Hydra Middleware for Developing Pervasive Systems: A Case Study in the e-Health Domain

Marco Jahn, Ferry Pramudianto, Ahmad-Amr Al-Akkad

Fraunhofer Institute for Applied Information Technology FIT, Schloss Birlinghoven,
53754 Sankt Augustin, Germany
{marco.jahn, ferry.pramudianto, ahmad-amr.al-akkad}@fit.fraunhofer.de

Abstract. In this paper we present from a developer's point of view lessons learned from building an ambient e-health system using Hydra middleware. Hydra provides a middleware framework that facilitates developers to build efficiently scalable, embedded systems while offering web service interfaces for controlling any type of physical device. It comprises a set of sophisticated components taking care of security, device and service discovery using an overlay P2P communication network. We describe in detail how we applied Hydra middleware for an e-health scenario aimed at supporting the routine, medical care of patients at home. Such a scenario illustrates the complexity of a pervasive environment that Hydra aims to solve. We elaborate the process of applying Hydra for building a pervasive environment, the problems we faced, and what we learned for future research.

Keywords: Pervasive Computing, Middleware, Networked Embedded Systems, SOA

1 Introduction

The seamless integration of technical devices becoming an integral part of our everyday life reflects many visions of pervasive computing [1]. Mobile computing has already prevailed as a common technology helping people achieving their routine activities. Going one step further, pervasive environments proactively support the user by providing ambient interaction with his surroundings and at the same time being minimally intrusive. For instance, in the area of home automation such a system could help people saving energy according to the their behavior and context information. In an e-health scenario a pervasive environment could monitor a patient's condition and constantly send these data to his doctor and notify an emergency service in a critical situation. Developing such pervasive environments covers a broad range of different technological challenges that developers have to consider such as [2]:

- Devices must be able to dynamically enter and leave the environment and at the same time they need to be aware of this frequent changes.

- Devices should be able to exchange information regardless of their communication technology, such as Bluetooth, RFID, Wi-Fi, Zigbee[1] etc.
- The communication flow among devices has to be secured to guarantee privacy and protect against misuse of information.

These points show that a pervasive environment implies a high degree of flexibility. Accordingly, the software architecture has to support this need. The service oriented architecture (SOA) paradigm represents a promising approach to face these challenges. It allows the integration of distributed software systems, which is highly desirable when trying to interconnect heterogeneous devices and systems. Services in a SOA are described by a platform-independent specification that abstracts from the complexity of the underlying systems, are loosely coupled, and in particular reusable [3].

The Hydra middleware framework strives to provide a SOA that enables developers to build scalable, embedded systems without having to deal with the complexity that pervasive environments require. In particular, Hydra makes benefit of Web Services as middleware technology to support the discovery, description and access based on XML and web protocols over the Internet.

In this paper we present our experiences gained from a developer's point of view, building an ambient e-health system, which is an appropriate test bed for Hydra since such a system requires the developers to deal with various sensors and actuators, and their communication protocols. Moreover, e-health is becoming an important factor improving our quality of life.

We show what the benefits and pitfalls are, when setting up such an environment based on the Hydra middleware. Our work shows, how well a middleware framework like Hydra applies to the requirements evolving from pervasive environment scenarios.

We first explain the motivation of our work by describing the problem and presenting an envisioned scenario. To outfit the reader with the current state of research we first give an overview of the main components of the Hydra middleware and present examples of related work. Next, we describe our testing environment and the experiences gained. This paper concludes with an outlook on the upcoming work.

2 Problem

To efficiently build pervasive environments that interconnect a broad range of devices and establish communication among these devices, developers need to be equipped with tools and frameworks that simplify the development process by abstracting from the technological heterogeneity. The Hydra middleware framework with its modular service oriented architecture claims to provide a solution to these problems. Developers have to carefully elaborate the requirements of their envisioned environment and make a decision whether developing everything from scratch or utilizing a framework that promises to meet their requirements. Making such a

[1] http://www.zigbee.org/ (last visited on 15/07/09)

decision is never easy and especially when the developers are faced many choices without supporting facts about the tools or frameworks.

