Noname manuscript No.
(will be inserted by the editor)

Design and Implementation of a Framework for Building Distributed Smart Object Systems

Fahim Kawsar · Tatsuo Nakajima · Jong Hyuk Park · Sang-Soo Yeo

Received: date / Accepted: date

Abstract A smart object system encompasses the synergy between computationally augmented everyday objects and external applications. This paper presents a software framework for building smart object systems following a declarative programming approach centered around custom written documents that glue the smart objects together. More specifically, in the proposed framework 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. There are three primary advantages of our approach- firstly, it allows developers to write applications in a generic way without prior knowledge of the smart objects that could be used by the applications. Secondly, smart object management (locating, accessing, etc.) issues are completely handled by the infrastructure thus application development becomes rapid and simple. Finally, the programming abstraction used in the framework allows extension of functionalities of smart objects and applications

Fahim Kawsar
Computing Department, Lancaster University.
Lancaster, LA1 4WA, UK
E-mail: fahim.kawsar@comp.lancs.ac.uk

Tatsuo Nakajima
Department of Computer Science, Waseda University.
3-4-1 Okubo, Shinjuku-Ku, Tokyo, Japan.
E-mail: tatsuo@dcl.info.waseda.ac.jp

Jong Hyuk Park
Department of Computer Science and Engineering, Seoul National University of Technology.
172 Gongreung 2-dong, Nowon-gu, Seoul, 139-742, South Korea.
E-mail: parkjonghyuk1@hotmail.com

Sang-Soo Yeo
Division of Computer Engineering, Mokwon University.
Daejeon 302-318, South Korea
E-mail: ssyeo@msn.com

very easily. We describe an implemented prototype of our framework and show examples of its use in a real life scenario to illustrate its feasibility.

Keywords Smart Object · Middleware · Pervasive Systems

1 Introduction

One of the consequences of pervasive technologies (e.g., miniaturization of the computer technologies and proliferation of wireless internet, short-range radio connectivity, etc.) is the integration of processors and tiny sensors into everyday objects. This revolutionized our perception of computing. We are in an era, where we communicate directly with our belongings, e.g., watches, umbrella, clothes, furniture or shoes and they can also intercommunicate. These everyday objects are designed to provide supplementary services beyond the primary purpose, an initiative that has been denoted as *Smart Object*[1] computing. It has drawn significant attention from the research community, primarily because of its promising potential in various industries e.g., supply chain management, medicine, environment monitoring, entertainment, smart spaces, etc.

In this paper, we look at the system issues for building a framework for smart objects. In particular we discuss how can we build smart object systems with suitable infrastructure to meet the dynamic and fluid nature of pervasive environment. Specifically, we focus on two issues 1) a suitable smart object architecture for representing smart objects and an application model to leverage the services of smart objects dynamically and 2) an infrastructure that supports such interaction while taking care of component (smart objects in this case) management issues away from the applications. The basic idea of our framework is to follow a declarative approach by using documents to externalize an application's requirements in a generic way without considering smart object management issues. Similarly, smart objects' services are externalized by structured documents. The runtime infrastructure provides a semantic association between the applications and smart objects using structural type matching of these documents. Because of such loose coupling, applications and smart objects can be built and extended orthogonally.

In section 2, we present the background and design issues of our framework and place our work against the state-of-the art. After that proposed design detail and implementation of the framework are described in section 3 and section 4 respectively. Then in section 5, we present the quantitative and qualitative evaluations of the framework through multiple application scenarios. Then we discuss some generic issues in section 6 before concluding the paper.

2 Background and Design Issues

Typically in a smart object system, context-aware applications run atop distributed smart objects embedded with awareness technologies (sensors, actua-

[1] In this article, smart objects and augmented artefacts carry similar meaning and will be used interchangeably.

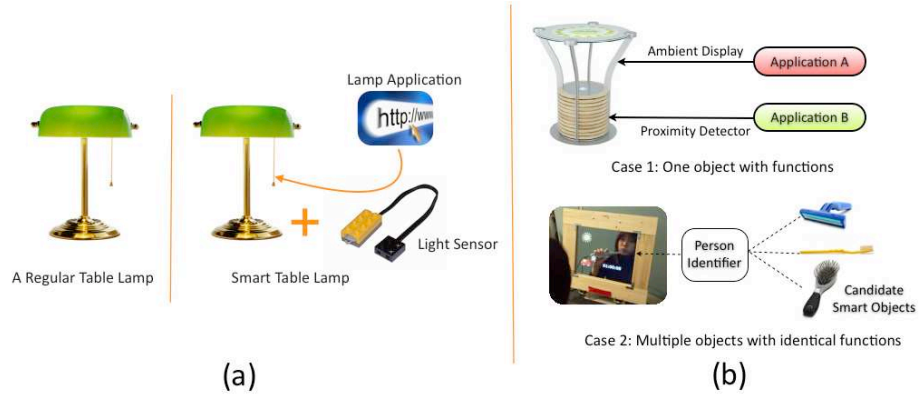

Fig. 1: (a) Any suitable application can be added to a compatible smart object (table lamp in this case) at any time; Smart object's features can be extended by adding new peripherals. (b) Augmentation variation of smart objects, a single smart object with multiple functions and multiple smart objects with identical functions.

