An Adaptive Network Architecture for Home- and Building Environments

Rolf Kistler, Stefan Knauth and Alexander Klapproth
Lucerne University of Applied Sciences and Arts, CEESAR
Technikumstrasse 21, 6048 Horw, Switzerland
{rolf.kistler, stefan.knauth, alexander.klapproth}@hslu.ch

Abstract

Fieldbus networks have significantly improved flexibility and management in classic building automation domains. However, the costs of such installations are still greatly affected by the commissioning efforts needed to bring them up and running. The dedicated cabling, initial engineering efforts and sophisticated binding and configuration tools often prevent home owners from investing in such networks. On the other side, many electronic devices from other domains have found their way into home- and building environments. And with IT equipment TCP/IP connectivity and "zero config" protocols were introduced. There is the vision of an intelligent home/building with one integrated network in which all devices work together seamlessly. It may positively influence factors such as energy efficiency, usability, flexibility, security and comfort. This paper proposes an Internet protocol based architecture thought to build up such a network. The result is a distributed, scalable hard- and software infrastructure that adapts to the context of use and comes up with goal-centric services provided by the numerous underlying devices. It allows standard mobile clients to act as remote controls with dynamic user interfaces generated on the fly. This text explores the different building blocks that make up the architecture and presents results derived from a first prototype.

1 Introduction

Numerous electronic devices populate a modern building today. An increasing number of them is equipped with network control interfaces and remote access features. The need to reduce operational costs led to more flexibility and optimized management through networked solutions in the domain of commercial building automation at an early stage of the development. KNX/EIB and Echelon LonWorks systems are now implemented in thousands of buildings. More or less recently, other domains (security and safety, energy metering, home automation, consumer electronics, domestic appliances...) have come up with their own network solutions. So in the homes and buildings of the next generation, all these networked devices will work seamlessly together for the convenience of the end user and to the benefit of solutions providers, integrators as well as investors? Looking at the current state, the sheer number of different technologies and standards waiting to be adopted in the networked building is overwhelming. Technological trends such as wireless systems or service oriented architectures (SOA) and the well known strengths of TCP/IP based protocols will probably help to some level of harmonization also in this field of application. However, a building network will continue to be a very heterogeneous environment for many years to come.

From an end user perspective, one finds that there is no common way to control all these devices. Today's user interfaces are specific (domain, manufacturer, device), inflexible and often proprietary. There are (still proprietary) solutions that harmonize control for selected high-end users in the home automation segment, but these Smart Homes are "hand-crafted". Bringing the heterogeneous zoo of devices together comes at expenditures that render those systems unprofitable for commercial buildings as well as the average home. A harmonized, intuitive and still cost-effective solution could bring real value to all stakeholders: End users are able to fulfil their tasks more efficiently using adaptive, context-sensitive, two-way interfaces. They can use their own familiar control device (e.g. smartphone) and every multimedia system looks the same providing only the services needed to fulfil a specific goal. New services like load control or multiroom media streaming become true. Devices from various vendors can be combined and integrated ad hoc with a minimum of engineering efforts. This is a very welcome feature in commercial buildings where changes are in the order of the day and also in homes where an ordinary user should be able to install a new device.

The UPnP forum [1], the OSGi Alliance [2], the Java Community and many others have come up with technologies and implementations ready to use for such purposes today. Researchers have provided building blocks with works in the fields of plug and play systems, ambient intelligence, pervasive computing, context-aware systems, task-based user interfaces and new forms of intelligent human machine interaction that could be part of such a system. However, there is still much work to do to close the gap to a truly integrated solution that follows the requirements posed on such systems. The proposed architecture brings these pieces together trying to stick to standards as far as possible. It adds extensions to existing protocols and comes up with new, pragmatic solutions where needed. Devel-

opment and implementation of the architecture are part of the CARUSO applied research project. The field tests of CARUSO are conducted in the iHomeLab [3], the national platform for research on intelligent living in Switzerland.

2 Terms

The **system** is a collection of target devices, control points and infrastructure components which provide low-level and high-level services to the user, so she can fulfil her tasks. The **user** of the system is a person with a predefined role (secretary, care taker, security officer, building engineer ...). The user has certain tasks (goals, needs) she wants to fulfil with help of the system. The system assists her with selected low-level and high-level services. A **control point (CP)** is a physical device that acts as human machine interface for the user, allowing her to interact with the rest of the system. A **target device (TD)** is a physical hardware device within the system that shall be controlled and supervised. The target device exposes its interface and provides **low-level services** to the user and the rest of the system. **Low-level services** are built-in device functions that have been predefined by the vendor and get into the system with a new TD. One TD can handle them all by itself. **High-level services** or **smart services** are situated one abstraction layer higher than the low-level services. They are independent from specific devices and resemble more closely the user needs (”user-centric”, ”task-based”, ”goal-driven”). We assume that in most cases, the task can be achieved with an intelligent aggregation of low-level services from one or more TDs and other high-level services within the system.

