TECHNISCHE UNIVERSITÄT MÜNCHEN
Institut für Informatik VI
Lehrstuhl für Echtzeitsysteme und Robotik

Model-Driven Tailoring and Assembly of Service Oriented Cyber-Physical-Systems

Stephan P. Sommer

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr.-Ing. Alin Albu-Schäffer
Prüfer der Dissertation:	
1.	Univ.-Prof. Dr.-Ing., Dr.-Ing. habil. Alois Knoll
2.	Hon.-Prof. Dr.-Ing. Gernot Spiegelberg, Universität Budapest / Ungarn

Die Dissertation wurde am 8.07.2014 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 18.08.2014 angenommen.

Zusammenfassung

Durch die kontinuierlich sinkenden Hardwarekosten ist es mittlerweile möglich, eine immer größere Anzahl an Sensoren und Aktoren mit intelligenten Kommunikationsschnittstellen auszustatten und somit zu "smarten", vernetzten Geräten zu machen. Im Gegensatz zu früheren, meist isolierten Installationen wachsen diese Geräte zu einem großflächigen, verteilten System zusammen. Dadurch verschiebt sich der Fokus bei der Entwicklung von den einzelnen, isoliert installierten Geräten, hin zu den vom Gesamtsystem zur Verfügung gestellten Diensten und somit zur ausgeführten Software mit ihren Eigenschaften. Um sicher zu stellen, dass die auf den Geräten ausgeführten Anwendungen herstellerübergreifend miteinander interagieren können, sind standardisierte Schnittstellen notwendig. Dies bedingt ebenfalls die klare Trennung zwischen Systemsoftware und Anwendung.

Der erste Beitrag dieser Arbeit ist die Definition einer **angepassten Dienste / Service orientierten Architektur** (SOA), die alle Schnittstellen der Anwendungen klar definiert und die auf den überwiegend stark ressourcenbeschränkten Systemen ausgeführt werden kann. Da als Basis für eine SOA üblicherweise eine Ausführungsumgebung bzw. Middleware als Infrastruktur benötigt wird, ist die Definition einer **modularen und anpassbaren Middleware** zur ressourcenschonenden Ausführung von Diensten auf leistungsschwachen Geräten der zweite Beitrag dieser Arbeit.
Da die Anpassungen der Middleware für jedes einzelne Projekt mit zunehmender Anzahl an Geräten und Diensten zu einer sehr komplexen Aufgabe wird, ist der weitgehende Support durch Werkzeuge und einen geeigneten Entwicklungsprozess Voraussetzung. Der Entwicklungsprozess muss die Beiträge verschiedener Benutzergruppen, nämlich der *Endanwender*, der *Plattform-Spezialisten* und der *Domänen-Experten* in einer eindeutigen Darstellung zusammenführen und gleichzeitig jeder Gruppe den für sie angepassten Ausschnitt des Systems zur Verfügung stellen. Die Definition dieses **Entwicklungsprozesses** und die Spezifikation eines dafür geeigneten **Systemmodells** stellen im Zusammenhang mit der Möglichkeit zur **Generierung** einer angepassten Middleware den dritten Beitrag dieser Arbeit dar.
Die Stichhaltigkeit des vorgestellten Ansatzes wird mittels einer Anwendung im Bereich Heim-/Gebäudeautomatisierung und einer Logistik-Anwendung evaluiert. In Anbetracht der Vorteile des erarbeiteten Ansatzes ist ein Transfer der Ergebnisse in andere Domänen, insbesondere in die Automobil-Industrie als weiterführende Arbeit zu betrachten. Dies ermöglicht das Ausnutzen der Skale-Effekte basierend auf der Massenproduktion um dann, im Anschluss, höher entwickelte Systeme zu einem geringeren Preis herstellen zu können, und diese in Bereichen einzusetzen, für die bis dahin Elektronik als zu kostenintensiv angesehen wurde.

