DYAMAND: DYnamic, Adaptive MAnagement of Networks and Devices

Jelle Nelis, Tom Verschueren, Dieter Verslype, Chris Develder
Dept. of Information Technology - IBCN, Ghent University – IBBT
Ghent University – IBBT, Ghent, Belgium
Email: {jelle.nelis, chris.develder}@intec.ugent.be

***Abstract*—Consumer devices increasingly are "smart" and hence offer services that can interwork with and/or be controlled by others. However, the full exploitation of the inherent opportunities this offers, is hurdled by a number of potential limitations. First of all, the interface towards the device might be vendor and even device specific, implying that extra effort is needed to support a specific device. Standardization efforts try to avoid this problem, but within a certain standard ecosystem the level of interoperability can vary (i.e. devices carrying the same standard logo are not necessarily interoperable). Secondly, different application domains (e.g. multimedia vs. energy management) today have their own standards, thus limiting trans-sector innovation because of the additional effort required to integrate devices from traditionally different domains into novel applications.**

In this paper, we discuss the basic components of current so-called service discovery protocols (SDPs) and present our DYAMAND (DYnamic, Adaptive MAnagement of Networks and Devices) framework. We position this framework as a middleware layer between applications and discoverable/controllable devices, and hence aim to provide the necessary tool to overcome the (intra- and inter-domain) interoperability gaps previously sketched. Thus, we believe it can act as a catalyst enabling trans-sector innovation.

I. Introduction

In the last two decades, the home network evolved quite rapidly. It started as a single computer that was responsible for Internet access (internal modem) which quickly evolved to a simple network in which an always-on device offered Internet access for one or two computers in the network. The evolution that can be seen now, is that nearly every device in the home offers interaction with its environment over a networked interface. Services are no longer limited to Internet access: television sets, networked hard drives, printers, laptops, etc. are offering their own specific services. Apart from aforementioned devices, also a wide range of sensing devices are able to provide context information.

Although a number of standardization efforts have tried to create a generic technology platform achieving device interoperability (e.g. UPnP [1] defines a common interface for networked devices, SLP [2] has been developed for general service discovery, etc.), the dream of seamless interoperation of all "smart" devices is still far from a reality. Even within certain application domains, problems still arise (e.g. varying UPnP implementations interpret the standard in another, sometimes faulty, way). Furthermore, due to political agendas or device limitations, certain device manufacturers choose one technology over another, which effectively divides the network in disjoint clusters of un-interoperable devices. Our work is motivated by the observation that inter-domain operability is still in its infancy, and the fact that device/service discovery is an essential generic building block to realize future "smart-house" services [3].

From an application developer's perspective, the problem of dealing with such a plethora of possible devices is difficult to solve. Mostly, the problem is ignored and the application developer will choose one or more popular technologies to support (hence possibly ignoring a considerable chunk of the market). On the other side of the spectrum, device manufacturers feel obliged to support as many technologies as possible to be compliant with as many applications as possible, which puts extra strain on the device's resources. Alternatively, closed ecosystems are formed combining devices with specific technologies and a set of associated applications. The framework we propose in this paper tries to bridge the needs of both application developers and device manufacturers: our DYAMAND framework acts as a middleware layer between the application developer and the controllable devices.

II. Service Discovery Protocols

A Service Discovery Protocol (SDP) is a protocol that enables dynamic discovery of services on a network. Technologies such as Universal Plug and Play (UPnP [1]), Service Location Protocol (SLP [2]) and Domain Name System Service Discovery (DNS-SD [4] are traditional examples of SDPs. A classification of a number of these popular examples can be found in [5]–[7].

However, the above list is in no way complete. Non-IP standards such as Bluetooth, USB and Zigbee[1] are technologies that we consider to be SDPs. This paper does not intend to provide an exhaustive list or comparison of all available SDPs. We will, however, discuss the general model of an SDP, which comprises up to three basic functions: (A) device and service discovery, (B) control, and (C) eventing. From the discussion thereof, we will derive the following **interoperability issues**:

1) different device and service models,
2) different service type representations,
3) different remote control protocols,
4) different service type semantics,
5) different eventing protocols,

[1]See resp. www.bluetooth.org, www.usb.org, www.zigbee.org

6) different event semantics.