3 Requirements

Looking at the needs roughly described in the introduction, we identified the main features the system should offer.

Adaptivity: Similar to ”flexibility” or ”context-sensitivity”, it here mainly stands for the ability of the system to dynamically adapt to parameters changing between different control scenarios. These changing parameters are: (1) Users, (2) Target devices or services, (3) Control points, (4) The physical environment. We mainly adopted the definition of the ”context of use” from [4].

Ease of use: If the usability of the system is not equal or higher than the state of the art, the user will not accept the new way.

Broad applicability: The system shall integrate a broad range of devices and services from different vendors. These vendors provide solutions for different domains in the building most of which where treated as separate islands so far.

Ad hoc extensibility: The system shall allow to add new target devices, services, control points and users at runtime and with a minimum of human interaction. Devices and services shall be automatically discovered and become part of the system. Changes in the system are updated on all relevant control points.

Scalability: A small home may not have more than few network nodes where as a commercial building can host hundreds or even thousands of devices and services. The range of devices reaches from high-end IT servers down to simple low-cost wireless sensor nodes operating on very limited resources. In terms of performance and interaction time, building automation applications put soft real-time requirements on the system. For lighting and jalousies, the reaction time between the user action and the visible result of the physical actors should lie below 200 ms. Also the other control interactions should not take considerably longer than using conventional control systems.

Security: In discussions around networked buildings, security issues always come up very early. Nobody wants a system that allows any person to shutdown a building using his PDA. No teacher presenting slides on his beamer likes to know that every pupil in the class room has remote access to it over the mobile phone. Some kind of authentication and authorization should be considered. It should also be possible to backtrack control actions and assign them to an individual, a control point or an automated service.

Mobile Standard Clients: The system shall provide the possibility to use standard mobile clients as control points. We chose four representative hardware devices: A newer mobile phone (Sony-Ericsson K800i), a smartphone (Qtek S200), a PDA (Siemens-Fujitsu Pocket LOOX 520N Series) and a Laptop PC (Thinkpad T43p). Such and similar devices are wide-spread, powerful, communicative and many users have become familiar with them. It’s important to state that they are thought as extensions to the existing conventional control devices and have to co-exist with them.

Standard Protocols: Where ever possible, standard mechanisms and protocols shall be used. This also takes into account that infrastructure in buildings have long life spans.

4 Architecture and Implementation

4.1 Network Topology and Infrastructure

First, the preconditions and assumptions we’ve defined: Every communication partner in the network provides a possibility to exchange data over IPv4 and UDP (including multicast), either directly or over a gateway. Thanks to the great popularity of TCP/IP and the Internet, gateways are commercially available for almost any translation from hard- and software used in the field (KNX/EIB, LonWorks, ProfiBus, PLC, RS232, ...) to Ethernet/IP. From this core IP network, we expect that it has been set up and is working properly.

Further, all communication partners must gain access to the IP network. One issue here is the connection of a control point to the network. It will most likely be a mobile device with wireless communication features. However, as public access without any security mechanisms is not an option, connecting WLAN, Bluetooth or ZigBee clients still lacks the convenience users expect. So the risk is high that configuration issues, such as acquiring and entering

network keys, will distract many of them from using their client. One way to increase usability and security is to apply emerging NFC and RFID technologies, which have a good chance to be included into future mobile phones. If the user enters a building with such a client, she just holds it near an NFC access point (like a badge) and is immediately granted access and connected to the network (or denied).

An important decision is the choice of the network topology. It affects many of the characteristics of the system. We found that neither a monolithically centralized client-server, nor a fully distributed peer-to-peer approach fits the needs (see Fig. 1). The following logical components are part of the network:

Control Points (CP): The purpose and characteristics of these have been described already. Under normal circumstances, the control points get their UIs delivered by the *UI server* and access the *target devices* through the *device controller*. If no server is found, a *CP* has the capabilities to discover and control *target devices* within its physical range directly and deliver a restricted user interface for them (peer-to-peer mode). This peer-to-peer mode, plug and play functionality, local data storage and other factors, which require an autonomous *CP* to some extent were reasons to install a dedicated control application on the *CP*.

Device Controller (DC): A server component that separates **Control Point Networks** from **Device Networks** thus restricting access between *control points* and *target devices*. Besides this, the device controller has access to the rest of the network and acts as router to other potential *device controllers* and their local control point-target device networks. Through the core network it also reaches the *services server*. The *DC* restricts access to the services exposed to the *control points* depending on the context and handles user sessions. It synchronizes and executes commands coming from *control points* for the *target devices* sending back their notifications. Depending on the size and type of the building, the number and performance of physical devices each holding a *device controller* instance may vary (it could be only one for the whole building or one in each room). It also segments the whole network when deciding what traffic is flowing to the core network or is handled between the local control point-device networks.

