Open Home Networks: the TEAHA approach

Hylke W. van Dijk
Hans J. Scholten
University of Twente
Enschede, The Netherlands
h.w.vandijk@utwente.nl
j.scholten@utwente.nl

Álvaro Tobalina
Víctor García Muñoz
Telefónica I+D
Madrid, Spain
alvarotv@tid.es
vmgm@tid.es

Stéphane Milanini
Antonio Kung
Trialog
Paris, France
stephane.milanini@trialog.com
antonio.kung@trialog.com

***Abstract*— The current trend for home appliances is networking. Although more and more of these appliances are networked, there is not a standard way of interaction, which restrains the development of services for in-home networks. The lack of standardisation is partly due to a legacy of business interests; white goods, audio video equipment, security, and personal digital appliances all have a different background and have different business models.**

Rather than profound standardisation we propose secure seamless interworking of technologies, applications, and business interests. In this paper we present an architecture which is embedded in legacy technology. Our approach combines known design patterns, augments existing technology, and facilitates so-called business clusters. Further, we discuss a prototype implementation that integrates as an example OSGI, ZIGBEE, and UPNP technology with CECED (white goods) business interests. The work reported in this paper has been executed in an international industrial project[1]: TEAHA.

I. INTRODUCTION

Modern homes are equipped with many electronic appliances. Nowadays many of these appliances can communicate in one way of another with other appliances: modern home appliances are networked. Creative minds can think of uncountable exciting applications and services by simply combining the huge potential of these networked appliances. However in practise implementing these applications is cumbersome and often requires a great deal of application specific *glue*.

Standardisation of interfaces and standardisation of protocols may help to ease the development of novel applications and services. A profound standardisation process, however, will be too time consuming and too inflexible to serve the industrial needs of today. The alignment of historically very different appliances and their networked solution is a difficult task because of the large installed base, the existing portfolio and market shares, and simply because solutions have been optimised for their typical application area. As an example, there is no need to implement high-data throughput remote control devices for controlling audio-video appliances over infra-red. An additional problem is that compliance to a new standard involves significant investments and it is hard to estimate the return value for unproven services. Standardisation efforts serve an inherently static purpose, typically induced from a predefined business model that brings together partners with mutual interests. In case of home appliances there are many fields of interests, such as white goods, audio-video, telecommunications, and house monitoring and control that all use their own, often proprietary, set of protocols. This makes it hard to define commonalities, while at the same time we can envision many novel emerging services and applications, disrupting existing business models and coalitions.

In the TEAHA project we envision a distributed home gateway that allows for seamless interworking of technology domains and business clusters. Devices in a technology domain share a mutual technology for communication and interaction, e.g., TCP/IP. Similarly appliances and services in a business cluster share a mutual interest, business wise or otherwise, however they are defined independently from the applied technology. Domains and clusters are formally defined a priori, yet the TEAHA architecture allows them to participate in the home network, by providing them seamless access to other domains and clusters as well as facilitating semantic translation of protocols.

TEAHA [23] is a European project with partners from industry and academia. In TEAHA we took a pragmatic approach. The TEAHA architecture and prototype implementation reuses existing and proven technology, including the implementation and fine tuning of known design patterns.

This paper is organised as follows. We start with articulating the general requirements for any architecture to accomplish the goals as set forth above. Then we introduce the building blocks in Section II. Section III describes the architecture in detail and presents implementation details. We discuss the resulting system in Section V. This section also includes future and related work. Section VI finally concludes this paper.

The main contributions of this paper are:

- creating an architecture for seamless interworking,
- that integrates basic building blocks,
- and assess its feasibility in a commercial setting.

In this paper we propose an architecture and assess its feasibility by means of (preliminary) profiling. Our assessment is in view of embedding the system in home appliances, which are constrained by their resources as well as the associated costs for updating the installed base.

[1]This work is sponsored in part by the European Commission (IST-507029 priority 2.3.1.8)

A. Requirements

The overall goal of the project is to create seamless interworking among services from different technology domains and to facilitate controlled interaction among services from different business clusters. In order to accomplish this generic goal, a feasible architecture shall embed legacy devices and legacy services. Seamless interworking requires a form of *semantic* abstraction in order to bridge the different underlying technologies, including their applied protocols. In addition it requires application specific *syntactic* transformations to let different business models interact.

Interaction alone, is usually insufficient for a successful business alliance. There are at least two additional constraints to create a controlled interaction: security and near zero-configuration. Security involves the secured interaction among services and users; communicating parties must be able to rely on the authenticity of their peers and may require encryption of their data with an agreed protection level to prevent eavesdropping. A service system that implements near zero-configuration allows for dynamically grafting, pruning, and updating of services almost without human intervention. Because of the security and near zero-configuration aspects, the architecture shall provide hooks to guarantee a policing mechanism when desired. The mechanism details, e.g, the encryption method, shall be specified on a session basis with an authenticated source, authenticated destination, a controllable duration (time or the number of discrete events), and a security level (plain, authenticated, or encrypted). In particular the application developer should indicate that policing is required but the implementation must be transparent. Near zero-configuration has to be accomplished in order to prepare the system for changes with a small burden on the end user of the system.

Special care has to be taken into the flexibility of the framework. Business opportunities come and go; time-to-market is as important as timely adaptation to changing business needs. The business clusters must be flexible in defining and updating their agreed protocol.

Finally, the architecture and corresponding implementation shall support the following identified business clusters

- Tele service providers: UPnP programming model and OSGi [1] based triple play (Telephone, Broadband Internet, and television)
- CECED [21]: white goods business cluster
- DLNA [22]: digital living network alliance for the audio-video business cluster

Typical technology domains include

- Wired Ethernet (IEEE 802.3)
- Wireless Ethernet (IEEE 802.11.x)
- Wireless personal area networks (IEEE 802.15.{1,4})
- EHS [14], low data-rate power line communication.

II. Concepts

In this section we introduce individual concepts and components of the architecture. The first subsections describe communication concepts, gradually introducing a more semantical interpretation. The last section presents a number of frequently used design patterns in the prototype implementation of Section III.