Abstract

Due to the decreasing cost of hardware, a huge amount of sensing and actuating devices are now equipped with communication interfaces and became "smart" and networked sensors or actuators. A sensor actuator network (SANet) can be seen as a huge distributed system with many interconnected devices. In contrast to the small and mostly isolated deployments in the past, nowadays many different devices and even networks are interconnected to form a large scale system, a system of systems. This evolution causes many changes to the well-known system development and maintenance techniques in the embedded domain as there will be different groups and companies involved in the development process. To assure that distributed applications on those devices can easily be interconnected and to get an uniform view of the whole network, standardized and uniform interfaces are necessary. This also introduces a clear separation between applications and the underlying execution environment.

This thesis targets the resulting challenges from the described trend. The first contribution of this thesis is to introduce an **adopted service oriented architecture** (SOA) providing a uniform notion of applications (services) by employing the well known SOA pattern. Key part of this contribution is the tailoring of the SOA idea of services for the mostly resource constraint embedded domain.

The foundation of a SOA is always an execution environment or middleware capable of housing the services and providing the required infrastructure. The definition of a **modular middleware** enabled for tool based **tailoring** and suitable for running on resource constraint embedded devices as well as for housing the services is the second contribution of this thesis.

As tailoring the middleware and so customizing it for each deployment can become a complex task, extensive tool support and a suitable development process need to be available. The proposed development process combines the contributions of distinct user groups namely the *end users*, *platform specialists* and *domain experts* by providing each group with a tailored view of the system model. The definition of this **development process** and the specification of a **system model** to support the system assembly as well as optimization and **extensive code generation** of the tailored middleware is the third contribution of this thesis.

The validity of the elaborated approach is evaluated using a home-/building automation and a logistics scenario. Considering the benefits of the presented approach, a transfer to additional domains especially the automotive domain needs to be considered as future work to employ the scale effect of mass production by developing more sophisticated systems for a much lower price. These systems can then be employed for tasks where electronics are now considered as to expensive.

Acknowledgements

First of all, I want to thank my supervisor, Professor Alois Knoll, for providing me the opportunity to prepare this thesis, for supporting discussions, and to work at his lab. I am also very thankful to Professor Gernot Spiegelberg for his helpful comments and for accepting to be my external reviewer.

Many thanks go to the entire embedded systems and robotics group at the Technische Universität München and the cyber-physical systems group at fortiss for the valuable discussions and the pleasant atmosphere.

Finally, I want to thank my parents for their continuous support, not only during this thesis, but also my whole life, and for all the opportunities they offered me.

Contents

List of Figures

CHAPTER 1

Introduction

Contents

1.1. Background and Motivation

Embedded networks containing a multitude of networked nodes with varying sensing, acting and processing capabilities as depicted in Figure 1.1 are gaining increasing importance in many application areas such as the automotive, building management or the factory automation sector. Mastering these large scale distributed applications has always been a complex and challenging task for which already a huge amount of experience was gathered over time. The challenges used to be concerning the development of suitable hardware devices and communication infrastructures. For new developments, the special characteristics of embedded networks, such as resource limitations, heterogeneous hardware, ranging from PCs over embedded controllers to primitive devices like switches, and the use of diverse communication protocols as well as in-

