Applying the Service Oriented Paradigm to Develop Sensor/Actuator Networks

Stephan Sommer, Christian Buckl, Alois Knoll
Department of Informatics
Technische Universität München
Garching b. München, Germany
{sommerst,buckl,knoll}@in.tum.de

Abstract

Programming sensor/actuator networks requires expertise in low-level programming. A major reason is the resource-constraint hardware that is typically used for these applications. The result are systems, where application logic and platform logic is interwoven. This minimizes code reuse and leads to major changes, in case the platform (hardware or software) is changed. In this paper, we present a model driven approach based on the service oriented paradigm to support the different expert involved in the development, namely platform experts, domain experts and end users. The goal of our approach is to enable the use of pre-implemented services in a potentially heterogeneous sensor/actuator network. The execution of these components is performed by a middleware. To address the resource constraints, this middleware is tailored for each application using a domain specific development tool. The platform experts can expand the code generator to support further platforms. Domain experts provide services and describe a potential interaction between different services. The end users can select, configure and combine adequate services to form a running application.

1 Introduction

Wireless sensor networks differ from standard computer systems. They are executed on resource-constrained devices, so that the efficiency of the code becomes essential. This leads to the fact that implementing sensor network applications demands expert knowledge in the application domain and in low-level programming. Modifications of the used platform[1], e.g. by changing the version of the operating system or by using different hardware, leads to the necessity of changing large parts of the system. Since the implementation depends on the used platform, the potential of reusing pre-implemented components is very limited. The wide variety of available platforms contradicts the approach to implement components independent of a concrete application.

[1]By the term platform, we understand the combination of hardware and operating system.

A promising approach is to rely on the concepts of MDA and Service Oriented Architecture (SOA): an application is interpreted as a set of data providing (sensors), data processing (application logic), and data consuming (actuators) services. However, the common implementation of these approaches using a generic middleware is not suitable for sensor/actuator networks as it is too resource consuming. Our solution to this problem is to use a code generator to tailor the middleware to perfectly match the application requirements [1].

This interpretation of SOA differs from other approaches in the sense that the service oriented paradigm is only used for the application logic. Features of the middleware such as communication is abstracted from the user / developer. This approach allows us to exploit the service-oriented paradigm to simplify the development process. In particular, we want to support the different expert groups that are typically involved in the development process of sensor / actuator networks. The first group, called the Platform Experts, have in depth knowledge of the involved platforms and are able to provide services for hardware interaction and middleware extensions. The second group we have in mind are the domain experts. They have expert knowledge to provide domain specific services for the third group, the End Users or Installers. This group does in general not have expert knowledge in implementation details neither in a certain domain nor in a platform. Within our approach, they can simply select services with desired capabilities and interconnect them with advanced tool support to build a concrete application. In combination with suitable hardware, the deployment will then be done automatically. In the following, we will elaborate this approach. The paper starts with a description of the different development groups and the associated development process. Afterwards, we give an overview of the middleware architecture in Section 3 and give in Section 4 a short introduction in the application we are using as example. The related work is referenced in Section 5. Finally, the paper is summarized and some future work is mentioned.

2 Development Process

In our approach the development and deployment process can be split into tree phases reflecting the three different development groups. Within each phase, the experts can focus on their expertise.

Domain Experts The Domain Experts implement the building blocks (Logic Services) that are used later on to build a specific application. In the Building Automation domain for example, a building block can be a heating / air conditioner control service based on temperature information from several sensors distributed across the building and a reference value given by the facility manager. The Domain Expert will have expert knowledge in his domain (e.g climate control in buildings), but does not have to take care of how to actually interface to specific sensors and actors. The interaction with other services is specified on a high abstraction level. A simple heating control might for example have one input reflecting the actual temperature, one input for the reference temperature and one output to control the heater. The in- and outputs are specified based on a domain-specific ontology to have a common understanding. In addition, it is possible to specify constraints like measurement resolution and minimal sampling rates.

Platform Specialist The interaction between different services is realized by the generated middleware. This middleware implements all non-functional services such as data transfer in the distributed system including QoS, service instantiation and execution. It is generated using a template-based code generator [1]. The templates are implemented by Platform Specialists. This group has in-depth knowledge of the hardware or operating system and can implement the relevant parts of the middleware. Due to the expandability of the code generator, new platforms can be easily supported by adding new templates or modifying existing ones. In addition, the Platform Specialist also provides the Basic Services for easy hardware access and extensions. The basic services reflect the software instances of the sensors and actuators that are provided by the hardware. The basic services abstract all implementation details and allow a *black box* usage of the hardware. In contrast to the transformation of a PIM to PSM model done by a platform expert, our experts each contribute artefacts to our model.