tors 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). 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 (a)). This scenario highlights two primary design challenges of building smart object systems:

1. **Independent Application Development:** Applications for smart objects need to be developed independently without considering which smart object from which manufacturer will be used in the application. 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. We can not expect 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. **Temporal Evolution of Smart Object System:** Unlike conventional distributed component systems where applications typically resides in the digital world along with their components, 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. In the scenario presented above, the table lamp's functions is extended over time. In a smart object system, such functional extensions can either be supported by upgrading the application or upgrading the object (or introducing new objects). Thus it is important to construct smart objects and applications in a way that allow developers (and ideally the end-users) to extend smart object systems' functionalities in an incremental fashion.

Another design aspect is the augmentation modalities of smart objects. Augmentations depend on the designer's intuition and it is hard to confine the augmentation scope. Consider Figure 1-(b), depicting two ideal situations; in case 1 we have a smart table providing two supplementary functions: an ambient display and a proximity detector. In case 2 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 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.

Considering smart everyday objects are distributed in a physical space, the development philosophies of a smart object system seem similar to the philosophies of encapsulation and reuse behind component based frameworks, e.g., Component Object Model (COM) [**?**], Java Beans [**?**] as well as more network friendly descendants like DCOM, Jini/EJB [**?**], OMG's CORBA [**?**]. However, these systems put significant development complexities due to the challenges of smart object systems mentioned above. For example, the challenge of application level component management is typically handled by existing frameworks using interface standardization. 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 (locating/spawnning/etc.). If the application functionlity 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. Because of such strict dependency, it is difficult to write applications that can run on any semantically compatible smart objects without prior negotiation and to extend both the applications and smart objects.

Some researchers have used mobile code approach, to dynamically download the heterogenous component interfaces at application ends [**?**,**?**]. In SpeakEasy

[?] mobile codes (typed data streams and services) are exchanged among heterogeneous devices to create an interoperable environment. However such approaches are impractical considering, for every new smart object an application encounters, it would need to download new codes, even for components that are semantically same, e.g., two identical chair from two different manufacturer. On a more lower granularity level, UPnP[2] defines a standard set of protocols for specific device types (e.g., audio/video devices) for interoperability. Jini describes devices using interface description and language APIs allowing applications to utilize those interfaces where as UPnP attempts to standardize protocols to allow devices to intercommunicate seamlessly. 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. Furthermore, these systems provide little support for extending applications or appliance services. For example, it is hard to add features in an existing smart object and using that feature immediately in the application with these infrastructures. Patch Panel [?] is a programming tool that provides a generic set of mechanisms for translating incoming events to outgoing events using EventHeap [?] communication platform. It allows new applications to leverage the services of existing components. This tool along with its visual programming environment have been used for prototyping ubiquitous environment [?] and mobile user interface [?]. Our overall approach is close to Patch Panel as we seek to support incremental integration. However, we exploit a distributed state model over generalized documents that enable incremental addition of features to both artefacts and applications. Also, our declarative approach enables developer to build applications in a generic fashion without requiring to know the target environment ahead of the execution. InterPlay [?] 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. Although our approach is very close to InterPlay as we employ similar mapping of tasks to device services, our challenge is to provide generic abstractions and to support incremental extension and deployment of both artefacts and applications. Our smart object model (described later in the paper) is a major leap from InterPlay which signifies our contribution.

A range of middlewares for pervasive systems [?,?,?,?,?] specify their application development processes strictly. These middlewares usually provide end-to-end API centric support for the application developer, i.e., smart objects are encapsulated into wrappers and an array of APIs is provided to the applications to manipulate them. The problem of this approach is that the applications and smart objects become virtually incompatible in other environments. For instance, Gaia is [?] meta operating system and its design philosophy is centered around the concept of Active Space that is capable of knowing all the avaialbe resources and providing services to users in a contextual manner. Although, Gaia has covered a lot of design requirements of

[2] Universal Plug and Play - http://www.upnp.org

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. Context Toolkit [**?**] 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. 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 [**?**]. Environment services can be dynamically loaded based on task requirement and can be adapted to dyncamic changes e.g., sudden badwidth drop. 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. JCAF [**?**] 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. 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.

