Abstract. We present a multilayer architecture for pervasive computing in heterogeneous environments: the General Adaptable Services Manager (GASM). This architecture allows a developer to build context-aware multi-platform applications by means of a set of core services such as device discovery and integration, flexible software component deployment, heterogeneous network communication broker and adaptable user interface presentations. We make use of the Open Services Gateway Interface platform (OSGi) as the underlying universal middleware that is available across many heterogeneous networks. Our architecture is defined in terms of OSGi components. A mobile agent manages the deployment and runtime of applications on top of the architecture.

Keywords. UIML, OSGi, eHome, pervasive computing, user interface

1. Introduction

Automated ambient environments [4] with eHome, eHealthCare, eSchool as prominent examples, require a reliable and intuitive mode to use their possibilities. In such environments, where a large number of devices participate, the user does not benefit from all available functionalities if these are not available at the right moment, and presented in a user friendly way to be manipulated.

Each pervasive computing environment [2][11][17] has a various number of complicated problems of fundamental importance that need to be tackled. General interconnectivity, monitoring, remotely controlling and dynamic tailoring of the displayed functionalities are all common aspects of immense importance to these systems.

Disregarding the problems in hardware diversity, the biggest technical challenge is the heterogeneous software platforms. If configuring the solutions is not sufficiently automated and trivial, an end-user of the system would have to be equipped with sufficient technical knowledge in order to use the system.

In a pervasive environment as eHome, our case study, information sources are spread over a wide physical area. The diversity of communication technologies like UPnP, ZigBee, WiFi, etc. makes the search for a generic and working solution very difficult. The complex and dedicated functionalities that are provided by each specific device and the diversity of manipulation possibilities, must be addressed in a common way for all, from a simple command like light on/off to a complex command to set an alarm system.

We have analyzed the system from three points of view: communication, integration and control. The system architecture presented here is service-driven with a functional dedicated and modular approach.

In this paper we present our system architecture named General Adaptable Services Manager (GASM) that combines OSGi and the User Interface Markup Language (UIML) followed by a discussion of the implementation of the case study based hereupon. The eHome implementation is realized as part of the OCoMIS project (OSGi as a framework for context-aware embedded devices) [12]. Our main goal is to enable the software developer to overcome a typical problem while building applications for pervasive computing environments: our system supports dynamic
heterogeneous environments at runtime without manual intervention.

A services manager monitors the registered services and assures their availability inside the setup environment. As this is built upon OSGi, it inherits the framework’s functionalities, including the possibility to install and update the bundles for each device and discovery and propagation of new services. An OSGi bundle consists of Java classes and resources which together can provide services to other framework bundles. An important part of the functionality is the ability to deploy remote control applications on different devices with an adapted user interface tailored according to the targeted end-user device. Using UIML we abstract the UI by describing it in a device-independent manner.

We have identified the necessity of using a central hardware component which is able to support the various protocols and connection types inside a network: the home gateway. The system manager running on this gateway is using the OSGi middleware. Its architecture is based on services grouped on an open multi-layer architecture. Each layer groups modules with a common task e.g. communication, presentation, etc. Therefore there is the possibility of having different modules with the same task but provided by different devices. The open multi-layer architecture is a stack of different well-defined layers where each layer can access the functionality provided by other layers, which happens in an arbitrated and supervised mode. The flexibility of the architecture is further increased by the possibility of dynamically adding new layers when there is a need for this. The open multilayer architecture grants a clear distinction between the different tasks of the system while avoiding the rigidity of layered systems where only the direct previous layer can be accessed.

Considering its features the OSGi specifications [13][15] have been proved to be valuable building stones for modular systems. Some of the most important characteristics that supply our system requirements are:
- Dedicated for networked applications and service-oriented software. The update of services can be done at run time over the network.
- The services can be adapted according to the device/user requirements (e.g. mail service, SMS service).

We have chosen UIML for the description of the user interface because of its characteristics [18] and the availability on various platforms [8][9][14]:
- The UI is independent from platform and programming language.
- The UI description can be customized for a specific device and for a specific user.
- The presentation of the UI is independent of the service being accessed. During the design phase, prototypes for different devices and users can be created easily.
- A new application can be build up faster from the combination and reutilization of previously developed modules.
- There is a clear separation between the user interface and the back-end application. This provides enough flexibility to change the back-end application without modifying the already developed user interface.

The remainder of this paper is structured as follows: we continue with an overview of the related work in section 2. Next, we introduce the General Adaptable Services Manager system design in section 3 followed by the eHome implementation in section 4. Finally we draw the conclusions and write down the future work.

