A Document centric approach for supporting Incremental Deployment of Pervasive Applications

Fahim Kawsar, Tatsuo Nakajima
Department of Computer Science
Waseda University
Tokyo, Japan
{fahim,tatsuo}@dcl.info.waseda.ac.jp

Kaori Fujinami
Department of Computer, Information and Communication Sciences
Tokyo University of Agriculture and Technology
Tokyo, Japan
fujinami@cc.tuat.ac.jp

ABSTRACT

This paper explores system issues for enabling incremental deployment of pervasive application - the problem of how to deploy and gradually enhance the functionalities of applications in a pervasive environment. We present a system architecture, FedNet that provides the foundation for incremental deployment and uses a document centric approach utilizing a profile based artefact framework and a task based application framework. Our artefact framework represents an instrumented physical artefact as a collection of service profiles and expresses these services in generic documents. Pervasive applications are expressed as a collection of functional tasks (independent of the implementation) in a corresponding document. A runtime component provides the foundation for mapping these tasks to the corresponding service provider artefacts. This mapping is spontaneous and thus enables gradual addition of services. Primary advantages of our approach are twofold- firstly, it allows end users to deploy pervasive applications and instrumented artefacts easily and gradually. Secondly, it allows developers to write applications in a generic way regardless of the constraints of the target environment. We describe an implemented prototype of FedNet, and show examples of its use in a real life deployment by the end users to illustrate its feasibility.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures

Keywords

Augmented Artefact, Pervasive Application, Deployment

1. INTRODUCTION

We envision that with the proliferation of low-cost sensors, smart artefacts and spontaneous communication technologies pervasive applications will find a universal place in our everyday life. A pervasive application usually involves physical artefacts, i.e. sensors, augmented artefacts, mobile devices, displays, etc. Ideally these applications should be similar to home appliances, i.e. easy to setup, adaptive to users' needs, styles and interchangeable with new models. An essential property of our living space is its evolutionary nature and receptibility to continual change. 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 [16]. Edward and Grinter echoed that the networked home of the future will not be custom designed from the beginning rather it will emerge in a piecemeal fashion [6].To support the evolutionary nature of our homes it is essential that pervasive applications and instrumented artefacts support the incremental deployment. A user can buy physical artefacts and pervasive applications and should be able to install these just like other home appliances. In addition, he/she can incrementally enhance an application's functionalities by purchasing new artefacts or upgrading the artefacts' functionalities, e.g. an application can provide a few basic services initially with the available artefacts, and can incrementally provide more richer services as new functionalities are added to the artefact.These observations raise two questions: i) How does one develop smart artefacts and pervasive applications, which can be deployed by the end users? and ii) How does an application adapt its functional behavior with incremental integration of physical artefacts providing richer services?

To address these concerns, we present a system, FedNet that provides the foundation for the incremental deployment of pervasive applications and physical artefacts. It uses a document centric approach utilizing a profile based artefact framework and a functional task-centric application framework. The artefact framework represents an instrumented artefact in a unified way by encapsulating its augmented functionalities in one or multiple service profiles atop a core (generic binary) and allows additional profiles to be plugged into the core incrementally. Such generality makes an artefact plug and play and allows end users to deploy it easily in a *Do-it-Yourself (DIY)* fashion. The artefact framework expresses these augmented functionalities of an artefact in descriptive documents that allows applications to interact with that artefact. Pervasive applications are represented as a collection of functional tasks (atomic actions) independent of their implementation and are exposed in a generic document . The task in our framework is an atomic action that requires an artefact's service, like *"sense current humidity", "turn on the lamp",* etc. Expressing application in this way allows developers to write applications without considering the target environments' constraints. Both the artefact framework and task-centric application framework are independent of any infrastructure, thus to create a spontaneous association among the artefacts and the applications at runtime, we utilize FedNet that maps the task specifications of the applications to the underlying artefacts functions by utilizing respective documents of the applications and the artefacts.

MobiQuitous 2008 July 21 - 25, 2008, Dublin, Ireland.

This spontaneity allows applications to leverage richer artefact services that are added gradually.

In the subsequent section we present the design decisions, followed by the technical detail of our approach. Then, we proceed to the feasibility of the proposed solution by illustrating a real life deployment experiment session. A smart mirror and a pervasive application for the mirror were deployed and incrementally enhanced by the end users using our system. Although, the end user experiment positively evaluated our system's support for the deployment activity, it revealed several usability problems. We report these findings along with the experiment descriptions. After which, we discuss some generic issues. Finally, we position our research with respect to the related work and conclude the paper.

2. DESIGN ISSUES

In this section, first we draw a scenario to illustrate the concepts of this paper. Then we elicit the design challenges and explain the design decisions to meet those challenges.

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. Furthermore, the application's information sources can be personalized if the user can be identified. For that the mirror needs to be augmented with an identification device, e.g. a finger print reader, etc. 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 a finger print reader and attached them to the mirror, so that she can interact with the application more closely and personally. Now, her mirror application is running in full fledged mode just as she liked.

2.1 Design Challenges and Decisions

The above scenario poses us with two fundamental design challenges. First, allowing end users to incrementally deploy the physical artefacts in a *Do-it-Yourself (DIY)* fashion without any complex configurations. Second, developing applications and physical artefacts in an independent manner so that applications can create a spontaneous federation at runtime and can leverage the richer services of the artefacts that are incrementally added. These two challenges lead us to the following design decisions.

1. **DIY Instrumented Artefacts:** Artefacts should be reusable and augmented features should not be tightly coupled with the scenario or the artefact itself. We need to abstract the augmented features in a way that can be applied to multiple artefacts and package the artefact in a generic binary so that the end users can deploy them. In addition, a common representation is needed to ensure that an artefact can participate in any environment. To meet these requirements, we adopted a profile based artefact framework that represents an artefact in a unified way using a generic binary with structured documents and allows plugging multiple profiles into an artefact. The profile is a single augmented functionality of an artefact. The separation of artefacts and profiles enables DIY support, i.e an artefact can be instrumented by a suitable profile in an ad-hoc manner. Because of this loose coupling and stand alone characteristics, our framework enables the incremental deployment support for end users, just like in the scenario: Alice can gradually enable the features of the application by adding multiple instruments to the mirror.

