DEVICE DISCOVERY VIA RESIDENTIAL GATEWAYS

Andrew Wils, Frank Matthijs, Yolande Berbers, Tom Holvoet, Karel De Vlaminck
DistriNet research group, Department of Computer Science, K.U.Leuven, Belgium

Abstract—This paper describes the discovery and interaction of devices in a home network. It defines an innovative Device Discovery and Description Framework called 3DF, that integrates existing networking and interoperability protocols. The 3DF framework provides abstract device descriptions with multiple views, dynamic capability changes and driver support in an advanced way, taking into consideration aspects such as reusability, extendibility and ease of use.

Currently, there is no true standard for home networking or device discovery. Many frameworks and protocols are under development, enabling interoperability or some form of device discovery: HAVi [1], OSGi [2], UPnP [3], [4], Salutation [5], ... The 3DF framework can leverage these, as it offers a design that is focused on a small and changing environment (the home network), while remaining independent of underlying protocols.

It will be shown that a residential gateway is an ideal candidate to run a discovery service on. 3DF itself has been implemented and validated on such a gateway using OSGi and UPnP.

I. Introduction

Although the field of home networks is still young, many initiatives and standardization efforts have already been made. Unfortunately, the myriad of standards and protocols used for home networks are mostly proprietary and do not operate easily with other networks. A lot of research can and needs to be done in this area. This paper tries not to add yet another middleware framework to those existing efforts, yet presents a way to bind several home network protocols, making the devices of one network discoverable in the other.

The growing demand for device discovery protocols can be explained by the rise of networks containing dozens and perhaps even hundreds of small, ubiquitous devices: these cheap devices need to be quickly found in order to make efficient use of tomorrow’s home and enterprise networks. The 3DF framework presents a number of features that can greatly facilitate device interoperation, bringing us one step closer to the “thinking home”, in which various appliances seamlessly work together with little or no user intervention. Along the way, we will introduce the residential gateway as a significant aid to help us take on this task.

More concrete, the goals of the 3DF framework are:
- offer an abstract description for devices and a structured way to query for them
- make it easy to connect a home networking standard to the framework
- only use existing devices, no devices should be modified to be discovered
- deal with intermittent connections
- try to make use of existing efforts and techniques.

II. Overview

In section III, we start with explaining what part of interoperability we are targeting in this paper and how device discovery fits into this. To illustrate, we will sketch a small example of the kind of device interaction we aimed at when researching the concepts outlined in this paper.

Section IV provides a short overview of the function of a residential gateway and how it will help us to do device discovery, to be followed by the structure and design of the 3DF framework in sections V and VI. Some more advanced topics such as dynamic descriptions, matching, executable actions and user interoperability will be dealt with in sections VII through X. In section XI, we briefly explain how to integrate UPnP in the 3DF framework. We conclude the paper with future work in section XII, related work in section XIII and a summary in section XIV.

III. Interoperability

To explain what we call a home network in which devices can easily interoperate which each other, consider the following example: A device that wakes a person in the morning. A common clock radio would normally sound an alert using its internal speaker to accomplish this task. A device or application that has access to the home network (e.g. an application running on the residential gateway) - let us call this the WakeUp service - could do much more than providing a buzzer function.
It could use the coffee machine in the kitchen to have a fresh pot of coffee ready, fire up the TV in the bedroom and gradually increase the volume to gently awake our user and open up the curtains to let the morning sun in. This kind of application is really not hard to realize, provided that all the devices can easily interoperate with the WakeUp service.

What then, is needed to ensure interoperability between WakeUp and the other home network devices? As a matter of fact, a number of approaches can be taken:

Scenario 1: the proprietary way would be: WakeUp, the coffee machine, the curtains, etc are all built by the same manufacturer. This would obviously enforce a serious lock-in for the customer.

Scenario 2: WakeUp and all network devices all support the same, open protocol. Standard interfaces for a coffee machine, curtains and TV’s are known to them all. Unfortunately, standardization is often a delicate process, especially considering the large range of devices present in a home network.

Scenario 3: WakeUp knows not about all devices present in the home, but supports at least one protocol that enables him to ask for curtains, TV’s and the like. A possible approach to ensure interoperability between devices of different networks would be to register each device in all other networks. Our curtains and TV, supporting the X10 and HAVi protocol, could also be represented as a UPnP device. When using this approach, applications do not have to support any other (device discovery) protocols as that of their own network. Figure 1 illustrates this: the lightly colored devices are representations of their counterpart in the other network. For devices that would fit easily into the network, this would be a solution that would require no extra functionality in possible clients.