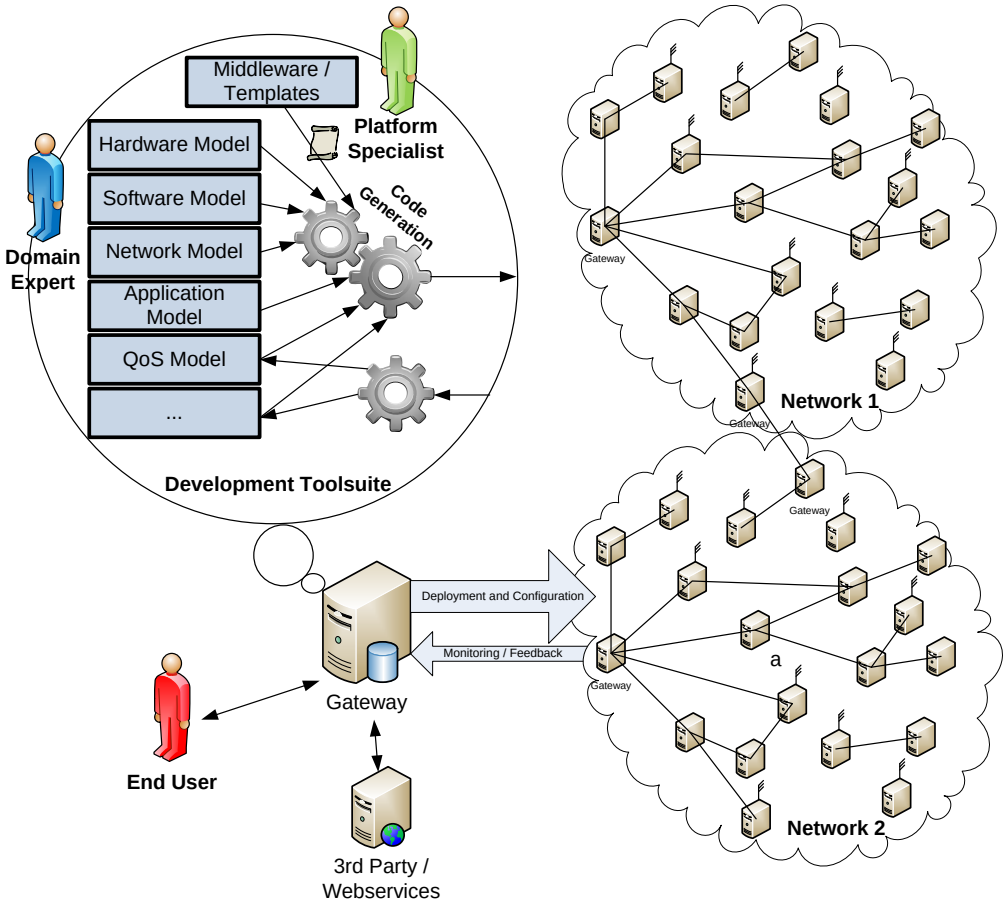


Figure 1.1.: Networked Embedded System Overview: Devices, User-Groups, Models

creasingly complex applications pose new and unique challenges. Analogous to other distributed systems, the development of customized solutions for every single installation is becoming too costly and time consuming.

Bringing the large scale and especially long living applications together with very resource constraint heterogeneous systems on the one hand and the internet on the other hand opens new challenges for research and industry products.

A promising approach is to rely on the concepts of a Service Oriented Architecture (SOA): an application is interpreted as a set of data providing (sensors), data processing (application logic), and data consuming (actuators) services. Nowadays, Web services [BDJ07] are the most prominent SOA implementation and have proven their suitability for building SOA based applications for the Internet. However, the notion of

SOAs known from the Web service domain is not applicable for embedded networks, mainly due to hardware constraints, and requires several adoptions.

This is the application field, where this thesis will contribute to lower the burden for new developments by providing adapted development methodologies as well as an implementation to show their feasibility.

1.2. Terms and Definitions

As terms and definitions in this area of research are mostly ambiguous, a selection relevant for this thesis is elaborated in the following paragraphs and provides a common understanding of the topic for the remainder of this thesis. The challenges as well as the contributions of this thesis cannot be directly assigned to exactly one of the topics as they mostly are overlapping. The discussion of the definitions will provide an introduction to the area of research and the related research topics. In particular these are (wireless) sensor actuator networks, cyber physical systems, systems of systems and networked embedded system.

1.2.1. Sensor Networks / Sensor Actuator Networks

Sensor actuator networks (SANets) in principle do have a long tradition in many different areas. Since many years, engineers employ sensor networks to get a better knowledge and finally also control over the processes, e.g. in industry, agriculture and the automotive domain. Almost all devices where electronics is employed are using sensors to monitor or to interact with the environment. The main focus of sensor actuator networks is the highly distributed acquisition of data using many sensor nodes to get profound knowledge of the environment and to use this information to monitor and control processes. John A. Stankovic introduced sensor actuator networks in the following way:

> *Wireless sensor and actuator networks (WSANs) constitute an important and exciting new technology with great potential for improving many current applications as well as creating new revolutionary systems in areas such as global scale environmental monitoring, precision agriculture, home and assisted living medical care, smart buildings and cities, industrial automation, and numerous military applications. Typically, WSANs are composed of large numbers of minimal capacity sensing, computing, and communicating devices and various types of ac-*

tuators. These devices operate in complex and noisy real world, real-time environments. [Sta08]