Target Devices (TD): The devices to be controlled. Access happens over a *device controller* or directly from the *control point*. The *target devices* expose their interface by providing functionalities to all components able to handle them. As already stated, the system offers - as a fallback - access to *TDs* from *control points* on a peer-to-peer basis. If - in this mode - part or all of the functionality of a device is critical to be presented to every user, its up to the *target device* to implement some kind of security mechanism to prevent this. If not in peer-to-peer mode, the user authenticates only with the *device controller*. Multiple login procedures for the same user shall be avoided.

UI Server: Services must be presented to the user somehow. The user interface for the end user is displayed on the *control point*, but the information on how to build up a UI is coming from the *UI server*. The *UI server* communicates with the *device controller* to get a list of target devices and services to present to the user.

"Plug and Play" Gateways: Many of the *target devices* will not be equipped with mechanisms such as device and service discovery and sophisticated event handling. Common field bus technologies do not provide them natively. To get the functionality for all devices in the system, further components are needed. *"Plug and Play" Gateways* poll for devices on the "field bus side" and map their services to the other side. CARUSO runs them as UPnP-fieldbus gateways in form of software components together with a device controller on the same device.

Services Server: This component is optional. It's where high-level services and supplemental resources are located. It holds binaries of high-level services that can be downloaded, installed and executed by *device controllers*. Moreover, a services server may provide a central user management console, logging services, translation and internationalisation services, services to fetch optional resources (icons, images, sounds, vendor skins, help texts) and Internet based external services.

Installation Server: A server which holds the control application to be downloaded and installed on the *control points* and notifies about updates.

For several reasons, OSGi has been chosen as technology for the server components. It has a sophisticated life cycle management, is well applicable for embedded systems, knows a predefined user administration concept, provides a services concept and comes up with standard services like logging and UPnP interoperability services. Network participants such as device controllers, UI servers, gateways, services servers and all high-level services are implemented as independent OSGi bundles. This approach allows for a scalable system in which the solution provider decides which software bundles to distribute on which physical nodes. For a home system, they may all reside on the same embedded server node. Some of the services may be optional and bought and installed automatically later on over the internet. On the control point side, the presentation and control application consists of a .NET compact framework binary.

4.2 Plug and Play Functionality

UPnP (Universal Plug and Play) was the first choice to implement plug and play functionality in CARUSO. It's very powerful and seems to be the most widely accepted protocol for this purpose. At least this is true for the consumer electronic, IT and mobile computing domains (Microsoft Windows Mobile). The cross-industry organization DLNA [5] has included UPnP as an integral part into their "guidelines based on open industry standards to complete the cross industry digital convergence". UPnP initially targeted home automation, but case studies and prototypes have indicated that it can also be adopted for commercial building automation over gateways or even in implementations on field level devices [6] [7].

UPnP bases on TCP/IP. The term *control point* has been taken from UPnP. UPnP *devices* are *target devices* and *ser-*