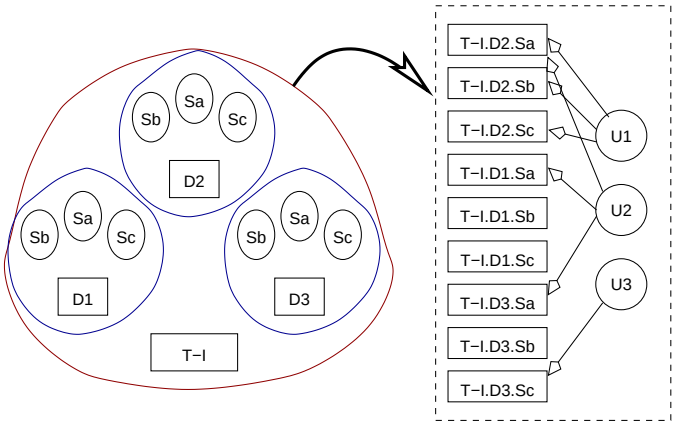


Figure 1: Technology domain

A. Ubiquitous Access

Ubiquitous access bridges technology domains and authenticates services and users. Bridging is conceptually accomplished by providing service access points in the virtual domain of potential users. The subsequently binding of users and services is traditionally the field of service discovery [10].

For the authentication of services and users we use a two stage approach. The first stage connects services or users to devices in a unique way, the second stage relies on a tamper proof secure engine of a device; sometimes referred to as a trusted module. A well known implementation of authenticated device to device protocol is the Diffie-Hellman station-to-station protocol [4].

Consider the diagram of Figure 1. Here we have a technology domain T-I with devices D1 to D3, each providing a set of services. The objective is to create the situation of the right-hand side of the diagram, where users U1 through U3 have ubiquitous access to these services. The accomplishment must be independent from the specific service discovery technology of domain T-I but with proper access regulation in place.

1) Service discovery: The diagram of Figure 2 presents an example of a generic model for service discovery. It is an extended version of a client-server model with a service agreement, see e.g. [20].

In this model, a manager retrieves service *descriptions* from a service (e.g. Sa). At a certain time instance a user (e.g. U1) puts a *request* to its managerial network, which consists of reachable explicit managers (e.g. M1) and implicit managers of reachable services (e.g. Sc). Given the current available set of descriptions the manager establishes a match, which is returned to the user as an *offer*. The match making processes involves a translation of service descriptions into the operation space of the user followed with an optimisation process. The translated service descriptions form the operation space, whereas the request holds the constraints under which

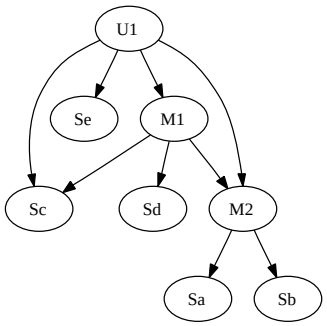


Figure 2: Hierarchically structured model for Service Discovery.

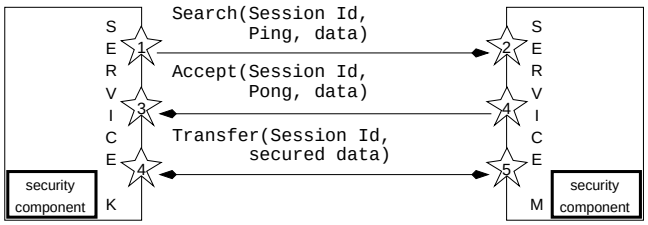


Figure 3: Secure service discovery.

the multi dimensional optimisation must be performed. The resulting offer is a set of Pareto optimal operation points; Pareto optimal points are those points in the operation space for which no points exist that are more optimal in all dimensions. In the use phase of the process the user provides an *agreement* to the manager, which in turn *configures* the service for use according to the current agreement.

The example model of Figure 2 includes two popular complementary discovery models: a broadcast and a directory based model. In the broadcast model, users contact every manager/service in reach to discover their services. In a directory based model, services register with a directory manager, whereupon users query the directory manager to retrieve service information. Hybrid models are used to increase the efficacy and agility of service discovery. For instance, in a situation with relatively static services and a reliable network, a directory based service discovery is known to be efficient (in terms of network load). In a more dynamic situation where services enter and leave the network frequently, the broadcast model gives the best coverage. In a situation with many communication deficiencies but a reasonable static service network, maintaining a network of directories (and backup directories) shows good results [18].

The typical situation for a TEAHA application is an in-home network, with a relatively static service network and a reasonably reliable communication network. The virtual domain of Figure 1 therefore uses a single directory to store service references. In case of supporting multiple technical domains it is the responsibility of the directory to delegate, whenever appropriate, service discovery requests from a user to the respective technology domains. Note that the single directory is a concept, its implementation may well be by means of a distributed directory in which the individual directories form a peer network as found in GSD [2].

2) Secure interworking: Service managers (and user managers) use a security engine to facilitate authenticated and possibly encrypted communication among a service and a user. In fact security becomes an integral part of the service discovery process, as explained in the following scenario. Let K be a service that actively searches for a service M. Service M will acknowledge the request provided that service K can be authenticated (is properly registered). Once accepted they decide to exchange messages in a secure way. The scenario is outlined in the diagram of Figure 3, which is an implementation of the station-to-station protocol [4] with a piggybacking service discovery protocol.

The steps are as follows: service K sends a `Search` request that includes a so-called *Ping* message. The Ping entails the key agreement request of service K and an authenticity proof, which allows service M to verify that service K posed the request. Once authenticated, service M replies with an `Accept` message that includes a *Pong* message. The Pong entails the key agreement response of service M and an authentication proof. After the (authenticated) receipt of the Pong message services K and M share a secret session key, which can be calculated from the exchanged key agreement information. The session key is used to encrypt data messages in the further communication (`Transfer`) among services K and M.