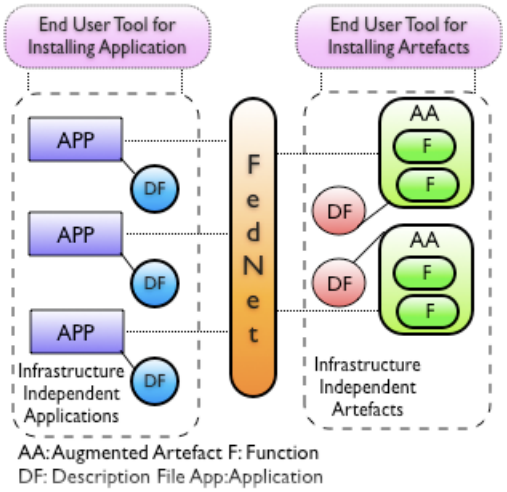

Figure 1: Basic workflow of our approach

2. **Infrastructure Independent Application:** Applications should be developed considering the functionalities only. To make an application independent of the Infrastructure, it is imperative to know an application's runtime requirement ahead of the execution. Furthermore, the application needs a generic access mechanism to interact with the environment. We have addressed these challenges by representing an application as a collection of functional tasks written in a task description file and allowing an application to access the artefact services using popular web techniques (SOAP for push and RSS Feed for pull). In our example scenario, the display application's runtime requirements were expressed in a document, which was utilized to enable its features when additional instruments were added to the mirror.

3. **Spontaneous Federation:** An intermediator is needed to create the runtime association among the applications and the artefacts, both of which are infrastructure independent. To form such a federation, it is essential to understand the semantics of the movable data ahead of the execution. In our approach, the infrastructure FedNet provides this intermediation. It analyzes the task description file to extract the service requirements and then maps these tasks to underlying service provider artefacts by matching artefact description files. FedNet then assigns a generic intermediation component to the application that allows the application to access the services of the artefacts. The spontaneous federation enables incremental integration of applications. In our example scenario, due to this, the application could use the added instruments whenever they are available. Figure 1 shows the basic workflow of our approach. An end user deployment tool works

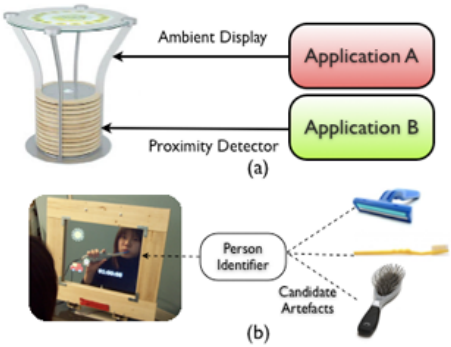

Figure 2: A single artefact with multiple roles and multiple artefacts with similar roles

atop FedNet that allows ordinary individuals to deploy and interact with instrumented artefacts and pervasive applications.

Next, we present the technical detail of our approach.

3. SYSTEM DETAIL

In this section, first we present the artefact framework and the task-centric application framework. Then, we will show how FedNet utilizes these frameworks to create a spontaneous federation.

3.1 Artefact Framework

Augmentations of physical artefacts depend on the designer's intuition and it is hard to confine the augmentation scope. Consider, Figure 2 depicting two ideal situations, a) a single everyday artefact capable of playing multiple functional roles and b) multiple artefacts sharing a similar functional role. In Figure 2(a) we have a smart table providing two supplementary functions: an ambient display and a proximity detector. In Figure 2(b) we have a mirror display [8] in a washroom which is triggered by any of the three augmented artefacts, e.g. a toothbrush, a comb or a razor. The suitable augmentation of these artefacts depends on the underpinned scenario, regardless of the multiple functionalities that can be afforded. Simultaneously, the characteristics of the application association with the artefacts require a new model for artefact presentation. Consider, Figure 3 where four different cases are shown. In case 1, artefacts are stand-alone providing a single or multiple built-in functions without any applications. Whereas in cases 2-4 three different modalities of application associations are shown. Although these latter cases are supported by existing middlewares [4, 9, 17] through the notion of device wrapper (these wrappers are tightly glued with the rest of the middleware), these middlewares have no clean support in case 1. Artefacts are inherently dependent on the middleware and can not run in a stand alone mode. Also, to use augmented artefacts in these middleware environments, applications are bound to follow the infrastructure semantics. Following these observations, we have adopted a profile based artefact framework in our system. Basic artefact functionalities are combined in a core component as a generic binary and additional augmented features can be added as plug-ins atop the core. Each augmented feature is called a profile in our approach. These profiles are artefact independent and represent a generic service. This design allows an artefact to act as a stand-alone artefact and to participate in an application scenario, thus supporting all four cases in Figure 3. Furthermore, the generic binary makes the artefact infrastructure independent enabling end users to deploy them spontaneously.

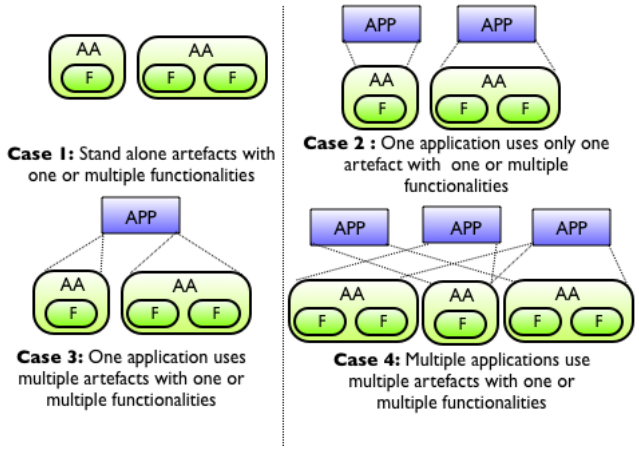

Figure 3: Different association cases between the artefacts and the applications

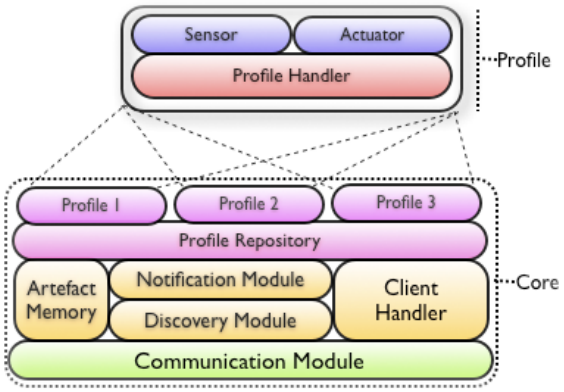

Figure 4: Architecture of Artefact Framework

3.1.1 Internal Architecture of Artefact Framework

The internal architecture of the artefact framework consists of the following (Figure 4):

1. **Core Component:** Typically instrumented artefacts have some common characteristics e.g. communication capable [2, 19], provides perceptual feedback [3], possesses memory etc. The core component of our artefact framework encapsulates all these functionalities in a generic binary. The communication module facilitates communication support and encapsulates the transport layer whereas the discovery module allows service advertisement. The notification module enables the rest of the modules to indicate their status. The artefact memory is a shared space that contains artefacts' property data, profile descriptions, and other temporal data. The client handler is the request broker for artefact services and delegates the external requests to specific profiles. Finally, the profile repository hosts the array of profiles. In the current prototype, the profile repository is implemented following a plug-in architecture. It has class loaders to load the artefact profiles dynamically when requested. The entire core is packaged in a generic binary thus runs independently and can respond to applications' requests. The profiles can be gradually added to this core.