2. Related work

From the point of view of Information and Communication Technology (ICT), interest has increased beginning with Weiser (1991) [11] in the “ubiquitous computing” vision direction. Other related terms that are often used are “pervasive computing”, “augmented reality” (Mackay, Wellener and Rich Gold 1993 [19]) “tangible bits” (Ishii and Ulmer 1997 [7]), “wearable computer” (Le Bass, 1997 [10]) and many others. The image of the desktop computer as we know it with its typical interaction is evolving from single input to a fully user friendly environment.

During the last decade, interest for ambient environments has increased, mainly targeting specific restrained topics. The choices made for our design coincide in many points with the presented research.

There are many presented solutions for service discovery [16]. [1] presents an architecture with plug-in support and services oriented technique for spontaneous composition, in pervasive environments. Considering our reduced scale it’s sufficient for us to use the built in querying facility of the OSGi framework.
There are many solutions for a smart home software infrastructure; e.g. [6] presents such a system able to discover the location of a certain device, using RFID (Radio Frequency Identification), to register a device in the OSGi based framework and to offer a central remote control. We enhanced the control possibilities by spreading the system control from one central remote point to many devices within the ambient environment.

A system architecture with a mobile-agent and an OSGi based three-tier control communication system architecture for the smart home has been presented in [5]. We acknowledge the similarities between this approach and our system requirements and related network communication design.

There is a clear need to separate the user interface description and the back-end application logic. Besides the benefits of using UIML for this purpose, which can be found in the introduction, UIML also enables us to have a clear separation of concern. The user interface can be created independently of the application logic, thus a user interface designer can create an appropriate user interface which can be easily integrated with the software components that are deployed. Furthermore the platform-independent aspect of UIML is indispensable for the system presented in this paper.

In GASM we propose a system that combines OSGi and UIML methodologies in a general and flexible solution for pervasive environments. The system architecture integrates functional components that arbitrate the dynamic distribution, migration, adaptation of the UI for the system remote control over the connected devices. This flexible mobile agent easily bridges the complex interaction between system, device and user.

The concept supports later maturity possibilities thanks to system modularity and functional concept.

3. General Adaptable Services Manager Design

The adaptability of our system is focused on the quality to deploy adapted UI on heterogeneous devices, the flexibility supporting complex devices and the goal of a dynamic number of system layers. The generality is given by the architecture adopted, that permits the basic system to be suitable for a eHome, eCareCentrum or other OSGi related pervasive systems.

We concentrated on providing three essential building blocks that are important parts of a pervasive computing environment: a heterogeneous communication infrastructure (“connectivity”), multiple different computing nodes (“devices”) and flexible adaptable interaction capabilities (“user interfaces”). Figure 1 shows a conceptual overview of the overall architecture based on the OSGi business model [3]. The communication with the outside world is supported (and protected) by gateway connections. The arrows indicate possible data communication paths between components.

![Figure 1. Communication architecture](image)

Three device service types are presented:

- **The device Services Provider**: can only offer remote manipulation services (e.g. a UPnP camera can register services as rotate, zoom, start, stop)
- **The device Services Client**: acts as clients for the registered services and has user interaction capabilities (e.g. acts as the remote control through the gateway – PDA control)
- **The device Services Provider-Client**: acts both as a service provider to the environment and as a service consumer.

The local area environment is represented simplified by the Gateway and the Provider/Client device Services. The internal communication possibilities are not limited to device-gateway-device communications; the gateway has been chosen as a solution covering the communication gaps between devices.

Our system is content-aware because it constantly processes the changes that take place with the components (layers, bundles, services, devices). It is able to automatically integrate new services and to manage the environment’s
devices in an unsupervised mode, thus functioning without the intervention of an actual user. This alleviates the user from the typical installation process coupled with each new device added to the environment. The Gateway connects the system with the Services Providers Repositories Gateways, from where the system can download suitable modules to install or update the system when new devices need to be supported. The Services Gateways allow the services and the environment to interact with each other (e.g. reading gas/water consumption; remotely commanding the HVAC system). Being able to communicate with different aspects of the user environment is a basic requirement for a context-aware system.

As shown in Figure 3 the GASM system is built on top of the OSGi-framework, thus inherits the OSGi service support. Its main attributes after the system is initially integrated, are enabling the interconnection of various devices, assuring the communication between devices inside managed local network and the Internet, to assure secure remote control and diagnosis from inside and outside the local network, and to sustain flexible software updates.

The architecture is an open multi-layer model that permits dynamic updates and component integration (Figure 2). Each layer is composed of elementary components which are mandatory to assure layer functionality and additional varying components. Additional components can be installed, updated and uninstalled without perturbing the layer’s functionality (e.g. on the communication layer the UPnP communication bundle can be installed or updated). The elementary components can only be updated, installed or uninstalled during the diagnose process. The diagnose process runs several tests to check the system consistency and it is designated for system elements update.