In summary, existing 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 the next section, a solution framework that addresses smart object specific features is introduced with explanations of its design rationales.

3 A Document Based Solution Framework

The characteristics of smart object systems raise a number of important questions - how will we build pervasive applications to manipulate smart objects with no prior knowledge? how will we manage those unknown smart objects? how will we build and extend application and smart objects for such dynamic environment? 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 access smart object services regardless of their types. From an implementation perspective, this can be achieved by applying the principle of reflection, i.e., allowing a program to access, reason and alter its own interpretation [**?**,**?**]. Application can access its components at runtime if it exposes its runtime requirement to an underlying middleware that provides unified access to its components. To do this, there must be an agreement between the participating entities, i.e., application, smart objects and the middleware. Documents encoded in XML and related technology can be successfully used to model this information. In the past, we have seen that XML based simple protocol and data format are successfully used for connecting arbitrary devices. XWeb [**?**] is an excellent example of such systems where 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 each other. From an abstract point of view, XWeb can be seen as a logical extension of HTTP for devices.

Building on these works, we have taken document based development framework in this work. 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. Similarly, a smart object is forced to expose its service specifications via documents. A secondary infrastructure then connects the application to the smart objects by matching these 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 software architecture. 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.

The design challenge imposed by the augmentation variation of smart objects is handled by following a core-cloud development model for smart objects

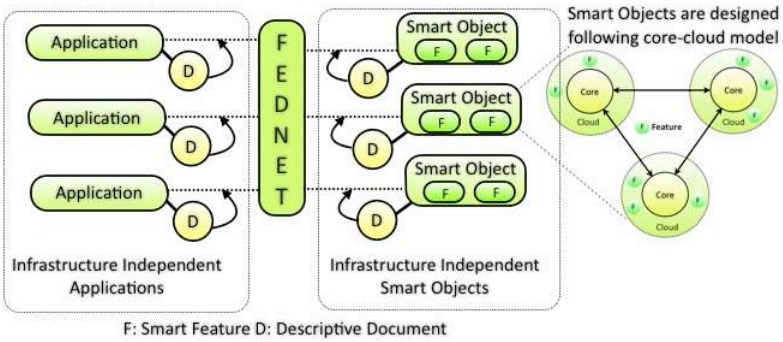

Fig. 2: A Conceptual Document based Framework

in the framework. 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 applicatios/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 allow 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 temporal evolution requirements for smart object systems.

The conceptual document based framework is depicted in Figure **??**. 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 in one or multiple ***Profiles*** (cloud) atop a runtime (core) and allows additions of profiles incrementally. Applications runtime specifications are exposed 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 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 develop application integrating smart objects without concerning interfaces and the management of smart objects.
- To extend both applications and smart objects.

In the next section a prototype implementation of this framework is discussed.

4 Framework Implementation

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

4.1 Smart Object Wrapper

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. Each augmented feature is called a service *Profile* in our approach. 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.

4.1.1 Internal Architecture of Smart Object Wrapper

The internal architecture of the smart object wraper is shown in Figure 3 and consists of the following:

1. **The Core:** It 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 generic binary and runs independently. Inside the core the communication module facilitates communication support and encapsulates the transport layer. In the current implementation, primarily IEEE 802.11x (TCP/IP) and Bluetooth (RFCOMM) are supported in this module. The discovery module allows service advertisement. The notification module enables the rest of the modules to indicate their status. The static memory contains property data, profile descriptions, and other temporal data. The client handler is the request broker for services and delegates the external requests to specific profiles. Finally, the profile repository hosts the array of profiles. The profile repository has dynamic class loaders to load the profiles dynamically when requested.

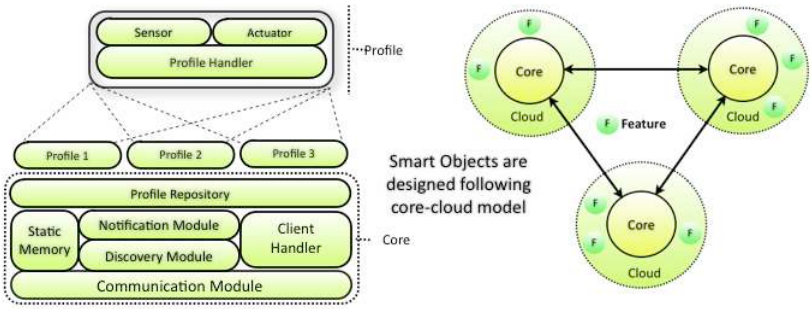

Fig. 3: Smart Object Wrapper Architecture

2. **Profile:** Each profile represents a specific functionality and implements the underlying logic of the functions, e.g., providing context by analyzing the attached sensors' data or actuating an action by changing the smart objects' states. Each profile is either sensor type or actuator type which is abstracted by a generic Profile Handler component. Each profile is packaged as generic binary and linked with the core dynamically.