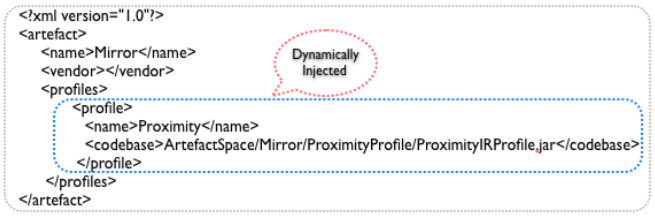

Figure 5: Artefact Description File for the Mirror (with *Proximity Profile*) of our example scenario.

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 artefacts' state (e.g., increasing the lamp brightness etc.). Each profile is a sensor or an actuator type and has a profile handler, a template to plug device code and context calculation or service actuation logic. Due to a variety of sensors and actuators, different artefacts are augmented with different sensors or actuators having different access and data semantics (like Smart-its [9], Mote[1], etc.) Since the same functionality can be achieved by different sensors and actuators, the profile handler has an abstraction layer that hides the heterogeneity of the underlying device platforms.

3.1.2 Documents to represent Artefacts

The artefact framework's core is packaged as a ready-to-run binary with a description document, Artefact Description File (ADF) as shown in Figure 5. This file contains the basic information about the artefacts, e.g., name, vendor, profile specifications etc. Whenever a new profile is attached to that artefact, the corresponding ADF is updated to reflect the added capabilities of the artefact. Profile is packaged as plug-ins that run atop the core and its services are expressed in a Profile Description File (as shown in Figure 6). This file specifies the data semantics of the corresponding profile. Each profile is a sensor or an actuator type. Therefore, the description file either contains a *detector* or an *actuator* node. The sensor type profile's description follows the specification of the Sensor Modeling Language (SensorML)[15] (Figure 6 (a)) and expresses profile's output (e.g., data format, parameters, etc.). The primary strengths of 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, our custom designed *Artefact Control Language* is used (Figure 6(b)) where the *state* attribute is used to abstract the operational states of the artefacts. This file specifies the required the input parameters and their data types to change artefact states. Each profile description also contains a quality of service(QoS) block which specifies the quality of the profile's service. Adding a profile to an existing artefact requires hardware attachment and installation of the plug-in implementing the profile atop the artefact core.

3.2 Task-Centric Application Framework

Usually pervasive middlewares [4, 17, 18] provide their own application model that the developers follow to utilize the environment resources. This dependency limits the deployability of the application. To create an infrastructure independent application that

[1]http://xbow.com/products/wirelesssensornetworks.htm

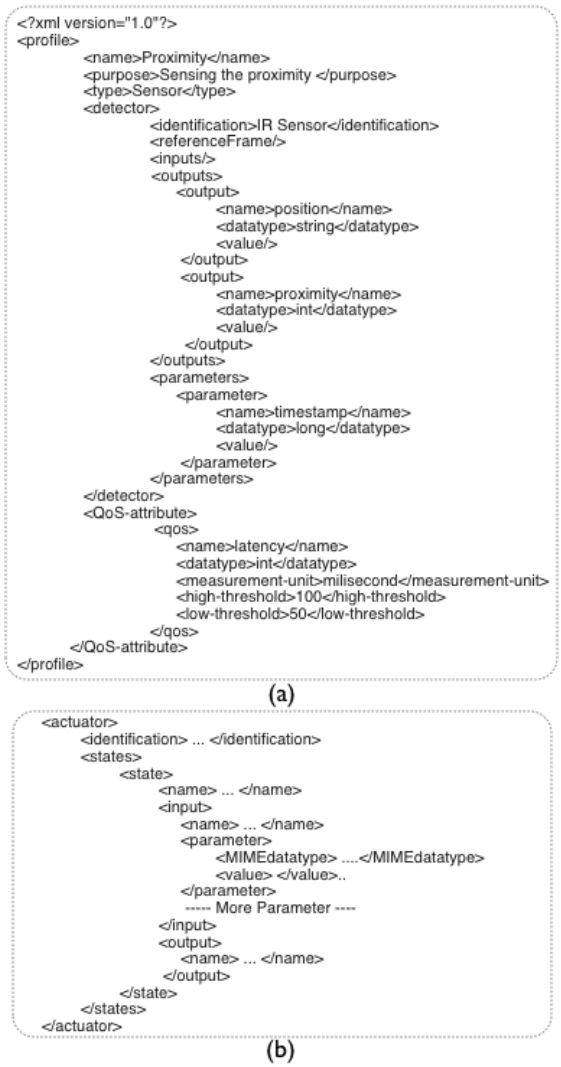

Figure 6: (a)Profile Description File for *Proximity Profile*. SensorML is used in the <detector> node. (b) Artefact Control Language is used for actuator profile, only the <detector> node is replaced with <actuator> node.

can still exploit the environment resources we utilize task specification of the applications. An application is expressed as a collection of functional tasks independent of the implementation and infrastructure. This specification allows the FedNet runtime to map the task to respective service provider artefacts. An application developer can follow any library and implementation language to code the execution logic of the application. The only two things necessary for an application to run in a FedNet environment are: i) expressing application's functional task list in a description file, and ii) utilizing an access point to manipulate the artefact services using generic web techniques.