Given an authenticated user and an authenticated service, the system must decide upon the interaction method. In case of TEAHA this decision tree is implemented by means of a hierarchical *registry*. New services and users get registered[2] with a directed registry tree. Each service and user comes with a level at which it is entered. A user has access to services that are reachable while descending the hierarchy of the (acyclic) registry tree. Users who are allowed to register at the root of the registry gain access to all services. The access method (plain, authenticated or encrypted) depends on the *weight* of the path from the user vertex to the service vertex.

More fine grain control can be gained when we associate roles to users and services. This way services and users can virtually appear at multiple instances in the tree, which gives users access to multiple clusters of services and services can appear in multiple clusters. The weight of the path from a user to a service is a maximum operation over the edge weights in the path; each edge is associated an ordered weight from the increasing set: plain, authenticated, encrypted.

As an example of a registry tree consider the tree of Figure 4. Here the root user has access to all services. User U1 for instance can compose applications that involve services Sb through Se. User U3 and user U1 both can gain access to service Se, yet U1 requires authentication whereas U3 has direct access; provided of course that the users successfully located the service before.

[2]The registration process and corresponding configuration management is an unsolved issue.

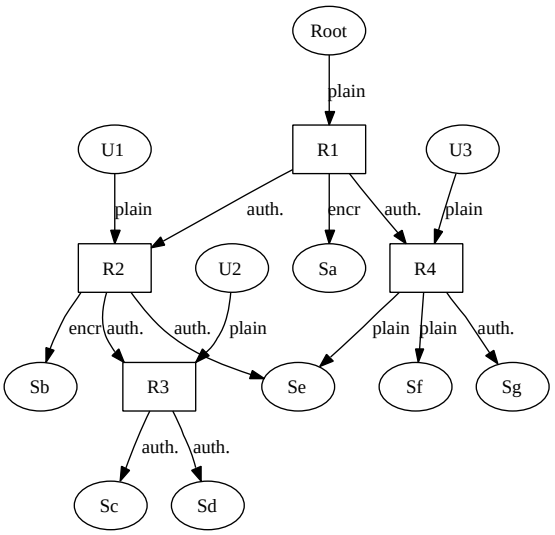


Figure 4: Registry tree example.

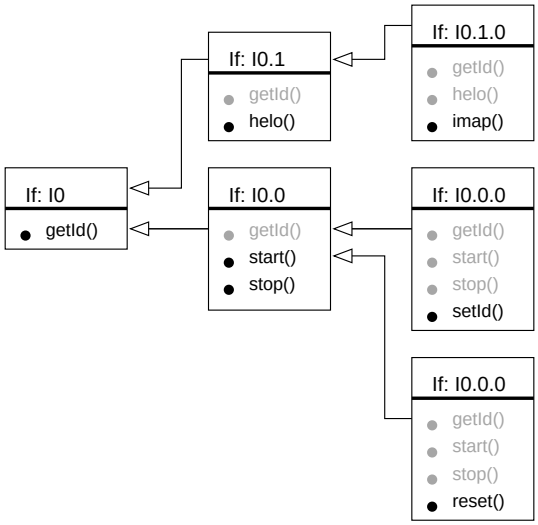


Figure 6: Transformer interface hierarchy.

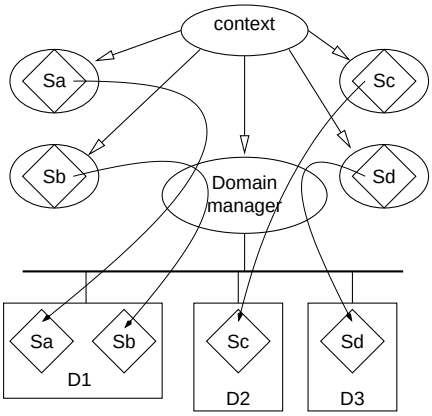


Figure 5: Surrogates.

B. Domain extension

In the previous subsection we have presumed seamless interoperability of services, users, and their managers. Here we extend the virtual domain by introducing surrogates and transformers. A surrogate is a representation of a service (its subject) in a remote technology domain. Surrogates mimic the original interface of their subject service. Transformers go beyond surrogates in the sense that they adapt the interface of their subject. In this case the adaption is defined by a business cluster.

1) Surrogates: A surrogate provides an interface to a remote object. The typical case has been drawn in Figure 5. Here the Domain manager acts as the interface between the context and the technical domain in which service Sa through Sd reside. Whenever the domain manager detects a new service it registers a service with the context with a similar interface (set of methods and fields) as the interface on the technology domain.

Service Sa through Sd can now be discovered and deployed by any user in the context domain. However, the user must be familiar with the protocol of the technology domain in order to successfully interact with the actual service.

2) Transformers: A transformer extends a surrogate in that it adapts the interface of the surrogate to provide an interface that is more natural to the user in the context domain of Figure 5.

Transformers are typically hierarchically structured. Consider as an example Figure 6. Here we show an interface that can be implemented by an adapter. The most straightforward interface only implements a `getId()` method. Depending on the precise nature of the service in the technology domain (e.g. see Figure 5) the domain manager selects an appropriate adapter to enrich the available functionality in the context domain. As an example suppose the technology domain provides a mail transport agent (MTA). Then depending on the capability of the MTA we can enrich the adapter with a plain `ping`, a query method, `helo`, or we may even retrieve messages from the service via the `imap` protocol.

In addition to the service itself, the objective of the business cluster can play a role in selecting an appropriate adapter. As an example, in the lower branch of Figure 6 the business cluster may opt for an adapter with either the `reset` of `setId` functionality.