Installer / End User Using the Basic Services provided by Platform Specialists representing the hardware infrastructure in combination with the logic services provided by one or more Domain Experts, the Installer / End User assembles the services in the same way he installs and wires the hardware components. After the hardware installation, the application can be launched and configured. This is done by the installer with full tool support. A very simple example would be the control of a shutter. The installer selects a shutter-control application capable of all the features he has in mind. In our application the installer would select, on the one hand, the hardware module for the shutter and on the other hand some push-buttons to allow the user to open and close the shutter. In addition, he can connect a central building control system to the shutter. So, in case of tempest, all shutters can be opened centrally.

3 Middleware Architecture

The result of our code generation (for more details on the generation process see [1]) is an optimized, tailored middleware with embedded and already configured services that implement the application logic. The main task of the middleware is to connect the different services involved independent of their location (local or remote). We decided to concentrate the configuration logic within the system in one component per node, the *Service Broker*.

The Service Broker is the central component of the system and is responsible for data flow and service interaction. Thus, it is also the component that has to be addressed if the system should be reconfigured during run-time. For the first version, we are currently only supporting dynamic data-flow reconfiguration and static placement for basic and control services due to the fact that we are using TinyOS as underlying operating system which does not support dynamic module loading. But our future goals are to support dynamic instantiation of services to increase flexibility and to

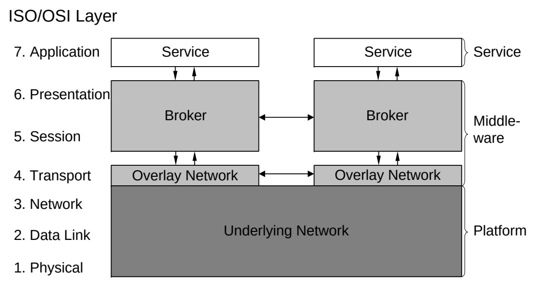

Figure 1. Network Stack

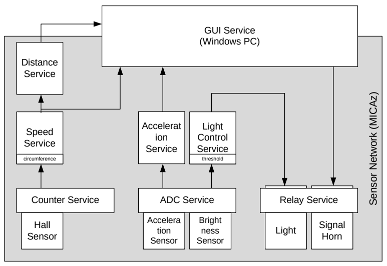

Figure 2. Application Example: Involved Services

address fault-tolerance issues as well as system triggered optimizations at run-time.

The code implementing the service instances is independent of the concrete service interaction. Thus, the service instance needs not to have knowledge about the location of the interconnected services and nodes. The data processing in a service is triggered by incoming data at an input port; after successful processing, the output data is sent to an output port. For basic services, processing could also be triggered by hardware events. Output events of a service are observed by the Service Broker which routes those messages to all connected input ports. The service even does not know where his input came from and where his output will be sent to by the Service Broker.

The interaction between middleware and services is depicted in Figure 1. Top down the involved Services (only one at each node in this example), the Service Broker and the Overlay Network layer are depicted. As described above the decision to which service a message has to be sent is made in the Service Broker. Based on the knowledge on which Node a Service is being executed, the Service Broker uses the Overlay Network layer to send the message to the remote Broker which is responsible for the destination service. The Overlay Network layer is used to abstract from any possible underlying network topology. This ensures us support for heterogeneous communication infrastructure. The underlying network can cover all layers from the physical layer to the transport layer. Our system supports currently UDP/IP, serial connection and I^2C.

Apart from some Services, we have implemented components for this middleware for the versions 1.1 and 2.0 of TinyOS[2] being executed on our MICAz motes and for a Windows PC. In addition, we also implemented some components for a Windows PC that allow the easy implementation of graphical user interfaces to permit easy user interaction with the sensor network. Due to the middleware approach, the services can be placed transparently either on the motes or the Windows PC.

[2]www.tinyos.net

4 Application Example and Evaluation

We evaluated the model-driven development approach, our template based code generator and the separation of system an application logic with different kinds of services in the context of an example application realizing the control of a model railway. As hardware, we used MICAz[3] sensor nodes from Crossbow in combination with a data acquisition board to expand the capabilities and support a wider range of sensors and actuators. For our example application we used brightness, acceleration and a hall sensor for speed measurement. As actuator we used two relays, one driving the train light and one driving a signal horn. To provide easy access to these hardware components, we implemented suitable basic services for those. These basic services can be seen as device drives for our middleware and are not application specific; they can be used in completely different scenarios. In addition to the basic services, we implemented hardware independent control services to calculate the current speed, covered distance, acceleration and brightness level, see Figure 2. Using different basic and logic services, we could compose the application and generate the complete middleware layer being executed on the MICAz motes. To demonstrate the interaction between the user and the sensor network, we implemented an application service for a Windows system which allowed the user to turn the signal-horn on. Summarily, we could show that the approach has significant advantages regarding development time and maintainability.