Any application is composed of several functional tasks, e.g. atomic actions. In pervasive applications, these atomic actions may be: *"get current light sensitivity","turn the air-conditioner on","sense the proximity of an object"* etc. We assume that each functional task explicitly manipulates one artefact. So, one artefact might be shared by multiple tasks but a single task can not use multiple artefacts. An application is expressed as a collection of such functional

```
<?xml version="1.0" encoding="UTF-8"?>
<application>                                          Dynamically Injected
        <name>Smart Display</name>
        <purpose>Providing Personalized Infromation with Situation Awareness</purpose>
        <binaryPath>ApplicationSpace/SmartDisplay/SmartDisplayApp.jar</binaryPath>
        <accesspointIP>10.0.1.3</accesspointIP>
        <accesspointPort>9824</accesspointPort>
 <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>milisecond</measurement-unit>
                                <high-threshold>70</high-threshold>
                                <low-threshold>60</low-threshold>
                        </qos>

                </profile-QoS-attribute>
        </task>
         ---------- More Tasks---------
 </task-list>
</application>
```

Figure 7: Task Description File (partly) for the display application used in the presented scenario

tasks in a Task Description File (TDF). Each task specifies the respective profiles, their quality of service and communication mode (e.g., synchronous and asynchronous) it needs to accomplish its goal. Figure 7 shows part of the task description file for the application presented in section 2. Each task may also contain Quality of Service (QoS) requirements for the target profiles.

The second requirement for an application is to use generic web protocols to manipulate the artefacts. When an application is registered in the FedNet system (see section 3.3), an access point is assigned to the application. An application needs to access this access point to send requests and receive responses from the underlying artefacts. In our current implementation the application uses a SOAP request for polling or sending an actuation request to the artefacts. For continuous polling (i.e. subscription), auto discoverable RSS feed is used. During the application's instantiation time, the required physical artefacts data semantics (<detector> and <actuator> nodes of the Profile Description File) are send to the application by the FedNet, so that applications can understand the data format of the artefacts and can request or receive data accordingly. In the current implementation, we have provided a simple library in Java comprised of a SOAP Client and Auto Discoverable RSS Parser, which the application developer can use to access the access point.

3.3 FedNet System

In our approach both the applications and artefacts are infrastructure independent and expressed in high level descriptive documents. Thus to create a runtime association between an application and the underlying artefacts, an intermediator is needed that can connect the applications and the artefacts. This intermediation is done by FedNet in our approach. FedNet does this intermediation by utilizing only the documents of these applications and artefacts. FedNet can contact the communicator module of the artefact core using the semantics described in the artefact documents for mapping application tasks, similarly application can contact FedNet using generic web access mechanisms. Since both the applications are artefacts are independent of FedNet and come as ready-to-run binary, end users can install them in the respective environment seamlessly. For supporting these installation, an end user tool is

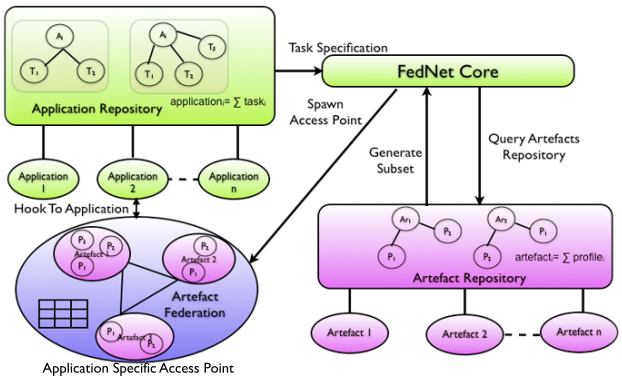

Figure 8: Architecture of FedNet

provided. FedNet itself is packaged in a generic binary and composed of four components as shown in Figure 8.

1. **Artefact Repository** hosts all the artefact running in the environment. During artefact deployment, the executable binary implementing the artefact framework and the ADF is submitted to this repository. When a profile is added to an artefact corresponding profile information is injected into the ADF (Figure 5) and the profile is attached to the corresponding artefact.

2. **Application Repository** hosts all the applications that are running atop the FedNet system. During application deployment, the binary executable and the TDF of the application are submitted to this repository and the identity of the corresponding application's access point is injected into the TDF (Figure 7).

3. **FedNet Core** provides the foundation for a spontaneous federation among the application and the artefacts. When an application is deployed the corresponding application's task descriptions are extracted from the application repository by the FedNet Core. Then it consults the artefact repository to identify the probable list of artefacts that can be federated considering the application tasks' profile and QoS requirements. Once the artefacts are identified, FedNet Core generates a template of the federation (collection of artefacts and their identities) and maps this federation into a generic access point component for that application. Then, FedNet Core assigns this specific access point to the corresponding application and injects the access point's identity in the TDF. When an application is launched, this access point is instantiated and the corresponding template is filled by the actual artefact available in the environment right at that moment thus forming a spontaneous federation.

4. **Access Point** is the generic component of FedNet that represents the physical environment (federated artefacts) needed by an application. Since each application's artefact requirement is different and each application might not be running all the time, FedNet assigns a unique access point for each application; meaning multiple federations of artefacts can co-exist in the environment. Simultaneously, each artefact can participate in multiple federations. When an application is lunched, it contacts it's Access Point to know the availability of the artefacts required by its task lists. The Access Point responds by specifying the tasks that can be supported

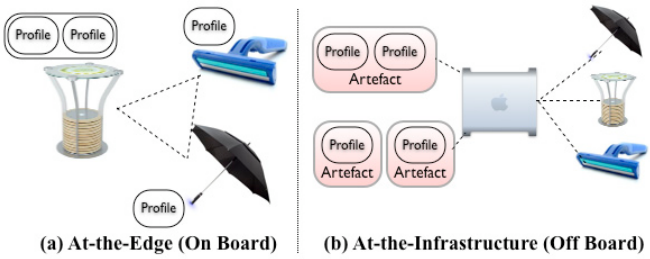

Figure 9: Location Modalities of Artefact Framework

in the current environment by filling its template with actual artefacts. It sends the mapped artefacts data semantics, i.e. *SensorML* and *Artefact Control Language* to the application which allow the 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 artefact. The artefacts' responds 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.

3.4 Distributed Management

In the earlier part of this section we have provided the explanation of the functional roles of the primary components of our infrastructure. From physical implementation point of view all these components could be distributed, i.e., instrumented artefacts can run in their own nodes, applications can run on the artefact nodes, or in a separate node integrating multiple artefact nodes, and FedNet can run in its own node to manage all other nodes.The artefact framework essentially is the digital identity of an artefact. So an obvious issue is the location of this digital part. We have two choices as shown in Figure 9: a) At-the-Edge (On-Board) b) At-the-Infrastructure (Off-Board). At-the-Edge means the artefact itself has a processing unit that hosts its digital representation where as the At-the-Infrastructure means a proxy, running in a separate location represents the artefacts and communicates with the artefact to retrieve sensor data or to actuate artefact's function 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 and self sustainable artefacts, it has minimal support for 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 advantage is the DIY support. Also, it enables rapid prototyping. In our current implementation we have adopted At-the-Infrastructure approach and each artefacts digital representation, i.e., artefact framework's binary core and profile plug-ins are deployed in a node that communicates with the physical artefact through some communication channel to retrieve the actual profile service via the hardware attached into the artefact. The same is true for the applications, i.e., the applications running on a single artefact can reside in the same node that represents the artefact and the application that integrates multiple artefact can reside on the any of those artefacts node. It is the FedNet components that organize these nodes in a distributed manner and manages the spontaneous federation. The FedNet components (i.e., Application Repository, Artefact Repository and FedNet Core) can reside in one or multiple nodes and manage the underlying artefacts and applications.