A. Discovery

As discussed above, the *sine qua non* functionality a Service Discovery Protocol must offer is discovery of devices and/or services. UPnP uses the following model: a UPnP root device can contain services and embedded devices which in turn can contain both services and embedded devices. SLP, in contrast, does not have a notion of devices. SLP only announces services identified by a given URL. The way information about these services is acquired, also differs among SDPs: for UPnP the so-called *description* phase needs to take place, i.e. the device and service descriptions need to be fetched and parsed to get the information about the discovered devices/services. These description files follow a UPnP-specific XML format. SLP on the other hand, embeds the service information in the announced URL.

Additionally, each technology uses its own definition of service types. UPnP uses the following form: *urn:schemas-upnp-org:service:**typename**:**version***, while DNS-SD uses ***typename**._tcp* or ***typename**._udp*. Apart from the different syntax, the service type names used are different as well. If we take the simple case of a printer, the type name is *PrintBasic* or *PrintEnhanced* in the case of UPnP, SLP uses *printer* and DNS-SD uses *_printer*.

The incompatibilities presented here lead to a first level of interoperability issues (1, 2) related to service awareness. This means that without any interoperability effort, with full SDP support, you will only be able to present the user with a random service type string, but it will not be possible to give any meaning to that service type.

B. Control

An SDP can contain support to control the discovered services (an example of one that does is UPnP). This implies that after discovering a service, all information is available to perform a syntactically correct action (even without grasping the semantics of that action). UPnP uses the Simple Object Access Protocol (SOAP) to actually support this. Alternatively, an external *Control Protocol* must be used. Which protocol to use is embedded in the service information discovered in the previous step: both SLP and DNS-SD are examples of such protocols and embed this information in the service type.

Service types are defined for a specific SDP. The UPnP Forum defines so-called *Device Control Protocols (DCP)* in Working Committees, the currently standardized DCPs can be found in [8]. SLP defines the way service templates must be registered in [9] and [10] keeps a list of all registered templates. Similarly, DNS-SD keeps a list of registered service types in [11]. In the same way, Zigbee defines *Application Profiles* and USB defines *Device classes* [12]. This discussion defines a new interoperability problem, namely that of service type semantics: we end up with additional issues 3 and 4.

C. Eventing

As with control, eventing can be supported by the Service Discovery Protocol itself or by an external protocol. UPnP embeds the General Event Notification Architecture (GENA) to support eventing. Others like SLP and DNS-SD rely on the external protocols as presented in section II-B. The interoperability issues encountered for eventing are similar to that encountered for remote control: the protocol used for eventing needs to be supported and the event semantics needs to be addressed (issues 5 and 6).

III. Related Work

In the previous section, we identified a number of interoperability issues of the concept of Service Discovery Protocols. We would like to stress that these issues are a problem for currently known SDPs as well as future technologies. An SDP always makes trade-offs specific for the area in which the technology will be used. An interoperability framework for networked devices should thus take into account current and future differences in technologies.

In [13] and [14] an architecture is presented to respectively map Jini and Zigbee to UPnP. Although this might be a start for interoperability, limiting your architecture to incorporate specific technologies inherently limits the applicability.

The architecture for interoperability of SDPs presented in [15] seems to not address the resolution of differences in device/service types across SDPs. Furthermore, the framework concentrates on service lookup and invocation (e.g. no eventing), where service access across varying SDPs is realized by dynamically binding an abstract control mechanism to the respective concrete SDP-dependent ones, which incurs a non-negligible performance penalty.

Another interoperability framework [16] abstracts service access, yet fails to explain how the semantic difference of services across SDPs is solved. Furthermore, applications need to be rewritten to take advantage of this framework.

In [17] an event-based SDP interoperability framework is discussed. It focuses on external interoperability, i.e. a client of SDP1 must be able to interact with a service of SDP2. It uses internal events to model the inner workings of an SDP and performs a direct mapping of an action in one SDP onto one or more actions in the receiving SDP. In our DYAMAND approach, we have an extra abstraction layer formed by an SDP-independent model, i.e. we would have an SDP-independent model in between SDP1 and SDP2. Thus we only need to relate each SDP-specific model just to the single SDP-independent model.