The drawback of this approach is that the discovery process is limited by the network protocol. Different home networks support different kinds of devices. Trying to fit in curtains into a network especially designed for audio/video use is not a trivial task, let alone getting them to be found easily. That is why we should extend these networks with a service enabling them to search in a broader range.

The purpose of this paper is to present a service that can describe devices in an independent way and to offer mechanisms for discovering them easily. All this is implemented in the 3DF framework, which, as explained in the next paragraph, we propose to run on a residential gateway. We will not define an extended range of APIs to enable this sort of interoperability (as described in Scenario 2), but rather discuss general ideas that deal with interoperating devices that know as little as possible of the way they operate.

IV. THE RESIDENTIAL GATEWAY

The primary purpose of a residential gateway (that will typically take the form of a set top box) is to offer internet services towards a home network. This includes basic services such as DHCP or firewall functionality. The residential gateway can also offer much more exciting services, making it a true internet portal. For example, a manufacturer of washing machines could have a piece of software running on a residential gateway that makes periodical diagnostics (or only when requested) and reports any malfunctions, so that a qualified technician can remotely determine what is wrong. By offering a discovery service, a residential gateway can serve not only as a gateway to the internet, but to all other devices on the network and other home networks as well. This paves the way for the realization of the extended home (accessing your home services when you are not in the home).

We have chosen to deploy our discovery service on a residential gateway as it is often the only device with the functionality and resources required to upload and run such a service. Also, it has been explicitly designed to run extra services. The resources available in a residential gateway contrast with those of the other devices connected to the home network. Last but not least, the residential gateway will most likely find its way to the typical households of tomorrow.

To be able to support small devices with limited resources, most of the complexity of the discovery service should be put on the residential gateway. The danger of this approach is the single point of failure it creates.
A solution to this would be to have multiple set top boxes, or to synchronize with, for example, the home PC.

The OSGi (short for Open Services Gateway Initiative) environment [2] is a framework written in Java, meant to facilitate the deployment of services and applications on a residential gateway. It achieves this by making use of Java’s reflection capabilities to permit dynamic installation of pieces of functionality. These pieces appear in the form of bundled collections of Java classes (or components, as you wish). In this way you can easily install new programs, drivers and devices without having to shut down the residential gateway or any of its components. Common applications of OSGi include allowing remote access to a device on the home network, configuration of devices or acting as a proxy to the device. The latter allows low-end devices, such as light switches, to become available as services without having the need to run a Java VM on the device itself, avoiding the need for modifications to the device. As mentioned before, this is also one of the goals of the 3DF framework. OSGi, as does 3DF, aims to complement existing and new networking standards and initiatives such as Bluetooth, 802.11b, Jini or UPnP.

An important component of OSGi is the device access part. OSGi supports the coordination of automatic detection and attachment of existing devices on an OSGi environment, an important part of device discovery. The basic concepts used here are drivers and devices. By making use of drivers that “monitor” the home networks the gateway is attached to, devices can be registered on the OSGi framework, whereas other driver services “refine” those devices by registering a different instance of the device. An example of this iterative procedure is a firewire printer that, when it gets physically plugged in, first is detected and registered as a generic firewire device on the OSGi framework by the base firewire driver. Then the framework interrogates other drivers. In our case, a printer driver supporting that kind of printer can register a new printer device.

V. DESCRIPTION OF THE 3DF FRAMEWORK

The 3DF framework complements the abovementioned OSGi device access specification and uses it to detect newly installed devices.

An logical extension of the specification would be to register a “refined 3DF device” for each device connected to the gateway. This way, each device would have an instance that is suitable for device discovery (e.g. there would be a “normal” HAVi device and a 3DF described one). We chose not to take this approach for 2 reasons. First, because this would hinder the normal iterative procedure: when a printer registered as a USB device has been refined as a 3DF device, OSGi would not try to refine the USB device a second time, rendering it useless towards services looking for a printer device. Second, having these instances in the framework does not directly facilitate the process of device discovery.

Let us now describe the device discovery process. When a client in need for a device connects to one of the home networks, it does not have to go on the look for a specific device, it can just ask the 3DF framework for it by providing it with a query object. In this way, the question: “what kind devices are present throughout the home networks, which ones can I use and how do I use them?” is reduced to: “I need this sort of device, can you get one for me and tell me how to use it?”.

Each device that offers one or more services, regardless of the way it is connected or which standard it adheres to, can register itself using a description object. Such a description consists of:

- a unique name and a basic description of the topic
- the actions the device can execute
- properties the device is willing to share (such as location, capabilities,...)