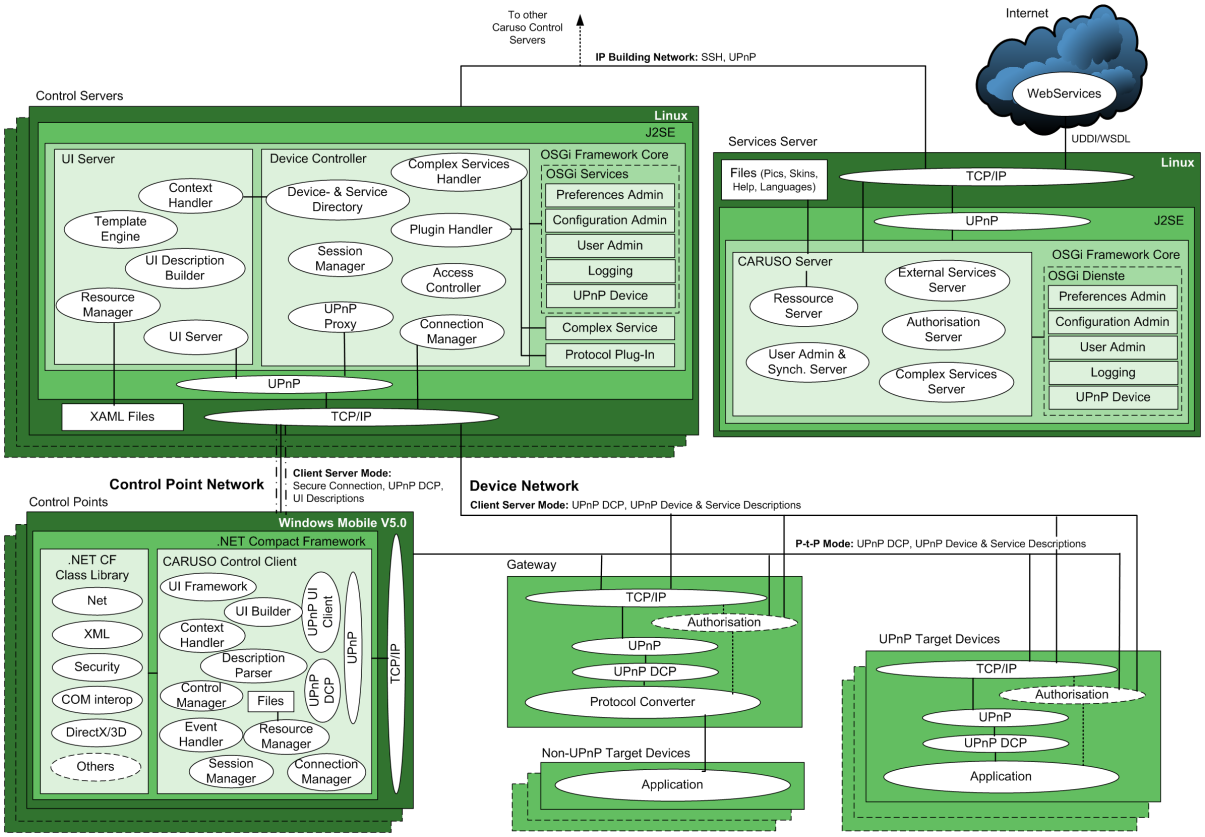

Figure 1. CARUSO Architecture Details

vices in UPnP are *low-level services* in CARUSO. UPnP defines protocols and procedures involved in (1) IP addressing (2) device and service discovery (3) device and service description (4) control (5) eventing and (6) presentation [1]. The presentation is limited to deliver an URL to an optional presentation page (mostly HTML). UPnP may post search messages to the network to selectively find devices and services (e.g. "all color printers", ...).

On top of that, the UPnP Forum has developed specifications for device classes, so-called "Device Control Protocols" (DCPs). Each DCP defines a common interface for a class of target devices and its services. Control over any UPnP DCP compliant device becomes easy with the knowledge of its well-defined interface and the ability to detect it. DCPs have been defined for many target device classes such as Media Servers, Media Renderers, Printers, HVAC devices, lighting equipment, UI servers/clients and others.

Worth examinig a bit closer are the DCPs *'Remote UI Client'* and *'Remote UI Server'*. Remote UI servers may supply UIs for remote devices and services. Remote UI clients find UI servers automatically and get a list of compatible UIs (in form of URIs). The list can be filtered and clients that speak multiple UI protocols are supported. By selecting an UI out of the list, the UI server returns a file. The file format is free, some known formats have been predefined. This scheme is used for the CARUSO UI Server and the control points (UI Client). If no UI server is present, the control point is forced to take the information directly from the target devices. They just deliver plain UPnP device and service descriptions. Besides the presentation link mentioned, these do not contain any UI related information (e.g. on how to represent, group and place UPnP actions and state variables). Besides the UI server, the device controllers as well as the services servers act as UPnP devices in the network. This allows other components to discover them and use their services.

UPnP solves a lot of problems. Two things it cannot do: UPnP is inherently device-centric and it has poor support for presenting user interfaces. For high-level services and user interface presentation, other solutions need to be found. And UPnP has another drawback that affects our system: It produces traffic on the network. As UPnP control messages are transmitted over SOAP over HTTP, the protocol overhead is quite massive. Sending dozens of bytes over the network to switch on a light is not very efficient. A proposal has been made to compress these messages between two gateways, that looks promising to be implemented in CARUSO [8].

It's worthy to mention that there exists a candidate for successor to UPnP - the Device Profile for Web Services (DPWS), sometimes called UPnP V2.0. It seeks to close the gap between the Internet based Web Services standard and the more on local networks restricted UPnP. DPWS includes missing device support and plug and play capabilities for web services, which on their terms add some of the missing features to UPnP. Time will show if DPWS reaches the wide acceptance of UPnP [9]. The gateway plug-in concept allows for CARUSO DPWS bundles.

4.3 Security and User Access

A network, which interconnects devices of a building, raises the security question. While native security is missing in UPnP, the forum delivered missing DCPs in form of "Device Security" and "Security Console". Device security secures a UPnP device while the console provides access to secured devices and manages access rights. Unfortunately, these DCPs came much later, are optional and not very easy to understand and implement. Most of the UPnP devices today ignore them. Well, in home environments, secure light switches may not be an important feature. And even in commercial buildings, passwords are often transmitted as plain text and access mechanisms can be overcome with common field bus tools. Still, our solution needs a concept of users and a way to protect the building from unwanted access.