4.1.2 Programming Model

In the current work, a ***profile*** based programming abstraction is offered to the developers and building smart objects involve two-step development effort:

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. Figure 4 shows the partial implementation of the proximity profile of the smart mirror. 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

```
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. }
```

Fig. 4: Sample Code implementing Sensor Type Profile

context information the ***setSML()***(line 11) function is invoked to generate profile output in framework specific way.

4.1.3 Documents to Represent Smart Objects

In the proposed framework the specifications of profiles are objectified in external documents that are used by the underlying infrastructure to allow discovery and interaction of external applications and other smart objects with the object in context. In addition, documents are also used by the core of a smart object to load its profiles dynamically. 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 5(a). 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 to discover the services of the smart objects and associate smart objects with applications.

2. **Profile Description Document (PDD):** Each smart object contains one or multiple profiles and usually with 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 move-

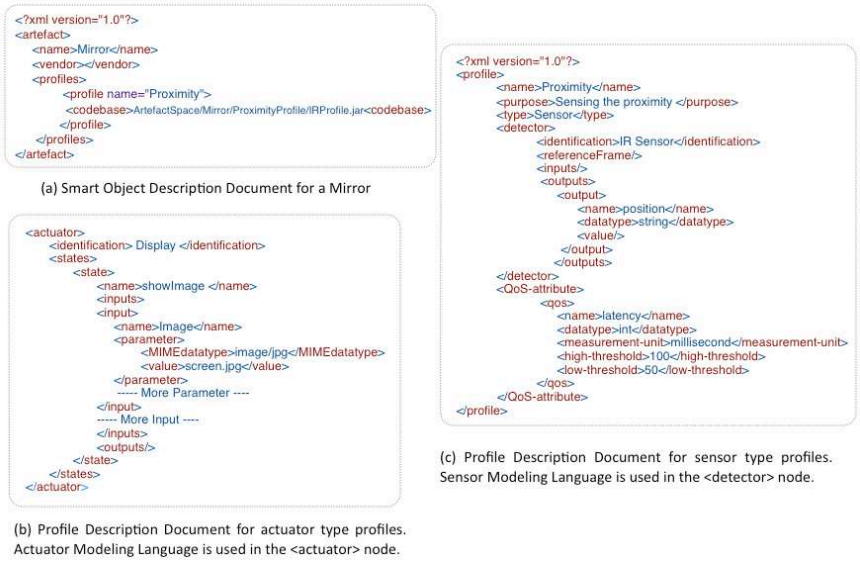

Fig. 5: Documents to represent Smart Objects

able 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. The sensor type profile's description follows the specification of the customized Sensor Modeling Language (SML) adopted from SensorML[3]. 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. For an actuator profile custom designed *Actuator Modeling Language (AML)* is used as shown in Figure 5(b). 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.

4.2 Application Development Process

Typical applications that run on smart objects are 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 and provides access and

[3] OpenGIS Sensor Modeling Language Specification: http://www.opengeospatial.org

management of smart objects. In the proposed document-based framework these recurrent tasks are supported through indirection using documents, i.e., application specifies its requirements in a high level structured document without considering how to attain those requirements. However, there is a 3-Step development process that an application developer has to follow in the current framework context.

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. To enable the application to interact with the underlying environments, however, a developer must ensure the followings:

1. ***Step 1 - Structuring Applications by Functional Tasks:*** The first step of the application development 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 6 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 provide a spontaneous federation. As we see in the Figure 6, 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. Applications tasks can also describe the required communication mode, i.e., synchronous (poll) and asynchronous (subscription) for aggregating context data from smart objects.
3. ***Step 3 - Accessing an Access Point in a RESTful Manner:*** The final step in application development is enabling an application to interact (discovery, access, etc.) with smart objects in a unified manner. In the current framework the discovery and management processes are eliminated from the application scope. FedNet performs the discovery on behalf of an application by utilizing its task description document, and allows application to access the smart object services by assigning a common point of interaction regardless of underlying smart objects' types and protocol heterogeneity. The identity of this common access point is injected into the meta data block of the Task Description Document as shown in Figure 6 during application deployment. When the application is launched required physical smart objects data semantics (*detector* and *actuator* nodes of the Profile Description Document) are provided to the application from this access point, 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

```
<?xml version="1.0" encoding="UTF-8"?>
<application>
        <name>Smart Display Application</name>
        <app-purpose>Providing Personalized Infromation 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>
```

Fig. 6: Task Description Document (partly)for a Smart Display Application

access point are based on HTTP and XML following the Representational State Transfer (REST) approach [**?**].