Using these rules a service can be described in many ways. A device can offer a service using a (limited) international standard, its own protocol or in a way directly aimed to the user. For example, our simple WakeUp service would like to access some volume controls of the bedroom TV. A simple interface with a setVolume method would suffice for this. In addition, different aspects of the functionality can be highlighted: the television can be a means of displaying images, or a device with a TV-tuner. A DVD-writer could be discovered and operated as a plain VCR. We call each “view” on the service a context.

One could also provide an additional context not towards other services, but towards the user. A context could be made human readable, so that it can be easily displayed, e.g. through a web interface (see [6]). We will explain later how this can be achieved.

In a similar way, a potential client of a device has to construct a query object which is nothing more than a description of a context the client requires. The description and query will then have to be matched by the discovery service.

VI. FRAMEWORK STRUCTURE

The 3DF framework roughly consists of 3 parts (see figure 2):

- a service that monitors newly registered devices and searches for drivers to make an advanced device descrip-
tion
- a registry to store all device descriptions
- a service that allows other devices to query it for device descriptions. It will do the actual matching.

To support all devices it is necessary to run 3DF driver software on, for example, the residential gateway, that extracts the necessary information from them and registers the appropriate device descriptions with the framework. That driver software can also be an adaptor that translates contexts to the native protocol, to be explained in section IX.

One of the goals of this framework is to use a black-box approach. Developers wishing to use it should not have to know the internals of the framework classes. That is why it is not based on extension and specialization inheritance, but rather on object interactions with well defined interfaces [7]. The OSGi framework is also structured in this way. In addition, the services deployed on an gateway running the OSGi environment are not structured in a fixed or layered hierarchical way. This enables each service to function as a component, using only those services that are available to it.

VII. THE MATCHING PROCESS

When a client asks the gateway for a device, it will hand over its query. The framework will compare this with all registered descriptions, and return the best matching devices. Naturally, if the match is not complete (e.g. not all actions or attributes are supported), the client must be informed.

As said before, the residential gateway will manage all descriptions. When a device gets plugged in to the OSGi system, the device discovery framework will try to find a “driver”, an entity that can make and register a device description. The description is made dynamically and, as we will explain, can change over time.

VIII. DYNAMIC DESCRIPTIONS

While describing a device, the question arises whether the data in the descriptions is valid for eternity. In many cases, yes. But there are a number of issues that can change over time, such as:
- is the service available? e.g. a VCR could be present, but it could be in timer mode.
- additional status info: e.g. is there a color ink cartridge installed in the printer?

These properties are bound to change. Therefore, the descriptions should be constantly refreshed. To reduce the involved overhead, our framework offers the possibility to specify whether a property is updated
- automatically by the registering instance itself
- on a regular basis (lease)
- every time the information is needed
- just not at all

Most of the times, this will be dependent on how the device is connected to the gateway.

An extreme example of changing capabilities is when a device suddenly becomes (temporarily) disconnected from the network. The framework should remember the device description, so that it can be recycled when the device reconnects. For short disconnects, this must happen transparently: a client examining the device will not notice it.
IX. Taking it one step further: executable actions

When a service has been found, naturally one wants to make use of it. When enough information has been collected about the service, one could try to:

- communicate directly with the device
- obtain and use the device’s OSGi device interface

However, an interesting way is using the already “agreed” on information in the descriptions and enable the actions to be invoked. The actions must then be mapped to the original API’s methods, be it an OSGi interface or a native protocol. This way one could truly operate any device without much trouble.

Unfortunately, as the actions become more and more complex, the ways to describe them increase in an exponential manner. There is also the issue of dealing with all sorts of datatypes, representation of complex parameters, recursive linking, etc. Executing actions falls outside of the goals of the 3DF framework, but other useful technologies already exist in this context. One could, for example, use SOAP, which has been designed to execute remote procedure calls.

X. Interoperability towards the user

Another form of interoperability has more to do with the user himself. Devices can be described so that they are easily accessible for the user. For example, a user could access his hi-fi system via his TV. A way of doing this is by using an abstract language with user interface elements, enabling devices to manipulate each other without really knowing each other’s functionality (cfr X in Unix). This is a versatile approach: the TV only knows about buttons and other widgets, it does not need any semantic knowledge of the services it controls. Although this was not the focus of the 3DF framework, it is perfectly possible to use it for this purpose. When devices are described in human readable terms, and simple actions are defined, the descriptions can be presented to the user using e.g. a web interface. This interface is also an easy way to verify that devices are correctly installed and recognized in the home network.

XI. Test Case: Universal Plug & Play

The framework has been verified by using it with Universal Plug and Play [3]. UPnP defines common protocols and procedures to guarantee interoperability among network-enabled PCs, appliances, and wireless devices. It is an ideal candidate to work in conjunction with a residential gateway.