3.5 Deployment Tool for End Users

The components described so far provide the system foundation for the end user deployment. However, to involve end users in the deployment process, a tool is needed that they can use to install the artefacts and applications into the corresponding repositories, and to add profile plug-ins into the artefacts. Furthermore, this tool should enable the end users to control (run and stop) these artefacts and applications. In our current implementation a web based tool is provided for the end users to deploy artefacts and applications in the environment. Using this tool, end users can add and remove an artefact; add and remove profiles to an artefact, and run an artefact. Furthermore, endusers can install, remove and run an application using this tool. Figure 10, shows some screen shots of this tool.

4. EVALUATION

In the introduction section we have pointed out two questions that we addressed in this paper: i) building artefacts and applications in a way that are independent of the underlying infrastructure and deployable by the end users and ii) enabling application to adapt its functional behavior as richer artefacts' services are introduced. We have explained our approach of specifying both the application and the artefacts through high level descriptions. A runtime component (FedNet) matches these descriptions to create a spontaneous federation. This is useful for both the end users and the developers. For end users, it allows incremental editing of the smart space in a DIY fashion and for the developers it enables the development of the infrastructure independent applications, which can incrementally leverage richer artefact facilities. To validate these claims, we have evaluated our approach following the guidelines of Edwards et al. [5]. A *proof-of-concept* ubicomp system [8] that include multiple artefacts, profiles and application are re-developed following FedNet's approach and are provided to end users for real time deployment in a DIY fashion. This deployment task is supported by the end user deployment tool. In this section we present the end user trial and result of the study.

4.1 Target Scenario and Apparatus

We used the scenario introduced in section 2 excluding the personalization feature. The scenario was picked because of its simplicity and strong visual appeal. A smart space, where many things happen autonomously without providing visual feedback, has little appeal to the end users [3]. Since, we solicit end users' effective responses from deployment point of view, it was very important that the end users are completely acquainted with the scenario at hand. The apparatus used in the experiment are explained below.

4.1.1 Mirror Display

The mirror is constructed using an acrylic magic mirror board and an ordinary computer monitor (Figure 11(a)). This mirror can be extended to improve its functionalities and is equipped with an extension board (Figure 11(b)). The mirror is hosted in a networked tablet PC following our At-the-Infrastructure approach as explained in section 3.4. We assume this augmented mirror with embedded computer can be bought from the store in the near future.

4.1.2 Display Application

The application shows some up-to-date information (weather, stock, currency exchange rate etc.) in a mirror display. This application is distributed as a executable binary along with a TDF (Figure 11(f)). The application can work in different combinations of the following modes depending on the level of functionalities available in the mirror.

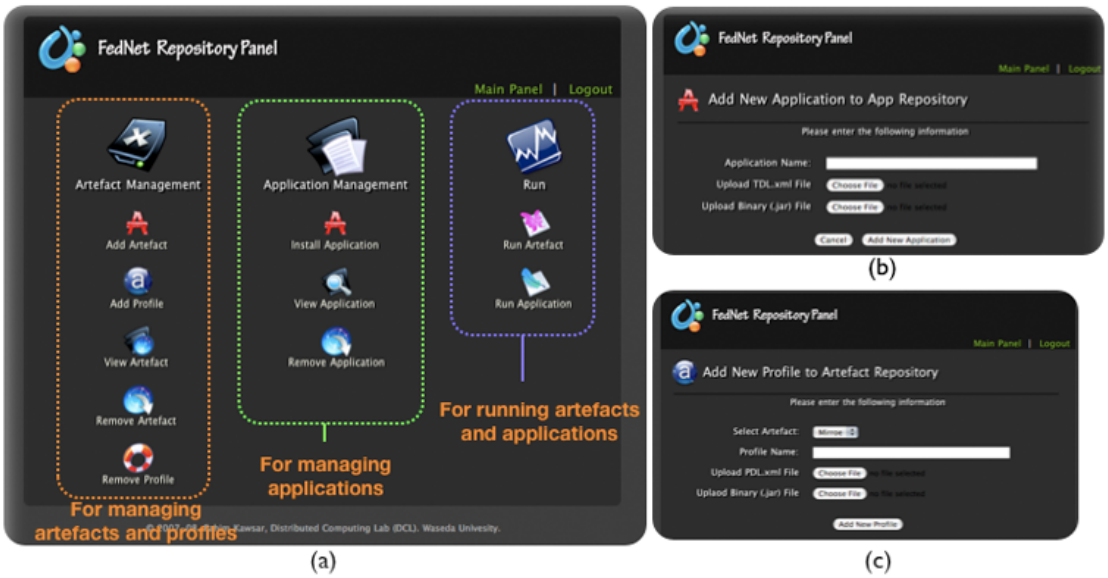

Figure 10: Snapshots of End User Deployment Tool (a) Main Panel, (b) Installing Application (c)Adding Profile

1. Basic Mode: The application shows information as pictorial widgets in this mode.

2. Sensor Mode: In this mode, the display is triggered only when someone is in front of the mirror, and the rest of the time it switches to the power save mode (blank display). To enable this mode the application needs the mirror to be augmented with a *Proximity Profile* that provides this positional context.

3. Standard Mode: The application provides two styles of presentation in this mode. Initially, the application shows the information in pictorial widgets which can be switched to textual presentation by interacting with the mirror. To enable this mode, the mirror needs to be augmented with a *Bi-state Interaction Profile*, which provides a two state switch functionality to the mirror.

4.1.3 Mirror Profiles

To work in a full fledged mode, the application needs two profiles in the mirror. A profile functionality can be achieved by multiple instrumentations. So, both profiles have multiple implementation choices. For each profile we have used two different implementations in this trial. Each profile comes with a hardware, a plugin binary and the Profile Description File. A user manual is also provided containing the installation instructions.

