Platform and language independent service life cycle management for device centric SOAs

Christian Fabian, Elmar Zeeb, Dirk Timmermann
Institute of Applied Microelectronics and Computer Engineering, University of Rostock
18057 Rostock, Germany
{christian.fabian, elmar.zeeb, dirk.timmermann}@uni-rostock.de

Frank Golatowski
Center for Life Science and Automation
18119 Rostock, Germany
frank.golatowski@celisca.de

Abstract – Service-orien ted architectures (SOA) are nowadays a wide-spread software building approach. The fields of application ranges from embedded devices up to enterprise and business solutions. For distributed services and devices it makes sense to have a mechanism for remote administration to manage their life cycle. Technological standards like OSGi (see section 3) provide such a life cycle management, but device-oriented Web Service technologies like the Devices Profile for Web Services (DPWS) do not specifies such a mechanism as a standard.

In this paper we present a service-based life cycle manager for* an *implementation* of DPWS *to manage* the services of *a device during runtime* through *remote* reconfiguration. *The advantage of the presented solution is the ability to be platform and language independent by using a script-based interaction between the life-cycle-manager and the individual services.

I. Introduction

With modern and lightweight middleware technologies the paradigm of Service Oriented Architectures (SOA) enters the application domain of device networks. These networks can consist of up to thousands of devices and require simple interaction concepts and interoperability. They can be found in factory automation, home automation, telecommunication and other domains. SOA allows the creation of modular and clearly defined software architectures that ensure a high grade of interoperability and reusability. The implementation of SOA in device networks simplifies the integration into enterprise systems or other systems that also implement the SOA paradigms in their architecture. As device networks have lower computing and power resources at the hardware level they differ from other fields of application of SOAs. Thus most tools and components used for development and management of SOAs can't be deployed or used on these devices. A typical problem of SOAs in device networks is the deployment and management of services. In this paper we will present a platform and language independent approach to deploy services on devices and to manage their life cycles. This approach will offer further flexibility for the implementation of SOAs in device networks.

This paper is organized as follows: Section 2 describes the issues with the implementation of SOAs in device networks. Alternative concepts for life cycle management of services on devices are described in Section 3. The presented approach and the technical reasons why this approach fits better into device networks are explained in section 4. In section 5 an example of use is demonstrated. A summary and future development is presented in section 6.

II. Device centric SOAs

The increasing complexity of device networks consisting of up to thousands of devices is demanding new technologies for simple device interaction and interoperability. Service-Oriented Architectures (SOA) [5] firstly addressed this issue for software components. The probably most popular implementation of SOA are Web services. For device to device communication, especially for resource-constraint devices, the Web services protocols often need too much resources and computing power. The Web services technology also lacks features like ad-hoc device discovery, device description and eventing channels needed in device networks. Thus, the Devices Profile for Web Services (DPWS) [10] was defined. DPWS uses some specific Web services protocols and restricts their usage because of resource limitations in embedded systems. So DPWS enables the usage of Web Services based technologies to implement device centric SOAs and this offers the same modular and clearly defined software architectures in device networks ([4], [9]).

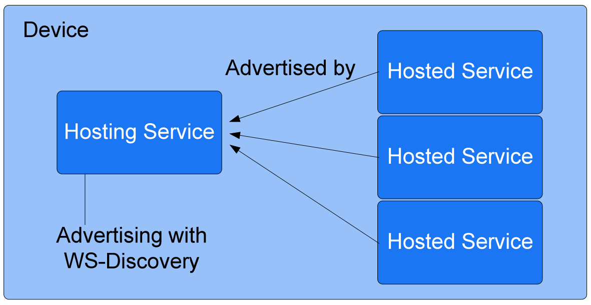

Figure. 1. separation of the hosting service from the hosted services

DPWS is based on well known protocols and Web Service specifications. It employs similar messaging mechanisms as the Web Services Architecture (WSA) [14] with restrictions to complexity and message size ([10], [15]). On top of the low level communication foundations it uses SOAP and XML Schema for the actual information exchange. DPWS specifies mechanisms for ad-hoc device discovery based on WS-Discovery, device and service description based on WS-MetadataExchange and WS-Transfer and a publish/subscribe mechanism using WS-Eventing [10]. The typical composition of a DPWS device is shown in Fig.1.