In general, the premise is that the device network is much less accessible than the control point network. Once the client is part of the control point network, it needs to authenticate to the server. This is where UPnP device security enters the scene. As all the traffic flows through the server and the device network is (more or less) save, the authentication needs to be done only once, based on the server's security service. The server may act as security console to the device network, to configure devices implementing UPnP's device security.

The user description and access right management is a compromise between the highest possible security solution and the one with lowest configuration efforts. For CARUSO, no individual users are defined but four typical roles with their own access schemes based on access levels. Access control information is stored on the server. The smallest access entity is an UPnP action. Discoveries and searches are currently not secured. The server holds the predefined access levels for all actions of all known UPnP DCPs. A global policy exists for all other (non-DCP) devices (allow all, deny all). As an example, this makes it possible to restrict access for a standard user to devices he could also control using conventional controls (e.g. light switches). If the client is aware of its location, access could be granted on the condition of physical presence in a room. While in those scenarios, no configuration is needed, introducing new user levels, new device classes or differentiating between instances of individual users or devices is possible, but results in human intervention. Decisions on the server are simple: If an authenticated SOAP action is received from the CP network, the access level of the user and the action are compared. If the access level of the user is equal or higher than the one of the action, the action is forwarded to the device.

4.4 User Tasks and Services

Users approach the system with certain tasks in their mind. These may be as simple as "turn on the light" or "set the room temperature to 20 degrees centigrade". But they can also become more complex, "watch a DVD" or "prepare Room 302 for a presentation meeting". The system places services at the users disposal to achieve her goals. The granularity and characteristics of these services greatly affect the usability.

On one end, an ideal system knows a minimal number of complex, intelligent high-level services, in which each service exactly maps to a user task. A minimum of interaction steps is required to finish a task. However, the chance is high that a one-to-one mapping cannot be achieved and thus the system becomes inflexible. A service that almost does what the user intends is probably more annoying than helpful. On the other end stands a system with many small and simple low-level services that could, in an intelligent orchestration, achieve any user task. Here, the user needs to know a number of these services and apply them in the correct order, which reduces the ease of use considerably.

Fact is that, in general, the user is not interested what devices are involved, he just wants to get the job done. Most of today's remote controls pose device-centric views on the user. That is helpful for a building engineer reading a parameter out of a device. However, most of the time, users would prefer to see the system as a collection of convenient services that closely resemble their needs.

CARUSO takes both views into account. The selection criterion is the role of a person in the building (she can have more than one role, but not at the same time). For this reason, roles can be defined (standard user, advanced user, fitter, engineer, administrator etc.). The role of the user defines the view and the security level. The device controller needs to know about these roles and their properties and filters devices and services the user is not allowed to see.

Besides of (UPnP) low-level services, CARUSO provides high-level services to its users. They come in the form of dynamically loadable plug-ins handled by the device controller. These services may be built into the controller already or fetched from a remote services server. A high-level service is an executable (OSGi bundle) that runs on the DC, triggered over a CP (or from another DC). The engineered service uses the DC for device discovery and executes low-level services to achieve a certain goal (other high-level services may also be involved). It's left up to the implementer of the service how intelligent the low-level services are chosen and aggregated. High-level services are published over UPnP too. A new UPnP device - the "Services Server" - adds those high-level services as its UPnP services. The services server may have a connection to the Internet and act as gateway to introduce external web services to the local UPnP system. In addition, the CP itself provides a facility for the user to record macros, store them locally and replay them later on. These are simple macros that allow to sequentially executing low-level functions on different devices without the ability to take influence on the timing or the execution flow.

4.5 User Interface

The user interface represents the system to the user and thus is naturally a very important factor for the acceptance of any control system. Although many possible human machine interaction schemes exist [10], CARUSO sticks to a 2D graphical user interface (WIMP) that can be shown on

Figure 2. Control Points: Designstudy (left) Smartphone QTek S200 (right)

the selected mobile clients. Also for the level of adaptation, a reasonable choice had to be made. The system adapts to: Display size, display resolution, input modality, user role, user language, computing performance and network bandwidth. The conclusion was made that dynamic user interface generation mechanisms best meet these requirements.

We know that the UI server sends a UI file to the control point. But its format is still open. Two main criteria influence this decision: (1) where the generation process takes place (2) when it does so. In the solution proposed here, the CP generates the UI and it does so on the fly at runtime. Other factors to consider in our UI design are the proper separation of the control application and the UI presentation/logic, a presentation that goes behind textual UIs or fixed standard widgets and usage of the same technology for all control points, if possible.