The results shown in the current paper are the result of follow-up research based on our earlier work [18] (where we positioned the seminal ideas of our DYAMAND work) and [19] (which in essence describes a precursor of the proof-of-concept application sketched in V-B).

IV. Architecture

A. Plugin framework

Apart from the functional requirements discussed in section II, the framework should deliver an easy-to-use and dynamic platform to develop value added services exploiting heterogeneous “smart” devices.

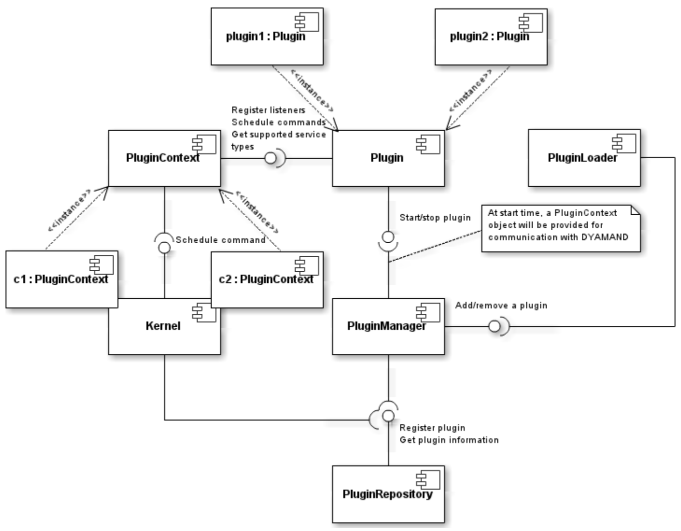

Fig. 1. Plugin architecture

Runtime modifiability is the most important quality attribute the framework needs to take into account.

The framework can be extended at runtime by using plugins. Figure 1 shows the DYAMAND architecture at the highest level. At start time, a plugin is provided with a *PluginContext* instance that enables the plugin to communicate with the framework. The major advantage of only allowing the plugins to communicate through one communication path is that it is easier to enforce security policies, e.g. only trusted plugins can perform SDP specific actions (see section IV-C).

B. DYAMAND data types

To solve interoperability issues 1, 3 and 5, DYAMAND defines the necessary data types to abstract the differences in modeling of a device and service, in terms of device/service types, control and eventing syntax.

A physical device is modeled as follows: a DYAMAND device is created for every physical device, uniquely identified based on the location information (e.g. IP address). A DYAMAND device will contain embedded devices that represent the SDP-specific devices and services present on the corresponding physical device. A service contains *Command* instances to abstract the control part of a service and *State-Variable* instances to model the state of a service, some of which can be evented, which abstracts eventing.

Up to now, the only problem that has been solved, is unifying the model that is used by different SDPs. However, the truly interesting part is providing the same semantics to similar services in different SDPs. To that end, the framework makes it possible to define generic service types that can be used by application developers to control whatever existing service. A service type is a generic blueprint that defines which commands a service of that type is able to implement and which state variables model the state. A *PrinterServiceType* might offer commands to schedule a print job and have a state variable *PrintState* indicating the current state of the printer (*Idle*, *Printing*, *WarmingUp*). This helps solve interoperability issues 2, 4 and 6.

C. Plugin types

Plugins can modify framework behavior by using the *Interceptor* pattern. Plugins are able to process devices, services, commands and state changes as they arrive in the system.

SDP plugins are responsible to implement (parts of) a Service Discovery Protocol. Discovery, control and eventing are completely separated, which means one plugin can implement discovery for a certain SDP while support for control and eventing can be supported by another plugin (possibly for only a subset of SDP-discoverable devices). Examples are the UPnP SDP plugin which responsible for discovery, control and eventing of UPnP devices. On the other hand, an SLP plugin will only offer discovery since control and eventing is not part of the SLP core protocol. Control and eventing for SLP services will be provided by a separate plugin.

To enable semantic interoperability, translation of generic services to SDP specific services is needed in terms of discovery, control and eventing. Translation in itself is quite simple. An interceptor receives a context object (device, service or state change) that it can manipulate. Suppose the *Dyamand-Printer* service type is defined in the framework, whenever the UPnP SDP plugin discovers a device of type *urn:schemas-upnp-org:device:Printer:1*, the UPnP-Printer translation plugin will translate this device to an instance of the *DyamandPrinter* service type. In the same way, an SLP plugin discovers a *service:printer:lpr://printerurl* after which the LPR printer will be translated to a new instance of the *DyamandPrinter* service type by the LPR translation plugin.