3 Scenario

Otto - a former professional e-sports gamer - has become old and now is a patient who is living at home and taken care of by a nursing service. Due to a heart disease, his health values have to be monitored by his family doctor so he can react to any aggravation of his patient's health. Otto sometimes is a little bit lazy, so his doctor sends him a message, that he should measure his temperature and blood glucose level and weigh himself. Otto heeds his doctor's advice but unfortunately he broke his blood glucose meter, so he just takes the temperature and weighs himself on his Wii Balance Board that he used to play a lot on, back in his better days. The resulting values are sent to his doctor's Tablet PC automatically. The doctor recognizes that the weight is ok, but the glucose level is missing and the temperature rose significantly since the last measurement. He decides it would be best, to have Otto checked through in the hospital. Therefore he sends this task to the nursing service with Otto's health information and a notification to Otto's phone that a nurse will come to pick him up. Otto agrees and generates a security token from his phone and sends it to the nursing service. The nurse receives her token and saves it to her nursing smart card while Otto saves his token on his Playstation 3 (he was a distinguished online soccer expert, but now uses it to control his front door). As the nurse arrives, she utilizes her smart card to open the door; the Playstation 3 (PS3) matches both tokens and she can enter Otto's house. Luckily the nurse brought a glucose meter that seamlessly integrates into Otto's home environment. She measures Otto's glucose level (again the result is sent to the doctor automatically) and takes Otto to the hospital. Since Otto's doctor knows best about his progress of disease he calls the hospital and discusses further treatment modalities.

4 The Hydra Middleware

To provide the reader with the basic concepts of Hydra we briefly introduce important terms and functionalities of the framework:

Hydra Devices Hydra separates between resource restricted devices that are not able to host Hydra middleware (non Hydra-enabled devices) and more powerful devices that are (Hydra-enabled devices). Restricted devices are connected to the Hydra network via a proxy mechanism. Such proxies are deployed on Hydra-enabled devices that are then called gateways. In a Hydra network, every device, whether it is Hydra-enabled or represented through a proxy, is called a Hydra Device.

Networking In Hydra the main component responsible for communication among devices is the Network Manager. It creates an overlay P2P network that all Hydra Devices are connected to. Since Hydra implements a service oriented architecture

using web services, the Network Manager employs SOAP Tunneling [4] as transport mechanism for web service calls. This allows Hydra Devices to communicate through firewalls or NAT.

Device Discovery As stated before, the detection of new devices is a key requirement to a pervasive environment. In our example, the nurse's glucose meter has to seamlessly integrate into Otto's environment. To this end the device has to be a Hydra Device that can be discovered by Otto's controlling application.

Security When sending private data over a computer network, security and privacy are always major issues. When Otto sends his health values to his doctor it is essential, that these sensitive data cannot be accessed by anybody but the dedicated recipient. Hydra supports security on different abstraction levels.

Context Awareness In a pervasive environment, devices and applications ought to be aware of the user and his surroundings, being able to react to changes in context. Hydra provides dedicated components that implement extendable rule based configurations for collecting, processing and combining context information.

Device and Application Development The Hydra architecture design allows for a clear division of work regarding device- and application development. A device developer is responsible for making Hydra Devices out of any kinds of physical devices. The application developer then should be able to easily integrate such devices into an existing Hydra network or build up a new one.

5 Related Work

Several middleware for networked embedded devices exist, for example SOCRADES[2], an effort to integrate embedded devices into SOA so that the information on the device level can be included easily to the business processes [5, 6]. It provides many features such as eventing, management, service composition and discovery. Unfortunately, they did not discuss about the possibility of semantic selection which is desirable when we deal with a large number of devices in an ad-hoc and dynamic environment. Another related work is AMIGO, an EU commission project aiming at development of an open, standardized, interoperable middleware and intelligent user services for the networked home environment [7].