Every application in the current framework is disseminated as a generic binary. Application uses 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 6, the ***binaryPath*** node specifies the path of application binary.

4.3 FedNet Runtime Infrastructure

Earlier sections showed that both the applications and smart objects are infrastructure independent and expressed in high level descriptive documents (i.e., task and profile specifications). FedNet provides the runtime association among them by utilizing only the documents of these applications and smart objects. FedNet itself is packaged in a generic binary and composed of four components as shown in Figure 7.

1. **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.
2. **Application Repository** hosts all the applications that run on FedNet environment. During an application's deployment, the binary executable

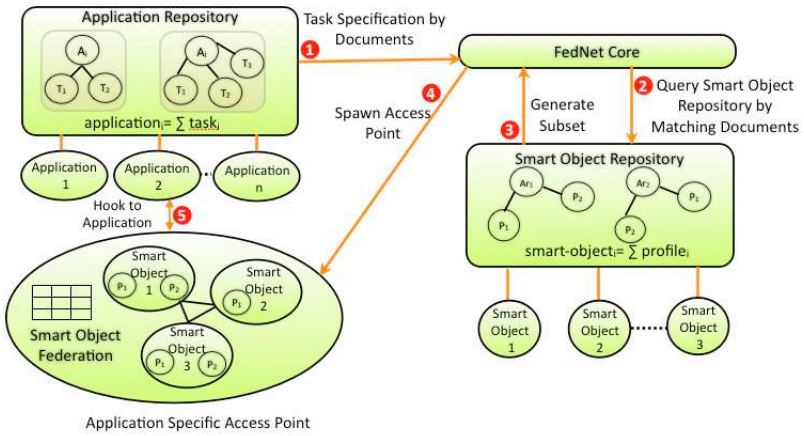

Fig. 7: Internal Architecture of FedNet

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.

3. **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 is performed by using XQuery[4]. When an application is launched, the corresponding 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. **Access Point** represents the runtime 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.).

[4] http://www.w3.org/TR/xquery/

This means multiple federations of smart objects can co-exist in the environment. Simultaneously, each smart object can participate in multiple federations. An 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.

5 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 section 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.

5.1 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.

5.1.1 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 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.

5.1.2 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 [**?**] which in turn is connected to a Gumstix[5] 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.
- *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 [**?**] that is connected to a Gumstix platform over Bluetooth.

[5] http://www.gumstix.com

- *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 CyberLink[6] implementation of UPnP.
- *Bluetooth Headphone:* A regular bluetooth headphone with Advanced Audio Distribution Profile (A2DP)[7] 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: 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.

5.1.3 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.

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

[6] http://cgupnpjava.sourceforge.net/
[7] http://www.a2dp.info

add any overhead to the overall system. Figure 8(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 8(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.

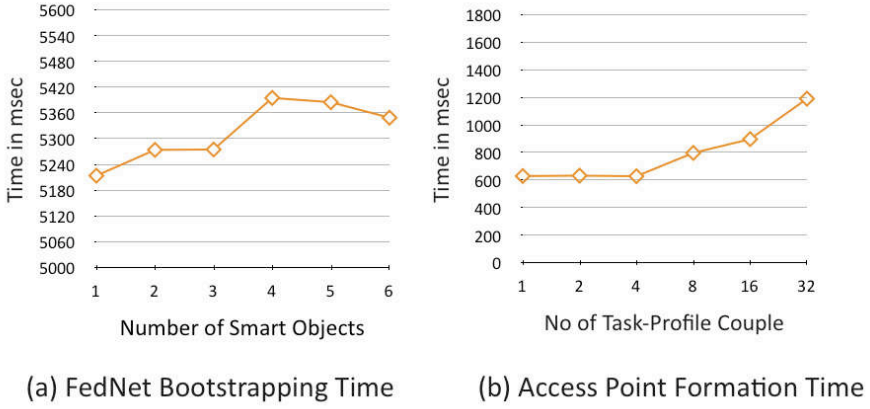

Fig. 8: Bootstrapping and Access Point Formation Time

Realtime Access Point Formation Time: The next concern is the Access Point formation time by FedNet for specific application. Recall from section 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 8 (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.