Our architecture supports evolutionary systems that have to deal with changes in the system features and available services. The division of the system in several layers leads to an abstraction of the underlying system and allows us to create dedicated layers for certain types of functionality. A multilayer system will have to go through the connection layer enabling the UPnP connection service to transmit data, thus hiding the details that are common with device integration. Although at the time of writing we have not implemented an agent for model-driven actions that is capable of making the connection between the user issuing command and executing actions. A model-driven action is a sequence of actions that can lead to an expected result. Intelligent systems are capable to learn models and generate new combinations.

The system is content-aware but could support such an upgrade to being context-aware. The number of layers is dependent on the system specializations that are required.

4. Implemented system: eHome

This section describes a home automation prototype implemented with the GASM system. Several layers were created that offer access to particular types of functionality required by the prototype:

- The Adaptable Interfaces Manager (AIM) layer contains the mobile agent that acts as the UI and the Abstract Device Manager.
- The Control System Manager (CSM) layer contains the main controller module which represents the kernel for the entire architecture; its role is to provide access to the system in a transparent mode to the AIM.
- The Services Manager (SM) layer contains the monitoring bundle, the authorization and the credits modules and state handling modules.
- The Communication Manager (CM) layer contains bundles that are in charge of the communication facilities that allow users to interact with their environment. It is also the most extensible layer because it permits specific communication services or protocols to be registered (e.g. SMS bundle, XMPP bundle, database support library and so on).

As can be observed in our design we have a clear separation of the remote application code and user interface that presents this remote functionality. The elementary elements are
represented by bundles because the system is built on an OSGi framework.

The CSM layer is the central part of the system and can be seen as the controller for all running system applications. Its responsibility is to control the discovery, registration and manipulation and to assure the rights arbitration over the available services and devices. This layer incorporates the system remote code application.

The SM main responsibility is to regulate and monitor the OSGi bundles execution states. The modules execution states and the OSGi bundles states are the same, since this is a one-on-one mapping (modules and bundles). The basic operations supported by the SM are: start, stop, install, uninstall, refresh and update [13]. Other tasks supported by this layer are to arbitrate device access to resource services and to manage user’s rights.

The central component of the AIM layer is the mobile agent that includes the Java UIML renderer module. The mobile-agent is capable of deploying the UIML UI on different devices and adapting the user interface according to the device and user profile. In previous work we extended the UIML renderer with support for MPEG 21-based user profiles [9] (e.g. color vision deficiency of a user can be described this way and taken into account when rendering the user interface). In general a mobile agent is able to migrate across the network, which inevitably involves difficulties to deploy it considering UI adaptability problems and to ensure enforced security constraints (e.g. requires a reliable deployment certificate mechanism). This approach that uses the UIML renderer has resolved the issue of adapting the UI to the device capabilities and user profile. The implemented run-time UIML-renderer offers support for Java AWT widgets, but we regard that it can also be implemented in a general fashion. In contrast with the flexibility of this approach is the rendering time required for the user interface to be built and presented on the end-user device. This depends on the complexity of the UI that is shown and the device resources available to the renderer. Initial experiments have indicated a high delay when using devices with limited resources: it takes between 1 and 2 seconds to render the UI and present it on screen. Because the processors of limited devices (PDAs, phones) are getting more powerful, we expect these delays will become significantly shorter in the near future. Furthermore the capability of representing a standard application is not going to be a problem for the majority of devices, but the incapacity to easily change the UI given the device capabilities and user profile, makes the UIML rendering methodology a future proof solution.

The abstract device interface is part of the AIM and is an interface for the real device driver implementation that can be implemented by different providers. This approach increases the transparency of the system.

From data service manager point of view the devices can be considered through abstraction of their offered services, as shown in Figure 5. In this figure we can see the control application interface rendered on a TV, a PDA and a mobile phone. Finally we can also see a simulation of the different devices on a desktop computer.
The flexibility of the system architecture gives the possibility to the user to decide when, where and how he/she wants to access and program the devices features.

5. Conclusions and future work

The purpose of the Home Automation System presented in this paper is to manage the centralized remote control and data flow for different interconnected networked home appliances (NHA) and to deploy adapted user control interfaces on different available devices.

The improvements that our system brings to other existing solutions are the combination of two technologies: OSGi and UIML. The flexibility of the architecture, the support for smooth system updates and the capability to tailor the UI without the necessity of code modification, are two benefits that are a consequence of combining these technologies.

Our future research efforts include the improvement of the rendering time that is required to adapt and present the UI. We intend to develop a healthcare home facility system using the presented architecture and, consistently shorten the development time by reusing modules that are already available and by building upon the kernel system we introduced.

6. Acknowledgements

The OCoMIS research project was funded by the Flemish Government through the IWT-Flanders. We also wish to thanks our industrial partners, for their valuable feedback.

Part of the research at EDM is funded by ERDF (European Regional Development Fund), the Flemish Government and the Flemish Interdisciplinary institute for BroadBand Technology (IBBT).

7. References