The University of Rostock - a partner of the WS4D [2] initiative - has developed the WS4D-gSOAP toolkit, which includes a DPWS protocol stack implementation and some software tools to create devices and clients. Other toolkits for different platforms and programming languages are also available. They can be found at the web site of the WS4D [2] initiative, at the web site of the SOA4D [1] initiative, or in Windows Vista and the .Net Framework [3].

III. Service life cycle management in device centric SOAs

A. *OSGi*

OSGi is a Java based Framework and was original developed for controlling and cross-linking embedded devices. A well known application using OSGi is Eclipse, but OSGi is also used in home automation, mobile phones, internet gateways or automobiles [16]. One of the basic concepts of OSGi are bundles. OSGi bundles simplify the modular design and administration of complex applications in Java. An application can consist of several bundles and each bundle can be administrated at runtime. The OSGi framework consists of 4 layers [17]: Execution Environment, Modules, Life Cycle Management and Service Registry. In this paper we concentrate on Life Cycle Management concepts. The Life Cycle Management offered by OSGi enables loading of bundles at runtime that are in fact Java class files. In a regular Java application classes in the classpath are accessible for all classes. The Life Cycle Management in OSGi restrict the access to a given rule. New services are transmitted to an OSGi framework as a jar-file or so called bundle. An OSGi bundle consists of a manifest, the resources and of optional documents.

B. *Gravity*

The Gravity framework is based on OSGi and was developed with the vision of a framework that can be customised during the runtime especially for client applications [7]. The basic concept of gravity is the modular composition of blocks called services. Available services can be combined to compose an application and it is not deterministic when services appear ore disappear. The focus of Gravity is the dynamic availability of services. A central component is the service binder. The service binder is a component to automate the dependency management. A service provides a XML file with his component and needed dependencies. For every component an instance manager will be created by the service binder to monitor, bind, unbind and manage the service dependencies. Referring to OSGi the management of dependencies are controlled by a central component and have not to be implemented by the service developer [7].

C. *Poco*

Poco is a collection of open source class libraries written in C++. The field of application range from embedded system up to enterprise solutions. Poco is not a service bases solution it is more for developing network-centric applications [6]. Several features are available like a XML parser, security, database access, file system access and so on. The core of Poco is the foundation library. This library contains components for memory management, error handling, logging, streams, threads, file system access, shared libraries, process management and utilities for strings and stream. The foundation library includes an abstraction layer to have a system independent API. Supported systems are Windows XP, Mac OS X, Linux, HP-UX, Tru64, Solaris and OpenVMS, in addition to Windows CE, QNX Neutrino and other POSIX-compliant embedded operating systems [6]. Additional libraries are the NetLibrary (for network programming), XML-Library (for reading and writing XMLData) and Util-Library (classes for working with configuration files and command line arguments). Additional components can be included during the runtime by linking dynamic libraries. Poco provides a template-based class loader analogue to the class loader in Java. The class loader in Poco needs a manifest. This manifest includes all implemented classes of the library and can be automatic created by a provided macro concept.

IV. Platform and langugage independent service life cycle manager

A. *Lightweight life cycle management integration in DPWS*

The concept introduced in this paper, implies a separation of the hosting service from the hosted services on a device. Figure 2 illustrates the services on a device and how they can be separated from the hosting service. With this separation service announcement and service producer can be implemented independently. This approach improves modularity and offers deployment of services during the runtime of a device. Furthermore a device can be distributed by announcing services running on different physical units.

