Developing Service Oriented Sensor/Actuator Networks using a Tailored Middleware

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, mainly because of using resource constraint hardware. The code of the resulting systems has typically no clear separation between application and system (infrastructure) logic. This minimizes the possibility to reuse code and leads very often to the necessity of major changes, in case the underlying platform (hardware or operating system) is changed. In this paper, we present a model driven approach based on the service oriented paradigm to support the different parties 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 that can be easily combined to form an application. The interaction of these components is implemented by a middleware. To address the resource constraints, this middleware is tailored for each application and platform using a domain specific development tool. The platform experts can expand the code generator to support further platforms and features. 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 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. Typical software designs for such systems provide no clear separation between application and system (infrastructure) logic. Modifications of the used platform[1], e.g. by changing the version of the operating system or by using different hardware, may therefore lead to the necessity of changing large parts of the system. Furthermore the potential of reusing pre-implemented components is very limited.

For standard IT systems, middleware approaches are used to allow for a clean separation between application and system logic and to increase the possibilities for reuse of components. Especially in sensor/actuator networks such a middleware approach would also solve the problem of heterogeneous platforms. However, due to resource constraints a standard middleware approach is not suitable as a generic middleware would be to resource consuming. A promising approach is to rely on the concepts of MDA and Service Oriented Architecture (SOA) in form of an actor oriented desing [10] in combination with a tailored middleware: an application is interpreted as a set of data providing (sensors), data processing (application logic), and data consuming (actuators) services. Based on a high-level description of the components and their interaction, the underlying middleware can be tailored to perfectly match the application requirements as shown in [2].

In contrast to other approaches, our interpretation of SOA differs in the sense that the service oriented paradigm is only used for application logic. Features of the middleware such as communication are abstracted from the user and service developer. This approach allows us to simplify the development process. In particular, we can support the different expert groups that are typically involved in the development process of sensor / actuator networks. The first group, called the

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

Platform Experts, have in depth knowledge of the involved platforms and are able to provide services that directly interact with the hardware and components for middleware extensions. The second group are *Domain Experts.* These experts have extensive knowledge to provide reusable and platform independent domain specific services. To support this group, it is necessary to generate platform specific skeletons for these services based on a description of the service interfaces. The resulting services and components of the first two groups can be used by the third group, the *End Users* or Installers, to form a concrete application. This group does in general not have in-depth knowledge in programming systems. Therefore, they must be supported by tools in building the applications. Within our approach, they can simply select services with desired capabilities and interconnect them. In combination with suitable hardware, the deployment will then be done automatically. In the following, we will elaborate this approach. After an overview of the middleware architecture in Section 2, we will discuss the different development groups and the associated development process. Afterwards, we will give a short introduction in the application, we are using as example in Section 4. The related work is referenced in Section 5. Finally, the paper is summarized and some future work is mentioned.

2 Middleware Architecture

The goal of our middleware approach is to assure code reusability and optimal size and run-time behaviour. To achieve this, we are using a domain specific, template based code generator. This allows an application-specific tailoring of the middleware by combining templates implementing middleware functionality and templates implementing the application logic. The result of our code generation (for more details on the generation process itself see [2]) is an optimized, tailored middleware with embedded and already configured services that implement the application logic. The main task of the middleware is to interconnect the different services involved independent of their location (local or remote). In the following we will describe our middleware architecture form bottom to top. At the lowest level we have the Communication component, followed by the Service Broker implementing the message routing. At the top level we have Facilities and Application Services, both connected to the Service Broker.

2.1 Communication

Because of the heterogeneity of our platforms, the capabilities provided by our middleware differ from platform to platform. For example on a Linux computer with full TCP/IP functionality there is usually no need for reliable transport including acknowledgement and retransmission provided by the middleware itself. In contrast, at the lower end of our supported platforms, on an 8-bit AVR[2] node, with a rs232 or I2C [17] communication link, there is no reliable communication provided by the platform itself. In the later case, we have to provide such features in our middleware when required by the application. In order to get small code-size and execution time, the application developer can specify if features like reliable communication are needed for a specific connection of services. These specifications are used to determine, which communication links need to be reliable and which not. Furthermore we provide different configuration options for a reliable communication link. For example we can provide acknowledged only transfer if it is only necessary to know if the message arrived at the sink or not. For reliable communication at specific communication links, an extension can be provided for retransmissions, similar to TCP, at the resource constraint devices and TCP can be used at the more powerful devices where these features are already provided by the operating system. After a valid message is decoded at the Communication Layer, it gets forwarded to the Service Broker for further processing.