In the chosen solution, the control point gets a file written in a User Interface Description Language (UIDL). It's a declarative description of the user interface. The CP parses this file and generates the final UI out of it. Numerous such languages have been created already (XIML, UIML, XUL, XAML, SVG, the authors counted 25 up till now). Besides most of them are in XML, they vary greatly in what they can do and how they do it (e.g. level of abstraction). The user interface description should build the bridge between the purely functional service description and the final UI. It must provide features like grouping of user interface elements, element hierarchies, internationalization etc. that are needed to build a sophisticated UI. Still, it should not make statements in the sense of "place a button with height/width to position x/y" as the decisions on what widgets to take and where to place them is left up to the control points. It best knows about its own possibilities to render the UI. The XAML language has finally been evaluated. It is an integral part of the Microsoft WPF (Windows Presentation Foundation). As the standard UPnP device is not likely to send this file, a UI server bundle generates the XAML description. In the original XAML approach, a designer draws the GUI in a design tool and exports it as XAML file. The file is then handed to the developer and compiled together with C Sharp code to produce a .NET binary. This separates the design process from the programming works. The link between the UI and the code is done over event handler information in the XAML description. So for a button, there is an event "Click" that points to the name of a C Sharp method containing the code to handle the event. In CARUSO, we actually take the XAML file and parse it on the control point. And the event handlers in the XAML point not to C Sharp code but to UPnP actions executed when a user interacts with the according UI elements. The file does not contain absolute location coordinates and sizes of elements and so on. The control point makes these decisions when it gets the XAML file and renders the UI.

As stated, .NET has been evaluated as technology for the control application and the rendering. Using this "thick client" technology on the mobile device was also influenced by an evaluation of other state-of-the-art UI technologies (HTML/CSS/JavaScript, AJAX, X-Forms, XUL, XAML, SVG, Flash(Lite), Flex, OpenLaszlo, .NET CF). In fact, we would have preferred a "zero deployment" solution, which means no installation procedure on the client. Unfortunately, at the time those investigations were done, we found that current mobile browsers and plug-ins are not ready to deliver the performance and the feature set needed for the UI we had in mind. Future generations of devices, browsers like Opera or iPhone's Safari and mobile Flash or Silverlight plug-ins may change that.

5 Scenario and Results

This section presents a short user scenario as a summary on how the system works.

1. A user enters the building for the first time. She has her NFC enabled smartphone (control point) with her. In the lobby, she takes it and places it near the registration device provided. She is registered and her phone is now configured to access the building network.

2. She gets an SMS with an URL to the installation server. She chooses to download, install and start it.

3. She reaches a room, the control point starts sending UPnP discovery messages. Over the device controller, acting as UPnP proxy, several devices respond: Two light switches, a temperature sensor, a beamer, a DVD player and a UI server. The CP automatically connects to the UI server. It delivers a list of URIs that lead to the UI descriptions of all services available to the standard user.

4. The control point lists the service names and a short description for each. It is possible to apply filters to the list so the user can further specify which services she wants to see. The standard user has the choice between three services: A lighting service, a temperature service and a service to watch a DVD on the beamer.

She decides to select the DVD service and presses on its icon.

5. The UI server returns a description of the UI for this service that is parsed by the control point. Taking the characteristics of the control point hardware into account, it renders the service on the smartphone display. There are controls to start/stop/pause the movie and adjust the volume. Additionally, there is a slider to dim all lights in the room and a possibility to control the jalousies. Beamer and DVD player are turned on.

6. She uses the slider to dim the lights. The DCP UPnP control message reaches the device controller, from there it is sent to a gateway and translated into a command to the LON node that controls the lights. Within 200 ms, the light visibly changes its state and the control point updates its light icon.

7. She puts the DVD into the player and presses play on her smartphone. The UI (and the beamer) indicates that the player has received and initiated the command.

A prototype has been built with which the scenario above can be reproduced. In a first step, a simulation system was used to represent a simple room scenario with lamps, sliders and a media renderer. Figure 3 shows the simulation that runs in a Flash movie for presentation and a Java based server to implement the logic and provide UPnP wrappers for CARUSO. In the front, a wireless based device shows the first tests with a ZigBee gateway bundle. Figure 2 depicts of the view the standard user when dimming a light as a design study (left) and a S200 QTek smartphone when controlling a media player (right). The next step will be to include a gateway for a LON WebServer with a SOAP interface including tests in a real building environment.