[2] www.socrades.eu (last visited on 15/07/09)

6 Test Bed and Lessons Learned

6.1 Architecture

Figure 1 visualizes the runtime architecture of our e-health environment. It shows that we integrated several non Hydra-enabled devices (depicted as images of the real devices) and four Hydra-enabled devices. Each non Hydra-enabled device is connected via a proxy on a Hydra-enabled device. To an application developer this is completely transparent; to him each device in this network is a Hydra Device, whether it is the Balance Board or the PS3. The complete setup is as follows (leaving out the glucose meter):

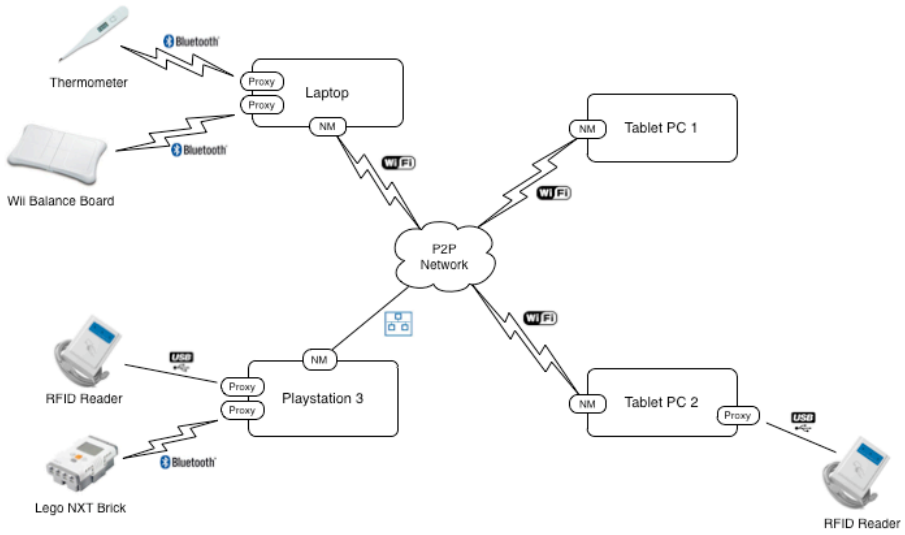

Fig. 1. Runtime architecture of the e-health environment

The laptop acts as proxy for the thermometer and the Balance Board. Both devices are connected via Bluetooth. The laptop itself is connected to the Hydra network via WiFi. The PS3 communicates with the Hydra network via LAN. Additionally it hosts proxies for the RFID reader (in the scenario it controls door access) and a Lego NXT brick that opens and closes the door of Otto's home. Tablet PC 1 (the doctor's PC) does not host any other devices, it may provide own services and invoke calls to other Hydra Devices. Tablet PC 2 is the nurse's PC that connects another RFID reader to the network. Recalling the scenario, the nurse needs to save her security token on a RFID card.

This visualization of the runtime architecture shows again that Hydra developers mainly deal with two tasks: Integrating non Hydra-enabled devices and connecting Hydra-enabled devices to a network. We try to put the Hydra middleware framework to a test by using a wide range of devices with different hardware specifications. In

the following we present our experiences gained from setting up the described environment. Following our approach of reporting from a device developer's perspective we mainly describe the process of integrating the two different types of devices: As examples of non Hydra-enabled devices we choose the Wii Balance Board and a RFID Reader. To show our experiences with a Hydra-enabled device, we report on the integration of Playstation 3.

6.2 Integrating Resource Restricted Devices

One major concept for making a non Hydra-enabled device a Hydra Device is the proxy. Such proxies offer web service interfaces for accessing the functionalities of the respective device as well as a set of basic functions that represent the behavior of a Hydra Device. Moreover, a proxy offers services running on top of UPnP protocol, allowing automatic device discovery.
For developing proxies, Hydra offers tool support suited for different platforms. One can either use Limbo [8], a Java-based web service compiler, or a set of discovery tools using .NET.
Limbo generates web service code for different target platforms (currently J2SE and J2ME) and different protocols allowing SOAP communication over TCP, UDP or Bluetooth. When using Limbo to integrate a device, developers have to specify the device's service interface and provide some Meta-information about the device. Limbo can then generate web service code that serves as basis for the device proxy. These services include Hydra standard services, device specific services, energy efficiency services and UPnP services to make the device discoverable.
When the device developer decides to build a Hydra Device using .NET, he extends the Hydra discovery tool by integrating the corresponding proxy into the discovery tool. This tool then instantiates the proxy when it discovers the respective device. The .NET library defines two abstractions for a device proxy: Device Device Manager and Device Service Manager. A Device Device Manager handles the creation of services that can be consumed from a Hydra network. A Device Service Manager is in charge of the communication between proxy and the physical device.