2.2 Service Broker

The Service Broker is the central component of the system and is responsible for the logical data flow and service interaction. Each messages which source or sink is on the local node is sent trough the Service Broker. In this component, messages can be split up, if there is more than one service at the local node requiring a specific message, to save network bandwidth and energy. The same feature can be used at the sending node. If the Broker determines that a message from a service at the local node is needed by more then one service at a different node, the broker can decide to send only one message and thus also save energy. The Service Broker is also the component that has to be addressed if the system should be reconfigured during run-time. After messages are processed by the Service Broker the is routed to the appropriate services. Beside the already

[2]We are using e.g. an AtMega8 with 8k flash, 0.5k sram and without operating system.

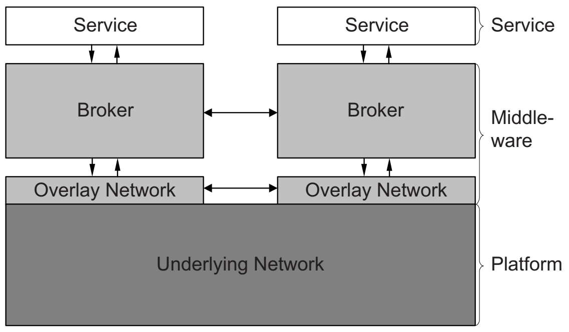


Figure 1. System Architecture

mentioned services providing the application logic, we also have facilities for management purpose.

2.3 Facilities

In our middleware, we use Facilities for network management, e.g. for reconfiguration. In contrast to Application Services, which will be discussed in the next paragraph, Facilities have precise knowledge of their location and communication partners. They can be tightly coupled with middleware components to provide an interface for reconfiguration and node management. For example, the Node Management Facility is used to announce the node to a central management component, to configure the Service Broker, and to send regular alive messages. It could also be used to keep track of the current node status e.g. in case of a wireless node, the battery level could be interesting for load balancing or reconfiguration.

2.4 Application Services

The code implementing the service instances is independent of a 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 one or more output ports. For basic services, processing could also be triggered by hardware events (e.g. timers). Output events of a service are observed by the Service Broker which serializes incoming data, forms a message and routes those messages to all connected input ports where the data is deserialized and routed to the addressed service. Since the routing is performed completely by the Broker, the service implementation can be designed independent of the concrete wiring in future applications.

2.5 Service - Middleware Interaction

The interaction between middleware and services is depicted in Figure 1. 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 to send the message to the remote Broker which is responsible for the destination service. The Overlay Network is used to abstract from any possible underlying network topology and can be seen as an hardware abstraction. It completely decouples the Broker from the underlying infrastructure and is part of the Communication component. Our system currently supports IP, where e.g. no routing has to be done and, as a physical / data link layer serial (rs232) and I^2C connections. A further step will be to evaluate 6lowpan[11] for communication to get a standardized and efficient way for communication between the internet and the sensor network.

We have implemented components for this middleware for the versions 1.1 and 2.0 of TinyOS[3] being executed on our MICAz and Tmote motes and also components 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 if there is an implementation of the service for the desired target platform. In our approach we provide this development infrastructure to support the three developer groups we have in mind for sensor network application development.

3 Development Process

The presented approach allows for splitting the development and deployment process into three phases as depicted in figure 2. Within the first two phases, experts for the application domain and for hardware components can focus on their expertise and create reusable components. These components can then be used by end users to build an application easily.

The development process is supported by a model-based development tools that allows the specification of component interfaces, the addition of templates to specify the functionality of components, and a code generator enabling the automatic generation of code for a concrete system.