1. Proximity Profile: This profile's sole purpose is to recognize the presence of an entity in front of the mirror. This functionality can be achieved in multiple ways, i.e. using an infra red sensor, a motion sensor, a camera, etc. In our test, we have provided two implementations with two different sensors for this profile (Figure 11(d)). The first one is with an Infra Red Sensor and the second one is with a Floor Sensor (Figure 11(c)).

2. Bi-State Interaction Profile: This profile enables a user to interact with the mirror. It provides a simple two-state input facility which is suitable for the display application since a user can navigate between the pictorial and the detailed presentation styles. There are multiple instrument choices for the profile implementation. In this test, we have provided two implementations (Figure 11(e)), one with a touch sensor and the other with a slider.

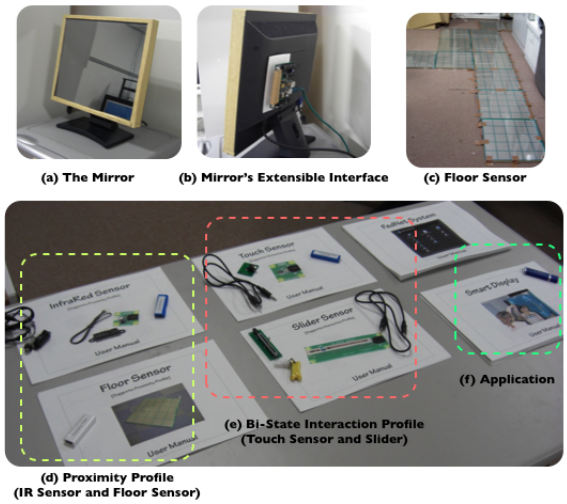

Figure 11: Mirror Artefact, Profiles and Applications with their manuals

4.1.4 FedNet

The FedNet infrastructure and the web based deployment tool for the end users are running in a laptop computer.

4.2 Experiment Detail

The goal of our experiment is to involve end users in a DIY fashion in deploying the mirror, installing the application, and then incrementally adding profiles into the mirror to enhance the application's functionality. We have invited 10 ordinary individuals

Figure 12: Participants deploying artefacts, installing application, adding profiles, etc.

(8 Male, 2 Female, Age Range: 21 35) with moderate computing skills through an open invitation in a social networking site. 9 of them did not have an engineering background and participated for the first time in this kind of experiment. The experiment had four phases. In phase one we introduced the concept, showed the apparatus and presented a tutorial on the web based deployment tool. In this phase, we also introduced the experiment tasks. In phase two, they were given 10 minutes to get familiar with the tools. Next, in phase three, they were asked to attain the given tasks.In this phase no direct assistance was provided except reference to the manual page containing the help.This phase included the following four tasks:

- **Task 1:** Deploying and and running the mirror.
- **Task 2:** Installing and running the application. The application runs in *Basic mode*.
- **Task 3:** Adding either the Proximity or the Bi-State Interaction Profile into the mirror by selecting one of the two implementations. The hardware installation requires attaching the sensor to the mirror using magic 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. The software installation required installing the plugin into the artefact binary. After this task the application either runs in *Sensor Mode* or in *Standard Mode* depending on the selected profile.
- **Task 4:** Adding the other profile into the mirror, and running the application combining the *Sensor Mode* and the *Standard Mode*.

Finally in phase four, we had a questionnaire and interview session.

4.3 Experiment Result

Figure 12 shows some snapshots from the experiment sessions. There were 40 tasks in total, four for each participant. All participants successfully finished the assigned tasks, though two participants needed active support in the early stages, primarily because of the unfamiliarity with the FedNet deployment tool. From a system's perspective, our approach provided a stable performance in

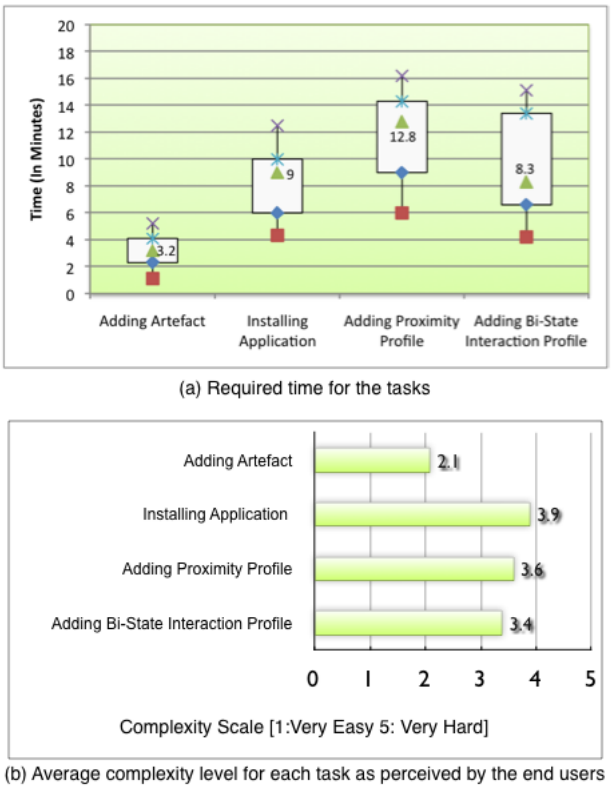

Figure 13: Average time taken and average complexities for completing experiment tasks

all the sessions and end users activities were properly converted into the system events accordingly, e.g. to add an artefact to the artefact repository, to add profiles, to form a spontaneous federation between the application and the artefact, etc. Regardless of the sensor type and implementation, profiles were seamlessly added into the artefact framework which highlights the capacity of our artefact framework design for hosting multiple profiles implementing different device interfaces. Furthermore, the application running in the mirror could successfully switch to respective advanced modes when the profiles were added signifying the spontaneous federation facilities of FedNet. Because of the runtime mapping of the applications' tasks to artefact profiles, whenever a new profile was added, the application could leverage that profile's service. This signifies the generality of our approach.

From usability perspective, each trial session was held for 120 minutes, on an average 44 minutes were required for the third phase (accomplishing the four tasks). Figure 13 shows the time (Figure 13(a)) required for each task and the corresponding complexity (Figure 13(b)) associated with the task. We measure the complexity in a 5 point scale with 1 as very easy and 5 as very hard. These complexity values are collected from the questionnaire sessions. All participants have shown progress in repeating tasks and on an average they required 43% less time in redundant activities, e.g. when adding the second profile plugin, attaching hardware, or restarting an application, etc. This indicates the fast learnability of our system from the end users perspective. In the following, we are reporting the implications of the subjective feedback that we received from the participants through formal interviews.

