild'98)*, pages 4--21.
- [Streitz et al., 2005] Streitz, N. A., Rucker, C., Prante, T., van Alphen, D., Stenzel, R., and Magerkurth, C. (2005). Designing smart artifacts for smart environments. *IEEE Computer*, 38(3):41--49s.
- [Strohbach et al., 2004] Strohbach, M., Gellersen, H. W., Kortuem, G., and Kray, C. (2004). Cooperative artefacts: Assessing real world situations with embedded technology. In *6th International Conference on Ubiquitous Computing (UbiComp 2004)*, pages 250--267.
- [Suutala et al., 2004] Suutala, E., Pirttikangas, S., Riekk, J., and Rönin, J. (2004). Reject-optional lvq-based two-level classifier to improve reliability in footstep identification. In *Second International Conference on Pervasive Computing (Pervasive 2004)*, pages 182--187.
- [Tandler, 2003] Tandler, P. (2003). The beach application model and software framework for synchronous collaboration in ubiquitous computing environments. *Journal of Systems and Software*, 69(3):267 -- 296.
- [Terrenghi et al., 2008] Terrenghi, L., Sellen, A., Patel, D., Marquardt, N., Harper, R., and Molloy, M. (2008). *The Time-Mill: An Interactive Mirror for Evoking Reflective Experiences in the Home*.
- [Tokuda et al., 2004] Tokuda, H., Takashio, K., Nakazawa, J., Matsumiya, K., Ito, M., and Saito, M. (2004). Sf2: Smart furniture for creating ubiquitous applications. In *International Workshop on Cyberspace Technologies and Societies at 2004 International Symposium on Applications and the Internet (SAINT 2004)*, pages 423-- 429.
- [Velde, 1997] Velde, W. V. D. (1997). Co-habited mixed reality. In *Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*.
- [Waldo, 1999] Waldo, J. (July, 1999). The jini architecture for network-centric computing. *Communication of the ACM*, 42(7):76--82.

- [Want et al., 1999] Want, R., Fishkin, K. O., Gujar, A., and Harrison, B. (1999). Bridging physical and virtual worlds with electronic tags. In *ACM Conference on Human Factors in Computing Systems (CHI 99)*, pages 370 -- 377.
- [Want et al., 1992] Want, R., Hopper, A., Falcao, V., and Gibbons, J. (1992). The active badge location system. *ACM Transactions on Information Systems*, 10:91--102.
- [Weiser, 1991] Weiser, M. (1991). The computer for the 21st century. *Scientific American*, pages 66--75.
- [Weiser, 1993] Weiser, M. (1993). Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):75 -- 84.
- [Weiser and Brown, 1997] Weiser, M. and Brown, J. S. (1997). The coming age of calm technology. *Beyond Calculation: The Next Fifty Years of Computing*, pages 75--85.
- [Winograd, 2001] Winograd, T. (2001). Architecture for context. *Human-Computer Interaction*, 16:401--419.
- [Yamabe et al., 2005] Yamabe, T., Takagi, A., and Nakajima, T. (2005). Citron: A context information acquisition framework for personal devices. In *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2005)*, pages 489--495.



Copy Right © 2009 by Fahim Kawsar

February 2009
Tokyo, Japan