The typical workflow is as follows: component providers (domain or platform experts) specify the in-

[3]www.tinyos.net

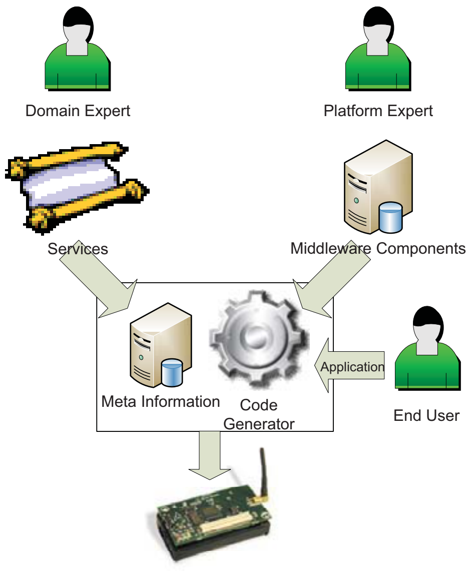
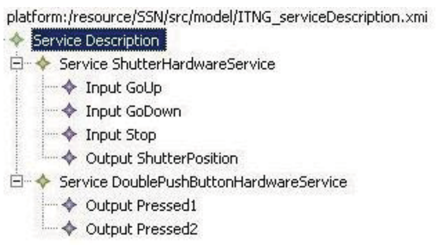


Figure 2. Developer Groups Involved

terfaces of services / components by stating for example input and output ports, as well as component constraints. Based on these descriptions, modeling artifacts and template skeletons are generated. The modeling artifacts are used by end users to model the concrete application. The component providers have to use the template skeletons to implement the functionality of the component. Based on the templates and the application model, the tool can generate the code. In the following, the different groups and their involvement in the development process is presented in more details.

Platform Specialist The interaction and communication 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, execution, configuration and management. It is generated using a template-based code generator [2]. The templates are implemented, maintained and extended by Platform Specialists. Members of this group have in-depth knowledge of the hardware or operating system for a specific platform 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, Platform Specialists also provide basic services for easy hardware access and extensions. Basic services reflect the software instances to access sensors and actuators provided by the hardware. The



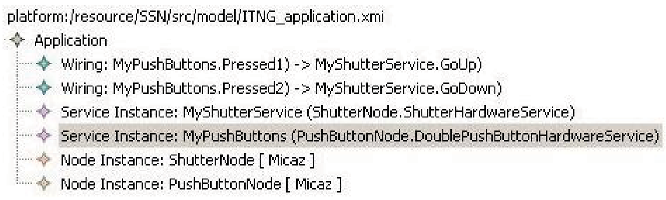
```
platform:/resource/SSN/src/model/ITNG_serviceDescription.xmi
Service Description
  Service ShutterHardwareService
    Input GoUp
    Input GoDown
    Input Stop
    Output ShutterPosition
  Service DoublePushButtonHardwareService
    Output Pressed1
    Output Pressed2
```

Figure 3. Services provided by Platform Expert

basic services abstract all implementation details and allow a *black box* usage of the hardware. An easy example is depicted in figure 3 where the software components for a shutter und two push buttons are modeled using our development tool.

Domain Experts In contrast to Platform Experts, Domain Experts have in-depth knowledge to implement the required functionality for a specific application domain. This functionality is encapsulated in so called (Logic Services) that are later on used to build a specific application. In the Home Automation domain for example, a building block can be a heating / air conditioner control service. Since basic services are available by platform specialists that allow measuring of the current temperature or that read user settings from a control panel, the implementation can be restricted to the pure functionality. The Domain Expert will have expert knowledge in his domain (e.g climate control in buildings) and can implement the pure application logic in ANSI-C for example. 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.