1. **Concept was difficult to comprehend:** The notion of arte-

fact, profile and application were difficult for the end users to comprehend and differentiate. For them the artefact and the application were the same, they could not separate the application from the mirror. This causes confusion while installing the application (task 2) in thinking it was already installed when the mirror was added in task 1. This is reflected in the median time of 9 minutes taken for installing the application as well as in the corresponding average complexity as shown in Figure 13(a,b) 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 in task 3 and 4 of adding profiles as shown in Figure 13 (a,b). Since they could not understand the notions clearly, it took time to install the profile into the appropriate artefact. However, the hardware installation was simple for them. These facts suggest that, our current notions are not self explanatory to end users and we need to provide a more comprehensive way of expressing these concepts.

2. **Installation process was difficult:** Although, all the participants were familiar with the internet, they found it difficult to use the web interface tool to install the artefact, the application and the profiles. Later interviews revealed that, it was not because of the interface directly but the process to accomplish a task, e.g. adding a profile, etc. Figure 13(a,b) also reflects these facts, on an average 12.8 and 8.3 minutes were required to add profiles with associated complexities of 3.6 and 3.4. The end users mainly struggled in installing the profile plugin, and we have found that the hardware installation was completed with less trouble. 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. This finding is crucial and has direct implications in our future work.

3. **Package with different options was preferred over the DIY approach:** Several participants pointed out that they can buy a product with different functional granularity according to their preferences. They concurred that the augmented artefact should be similar, 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 or to upgrade their existing package. Although,they agreed that the DIY approach is fun, interesting and inexpensive, but it limits the acceptability of the product to a mass population. A participant pointed out *"I don't think my 58-year old mom could use your whole system. Maybe, it was ok for me. But not for her. I don't think she will be able to attach sensors or even install anything. But she can use the microwave, because it just works....."*. Similar views were received from other participants which indicate that the DIY approach is suitable for a specific class of users familiar with technology. These facts signify that to make augmented artefacts available to a larger user base, packaging with variant options is needed. The incremental DIY approach can further extrapolate the packaging scheme.

4. **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 our approach software installation is required. Although, our 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 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. **Instantaneous feedback is necessary:** It is essential that when a profile is added, the new functionalities are reflected in the artefact instantly and erroneous installation is reported immediately to confirm the users' actions. In our 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 our artefact framework needs to have an instant feedback facility for the users' actions.

6. **Intuitive hardware interface is needed:** Considering, our participants performances and subjective feedback, we concurred that the intuitiveness of the profile hardwares are essential for the success of the DIY approach. For example, in our 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. 4 of the participants made mistakes in picking the right port (These cases are reflected in the max values of Figure 13(a) for task 3 and 4) Although it was clearly written in the manual, they did not consult it and tried to do it intuitively. Of course, If the artefact 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, we 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 artefacts 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.

Although the feedback from the participants did not positively validate the current state of our approach from the usability perspective, we consider these results promising for future research on pervasive applications and augmented artefact deployment. However, from a system point of view our approach provided stable performance and met the primary goals that we attempted to reach in this work.

5. DISCUSSION

For the purpose of discussion, we would like put forth a few issues in this section.

5.1 Modality of End Users' Support

One important discussion point is the type of support that is offered to the end users in our approach. The artefact framework is a generic binary with the support for plugging-in multiple profiles. So, a new artefact either with profiles or without profiles come with a ready to run component that users can deploy using FedNet end

user tool. Also, end users can gradually add new profiles to an artefact without complex configurations. As we have shown, end users have the flexibility to select the appropriate profile implementation that matches their preferences. The multiple implementations of the same profile (as shown in section 4.1.3) essentially highlights the abstraction layer of profile handler. Similarly, end users can install any application that adheres to the FedNet requirement, i.e. comes with a Task Description File and implements generic web techniques to access the access point. Applications functionalities can be enhanced by gradual addition of profiles into artefacts. The combination of these approaches enables lucid support for end users to build and enhance a smart space incrementally.

5.2 Profile Ontology

Our artefact framework is organized as a collection of profiles and these profiles are derived from the designers of the system. This profile notion has serious drawbacks from the standardization point. Since, we do not have a common vocabulary or ontologies that can be used to define profiles, a pitfall of our approach can be seen in the profile based unification. However, by profile abstraction, we are not trying to define the ontology for profiles. In stead we are providing a structure that designers can use to disseminate their implemented ontology. Defining the conceptual ontology in a standard way is the hardest part of pervasive computing not the encoding. We are fully aware of that, and do not claim that our platform is providing a solution to that. Our contribution is providing an architecture that can glue the encoding structures with rest of the systems seamlessly.

5.3 End User Deployment Process

Our end users experiment revealed that the deployment process has to be as seamless as possible. We found that the end users had difficulty using current web based deployment tool. As we mentioned in section 4.3, ideally end users would only do the hardware installation which itself has to be self explanatory and the software installation has to be autonomous to ensure the balance with end users current practices with home appliances. Considering these facts we are now working on a tangible interaction mechanism for software installation, which will allow end users to touch a specific RFID tag (embedding a URL) that comes with the artefact and profile into a reader attached to the FedNet system that will automatically collect the artefact binary and profile implementation with corresponding Description File from a remote location and will install it into the specific artefact space. Similar approach will be applied for installing the applications. We reckon this will make the software deployment process for physical artefacts and pervasive applications more lucid from end users perspective. Also, we are working on providing a instantaneous visual feedback to end users for their deployment actions.

6. RELATED WORK

The end users' support to entail a smart spaces incrementally, relies on instrumented artefacts and device integration technologies. We will look at the related work in these areas.

6.1 Augmented Artefacts

One of the very first prototypes of smart object was Mediacup [2] where a regular coffee cup was instrumented to provide the state of the cup as context information. Although the Mediacup project and its succeeding SmartIts [9] provide solid insight into the augmentation of physical artefacts with sensing and processing, they did not provide any generic representation model that can make them usable with any general purpose applications. Tokuda and his group introduced Smart Furniture and u-Textures to build custom furniture [11], 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 [8, 19]. These objects work fine in a specific scenario, however this assumption of scenario specific objects leads to a less reusable and closed development model. The artefact framework presented in this paper takes a generic approach to solve this problem.We present a service profile based framework to represent the instrumented features of the artefacts in an application independent way. We express this augmented features in generic languages which any application can use without prior understanding. By doing so, we make the instrumented artefacts plug and play this allows end users to deploy them freely.