C. Interoperability

So far we established interworking but only to the point where either the native technology domain of the service defines the interface (surrogates) or the business cluster defines the interface (transformer). In this section we regard TEAHA as a business cluster. TEAHA is a consortium of telecommunications operators, audio/video manufacturers, and white good manufacturers. The consortium has been inspired by the upcoming UPNP standard for the definition of a business cluster interaction model. The TEAHA business cluster uses an event-based programming model and an XML based attribute value pair model to describe services, including their interfaces

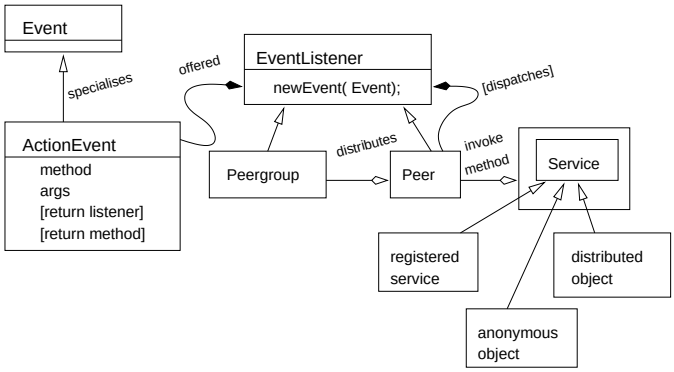


Figure 7: TEAHA programming model.

and their capabilities.

Figure 7 outlines the TEAHA programming model. The model supports a network of peer groups in which events are dispatched and distributed. A peer group that receives an event distributes copies of the event to its peers, which maybe a peer group again.

Peers handle events. A particular event is an `ActionEvent` which specifies a method that is to be invoked on the subject of a peer. The event also specifies a return event and the peer group to which this event has to be dispatched by the handling peer. An `ActionEvent` essentially creates a synchronous method call between a user and a service. Special care has to be taken in order to prevent out of order processing and overtaking of events by the peer group network. A robust method using relative clocks has been described in [15].

D. Patterns

In this section we review a selection of design patterns that we use to implement the introduced concepts.

1) Proxy pattern: The proxy pattern, [5] and Figure 8a, guards a subject. A client uses a proxy object in the same way as the actual subject. The proxy then decides either to invoke the real subject or to raise an exception, for instance in case of a security violation.

2) Factory pattern: The factory pattern, see [5] and Figure 8b, creates a dynamic way of creating real products. The actual product will match a set of predefined attributes. In our case we use a factory method to create a proxied subject if policing is required.

3) Composite pattern: The composite pattern, [5] and Figure 8c, creates a recursive set of components. Invoking the operation method `Op()` on a composite results in an avalanche of operation invocations of its children. Invocation of `Op()` on an entity does the actual work.

4) Custodian pattern: A custodian pattern, see [11] and Figure 8d, is a pair of access and exit points. Together they implement a policing functionality. Since the user and the service will never communicate directly the custodian can enforce the implemented protocol. Custodian access points and exit points are in fact proxy implementations, which create transparency for their clients: a user and a service.

5) Driver Extension Pattern: OSGI [1] defines a device attachment algorithm to enable flexible attachment of extended drivers to idle devices. The algorithms locates drivers and then gets a matching value. It consequently selects and attaches the driver that reports the highest matching value. The pattern is also known as an extension framework [8].

III. INTEGRATION

The base platform for our prototype implementation is a service platform, which allows for dynamic loading, unloading, and configuration of services [8]. The OSGI middleware [1] provides a small core that allows the different components of a project to cooperate seamlessly inside a Java Virtual Machine. We refer to this core as the OSGI context. OSGI is an obvious choice to create a near zero-configuration service platform (see also Section II-A) because of its mechanism for the deployment of services; it goes beyond the concepts of devices and it allows for different components of the control software, while it unites everything in a single service directory. At the same time the OSGI context facilitates the implementation of software applications for various sources. Currently OSGI includes features such as a driver search system, which is an essential ingredient for supporting flexible business models.

A. Ubiquitous access

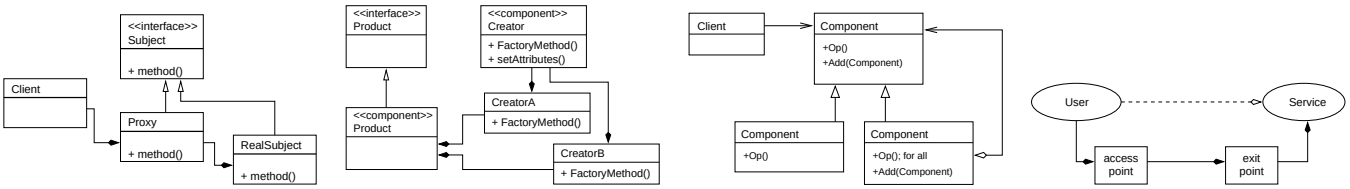
Applications, services, and users that reside in the same *context* will be able to interact. Whether a service representation is a surrogate or transformer is irrelevant in this discussion. Policing however requires a single point of access. In Figure 8d we described the general – distributed – case, which requires separate access and exit points. Here we describe the simple case with one OSGI context.

Figure 9 outlines the approach. The directory represents the OSGI context, a TEAHA user is an authenticated user and a TEAHA manager is an authenticated manager of services; their authentication is securely stored in their wallet. Plain users and managers cannot show proper identification, however, we still would like them to interwork securely with TEAHA users and TEAHA managers.

Here the plain manager would register a service a with the OSGI context, whereas a TEAHA manager would register a service B with the TEAHA context. In turn the manager registers a factory method (see Section II-D) with the OSGI context. When a user requires a reference to the service B, the factory method will dispatch a proxy (see Section II-D) of the actual service. Access to service B is now controlled by the proxy, implementing effectively policing, as required.

Similarly, a TEAHA user may request policed access to a service. In case of a TEAHA user that requests a reference to – the uncontrolled – service a through the TEAHA context, the context will return a proxy of service a, similar to the aforementioned factory method.

The simple custodian has been implemented using a proxy pattern. Each time an authenticated user accesses an authenticated service, the custodian queries a policing registry to



(a) Proxy. **(b)** Factory. **(c)** Composite. **(d)** Custodian.

Figure 8: Design patterns.