Application plugins are nothing more than plugins that implement a particular use case. This can be a very simple plugin that performs a simple action whenever a certain event occurs, e.g. switching of all lights when the TV starts playing. The power of this concept lies in the fact that simple application plugins can together implement a complex use case. An application plugin can execute generic commands without the need to know which SDP offers the specific service. *CommandInterceptor* instances are given the possibility to translate the generic command to an SDP specific command that will get executed by the responsible SDP plugin. The same concept is used whenever an SDP-specific state change arrives in the framework.

D. Performance tests

To get a view on the overhead introduced by the interoperability framework, compared to native SDP calls, the discovery part of the framework has been tested in terms of overhead in syntactical as well as semantic interoperability. Results show that this functionality can be offered without a noticeable performance penalty.

V. Examples

The framework discussed in this paper has been used in a number of diverse projects, two of which will be discussed in this section.

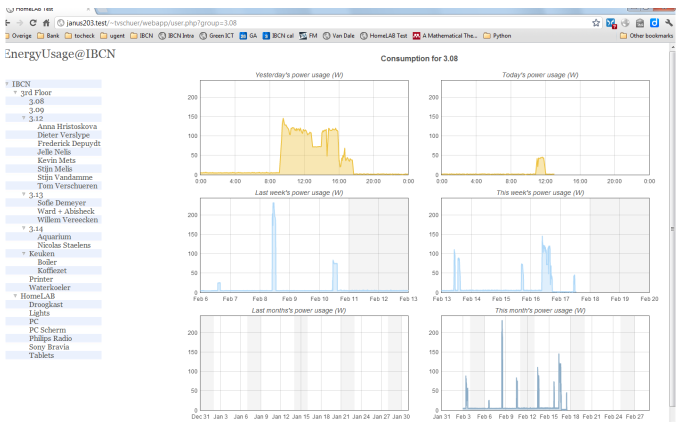

Fig. 2. Energy usage monitoring

A. Energy Usage Monitoring

At our research department, we deployed an energy usage monitoring tool that uses DYAMAND as underlying framework. Smart electricity plugs controlled through Zigbee are monitoring the energy usage for every person, DYAMAND continuously polls these plugs to offer a fine-grained visualization of the energy used per person, per office, per floor and for the entire department. This information helps raise awareness of personal energy usage. Figure 2 shows the interface of this application.

B. Aggregation of Multimedia Content

As a follow-up project to [19] a multimedia aggregation application has been developed. The application detects all media content, scattered across multiple devices (NAS, laptops, etc.), offering the user an integrated view of all available media. Furthermore, all media playing devices (audio and/or video) are discovered, and playout of media items can be triggered on any device that is capable of rendering it. Media is automatically enriched with information obtained through web services (IMDB for movies, FreeDB for audio, etc.).

VI. Conclusions

In this paper we presented an interoperability framework for Service Discovery Protocols. Interoperability issues were identified and we argued how the DYAMAND framework can solve them. DYAMAND realizes easy-to-use access to a myriad of different devices, in terms of supported Service Discovery Protocols, as well as different service types. In addition, the information gathered by all SDPs is available for use in application plugins that are interested in all services (e.g. a helpdesk application for troubleshooting).

This paper has indicated that the presented DYAMAND framework can solve interoperability issues encountered when developing applications that want to leverage a heterogeneous set of "smart" devices. Future work will mainly imply demonstrating DYAMAND in trans-sector applications, combining services from traditionally different domains (e.g. energy and media), as well as showcasing cross-SDP interoperability (i.e. clients and services of different SDPs should be able to communicate).

References

[1] A. Presser *et al.*, "UPnP device architecture 1.1," 15 Oct. 2008. [Online]. Available: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf

[2] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service location protocol, version 2," RFC 2608 (Proposed Standard), Internet Engineering Task Force, Jun. 1999, updated by RFC 3224. [Online]. Available: http://www.ietf.org/rfc/rfc2608.txt

[3] F. den Hartog, T. Suters, J. Parsons, and J. Faller, "Production of a roadmap for an integrated set of standards for smarthouse and systems related to it and an open event: Final report," CENELEC, Project Report Smart House Roadmap SA/CLC/ENTR/000/2008-20, 9 Feb. 2011.