The life cycle manager in the WS4D-gSOAP toolkit is a normal hosted service and runs under the same condition like the other hosted services, which are supervised by the life cycle manager or standalone. When no life cycle management is needed, this service can be omitted to save resources. The life cycle manager is not a regular part in DPWS. For a better differentiation between regular hosted and managed services, the special notation lcmC-services for managed services will be introduced. The shortcut lcmC is a synonym for "life cycle manager controlled".

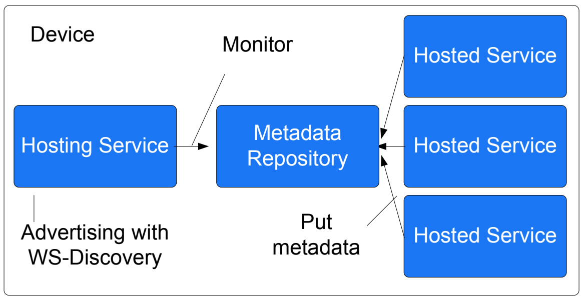

Figure. 2. separation of the hosting service from the hosted services

The life cycle manager, presented in this paper, is able to add, start, stop, update and remove services during runtime. To realize this, there are provided the following methods: install, deinstall, start, stop, update and a function to resolve all dependencies. The dependencies have to be resolved to check if every required service, device or hardware component is available on start-up of the lcmC-service. The amounts of services which can be managed by the life cycle manager depend on the available memory of the host system and the size of the specific lcmC-services. The life cycle manager can be used among several systems like Linux, Windows and on embedded systems like the Fox-Board [12] and the Cool MoteMaster board from LiPPERT [18].

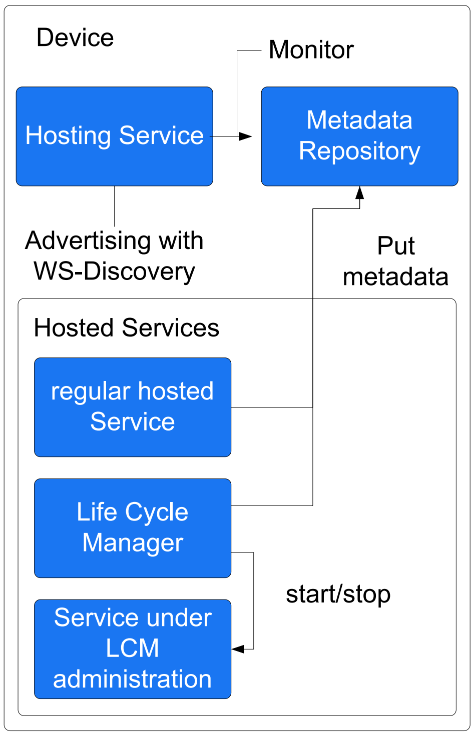

Figure. 3. **device configuration with life cycle management**

To install a new lcmC-service, a Bundle is expected. This bundle is a standard zip-archive and specified in chapter 4.2. All bundles will be analysed for consistency to increase the system stability. That means all needed components have to be included in the bundle. If the performed analysis was successful and no inconsistency was found, than the lcmC-service state is set to "install" and the life cycle manager is ready to resolve all dependencies before the service can be started. A serviceID is used to distinguish several installed lcmC-services. According to the DPWS specification [10] the serviceID has to be unique within a device. This serviceID is defined in the metadata file of a service and clearly identifies a hosted service. It has to be "persisted across re-initialization" [10]. The serviceID is also used in the life cycle manager as an identifier for the installed lcmC-services. Clients are now able to control lcmC-services by the usage of this identifier for service configuration. The basic configuration of a device with life cycle management is shown in Fig. 3 including the separation of the hosting service from all hosted services. The specification [10] define that the hosting service of the device has to send-out a new "Hello"-message when new services are available. For the DPWS-stack of the University of Rostock a mechanism was implement to inform the hosting service that a new service are available on the device. The hosted services store their metadata description into a special metadata repository. The hosting service observes this repository and recognises any modifications. When a metadata description is missed or a new one is detected the device description has to be updated by this hosting service. To start a new service by the life cycle manager, the metadata description has to be stored in the metadata repository and will be done automatically.