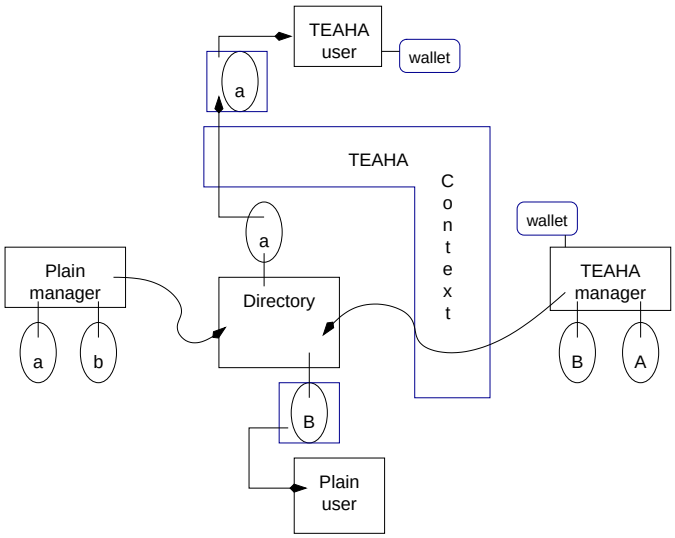


Figure 9: TEAHA secure service discovery

determine the desired protocol for interaction and the status of the current session. The user and service credentials are stored in the respective secure engines of the devices on which the user and service processes are executed, whereas the session status data is securely stored in secure engine of the device on which the custodian executes. Because of the single point of access, session management becomes feasible, without being intrusive or expensive.

B. Domain extension

Following Section II-B the implementation of different technology domains uses the model of surrogates and transformers with the help of the tools provided by the OSGI platform. Here we will present two different implementations, one for UPNP and another for ZIGBEE.

1) UPNP *:* UPNP is a well known standard that has been adopted by many vendors recently. Its use relies on the creation of drivers and control points. In [1, pp 503–528] a group of interfaces is defined to ease the implementation of a base driver, which is a bundle of Java classes. Some vendors developed proprietary base drivers but open source distributions are readily available. In TEAHA we adopted an existing one. The base driver creates and registers services with the OSGI directory, i.e., creating the surrogates for this technology domain. The current implementation uses a tracker of OSGI services (see [1, pp 391–402]) that for each surrogate registers a new TEAHA service that in addition follows the necessary requirements to fit the TEAHA programming model. With this extended interface a new service becomes a transformer of the UPNP domain.

2) ZIGBEE *:* ZIGBEE [24] is a set of protocols designed to be used by small low-energy wireless communication devices. A special purpose module has been used to connect the ZIGBEE protocol through a RS-232 protocol to an OSGI gateway. The ZIGBEE driver that registers services has been designed and implemented into OSGI. The driver operates in two stages. First it registers a surrogate that provides some basic information about the newly discovered device in the ZIGBEE net. This representation follows the set of specifications for a device as described in [1, pp 223–252] and has the necessary methods to transmit data to the serial port, but it does not implement device specific methods. Second the driver searches for a plug-in, as a suitable transformer. When found, the device is registered and a link is created with the corresponding surrogate or transformer, depending the capabilities of the selected plug in.

3) Cluster plug-in: The cluster plug-in implements the driver extension pattern to select dynamically the best matching cluster plug-in. In our prototype we created an extension manager that is integrated with an EHS driver. EHS is a power line communication protocol. The base driver works similar to the drivers described in the UPNP and ZIGBEE subsections above. It creates surrogate services that represent devices hooked on to the EHS power line.

In this case the surrogate implements an extendible transfer interface. The diagram of Figure 10 outlines an example of the extension in case of a CECED business cluster.

In the diagram the base driver, EHS proxy, queries the context to locate an EHS cluster chooser. In the example it locates a cluster chooser with three business clusters: CECED, FM and AV. The base driver offers each of the clusters a reference of the surrogate while calling their `match` method. In turn the cluster object will query internally a hierarchy of plug-ins providing them access to the surrogate. The better the match, the more information can be retrieved from the surrogate. The best match is a tree of plug-ins specifying the details of the surrogate and therefore opening the opportunity to use the service in more depth.

The depth of the match tree depends on two parameters: the functional match and the business protocol match. The func-

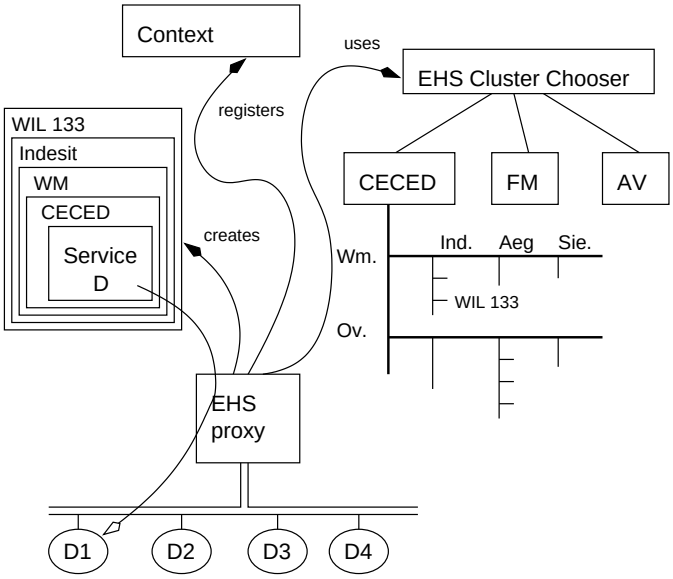


Figure 10: TEAHA cluster plug-in example

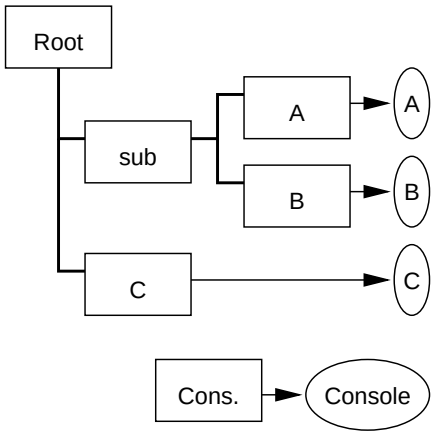


Figure 11: Short message system

tion match specifies whether a surrogate exposes an interface (functionality) that a plug-in can exploit, whereas the business protocol match regulates whether the plug-in is allowed to participate in the extension, security wise. Note that the actual policing is done as described in Section III-A.