1.2.2. Cyber Physical Systems

Over the time, the systems became more and more complex. If there was at the beginning only one sensor and one actuator which were connected to a control unit, the systems began to grow and the complexity began to rise. With the rising number of sensors, controllers and actuators, the possibility to exchange data between different controllers and of course to use the sensor readings provided by one sensor for many different control applications became an important design factor and finally lead to the introduction of bus systems for data exchange as a first step. Although sharing of data was then possible, the devices on a bus usually stayed quite homogeneous at least in the sense of communication.
Although there are different definitions for the term cyber-physical system (CPS), most of them agree that CPS focus is in the interaction of the physical (using sensor and actuators) and the cyber (software) world to control processes. The distributed acquisition and control is in contrast to SANets, not a key focus for CPS. In one of his papers about CPS, Edward A. Lee summarized the challenges coming from the interaction of the real world and the cyber world with the following statement:

Cyber-physical systems (CPSs) are integrations of computation with physical processes. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa. In the physical world, the passage of time is inexorable and concurrency is intrinsic. Neither of these properties is present in today's computing and networking abstractions. [Lee08]

These CPSs usually have one manufacturer or at least a contractor who is responsible for the whole system, who decides how the interfaces to the external world are, and how the components in the system can interact. This developer also decides which services a system can provide to external users or which services it my use.

1.2.3. System of Systems

As soon as we go one step forward, systems consisting of many individual CPS and SANets can be built and responsibilities start to blur. This is when systems of sys-

tems appear and provide new functionality and - most possibly - emergent behavior. System of systems (SoSs) [Kot97] usually consist of many individual systems (CPSs, SANets) which are controlled by individual persons or organizations. The SoS are usually formed at run-time using information provided by the individual systems and the requirements given by them or by a user. Based on these requirements, SoS are assembled at run-time based on contracts between different systems and service providers. Usually for all the participants of a SoS the cooperation provides an additional value to them; this can be the payment for the usage of a system or the additional data that is available due to the cooperation [SSH+07]. In order to allow a flexible system composition, a common knowledge or at least a common understanding of the data and the communication interfaces is required. Thus an ontology is required to describe the system, the services and the requirements and capabilities. In addition, an ecosystem for SoSs needs to provide several services for the users to increase development speed and quality. Although there are many definitions of SoS (Kotov [Kot97], Keating [KRU+03], Manthorpe [Man96]), there is no broadly accepted one, but many of them agree, that the main focus is the interconnection of different systems to provide more use than a single systems could do. In addition, the behavior and composition of these systems can change regularly. Sage and Cuppan summarize the key characteristics as follows:

> *The component systems achieve well substantiated purposes by themselves and continue to operate in this way and accomplish these purposes even if detached from the overall system. The component systems are managed in large part for their own purposes rather than the purposes of the whole. Yet, they function to also resolve purposes of the whole that are generally unachievable by the individual systems acting independently. In other words, these ultimate purposes "emerge" from the SoS. It is not the complexity or size of the component systems or their geographic distribution that makes them a "system of systems", although many contemporary systems of systems will be geographically distributed. Thus, geographic distribution can be viewed as a third characteristic. Another major characteristic that is useful in distinguishing very large and very complex but monolithic systems from a true system of systems is evolutionary development. A system of systems may not appear fully formed and functional initially. Its development is evolutionary in the sense that functions and purposes are added, removed, and modified with experience in use of the system. As a consequence of this evolutionary development, the resulting system of systems will have the property of emergent behavior whereby it functions and carries out purposes that are not possible by any of the component systems. These are the five characteristics of systems of systems detailed earlier.*

> *A system will be called a SoS when all or a majority of these characteristics are present. [SC01]*

1.2.4. Term used in this Thesis

As elaborated above, sensor actuator networks (SANets), cyber-physical systems (CPSs) and system of systems (SoSs) cover an overlapping number of topics. In which of these three definitions a project or deployment fits, is quite blurred and has no deep impact on the topics discussed and contributed by this thesis. Hence the term **networked embedded system** will be used. This also includes the interoperation of the networked embedded system and the Internet.