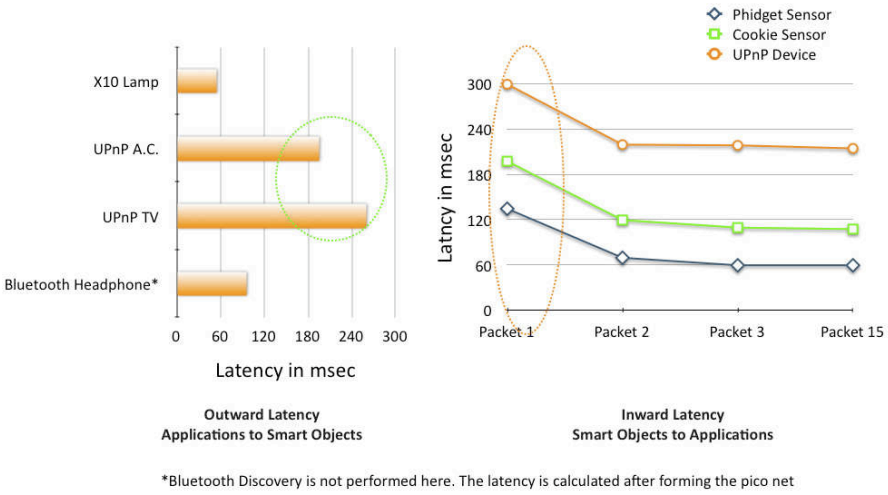

Fig. 9: Communication Latency

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 9 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.

5.1.4 Summary of the Quantitative Evaluation

In the above section, the runtime performances 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.

6 Qualitative Evaluation of the Framework

In the earlier sections we have claimed that the proposed framework enables i) building smart objects and applications independently, ii) shifting smart object management issues away from the applications and iii) extending the functional features of smart object systems over time. To validate these claims, we have evaluated our approach following the guidelines of Edwards et al. [**?**] by building effective prototypes that expose and evaluate the prime features of a pervasive computing infrastructure. Over the years we have developed several smart object systems that include multiple smart objects, profiles and applications [**?**,**?**,**?**]. These applications are re-developed and extended following the proposed approach during the course of this work. In this section, we present *proof-of-concept* one of those smart object systems that is very practical, and useful.

We constructed a smart mirror by augmenting a regular laptop with acrylic magic mirror (Figure 10(a)). Initially this mirror has a display profile. We wrote an application for this mirror where the application can show some personalized information (e.g. weather, stock quote, movie listing etc.) into the mirror display (Figure 10(c)) [**?**] utilizing its display profile. However, this application can proactively show information only when someone is in-fornt of the mirror. But for such proactivity it requires a proximity profile. To enable

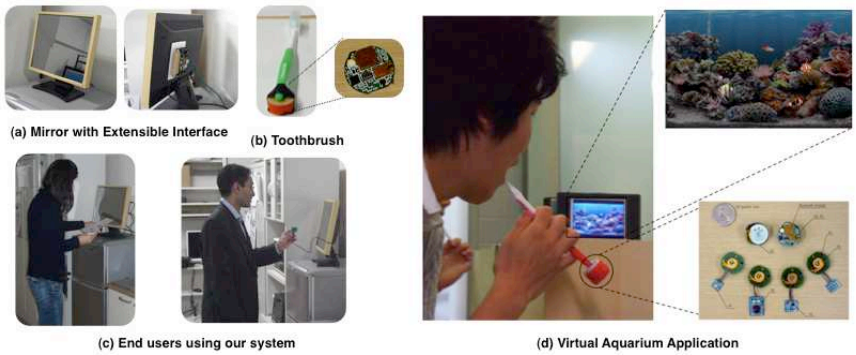

Fig. 10: A Prototype Smart Object System

this application feature, later we have added a proximity profile to this mirror by attaching an Infra-red sensor. This improved the applications interactivity. Afterwards we built a completely separate application for the mirror where user's dental hygiene is reported in a persuasive way utilizing the metaphor of a clean and dirty aquarium (Figure 10(d)) [**?**]. We replaced the previous application running on the mirror with the new one. This new application requires a smart toothbrush that can detect its state-of-use (Figure 10(b)). We constructed the smart toothbrush by attaching a wireless accelerometer sensor and deployed it in our environment with corresponding profile. Thus the mirror shows an aquarium reflecting users brushing practice whenever the user brushes his teeth in front of the mirror.

Both the applications were built independently and deployed with corresponding documents expressing the tasks, similarly the two smart objects e.g., the toothbrush and the mirror were built independently and deployed with corresponding documents. FedNet provided the runtime association among them thus freeing application from smart object management. Furthermore, this scenario highlighted the service extension feature of our approach. We have added new profiles to an existing smart mirror allowing an existing application to leverage new functionalities. Importantly, the application did not have to take into account the heterogeneity issues introduced by the addition of an Infra-red sensor as it was handled by the proximity profile implementation. Similarly, the installation of the second application into the mirror enabled the mirror to play different role by co-operating with the toothbrush, which was seamlessly integrated into the setup.