An alternative view reveals that the tree of plug-ins represents a hierarchy of business clusters. At the top we have CECED, an alliance of white good manufacturers. Within in this cluster we recognise a cluster of washing machines, WM. This cluster is split up along manufacturers and finally along the cluster of a specific type. In this case the hierarchy adds more and more functionality. At CECED level we can identify, start, and stop the EHS device, at washing machine level we can read the phase of the current washing cycle. The manufacturer level opens up a functionality to update the firmware and finally at type level we give access to type specific functionality such as controlling the automatic detergent dispenser.

C. Interoperability

The programming model and corresponding XML description of services closely follows the UPNP model. This is a strategic choice for industrial partners. Note that, although the details differ this model has many similarities with ZIGBEE protocol and service profiles as well as the CECED protocol and service description.

The programming model uses events for communication among services. Events are distributed in peer group. Peers register with a peer group to receive events and peers interact with service objects; they effectively implement the composite pattern. Thus a peer that receives an asynchronous action event (see Figure 7) will invoke the specified action synchronously. If requested the peer will thereupon return an action event to the specified peer (or peer group).

Peers implement the proxy pattern for their services. It is irrelevant for a peer whether a service is a plain object or a registered service. The proxy pattern hides this. If a registered OSGI service is used then it is also policed.

Peers also have an XML description of the service they represent. On demand they will offer that. The description is a simple attribute value tuple. If the peer has been registered as a service, policing is also in place for the service description query. Our attribute, value tuple are consistent with the UPNP XML service description, although we do not mandate a full specification. Recall that many service descriptions are based on attribute, value tuples and queries on that. Examples include ZIGBEE, SLP, CECED. Others, such as SDP and GSD make use of a URN.

IV. PROTOTYPE EXPERIMENTS

In this section we present two prototype applications to demonstrate the feasibility of our architecture. A short message system (SMS) demonstrates the programming model and the interaction with the UPNP technology domain, whereas a clock synchronisation application demonstrates the business cluster plug-in technology interacting with the EHS technology domain.

The efficacy and agility of this prototype implementation has been derived from the prototype applications. The efficacy models the amount of used resources to execute the applications in terms of memory and processing bandwidth. The agility models the lag between the occurrence and completion of two events. The latter is an important parameter to assess the type (and quality) of applications that can be controlled by the current implementation.

A. SMS system

In this application we consider a short message system with three terminals A, B, and C. The terminals are accessible through their respective peers, which in turn have been registered with peer groups: root or sub (Figure 11).

In addition the application has a console, which is accessible through a console peer in case we request an acknowledgement. All peers and peer groups in the application have been

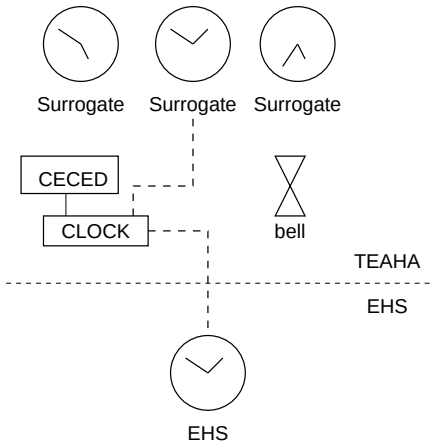


Figure 12: Clock synchronisation system

registered with the context. Thus from the console we can disseminate a message to any peer group or peer; in case of an acknowledgement we specify the control peer as the receiving peer, presenting the console as a terminal.

Further to the SMS terminals, any UPnP service that gets registered through UPnP driver becomes available through the TEAHA programming model. In effect one can send an action event from the console user to a UPnP service, while the actual dispatch of the event is policed by the custodian of the UPnP transformer.

B. Clock synchronisation

In this application we consider a "bell" service that acts as the master clock of the system. The bell service locates all possible clocks in the system and sends them their current time. One of the clocks in this application is a clock on the EHS power line communication technology that will be registered under the flag of the CECED business cluster.

The diagram of Figure 12 outlines the setup of the clock application. In the event that a clock becomes available on the EHS bus this event will be observed by the EHS driver. In turn the driver will query the EHS-CECED chooser to locate an appropriate plug-in for this service; in this case a generic clock plug-in suffices. The bell is a separate service that runs in the background. Once in a while it becomes active and locates all clocks and surrogates (or transformers) that registered a clock interface. Subsequently the bell service invokes the `setTime()` method to inform the clocks of the current accurate time.

C. Preliminary Profiling

Preliminary profiling of the prototype applications is done on a PC (1186 bogo Mips, 512 MByte) running a recent Linux kernel, Java 1.4, and Oscar 1.5 as the OSGi implementation. Furthermore the OSGi framework is equipped with `log4j` logging service, SWT widgets, jetty web server for service configuration and RxTxcomm for serial communication.

The typical prototype application shows a residential memory set of 12.6 MByte, while using up 212 MByte of memory maximum. These figures include the Java virtual machine and OSGi framework. The required amount of CPU resources were measured up to 18.5 % maximum of the available bogo Mips, which equals to 220 bogo Mips.

In order to test the agility of the system we profiled a series of experiments. First the programming model was tested by sending around log messages as in the SMS prototype application. The time elapsed between sending a message and the subsequent reception turns out to be measurable at ms. scale; 1 ms. maximum. The reason for this delay is due to the length of method invocation chain because of the frequent use of the proxy pattern and the corresponding reflection in Java. Interestingly, we found a severe penalty for abnormal situations. In the implementation we rely on the Java exception handling to detect distribution of events that cannot be handled by the respective peers. This situation occurs if we disseminate an action event in the system peer group. Obviously not all registered peers implement the requested action and hence raise an exception. We typically measured an additional penalty of 20 ms.