A UPnP device announces itself by multicasting some HTTP messages containing an URI (Uniform Resource Identifier [8]) pointing to an XML-file. In this file, its actions and attributes are presented, which map partly to the device description of our framework. In this way, a UPnP device can be automatically detected and a UPnP-specific context can be automatically registered without any user intervention. Of course, to add additional contexts, one does need to provide the framework with that.

Conversely, devices discovered by our framework can be announced towards the UPnP framework too, so that devices only supporting UPnP (having no knowledge of an OSGi residential gateway) can use these (also see section III and [9]). This involves creating an XML file containing a UPnP like description and actions, as well as mapping UPnP SOAP invocations to the 3DF actions.

XII. Shortcomings and Future Work

The following paragraphs present a number of possible improvements for the 3DF framework.

Technologies like OSGi would benefit greatly from a large set of well-defined interfaces to control various devices. As said before, this is not the goal of 3DF, nor is it very likely that everyone will use these. OSGi itself is currently defining a number of interfaces for various devices. Being able to add these as an additional context to a device description can greatly increase interoperability between a device and an OSGi-aware entity.

Another issue is that, in order to be able to discover devices, OSGi and 3DF base drivers have to be written for every network protocol (e.g. UPnP, HAVi, X10, ..). Although this can be cumbersome, it could avoid the need to make bridges for every 2 network technologies. The Device Expert Group (DEG) is working on extending the already defined device capabilities of the OSGi model to create specific mappings for different device protocols and standards such as UPnP, Jini, and USB. Using technologies like OSGi and frameworks such as this one avoids having to create a multitude of bridges from and to every networking technology.

The 3DF framework does not address security. Access control is important, especially when considering the extended home environment. Next to this, outsiders should even not be able to know about the presence of the home network devices. Devices cannot be used nor discovered by everyone.

Although the matching algorithms of 3DF are not trivial, there is a more intelligent matching possible. Consider a user that wants to ask the service for the VCR having the most free space on its tape. Although this can be useful, the ability to find a usable VCR on itself is far more important. When desired, this kind of
functionality could be implemented on top of 3DF.

XIII. RELATED WORK

Today a lot of home networking standards, protocols and frameworks enabling interoperability of services and devices are under development: HAVi [1], OSGi [2], UPnP [3], [4], Salutation [5], ...

In [10], a bridge between HAVi, Jini [11] and UPnP was implemented. Using a a concept they call Virtual Overly Networks, they have found a systematic way to create gateways between different specifications for home computing. No attempts were made to bring extra device discovery into the networks.

In [4] a secure Service Discovery Service was implemented, also using complex descriptions and complex queries for locating services, designed to be scalable and fault-tolerant, using XML and announcement based dissemination.

XIV. SUMMARY

The 3DF framework permits an easy, ad-hoc way to integrate devices supporting different networking technologies using a residential gateway and abstract device descriptions. Matching of descriptions and queries is not based on interfaces, classes, or types and thus has no problems with versioning: what you seek is what you get.

Devices can be described using several contexts, depending on the way one wants the devices to be discovered. For example, a human readable context provides an easy way to have an overview of the devices connected to one’s home network, and permits those devices to be easily manipulated through a web interface.

XV. BIOGRAPHY

Andrew Wils is a PhD student at the Computer Science department of the K.U.Leuven and a member of the DistriNet group. Currently at the end of his first year, his research has formed around home networks and evolution of component-based software in embedded systems.

Frank Matthijs is a post doctoral fellow of the K.U.Leuven. After his research in the DistriNet group, which included network protocol stacks and application integration, he went on to help form a company called QMedit, focusing on the development of workflow software for healthcare driven by clinical pathways.

Tom Holvoet’s teaching activities as a professor at the K.U.Leuven include introductory courses in programming and object-oriented programming at the Computer Science department of the K.U.Leuven. His research handles about multi-agent systems (biologically inspired MAS, emergent behavior, etc), software engineering and embedded systems.

Yolande Berbers’ research interests include embedded and real-time systems, multi-agent systems with emergent behavior, system support for distributed applications (a.o. mobile agents), distributed and parallel systems, distributed computing, and computer architecture. A professor at the K.U.Leuven and a member of the DistriNet group, she teaches several courses on programming, structure and organization of computer systems and system-integrated software.

Karel De Vlaminck is professor at the Computer Science department of the K.U.Leuven, where he teaches courses on object-oriented programming and software engineering. His research interests are object-oriented and component-based development methodologies, specifically oriented towards embedded software. Currently, he is the head of the Computer Science department.

REFERENCES