B. Combined packaging of service and administrative components

The central component of the installation process is a bundle. The bundle is configured as a ZIP-archive and contains the following parts:

- service: the executable file which includes the functionality (more than one file is possible to increase system independency)
- WSDL: the interface description of the service
- XML: metadata description, fundamental for the hosting service to announce the hosted service
- scripts (resolve-, start-, stop- and isAlife script): to interact with the hosted services, especial for starting and stopping
- further files and subdirectories are possible

The life-cycle-manager decompresses the bundles and analyzes their content. If all necessary components exist, then the installation process continues. To compress and decompress bundles the free zLib library [11] is used. ZLib is available for a great variety of hardware platforms. The memory usage by zLib is predictable and performance for compressing/decompressing is nearly equal to CRC-32 (cyclic redundancy check) [11]. The advantages of zLib are the wide distribution in more then 500 applications using it, adjustable compression rate and the low memory footprint [11]. It is optimal for the usage in embedded systems under the directive of low memory consumption and system-independency.

Innately zLib is not able to create ZIP-archives. With the help of the minizip-package by Gilles Vollant [13] it is possible to create ZIP-compliant archives. This package was included into the life cycle manager after some necessary API-customisations. The complete internal structure of the bundle will be observed and controlled by the life cycle manager. For every new service a temporary folder will be created and all needed components generated. After this procedure the service folder will be renamed with the unique serviceID.

C. Mechanism to operating with platform and language independent services

The intention was the realization of a mechanism for a system independent starting of managed services. This was done with the WS4D-gSOAP toolkit from the University of Rostock. The life cycle manager should be implemented in a system-independent manner and has to support the language independent control of services. One opportunity was the implementation of a customised functionality to start ore stop lcmC-services for each combination of hardware platform and programming language, but this is not a good solution with respect to modern software engineering. It is very comprehensive to implement functionalities for all possibilities and not holistic for the reuse of software modules. Furthermore it is a static solution with no room for dynamic adjustments or loosely coupled software principles like SOA. A smarter solution is the usage of an internal broker service to translate between the life cycle manager and the lcmC-services. This broker service is a scripting language interpreter and runs the special management scripts included in the bundle of an lcmC-service. For every lcmC-services, that has to be installed, exists individual scripts. There is a wide range of available scripting languages like Lua, ecmascript, guile, python and LISP. After an evaluation Lua [8] seems to be the best option for the proposed context because it has a small sized interpreter (120kB) with good performance results, released under the MIT licence, flat learning curve and it is common used to realize dynamic software solutions [8]. Especially for embedded systems and small devices Lua fits best of all analyzed opportunities.

The sequence diagram in Fig. 4 shows how the script based interaction works. From step 13 up to 22 the starting sequence of an lcmC-service via the life cycle manager is illustrated. First the status of the lcmC-service must be resolved before it can be started. That mean all dependencies must be accessible. After that the life-cycle-manager calls the start-script (Fig. 4 step 16) of the lcmC-service. The lcmC-service will be started by this script (step 17) and after the script execution has finished a value will be returned to advertise whether the service started successful or not. After the service started successful the metadata description will be transmitted to the metadata repository (step 19). Now the hosting service is able to announce the new service. The isAlive script will be called (step 20) before the installation process has finished. This script will test if the lcmCservice started and announced correctly. When the service call fail then the lcmC-service will be deleted. The result of the installation process is included in the response message (step 22).