A second experiment involves the use of the secure engine. The secure engine participates in the security protocol; it stores credentials and signs and encrypts messages on request. Part of the secure engine is a tamper proof module. In our experiments we used an integrated USB smart-card and reader for this purpose (Gem eSeal @ 5 MHz.). The initial connection to the card is slow (typical delay 400 ms.), reading credentials (8 Bytes) from the card takes about 22 ms., while 3-DES encrypting of 64 bits takes 35 ms, excluding the data transfer. RSA signature generation is slow (250 ms.), but it is only required during session set up and service registration.

A third experiment involves the UPnP and ZigBee drivers, measuring the time elapsed between the event that new hardware is detected by the driver and a surrogate has been effective installed. The results for the UPnP devices (an emulated clock and a light device) were consecutively 991, 110, 2624, 301, 80, and 150 ms., whereas for the ZigBee devices (an emulated washing machine controller) we measured 100, 251, and 80 ms. The non-deterministic character of the measurements can partly be accounted to the platform. Although we emulated UPnP devices they participate in a domain with real devices that all compete for registration. This interferes with the registration of the light and clock services, e.g, the driver waits for time outs. The emulated ZigBee devices show less variation, however it can be expected that the lag when using the actual service on the target technology has more variation because of technology dependent delays.

A fourth and final experiment involves the cluster plug-in selection and subsequent use as described in the clock synchronisation example. In this experiment we emulated an EHS bus with several devices switched on and off. In this case the EHS/CECED chooser offered a plug-in for each device. We measured lags of 1070 ± 40 ms; subsequent use of the clock device was fast: 1 ms.

V. DISCUSSION

In this section we discuss the proposed architecture and implementation for seamless interworking among technology domains an for enabling business models. We discuss the virtues of the proposed solution, known solutions from literature, and possible future enhancements.

A. Lessons

The identified issues and highlights with the proposed solution in this paper are structured along the degree of integration: from architecture via implementation to performance.

1) Architecture: The requirement that legacy technology must be supported turns out to be the driving requirement on the architecture. As a result the architecture is a set of building blocks that is made available to the application engineer. For instance the programming model is a library that generates a particular view on the installed base. This view subsequently allows the engineer to use installed services either by their native interface or by the TEAHA programming model.

Policy management remains a difficult topic. We found that once policies have been properly registered, they can be enforced through the custodian pattern. However this bypasses the actual management of the policies, which involves system configuration by the human user. A complicating factor in this respect is that policies are enforced by business clusters. One of the fundamental issues is to decide on the priority of policy management: the human owner or the business owner. A simple solution in our case is to make the business owner explicit in the policy hierarchy of Figure 4.

2) Implementation: Implementing and integrating the proposed architecture went reasonable smooth. Our understanding of concepts and choice of reference implementations turned out to be sound. In particular the OSGI service platform, UPNP, and the framework extension technology are very useful.

Because of the development platform (PC based Java Virtual Machine) there is a danger of over dimensioning the system; adding too much functionality, being too flexible, and even installing redundant functionality. From a maintenance point of view, this is a highly undesirable situation. On the other hand, business models and legacy technology bring their own culture of doing things. Providing this culture with the necessary modules (bundles in case of OSGI) is often desirable and adheres to our strategy of enriching the system with modules rather than enforcing a TEAHA way of working.

3) Performance: The performance of the preliminary prototype, although functional correct, is neither fast nor low profile. The residential gateway is a mere PC, but we are currently in the process of porting the architecture to platforms with more limited resources, in particular an XScale a StrongArm platform.

The use of Java limits the use of the platform for real-time control. Real-time data streaming has never been considered; our platform is a control-only system. The preliminary profiling figures suggest that synchronisation is possible at deci seconds (10^{-1}) rather than milli seconds (10^{-3}). While this is sufficient to control white goods like washing machines and ovens, it might pose problems for regulating streaming audio and video applications.

Applications should be aware of the time synchronisation limits. For instance the clock application requires to estimate the delay time from the bell service to the actual clock device. The bell service may then account for the extra delay in order to provide a more accurate time. The observed undeterministic character of the delays is a complicating factor here.

B. Related work

There are a number of initiatives that target seamless interworking among different technologies. Neither of these initiatives exceeded the level of technology interworking, it is unclear whether and how we can raise the interworking to the level of business clusters. We feel however that they are important enough to be discussed in brief here.

In [6], [12], [19] architectures based on OSGI gateways are presented. They provide a clear overview of add-on technology for OSGI and alternatives like HAVI, .NET, and MHP.

In [9] a home networking platform is presented based on Jini. Jini bridges (federates) technology through downloadable proxies. One particularly interesting aspect is their use of surrogates that off-load the burden from tiny devices of running a Java virtual machine.

In [16], [17] two architectures based on CORBA are presented. While the aforementioned, as well as our solution, concentrate on controlling services and devices in the home environment, CORBA based architecture also process the data themselves. The CORBA inter object request broker protocol provides interoperability between different technologies. They enforce a programming model, unlike in our approach where the programming model is optional. Especially in [16] the architecture describes a solution to implement a connection based CORBA protocol over an intrinsically connection less IEEE 1394 protocol.

As a final example, in [13] an architecture based in SOAP is described. This architecture is a service aggregater rather than a home network. Nevertheless, SOAP is one of the underpinning technologies of .NET.

There are only a few papers that describe experimenting with a service platform supported with figures. In fact we only found one, [17], which we discussed above.

Integrated security is another important topic. While modern middleware provides many ways of securing transactions and methods for authorisation [7], their configuration is usually left unspecified. The lack of a proper configuration scheme conflicts with our requirement for near zero-configuration. In [17] the authors propose and experience with a policy hierarchy, similar to the one we proposed in this paper.

In [7] the security methods and implementations for three major distributed component systems are evaluated: CORBA, J2EE, and .NET. In view of our requirements there are a few highlights. J2EE and .NET do not provide a standard method for non-repudiation, which is an anomaly in case of business clusters. Further, the standard authorisation methods of J2EE and .NET rely on predefined roles, which is too inflexible

for our purposes. The close relation of .NET with Microsoft Windows is a concern because of the required support for legacy technology.