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 configured and launched. This is done by the installer with full tool support. A very simple example would be



```
platform:/resource/SSN/src/model/ITNG_application.xmi
  Application
    Wiring: MyPushButtons.Pressed1) -> MyShutterService.GoUp)
    Wiring: MyPushButtons.Pressed2) -> MyShutterService.GoDown)
    Service Instance: MyShutterService (ShutterNode.ShutterHardwareService)
    Service Instance: MyPushButtons (PushButtonNode.DoublePushButtonHardwareService)
    Node Instance: ShutterNode [ Micaz ]
    Node Instance: PushButtonNode [ Micaz ]
```

Figure 4. Application assembled by End User

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. The selection can be based on the specification of the interface and a textual description. Most important, implementation details are completely abstracted by this approach. The End User only imports the pre implemented Basic Services representing the hardware (figure 3), performs a configuration of the components, and builds his application by interconnecting the involved services as depicted in figure 4.

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 implementing the control of a model railway. As hardware, we used MICAz 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 5. Using different basic and logic services, we could compose the application and generate the complete middleware layer being executed on the MICAz

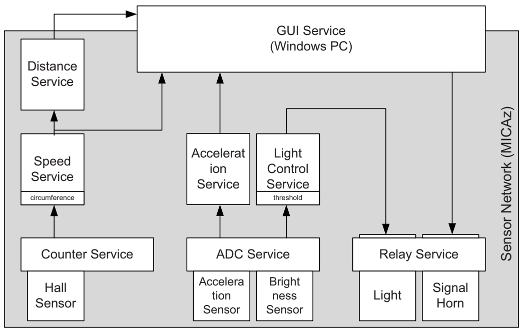


Figure 5. Application Example: Involved Services

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 [5, 16].
CORBA [14] is a widely used middleware standard, but the implementations are typically too resource consuming to be used in the context of wireless sensor networks. The standards Minimum CORBA [13] and Real-Time CORBA [12] define a smaller subset in respect of the constraints in these networks. Nevertheless with a footprint of about 100 kB, the use of CORBA is not feasible for wireless sensor nodes like MICAz that provide 4kB of SRAM and 128kB of FLASH. With about 300kB footprint, the .net MicroFramework [18] is about in the same order of magnitude.
The OASiS Framework[9] and the SIRENA[8] 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 [15] 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[3] 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.
CoSMIC[4] is also focusing on a tailored middleware layer providing only required aspects. In contrast to our approach, they do not focus on different developer groups.
For home automation, the Konnex (KNX) [7] standard, for industrial-process measurement and control systems the IEC 61499 [6] standard and for automotive applications AUTOSAR[1] is used to ensure the interoperability of different devices. However, these standards do not address issues like different developer groups in combination with automatic service composition 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 independent services that interact via an event based push model. The presented approach allows different expert group to focus on their expertise: platform specialists can implement basic services that are used to access hardware devices and offer an abstraction of the low-level implementation. Domain experts provide logic services that implement functionality to create the actual application logic and can be implemented platform-independent. End users without knowledge in programming can finally select and combine the provided components to build a concrete application.
Future work is mainly focused on the features of the middleware. 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. Furthermore, we plan to integrate Quality of Service (QoS) assurance in the middleware. Concerning the support of component developers, we are currently extending our approach by exploiting meta-modeling techniques to support an even easier extensibility of the middleware and addition of services.

References

[1] AUTOSAR – Automotive Open System Architecture. http://www.autosar.org/.

[2] 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.

[3] 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.

[4] A. Gokhale, K. Balasubramanian, A. Krishna, J. Balasubramanian, G. Edwards, G. Deng, E. Turkay, J. Parsons, and D. Schmidt. Model Driven Middleware: A New Paradigm for Developing Distributed Real-time and Embedded Systems? *Science of Computer programming*, 2005.

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

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

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

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

[9] 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.

[10] J. Liu, J. Eker, J. W. Janneck, X. Liu, and E. A. Lee. Actor-Oriented Control System Design: A Responsible Framework Perspective. *IEEE Transactions On Control Systems Technology*, 12, No. 2, March 2004.

[11] G. Mulligan. The 6LoWPAN architecture. In *Proceedings of the 4th workshop on Embedded networked sensors*, pages 78–82. ACM Press New York, NY, USA, 2007.

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

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

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

[15] 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.

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

[17] P. Semiconductors. The i 2c-bus specification. Technical report, 2000.

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