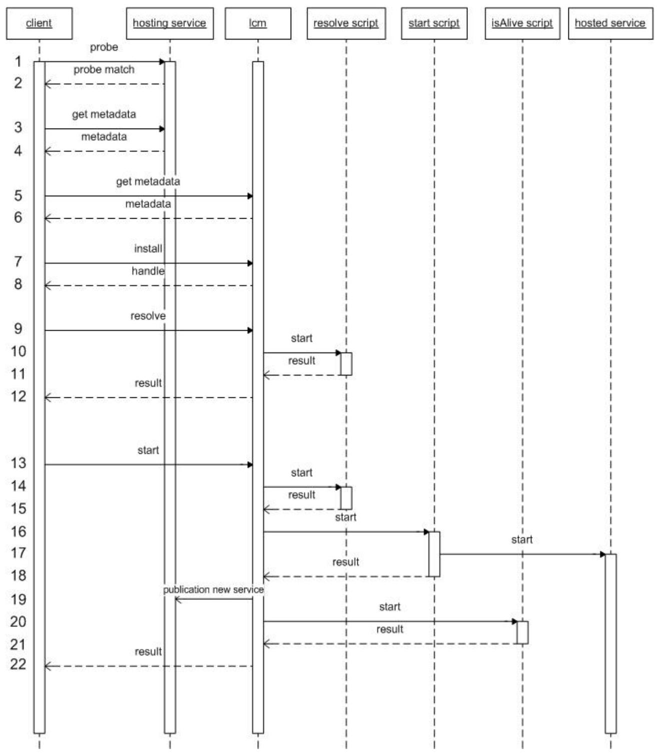

Figure. 4. sequence diagram to start a service platform and language independent

A bundle can include several components for different operating systems. To choose the correct component it is possible to read out the environment variables by Lua. Lua provide the method “os.getenv()” to read a given environment variables and return the assigned value. An example how to differ between operating systems is shown in Listing 1. In line 2 a request demonstrates this check if the operating system is Windows NT. The check for other systems can be done similar. Depending on the platform the dedicated service can be started with the command “os.execute()”. This command is a specific Lua command and platform independent. This provides a simple way to start services on different platforms with one bundle for all operating systems.

It could be also possible to manage lcmC-services across different systems. The life cycle manager can run on a different physical system than the lcmC-services. Every lcmC-service works in its own process to reduce the influence between different concurrent running services. This concept allows very flexible service architectures for device networks.

```
-- test the operating system
if os.getenv(”OS”) == ”Windows NT” then
    print(”Windows”) ;
    if Resolve( ) == 0 then
        -- start the service
        os.execute(”. /simpleExample.exe ” )
    else
        print(”Resolve not successful!”)
        return 0
    end
else
    print(”OS is not Windows !”)
    return 0
end
return 1
```

Listing 1. Lua script example

V. Example of Use

To show the operational capability of the life cycle management approach we ran an implementation of it on an embedded system called Fox-board supplied by Acme Systems. This is a minimal embedded system including a 100 MHz 32-Bit RISC processor, 1 Ethernet (10/100 Mb/s), 2 USB 1.1, 1 serial console port, 8MB FLASH and 32MB RAM [12]. The operating system is an embedded linux distribution including the linux kernel of version 2.6.19. The memory usage for the binaries of each needed service, in the above described configuration, is shown in Table 1. Table 1 also shows the advantage of the service deployment in a zipped fileformat. A bundle is approximately 70 % smaller then the binary equivalent (compared acs_service and acs_service.zip). This decreases the needed message size and by the way reduces the network load and the transmission time.

Service name	Platform	Memory size
hosting service	AMD x64	413,8 kB
	Fox-Board	359,3 kB
life cycle manager	AMD x64	667,5 kB
	Fox-Board	634,7 kB
acs_service	AMD x64	460,8 kB
	Fox-Board	580,2 kB
acs_service.zip (bundle)	AMD x64	150,8 kB
	Fox-Board	175,1 kB

Table 1: service binary size for different platforms

With the used platform it is possible to run the hosting service and the life cycle manager at the same time in parallel. Furthermore it is possible to install 20 different lcmC-services on the Fox-Board and to start 6 of them before the system runs out of memory. The CPU usage was not taken into consideration. The main time during the installation process is needed to decompress and analyze the bundles. This depends and is closely related to the read-write-speed of the system memory.