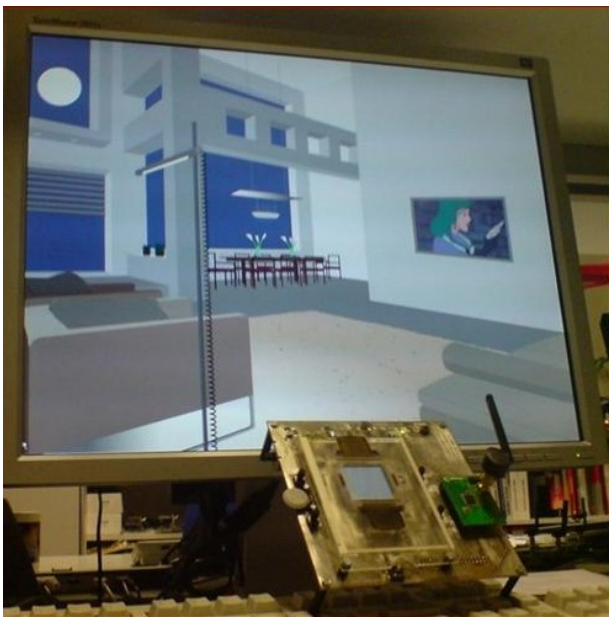

Figure 3. Room Simulation for CARUSO

6 Related Work

Many have contributed valuable ideas, concepts and technologies that were included into CARUSO. Some quite sophisticated universal remote controls for homes can already be bought and there are research projects that come close to what this proposal suggests. Apart from common universal remote controls and remote control software for PDA, the most notable products on the market for home environments are the **Philips Pronto Series**, **Logitech's Harmony**, **AMX R4 ZigBee** and products from **Xabler** and **Nevo**.

The **PECo (Personal Environment Controller)** [11] comes with an intuitive and novel interaction metaphor that provides 3D-sights of the rooms controlled on a PocketPC. The user can thus simply drag his electronic media like a presentation file and drop it to the beamer device on the screen. The creators of PECo have also proposed a generic UPnP architecture for their CP [12] and mention EIB. They have implemented their concept in an ambient intelligence meeting room. The 3D presentation needs a detailed model of the physical environment and the devices in it. To create and deploy these descriptions still needs too much effort for the CARUSO requirements. **PUC (Personal Universal Controller)** [13] [14] is a peer-to-peer approach to control devices over various standard clients. The UIs are generated on the client with help of a user interface description in an XML dialect that was developed for this project. A prototype has proved the concept on a Palm device and a smartphone. The new **URC (Universal Remote Console)** [15] standard proposes a "Protocol to facilitate operation of information and electronic products through remote and alternative interface and intelligent agents". The first attempt to really standardize remote UIs. The creators of the standard have proposed an **URH (Universal Control Hub)** architecture based on UPnP using their concept of UI Sockets and task based user interfaces that is close to CARUSO [16]. Proposals have been made on how to integrate UPnP into building automation environments [6] [7] [8]. A lot of work has been done concerning smart services, intelligent device ensembles and how to get to them (e.g. using pattern matching or distributed intelligent agents) [9] [17] [25] [26]. Model-driven user interface techniques have delivered many ideas on how to dynamically generate user interfaces. Interesting work has been done concerning multi-modal [18] and multi-target user interfaces [4] also especially related to mobile devices [19]. **iCrafter** [20] provides a service framework for ubiquitous computing environments that supports UI selection, generation and adaptation. It also includes a service concept with patterns for on the fly aggregation of services. CARUSO related research is being conducted in the projects **AMIGO** [21], **TEAHA** [22], **DynAMITE** [23] and **Pebbles** [24].

7 Conclusion

This paper introduced an adaptive architecture for an integrated network in a home- and building environment. Taking the special requirements of such a system, we identified all network participants and proposed a topology to connect them. The resulting IP network provides a modular server concept with distributed OSGi bundles for tasks such as device proxies, access control, UI generation, field protocol plug-ins and high-level services. It also includes a restricted peer-to-peer scenario without servers. UPnP introduces the required plug and play capabilities and the architecture assures that standard based target devices can be integrated into the system without modification. Dynamically generated, bi-directional user interfaces and a service based approach provide the flexibility and user friendliness critical for the acceptance of such a system. We think that the proposed pragmatic concept enables a new generation of control systems for home- and building environments that can be realized not too far from now.

8 Acknowledgements

This work is funded by Siemens Switzerland Ltd, Building Technologies Division and the KTI/CTI programme of the Swiss government.

References

[1] UPnP Forum, (July 2006), UPnP Device Architecture [Online]. Available: http://www.upnp.org/specs/arch/UPnP-DeviceArchitecturev1.0.pdf

[2] OSGi Alliance, (), The Dynamic Module System for Java [Online]. Available: http://www.osgi.org/

[3] iHomeLab, (), Intelligent Living Begins Here [Online]. Available: http://www.ihomelab.ch/

[4] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt, "A unifying reference framework for multi-target user interfaces", *Interacting with Computers*, vol. 15, no. 3, pp. 289–309, June 2003.