Integrating Devices with Limbo The Wii Balance Board itself is not capable of hosting parts of the Hydra middleware. Thus, we had to build a proxy to make it available as Hydra Device. For our scenario we decided to let Limbo generate the web service code for UPnP discovery, Hydra standard services, and device specific services. The device specific web service represents the functionality offered by the Wii Balance Board, consumes incoming calls from the Hydra network and delegates them to the logic that handles low level Bluetooth communication between proxy and Balance Board.
Before letting Limbo generate any code, we had to do two things: Specify the device service's interface and provide some basic Meta-information about the device itself. For the service interface Limbo supports WSDL, the device information has to be provided in OWL, both well-known standards. The Balance Board's interface was kept quite simple, providing only one method for accessing the weight value. Meta-information included things like manufacturer, model name, device description etc.

Outfitted with this information we took Limbo to generate the three services that are required to integrate a device: The device specific service (depicted in the WSDL), the Hydra standard service and the UPnP service. The next step was to register these services at the Network Manager running on the proxy machine, to ensure that device-to-device communication is completely under the control of Hydra. At that point the Balance Board proxy is visible and accessible inside the Hydra network. What remained to be done was implementing the low-level communication between proxy and Balance Board. We used the BlueCove[3] Library to provide Java-access to the Balance Board via Bluetooth.

Integrating Devices with .NET Just like the Balance Board, the RFID readers are non Hydra-enabled devices and have to be integrated via proxies. The .NET tools comprise a set of libraries (including the aforementioned Device Device- and Device Service Manager) providing reusable functionality for device developers. To enable UPnP-based device discovery we used the Intel UPnP Tool[4] to generate UPnP service code (C#) and then adopted that code inside the Device Device Manager. Doing so, the proxy can announce itself representing the corresponding physical device in the Hydra network. Next, we wrote a service contract for the RFID reader as well as for the Tablet PC with the methods that we would like to offer. We inherited the class library `BasicHydraDevice`, and sent our service contract to this parent class, which then generates a web service stack. In the Device Service Manager we use an external library that handles the communication with the RFID reader and our Tablet PC UPnP device.

Comparing both approaches, we feel that none of them has major advantages over the other. Both provide good support for automated generation of UPnP services, which is essential to make devices discoverable. Also implementing device specific services is well supported by both approaches.
Nevertheless, there are some unique features to each of the tools. Limbo is able to generate web service code for several platforms including J2ME. This allows developers to integrate mobile phones supporting J2ME directly, without having to build proxies. In that case all the Limbo-generated code will run on the device itself. As described in the Balance Board report, when integrating a device with Limbo, developers have to deal with the underlying low-level communication themselves, which might not be a too big drawback because established libraries for many protocols exist. Nonetheless, the Hydra .NET libraries offer a bit more comfort, providing implementations of various device protocols, e.g. Bluetooth, Zigbee, and RFID that can be used out of the box. The decision which approach to choose might mostly be influenced by the developers' programming language preference. Of course a dedicated Java developer might go for the Limbo approach while a .NET developer might probably make a different decision.

[3] http://www.bluecove.org/ (last visited on 15/07/09)
[4] http://software.intel.com/en-us/articles/intel-software-for-upnp-technology-technology - overview/ (last visited on 26/06/09)

6.3 Building the Hydra Network

Since the focus of our work lies on device development, our application logic is still on a rather basic level. Nevertheless, we have all our devices connected to the Hydra network; communication is completely realized over Hydra mechanisms. This means, that all web service calls are routed through the corresponding Network Managers that take care of SOAP Tunneling.
As depicted in Figure 1, each Hydra-enabled device hosts a Network Manager. The installation of these components on the PCs worked very well; not least because these managers are deployed in an OSGi[5] runtime supporting the need for a modular and pure service oriented environment. To test Hydra under harder conditions we took the Playstation 3 as an example of a non-standard device.