7 Discussion and Conclusion

There are primarily two abstractions underneath the documents that we have utilized in our framework. From the 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 plugin architecture, a smart object service can be extended anytime by adding new sensors or actuators and attaching corresponding profile into the smart objects core. Also, if a specific service needs to be updated only the corresponding profile need to be replaced, not the entire smart object or the applications utilizing them. Furthermore, a profile may provide services in various granularities thus supporting multiple applications requiring services at different scale (i.e., some applications may ignore some service features). 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. However, by profile abstraction, we are not trying to define the ontology for profiles. Instead we are providing a structure that designers can use to disseminate their implemented ontology and glue it with rest of the infrastructure.

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. 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 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 our approach, such flexibilities are provided elegantly by only expressing applications' task specifications in documents and ignoring smart object management issues at the application level. FedNet provides the appropriate mapping of these documents with smart objects documents expressing their services.

This disassociation of applications from the smart objects they reference is identical to the Model-View-Controller (MVC) architecture from Smalltalk. 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 our middleware act as the glue that associates smart objects services to applications that manipulate the services. Such separation of concern (i.e., both the applications are artefacts are independent of FedNet and come as ready-to-run binary), and data centric approach also enable us to provide additional services orthogonally in our system. For example, we have implemented several end user tools atop FedNet that enable end users to deploy, configure and manage the applications and smart objects running in the FedNet environment.

In this paper we have presented a document based approach to build smart object systems. Applications' requirements and smart objects' services are externalized using structured documents utilizing *Task* and *Profile* abstractions respectively. A runtime framework FedNet provides the dynamic association by structural type matching. The contributions of our approach are two-fold: firstly, it allows developers to write applications in a generic way regardless of the constraints of the target environment utilizing the abstractions that are realized through documents Secondly, it allows extension of functionalities

of smart objects and applications very easily. We have described an implemented prototype of our approach with an application scenario that highlight the power and flexibility of our framework. We consider our approach is very useful for the ubiquitous computing domain, particularly one that involves smart objects.

References

1. R. Ballagas, F. Memon, R. Reiners, and J. Borchers. istuff mobile: Rapidly prototyping new mobile phone interfaces for ubiquitous computing. In *Conference on Human Factors in Computing Systems (CHI 2007)*, pages 1107–1116, 2007.
2. R. Ballagas, M. Ringel, M. Stone, and J. Borchers. istuff: A physical user interface toolkit for ubiquitous computing environments. In *Conference on Human factors in computing systems (CHI 2003)*, pages 537–544, 2003.
3. R. Ballagas, A. Szybalski, and A. Fox. Patch panel: Enabling control-flow interoperability in ubicomp environments. In *2nd IEEE Annual Conference on Pervasive Computing and Communications (PerCom 2004)*, pages 241– 252, 2004.
4. J. E. Bardram. 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, 2005.
5. L. Capra, W. Emmerich, and C. Mascolo. Exploiting reflection and metadata to build mobile computing middleware. In *Workshop on Middleware for Mobile Computing. Co-located with Middleware 2001*, Heidelberg, Germany, Nov. 2001.
6. A. K. Dey, G. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2-4):97–166, 2001.
7. W. K. Edwards, V. Bellotti, A. K. Dey, and M. W. Newman. 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)*, 2003.
8. W. K. Edwards, M. Newman, J. Sedivy, T. Smith, and S. Izadi. Challenge: Recombinant computing and the speakeasy approach. In *8th Annual International Conference on Mobile Computing and Networking (MobiCom 2002)*, 2002.
9. F. Eliassen, A. Andersen, G. Blair, F. Costa, G. Coulson, V. G. O. Hansen, T. Kristensen, T. Plagemann, H. Rafaelsen, K. Saikoski, and W. Yu. Next generation middleware: Requirements, architecture and prototypes. In *7th IEEE Workshop on Future Trends in Distributed Computing Systems*, 1999.
10. R. Englander. *Developing Java Beans*. O'Reilly and Associates, 1997.
11. R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.
12. K. Fujinami, F. Kawsar, and T. Nakajima. Awaremirror: A personalized display using a mirror. In *3rd Third International Conference on Pervasive Computing (Pervasive 2005)*, pages 315–332, 2005.
13. S. Greenberg and C. Fitchett. Phidgets: Easy development of physical interfaces through physical widgets. *14th Annual ACM Symposium on User Interface Software and Technology (UIST 2001)*, pages 209 – 218, 2001.
14. K. Hanaoka, A. Takagi, and T. Nakajima. 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, 2006.
15. D. Iseminger. *Com+ Developer's Reference*. Microsoft Press, 2000.
16. B. Johanson, A. Fox, and T. Winograd. The interactive workspaces project: experiences with ubiquitous computing rooms. *IEEE Pervasive Computing*, 1(2):67 – 74, 2002.
17. F. Kawsar, K. Fujinami, and T. Nakajima. 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)*, 2005.