[5] DLNA - Digital Living Network Alliance, (2007), DLNA Overview and Vision Whitepaper 2007 [Online]. Available: http://www.dlna.org/news/DLNA_white_paper.pdf

[6] W. Kastner and H. Scheichelbauer, "UPnP Connectivity for Home and Building Automation", in *Parallel and Distributed Computing and Networks*, 2004.

[7] D. K. A. Klapproth and T. Peter, "Lowcost Wireless Webserveer", in *ZigBee Entwicklerforum 2005 Munich*, April 2005.

[8] S. Knauth, D. Kaeslin, R. Kistler, and A. Klapproth, "UPnP Compression for IP based Field Devices in Building Automation", in *11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'06)*, September 2006, pp. 445–448.

[9] M. Hellenschmidt and T. Kirste, "SodaPop: A Software Infrastructure Supporting Self-Organization in Intelligent Environments", in *IEEE International Conference on Industrial Informatics (INDIN'04)*, 2004, pp. 479–486.

[10] A. A. N. Shirehjini, "Klassifikation der Human-Environment-Interaction in intelligenten Umgebungen", *Informatik 2006. Informatik fuer Menschen. Beitraege zur 36. Jahrestagung der Gesellschaft fuer Informatik e.V., Bonn : Gesellschaft fuer Informatik*, vol. 2, pp. 382–389, 2006.

[11] A. A. N. Shirehjini, "A novel interaction metaphor for personal environment control: Direct manipulation of physical environment based on 3d visualization", *Computers & Graphics, Special Issue on Pervasive Computing and Ambient Intelligence*, vol. 28, pp. 667–675, 2004.

[12] A. A. N. Shirehjini, "A Generic UPnP Architecture for Ambient Intelligence Meeting Rooms and a Control Point allowing for integrated 2D and 3D Interaction", in *Smart Objects and Ambient Intelligence. SOC-EUSAI'05*, 2005, pp. 207–212.

[13] J. Nichols, B. A. Myers, M. Higgins, J. Hughes, T. K. Harris, R. Rosenfeld, and M. Pignol, "Generating remote control interfaces for complex appliances", in *15th annual ACM symposium on User interface software and technology*, 2002, pp. 161–170.

[14] J. Nichols and B. A. Myers, "Controlling Home and Office Appliances with Smartphones", *IEEE Pervasive Computing, special issue on SmartPhones*, vol. 5, pp. 60–67, July 2006.

[15] G. Zimmermann, T. Nixon, M. Beard, E. Sitnik, B. LaPlant, S. Trewin, S. Laskowski, and G. Vanderheiden, "Toward a unified universal remote console standard", in *Extended Abstracts on Human Factors in Computing Systems, CHI2003 Conference on Human Factors in Computing Systems*, 2003, pp. 874–875.

[16] G. Zimmermann, G. Vanderheiden, and t. . C. Rich, ()

[17] M. Valle, F. Ramparany, and L. Vercouter, "Flexible composition of smart device services", in *The 2005 International Conference on Pervasive Systems and Computing(PSC-05)*, 2005.

[18] M. Valle, F. Ramparany, and L. Vercouter, "Dynamic service composition in ambient intelligence environments: a multi-agent approach", in *First European Young Researcher Workshop on Service-Oriented Computing*, 2005.

[19] M. Vukovic and P. Robinson, "Adaptive, planning-based, Web service composition for context awareness", in *Second International Conference on Pervasive Computing, Vienna*, April 2004.

[20] P. Shroff and J. Winters, "Generation of Multi-Modal Interfaces for Hand-Held Devices Based on User Preferences and Abilities", in *IEEE D2H2 Distributed Diagnosis / Home Healthcare*, 2006.

[21] J. Eisenstein, J. Vanderdonckt, and A. Puerta, "Applying model-based techniques to the development of UIs for mobile computers", in *6th international conference on Intelligent user interfaces*, 2001, pp. 69–76.

[22] S. R. Ponnekanti, B. Lee, A. Fox, P. Hanrahan, and T. Winograd, "ICrafter: A Service Framework for Ubiquitous Computing Environments", in *3rd international conference on Ubiquitous Computing*, 2001, pp. 56–75.

[23] IST-AMIGO, (), Ambient intelligence for the networked home environment [Online]. Available: http://www.hitech-projects.com/euprojects/amigo/

[24] IST-TEAHA, (), The European Aplication Home Alliance [Online]. Available: http://www.teaha.org/

[25] DynAMITE, (), DynAMITE Dynamisch Adaptive Multimodale IT-Ensembles [Online]. Available: http://www.igd.fhg.de/igd-a1/dynamite-project/

[26] Pebbles, (), PDAs for Entry of Both Bytes and Locations from External Sources [Online]. Available: http://www.pebbles.hcii.cmu.edu/