6.2 Device Integration

To date several methods have been proposed to address device integration mechanism. One approach is interface standardization as attempted by Jini [20] and UPnP [14]. They describe devices using interface description and language APIs allowing applications to utilize the interfaces. However, they do not provide any artefact framework that enable incremental deployment in a meaningful way. Furthermore, as new services or features are added into artefacts, they can not be utilized by the applications because of the limited interfaces. Patch Panel [1] is a programming tool that provides a generic set of mechanisms for interoperating and translating incoming events to outgoing events, enabling interoperation among any set of devices that communicates using their EventHeap [10] communication platform. Although, this approach is seemingly lucid, it does not specify how to support the development of artefacts incrementally and how to express their semantics in a generic way so that any application can use those artefacts. In SpeakEasy [7] mobile codes were exchanged among heterogeneous devices to create an interoperable environment. Their approach requires a special runtime environment to be available at each device to exchange and execute mobile codes. It is hard to expect that such a specialized mobile code at the device end can be added incrementally and it is impractical for augmented artefacts deployable by end users. Although he has not considered the representation of artefacts and applications, his system is quite useful and can be integrated into our approach. InterPlay [13] is a middleware for home A/V networking and is similar to our approach. InterPlay 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 which contains property and grounding protocol information. However, InterPlay predominantly focuses on A/V devices thus does not provide any support for building artefacts incrementally. Our artefact framework is a major leap from InterPlay which signifies our contribution. Also, we do not consider user oriented tasks rather we express applications as a collection of functional tasks which enables developers to write applications without considering the constraints of the target environment. Task Computing [12] initiative allows users to select a basic service or compose a complex service by combining multiple basic services, but it does not provide a good user-centric approach as the user spends a lot of time in understanding the services. Also, it has no support for incremental deployment of artefacts for end users. A range of middlewares have been proposed in the pervasive literature [17, 18, 4] specifying their application development processes. These middlewares usually provide end-to-end support for the application developer, i.e. instrumented artefacts are wrapped into middleware specific wrappers and a range of APIs is provided to the applica-

tions to manipulate them. However, the problem of this approach is that the applications and the instrumented artefacts become virtually incompatible in other environments. In our approach, we have adopted a document centric approach allowing development of infrastructure independent applications and artefacts and FedNet provides the runtime association among them using respective documents.

7. CONCLUSION

We believe that in the near future, end users will be involved in associating smartness in the home and this involvement must support the evolving nature of the home, i.e. incremental deployment. From a practical point of view, this should be achieved by the end users. To enable this, we have presented, FedNet, a system infrastructure that enables incremental deployment support for end users utilizing an artefact framework and a task-centric application framework. The contribution of this paper can be seen as twofold. First, the plug and play artefact framework allows end users to deploy and to incrementally enhance a smart space without going through complex authoring or configuration steps. Our approach also allows an application developer to write applications considering the functionalities only regardless of the constraints of the target environment. Since, the profile abstraction essentially represents an artefact's functional capabilities, applications can be associated with the environment at runtime by the FedNet system. Second, We have reported a real life end user deployment session and findings from the experiment. From a system's perspective our approach was successful in enabling the incremental deployment by the end users. However, the experiment also exposed several usability aspects of end user deployment process, e.g., confusion arising from the notion of profiles and manual installation process, need for different packaging, intuitive hardware interface, instantaneous feedback, etc. We consider these findings from our experiment are very useful for further research exploration in the pervasive computing domain, specially one that involves augmented artefacts.

8. ACKNOWLEDGEMENT

This research was supported by "Ambient SoC Global COE Program of Waseda University" of the Ministry of Education, Culture, Sports, Science and Technology, Japan.

9. REFERENCES

[1] R. Ballagas, A. Szybalski, and A. Fox. Patch panel: Enabling control-flow interoperability in ubicomp environments. In *Second Annual IEEE International Conference on Pervasive Computing and Communications*, 2004.

[2] M. Beigl, H. W. Gellersen, and A. Schmidt. Media cups: Experience with design and use of computer augmented everyday objects. *Computer Networks, special Issue on Pervasive Computing*, 35-4, 2001.

[3] V. Bellotti and K. Edwards. Intelligibility and accountability: Human considerations in context-aware systems. *Human-Computer Interaction*, 16(2-4), 2001.

[4] 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.

[5] 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 *The ACM Conference on Human Factors in Computing Systems (CHI '03)*, 2003.

[6] W. K. Edwards and R. Grinter. At home with ubiquitous computing: Seven challenges. In *The Third International Conference on Ubiquitous Computing*, 2001.

[7] W. K. Edwards, M. Newman, J. Sedivy, T. Smith, and S. Izadi. Challenge: recombinant computing and the speakeasy approach. In *The Eighth Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2002.

[8] K. Fujinami, F. Kawsar, and T. Nakajima. Awaremirror: A personalized display using a mirror. In *Third International Conference on Pervasive Computing*, 2005.

[9] H. Gellersen, G. Kortuem, A. Schmidt, and M. Beigl. Physical prototyping with smart-its. *IEEE Pervasive Computing*, 03(3):74–82, 2004.

[10] B. Johanson, A. Fox, and T. Winograd. The interactive workspaces project: experiences with ubiquitous computing rooms. *IEEE Pervasive Computing*, 1-2, 2002.

[11] N. Kohtake, R. Ohsawa, M. Iwai, K. Takashio, and H. Tokuda. u-texture: Self-organizable universal panels for creating smart surroundings. In *The Seventh International Conference on Ubiquitous Computing*, 2005.

[12] R. Masuoka, B. Parsia, and Y. Labrou. Task computing - the semantic web meets pervasive computing. In *The Second International Semantic Web Conference*, 2003.

[13] 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 *Fourth Annual IEEE International Conference on Pervasive Computing and Communications*, 2006.

[14] Microsoft Corp. *Universal plug and play device architecture reference specification*.

[15] O. G. C. Inc. *Sensor Model Language (SensorML) implementation specification*.

[16] T. Rodden and S. Benford. The evolution of buildings and implications for the design of ubiquitous domestic environments. In *The ACM Conference on Human Factors in Computing Systems (CHI '03)*, 2003.

[17] M. Roman, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. Gaia: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing*, pages 74–83, 2002.

[18] 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*, 2002.

[19] M. Strohbach, H.-W. Gellersen, G. Kortuem, and C. Kray. Cooperative artefacts: Assessing real world situations with embedded technology. In *The Sixth International Conference on Ubiquitous Computing*, 2004.

[20] Sun Microsystems Inc. *Jini Specification, Nov. 1998.*