Integrating Playstation 3 As the PS 3 is powerful enough to host a Network Manager it does not need a proxy to become a Hydra Device. As a first step we set up Equinox as OSGi runtime environment, which did not cause any problems. At first, we needed adapted versions of the Network Manager and related bundles for the Power PC architecture and Java 1.5 SDK. And second, this was the problem we couldn't fix for this installation: the constant interchanging of data with Hydra network consumed too much working memory. The working memory of the PS3 is 256MB, but the actual one was just 128MB. We solved these problems by installing a simplified Yellow Dog[6] Linux dedicated for the PS3 called Helios[7]. All in all, the PS3 with its Power PC architecture and relatively low amount of memory was not so easy to integrate. The problems we faced show that there is still some work to be done, to make Hydra available on as many devices as possible.

7 Conclusions and Future Work

Building a demonstrator for a pervasive environment in the e-health domain proved to be a good use case for evaluating the Hydra middleware framework. As this paper shows we are still at the beginning of building such an environment. Thus, our focus lied on testing Hydra against the fundamental requirements of such an environment, namely device integration and basic networking capabilities. We report from a device developer's perspective and show what it takes to make various kinds of devices usable for a Hydra application developer. The obtained results show that Hydra provides a good foundation for integrating heterogeneous devices. The tested tools and software components significantly ease the tasks of a device developer for pervasive environments. In the near future we will integrate the G1 mobile phone as a mobile device that is capable of hosting the essential parts of the Hydra middleware.
In this paper we evaluate Hydra mainly from a device developer's perspective. The next step is to move the focus to application development. Once we have a set of

[5] http://www.osgi.org/Main/HomePage (last visited on 26/06/09)
[6] http://us.fixstars.com/products/ydl/ (last visited on 26/06/09)
[7] http://www.helios.de/news/news07/N_04_07.phtml (last visited on 26/06/09)

devices and are able to easily integrate new ones, we can evaluate the features that Hydra offers for application developers. One major task will of course be to integrate Hydra security components, since secure communication is one key requirement in a pervasive e-health environment.

Acknowledgements. The work presented in this paper is supported by the European research project Hydra IP (IST-2005-034891).

References

1. Satyanarayanan, M.: Pervasive Computing: Vision and Challenges. IEEE Personal Communications, August (2001) 10-17
2. Eikerling, H.-J., Gräfe, G., Röhr, F., & Schneider: Ambient Healthcare Systems - Using the Hydra Embedded Middleware for Implementing an Ambient Disease Management System. In: Luís Azevedo & Ana Rita Londral, ed., HEALTHINF, INSTICC Press, pp. 82-89 (2009)
3. Sanders, D. T., Hamilton, J. A., and MacDonald, R. A. 2008. Supporting a service-oriented-architecture. In Proceedings of the 2008 Spring Simulation Multiconference (Ottawa, Canada, April 14 - 17, 2008). Spring Simulation Multiconference. The Society for Computer Simulation International, San Diego, CA, 325-334.
4. SOAP Tunnel through a P2P Network of Physical Devices
   Francisco Milagro, Pablo Antolín, Peeter Kool, Peter Rosengren, Matts Ahlsén
5. Karnouskos, S.; Baecker, O.; de Souza, L.M.S.; Spiess, P., "Integration of SOA-ready networked embedded devices in enterprise systems via a cross-layered web service infrastructure," Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on , vol., no., pp.293-300, 25-28 Sept. 2007.
6. Kirkham, T.; Savio, D.; Smit, H.; Harrison, R.; Monfared, R.P.; Phaithoonbuathong, P., "SOA middleware and automation: Services, applications and architectures," Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on , vol., no., pp.1419-1424, 13-16 July 2008
7. Ressel, C.; Ziegler, J.; Naroska, E., "An approach towards personalized user interfaces for ambient intelligent home environments," Intelligent Environments, 2006. IE 06. 2nd IET International Conference on , vol.1, no., pp.247-255, 5-6 July 2006.
8. Klaus M. Hansen, Weishan Zhang, and Goncalo Soares. Limbo: an ontology based web service compiler for networked embedded devices. In Proceedings of the International Conference on Software Reuse, 2008.