[4] S. Cheshire and M. Krochmal, "DNS-based service discovery," 9 Dec 2011. [Online]. Available: http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.txt

[5] F. Zhu, M. Mutka, and L. Ni, "Service discovery in pervasive computing environments," *IEEE Pervasive Comput.*, vol. 4, no. 4, pp. 81–90, Oct.–Dec. 2005.

[6] R. S. Marin-Perianu, P. H. Hartel, and J. Scholten, "A classification of service discovery protocols," http://eprints.eemcs.utwente.nl/735/, Centre for Telematics and Information Technology University of Twente, Enschede, Technical Report TR-CTIT-05-25, June 2005.

[7] C. Bettstetter and C. Renner, *A comparison of service discovery protocols and implementation of the service location protocol*. Enschede, Netherlands: Citeseer, 13–15 Sept. 2000, pp. 13–15. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.115.4652&rep=rep1&type=pdf

[8] Apr. 2012. [Online]. Available: http://upnp.org/sdcps-and-certification/standards/sdcps/

[9] E. Guttman, C. Perkins, and J. Kempf, "Service templates and service: Schemes," RFC 2609 (Proposed Standard), Internet Engineering Task Force, Jun. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2609.txt

[10] Apr. 2012. [Online]. Available: http://www.iana.org/assignments/svrloc-templates.html

[11] Apr. 2012. [Online]. Available: http://dns-sd.org/ServiceTypes.html

[12] Apr. 2012. [Online]. Available: http://www.usb.org/developers/devclass_docs\#approved

[13] J. Allard, V. Chinta, S. Gundala, and I. Richard, G.G., "Jini meets UPnP: an architecture for Jini/UPnP interoperability," in *Proc. Symp. Applications and the Internet (SAINT 2003)*, Orlando, FL, USA, 27–31 Jan. 2003, pp. 268–275.

[14] S. H. Kim, J. S. Kang, K. K. Lee, H. S. Park, S. H. Baeg, and J. H. Park, "A upnp-zigbee software bridge," in *Proceedings of the 2007 international conference on Computational science and its applications - Volume Part I*, ser. ICCSA'07. Berlin, Heidelberg: Springer-Verlag, 26–29 Aug. 2007, pp. 346–359. [Online]. Available: http://dl.acm.org/citation.cfm?id=1802834.1802867

[15] P. Grace, G. S. Blair, and S. Samuel, "A reflective framework for discovery and interaction in heterogeneous mobile environments," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 9, no. 1, pp. 2–14, Jan. 2005. [Online]. Available: http://doi.acm.org/10.1145/1055959.1055962

[16] P.-G. Raverdy, V. Issarny, R. Chibout, and A. de La Chapelle, "A multi-protocol approach to service discovery and access in pervasive environments," in *Mobile and Ubiquitous Systems - Workshops, 2006. 3rd Annual International Conference on*, San José, California, 17–21 Jul. 2006, pp. 1 –9.

[17] Y.-D. Bromberg and V. Issarny, "INDISS: Interoperable discovery system for networked services," in *Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware*, ser. Middleware '05. New York, NY, USA: Springer-Verlag New York, Inc., 28 Nov.–2 Dec. 2005, pp. 164–183. [Online]. Available: http://dl.acm.org/citation.cfm?id=1515890.1515899

[18] D. Verslype, J. Nelis, T. Verschueren, W. Haerick, F. De Turck, and C. Develder, "Framework for ubiquitous discovery and access to home services," in *Proc. 1st Int. Conf. on Advanced Service Computing (Service Computation 2009), part of ComputationWorld 2009*, Athens, Greece, 15–20 Nov. 2009, pp. 398–403.

[19] J. Nelis, D. Verslype, and C. Develder, "Intelligent distributed multimedia collection: Content aggregation and integration," in *Proc. 36th IEEE Conf. Local Computer Networks (LCN 2011)*, Bonn, Germany, 4–7 Oct. 2011, pp. 203–207.