VI. Conclusions & Future Work

In this paper a platform independent life cycle manager for device centric SOAs was introduced. Firstly we described the implementation of SOAs in device networks and alternative implementations of life cycle management in different technologies were presented. We have shown the ability and benefits of the life cycle manager in WS4D-gSOAP toolkit from the University of Rostock to deploy and manage services from remote. Furthermore we have presented a mechanism for a life cycle manager to control services in a language independent manner. This was possible with the integration of a broker service controlled by the scripting language called Lua. We were also able to transfer and run the concept on an embedded system. In conclusion we can state that the presented concept works well and it is a good base for ongoing researches on the field of platform and language independent approaches.

Concerning future work, some security mechanism should be included into the life cycle manager. The existing solution permits everybody to control all installed lcmC-services. A further improvement would be the opportunity to move a service from one device to another. This requires the ability to break and resume a service. The platform and language independent life cycle manager can be used in various ways. Possible fields of applications are grid computing or industrial production lines.

The lcmC-service and the life cycle manager memory consumption will be reduced soon and so it will be possible to run the presented life cycle manager on smaller embedded devices.

Acknowledgment

This work has been funded by German Federal Ministry of Eductation and Research under reference number 01—SF11H.

References

[1] Service-oriented Architectures for Devices, 2008. http://www.soa4d.org.

[2] Web Services for Devices, 2008. http://www.ws4d.org.

[3] Web Services on Devices, 2008. http://msdn.microsoft.com/en-us/library/aa826001(VS.85).aspx

[4] H. Bohn, A. Bobek, and F. Golatowski. SIRENA – Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains. In International Conference on Networking (ICN), 2006.

[5] W. Dostal, M. Jeckle, I. Melzer, and B. Zengler. Serviceorientierte Architekturen mit Web Services. Elsevier, 2005.

[6] A. I. S. E. GmbH. POCO C++ Libraries, 2008. http://pocoproject.org/.

[7] R. S. Hall and H. Cervantes. Gravity: Supporting dynamically available services in client-side applications. 2003.

[8] R. Ierusalimschy, W. Celes, and L. H. de Figueiredo. Lua - the programming language, 2008. http://www.lua.org/.

[9] F. Jammes, A. Mensch, and H. Smit. Service-oriented device communications using the devices profile for web services. In 3rd International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC05) at the 6th International Middleware Conference, 2005.

[10] Microsoft, Intel, Ricoh, Lexmark. Devices Profile for Web Services, 2006. http://schemas.xmlsoap.org/ws/2006/02/devprof/.

[11] G. Roelofs, J. Gailly, and M. Adler. zLib Technical Details, 2006. http://www.zlib.net/zlib_tech.html.

[12] A. Systems. FOXLX home page - Embedded Linux single board computer. http://foxlx.acmesystems.it/?id=4.

[13] G. Vollant. Minizip: Zip and UnZip additional library, 2005. http://www.winimage.com/zLibDll/minizip.html.

[14] World Wide Web Consortium (W3C). Web Services Architecture, 2004

[15] E. Zeeb, A. Bobek, H. Bohn, and F. Golatowski. Service-Oriented Architectures for Embedded Systems Using Devices Profile for Web Services. In 2nd International IEEE Workshop on SOCNE 07, 2007

[16] OSGi – The Dynamic Module System for Java, 2008, http://www.osgi.org

[17] About the OSGi Service Platform, Technical Whitepaper, June 2007, http://www.osgi.org/wiki/uploads/Links/OSGiTechnicalWhitePaper.pdf

[18] LiPPERT the embedded PC company, http://www.lippertembedded.de/