C. Future work

While developing the architecture and implementing the prototype we necessarily took a number of design decisions that limit the scope of our results. In this section we present a couple of issues that we plan to work on in the (near) future.

The presented architecture uses a single directory for service location. A virtual, physically distributed, directory yields an improvement of high practical value. Think of the intuitive way that a service can be registered and managed by the human end user. Another example is the opportunity to locate technology drivers in close range of the actual appliances. We are working on such a distributed version. The problem here is that the custodians, which enforce (and guarantee) policing, become distributed too. In this work we consider JXME as a candidate for creating a peer network of gateways.

Our architecture uses a hierarchy for registering the policies by which services and users interact. In practical situations the registry data will be distributed over multiple devices. We consider an approach that uses a visitor and traveller pattern [3], to query the distributed registry. In addition, registration configuration must be integrated.

When services are dynamically created and used, QoS becomes an issue, because service deployment needs resources. We consider an ARC [20] approach for dynamic interworking of service.

Privacy becomes an issue now that more and more services share resources. Potential burglers, for instance, can easily deduct whether someone is present in a home by monitoring the gateway activity.

VI. Conclusion

In this paper we presented an architecture for seamless interworking of technologies and appliances for dedicated business clusters. This concepts goes beyond the state of the art of seamless interworking among technology domains.

For the successful application of business clusters it is important that the interworking among appliances can be regulated. We proposed to organise the policy registration hierarchically, in order to make intuitive configuration possible for the human end user. Policing of interactions is implemented by adopting a known design pattern: the custodian pattern. In fact the entire architecture combines proven patterns and proven technology, which smoothened the development process of a prototype.

We also presented preliminary profiling figures from experiments we conducted with a prototype implementation. The prototype demonstrates the feasibility of seamless and controlled interworking among technology domains. The profiling figures suggest that an OSGi based implementation is, under constraints, feasible for commercial application. The eventual gateway platform approaches that of an embedded PC and the application domain must not require real-time accuracy better than 100 ms.

Acknowledgement

The authors wish to thank the partners of the TEAHA project for their contribution. In particular we appreciate the vivid discussions with Victor Poznanski (Sharp Ltd.), Tim Wilson (Sharp Ltd.), Ardjan Zwartjes (University Twente) and Danny De Cock (Katholieke Universiteit Leuven).

References

[1] T. O. Alliance. *OSGi Service Platform, Release 3*. IOS Press, 2003. also available from www.osgi.org.

[2] D. Chakraborty, A. Joshi, Y. Yesha, and T. Finin. GSD: a novel group-based service discovery protocol for MANETS. In *Mobile and Wireless Communications Network, 2002. 4th International Workshop on*, pages 140–144, 2002.

[3] A. v. Deursen and J. Visser. Source model analysis using the JJTraveler visitor combinator framework. *Software: Practice and Experience*, 34(14):1345–1379, 2004.

[4] W. Diffie, P. C. Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography (Historical Archive)*, 2:107–125, June 1992.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[6] L. Gong. A software architecture for open service gateways. *Internet Computing, IEEE*, 5:64–70, 2001.

[7] G. Gousios, E. Aivaloglou, and S. Gritzalis. Distributed component architectures security issues. *Computer Standards & Interfaces*, 27:269–284, March 2005.

[8] O. Gruber, B. Hargrave, J. McAffer, P. Rapicault, and T. Watson. The eclipse 3.0 platform: Adopting osgi technology. *IBM Systems Journal*, 44(2):289–299, 2005.

[9] R. Gupta, S. Talwar, and D. P. Agrawal. Jini home networking: a step toward pervasive computing. *Computer*, 35:34–40, 2002.

[10] E. Guttman, C. E. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. RFC 2608, IETF Network Working Group, July 1999.

[11] U. Halfmann and W. E. Kühnhauser. Embedding security policies into a distributed computing environment. *SIGOPS Oper. Syst. Rev.*, 33(2):51–64, 1999.

[12] B. Horowitz, N. Magnusson, and N. Klack. Telia's service delivery solution for the home. *Communications Magazine, IEEE*, 40:120–125, 2002.

[13] V. Kapsalis, K. Charatsis, M. Georgoudakis, E. Nikoloutsos, and G. Papadopoulos. A SOAP-based system for the provision of e-services. *Computer Standards & Interfaces*, 26:527–541, October 2004.

[14] A. Kung, B. Jean-Bart, O. Marbach, and S. Sauvage. *The EHS European Home Systems Network*. Trialog, 2004.

[15] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558 – 65, 1978/07/.

[16] J.-Y. Oh, J.-H. Park, G.-H. Jung, and S.-J. Kang. CORBA based core middleware architecture supporting seamless interoperability between standard home network middlewares. *Consumer Electronics, IEEE Transactions on*, 49:581–586, 2003.

[17] J. H. Park, M. J. Lee, and S. J. Kang. Corba-based distributed and replicated resource repository architecture for hierarchically configurable home network. *Journal of Systems Architecture*, 51:125–142, February 2005.

[18] V. Sundramoorthy, P. H. Hartel, and J. Scholten. On consistency maintenance in service discovery. In *20th IEEE Int. Parallel & Distributed Processing Symp. (IPDPS)*, April 2006.

[19] D. Valtchev and I. Frankov. Service gateway architecture for a smart home. *Communications Magazine, IEEE*, 40:126–132, 2002.

[20] H. van Dijk, K. Langendoen, and H. Sips. ARC: a bottom-up approach to negotiated QoS. In *3rd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2000)*, pages 128–137, Monterey, CA, Dec. 2000.

[21] European committee of manufacturers of domestic equipment. www.ceced.org/.

[22] Digital living network alliance. www.dlna.org.

[23] TEAHA: The European Application Home Alliance. www.teaha.org.

[24] Zigbee:. www.zigbee.org.