5 Related Work

Different research teams addressed recently the discussed issue by using macro-programming languages,

middleware and service-oriented approaches for sensor networks [4, 13].
CORBA [11], Minimum CORBA [10], Real-Time CORBA [9] and the .net MicroFramework [14] are widely used middleware standards, but are typically too resource consuming for devices we have in mind.
The OASiS Framework[8] and the SIRENA[7] project aim at developing a framework that allows designing service-oriented sensor or automation network applications with an object-centric point of view; In contrast to our approach, they do not provide automatic code generation.
In contrast to BOTS [12] which also uses generative programming we see our platform as collaboration of loose coupled services provided by multiple vendors and not as a static system image.
The RUNES[2] middleware provides a component oriented programming platform for sensor network applications. However, the design and composition of the individual components is still the task of an expert and cannot be done by the end-user himself.
For home automation, the Konnex (KNX) [6] standard and for industrial-process measurement and control systems the IEC 61499 [5] standard is used to ensure the interoperability of different devices. However, these standards do not address issues like automatic code generation or transparent heterogeneous communication.

6 Conclusion

In this paper, we proposed an approach using domain specific languages and a template-based code generator to accelerate the development of sensor actuator network applications and to increase reusability.
For the domain specific language, we are using a service-oriented approach. The sensor network application is interpreted as a set of services that interact via an event based push model. Basic services are used to access hardware devices and offer an abstraction of the low-level implementation. Logic services in combination with the service composition done in the Service Brokers realize the actual application logic and can be implemented independent of the used hardware and operating system. Thus, also developers without expert knowledge in low-level programming can implement these systems. By introducing an Overlay Network Layer, we established a seamless communication layer to transparently connect multiple heterogeneous devices.
Since we just have started this work, there are a lot of features, we have in mind but could not yet implement. The next step will be to realize a dynamic instantiation of new services in the sensor network at runtime to cope with node failures and improve flexibility. To increase the support for the component developers, we are currently improving our modelling tool splitting up the hardware description into three consecutive steps. Furthermore, we also plan to implement Quality of Service (QoS) methods into the network.

References

[1] C. Buckl, S. Sommer, A. Scholz, A. Knoll, and A. Kemper. Generating a tailored middleware for wireless sensor network applications. In *2008 ieee International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, 2008.

[2] P. Costa, G. Coulson, C. Mascolo, G. P. Piccoand, and S. Zachariadis. The RUNES Middleware: A Reconfigurable Component-based Approach to Networked Embedded Systems. In *Proc. of the 16th Annual IEEE Intl. Symposium on Personal Indoor and Mobile Radio Communications (PIMRC'05)*, 2005.

[3] I. Crossbow Technology. Micaz, wireless measurement system.

[4] S. Hadim and N. Mohamed. Middleware: Middleware challenges and approaches for wireless sensor networks. *IEEE Distributed Systems Online*, 07(3), 2006.

[5] International Electrotechnical Commission. IEC 61499: Function blocks.

[6] International Organization for Standardization. ISO/IEC 14543-3: Information technology - Home Electronic Systems (HES) Architecture - Part 3: Communication Layers and Initiation.

[7] F. Jammes and H. Smit. Service-oriented Paradigms in Industrial Automation. In *IEEE Transactions on Industrial Informatics*, volume 1, pages 62–70, 2005.

[8] M. Kushwaha, I. Amundson, X. Koutsoukos, S. Neema, and J. Sztipanovits. OASiS: A Programming Framework for Service-Oriented Sensor Networks. In *International Conference on Communication System software and Middleware (COMSWARE 2006)*, 2007.

[9] Object Management Group. Real-time corba specification, Jan 2005.

[10] Object Management Group. Corba for embedded specification, version 1.0 beta 1 specification, Aug 2006.

[11] Object Management Group. Common object request broker architecture (corba) specification, version 3.1, Jan 2008.

[12] R. Pandey and JeffreyWu. BOTS: A Constraint-based Component System for Synthesizing Scalable Software Systems. In *ACM Conference on Languages, Compilers and Tools for Embedded Systems (LCTES)*, 2006.

[13] A. Rezgui and M. Eltoweissy. Service-oriented sensor-actuator networks: Promises, challenges, and the road ahead. *Comput. Commun.*, 30(13):2627–2648, 2007.

[14] D. Thompson and C. Miller. Introducing the .net micro framework, 2007.