18. F. Kawsar, K. Fujinami, and T. Nakajima. Experiences with building intelligent environment through sentient artefacts. In *3rd IET International Conference on Intelligent Environments (IE'07)*, 2007.
19. F. Kawsar and T. Nakajima. Persona: A portable tool for augmenting proactive applications with multi-modal personalization support. In *6th International Conference on Mobile and Ubiquitous Multimedia (MUM 2007)*, 2007.
20. A. Messer, A. Kunjithapatham, M. Sheshagiri, H. Song, P. Kumar, P. Nguyen, and K. H. Yi. 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, 2006.
21. T. J. Mowbray and R. Zahavi. *The Essential Corba: System Integration Using Distributed Objects*. John Wiley and Sons, 1995.
22. T. Nakajima, V. Lehdonvirta, E. Tokunaga, and H. Kimura. Reflecting human behavior to motivate desirable lifestyle. In *The Conference on Designing Interactive Systems (DIS 2008)*, pages 405–414, 2008.
23. J. Nakazawa, W. K. Edwards, U. Ramachandran, and H. Tokuda. A bridging framework for universal interoperability in pervasive systems. In *The 26th International Conference on Distributed Computing Systems (IEEE ICDCS 2006)*, 2006.
24. D. Olsen, T. Nielsen, and D. Parslow. Join and capture: a model for nomadic interaction. In *4th annual ACM symposium on User interface software and technology (UIST 2001)*, pages 131–140, 2001.
25. M. Roman, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74–83, 2002.
26. J. P. Sousa and D. Garlan. Aura: an architectural framework for user mobility in ubiquitous computing environments. In *3rd Working IEEE/IFIP Conference on Software Architecture*, pages 29–43, 2002.
27. J. Waldo. The jini architecture for network-centric computing. *Communication of the ACM*, 42(7):76–82, July, 1999.

Authors

Fahim Kawsar is a PostDoc in the Computing Department of Lancaster University,UK. He received his Ph.D. and M. Engg. at the Distributed Computing Lab of Waseda University in 2009 and 2006 respectively. His research interests evolve around ubiquitous computing with specific interest in smart object systems, human-centric system infrastructures and tangible interfaces. He has published in the areas of distributed middleware, smart objects, personalization, and physical interfaces. He was a recipient of 2006-08 Microsoft Research (Asia) fellowship.

Tatsuo Nakajima is a professor in Department of Computer Science and Engineering of Waseda University. His research topics are operating systems, distributed systems, real-time systems, ubiquitous computing, and interaction design. Currently, he is is leading two projects: an operating system for future multicore based information appliances and persuasive technologies for motivating desirable lifestyle.

Jong Hyuk Park received his Ph.D. degree in the Graduate School of Information Security from Korea University, Korea. He is now a professor at the Department of Computer Science and Engineering, Seoul National University of Technology, Korea. He has published about 100 research papers in international journals and conferences. He has been serving as chairs, program committee, or organizing committee chair for many international conferences and workshops. He was editor-in-chief of the International Journal of Multimedia and Ubiquitous Engineering (IJMUE), the managing editor of the International Journal of Smart Home (IJSH). He is Associate Editor / Editor of 14 international journals including 8 journals indexed by SCI(E). In addition, he has been serving as a Guest Editor for international journals by some publishers: Springer, Elsevier, John Wiley, Oxford Univ. press, Hindawi, Emerald, Inderscience. His research interests include security and digital forensics, ubiquitous and pervasive computing, context awareness, multimedia services, etc. He got the best paper award in ISA-08 conference, April, 2008.

Sang-Soo Yeo received his bachelor's, master's and Ph.D. degrees in Computer Science from Chung-Ang University, Seoul, Korea. He previously taught at Dankook University, Seoul, Korea. He has joined Kyushu University in Japan as a visiting scholar at the Graduate School of Information Science and Electrical Engineering (ISEE). And then he came back to Korea and he worked for BTWorks, Inc. as a General Manager and he was involved in Hannam University as a visiting professor at the same period. Now he is a professor at Division of Computer Engineering, Mokwon University, Korea. Dr. Yeo has been serving as Chairs for a number of conferences and workshops; MUE 2007, IPC-07, FBIT 2007, FGCN 2007, WPS 2008, SH'07, MUE 2008, ISA 2008, CSA 2008, UMC 2008, BSBT 2008, FGCN 2008, ASEA 2008, SecTech 2008, and ICUT 2009. He is an associate editor of the International Journal of Multimedia and Ubiquitous Engineering (IJMUE) and he has been served as a Guest Editor for the International Journal of Security and Its Applications (IJSIA), the International Journal of Smart Home (IJSH), and the International Journal of Autonomous and Adaptive Communications Systems (IJAACS). Dr. Yeo's research interests include Security, Ubiquitous Computing, Multimedia Service, Embedded System, and Bioinformatics.