1.3. Challenges for Networked Embedded Systems

As an introduction the basic challenges for embedded networks will be summarized and later on be used as foundation for the adoptions required to the SOA approach for a feasible deployment to embedded networks. The following discussion will focus on embedded SANets, which are used to perform control and automation tasks. Starting from these rough characteristics introduced above, detailed characteristics are identified in the followings paragraphs:

Heterogeneity

A network built for automation purposes will typically contain nodes with a broad range of different capabilities. Depending on their task, nodes possess a diversity of processing, storage, sensing, and acting capabilities stemming from hardware components supplied by various manufacturers. Another source of diversity are user supplied devices which are used by the end-user to interact with the system, such as cell phones, PDAs, PCs, etc. This *heterogeneity* requires tools that allow building applications without prior knowledge of the exact hardware configuration, while simultaneously exploiting the given hardware resources as efficient as possible.

Distributed and Reconfigurable Architecture

In a control oriented network multiple distributed applications are simultaneously executed, each of them accessing a subset of the available sensors and actuators. As a consequence, a *decentralized network structure* is beneficial for these control applications. It

avoids the bottleneck of a single central node and ensures that not all applications cease to work if a single node fails. A distributed execution is also beneficial from an optimization point of view, because often the amount of transferred data can be reduced by placing the data consuming control logic nearby the data producing sensors. Furthermore, control networks have to be *reconfigurable at run-time*. At any time, new nodes with previously unknown functionality can be added. To support these dynamics, the network has to provide a repository of the available devices and a logging facility that allows retracing changes. Because new applications can be installed at run-time, the purpose of individual nodes in the network is not fixed, but changes throughout the lifetime of the network. This requires a dedicated *life cycle management* that supports the installation, startup, shutdown, and removal of applications on the nodes in the network.

Resource Limitations

Hard boundary conditions of sensor networks are *resource limitations* imposed by the underlying hardware. Consequently, an efficient execution of applications and compact network protocols are important. The diversity of the available hardware additionally requires *scalable functionality*. Small devices should only contain the bare minimum of functionality needed to perform their tasks, whereas more powerful nodes should be flexible enough to provide run-time adaptability.

Error Detection and Recovery

Node failure or communication problems are likely to occur in embedded networks, especially if battery powered devices or wireless links are used. Some problems can be compensated by the used network protocols, e.g., by re-routing data on alternative paths. Other exceptional situations, e.g., a non-functional sensor or actuator, may be compensable if redundant hardware is available. Development tools should support the creation of *robust applications*, which benefit from redundantly available hardware. Furthermore, foreseeable exceptional situations, e.g., energy depletion, should be detected and reported before an actual failure occurs.

End-User Programming

The applications running on a sensor network are typically not known in advance and often no trained personnel are available for the installation of new applications. E.g., an end-user, who wants to configure the mapping of lights and switches in his automated

home, has neither programming experience, nor detailed knowledge about the used hardware. Additionally, the applications executed on the network vary from installation to installation, because they depend heavily on user preferences and the available nodes. The opening of a broad mass market requires concepts which support an easy *end-user programming*, i.e., enable an end-user to intuitively install, (re-)configure, and extend applications. Furthermore, *automation support* for the installation and configuration of applications in large scale installations is important. Subnets with similar functionality should only have to be configured once and similar installations should be configured analogously to existing ones.

Bridging

Embedded networks do not operate in isolation but often possess access to wide area networks or the Internet. An easy integration of embedded networks with external components requires *web service (WS) based interfaces*, as these are the de-facto standard for the communication with external services. The challenge thereby is to connect the WS domain with its high resource demand and its highly available components to the embedded network domain with its small footprint nodes, which might fail from time to time. WS interfaces alone are not sufficient for the integration of embedded networks with enterprise back-ends, e.g., in a shop floor integration scenario. Additionally, the data delivered by the sensor network has to be integrated into the enterprise knowledge domain. This requires *semantic information* that allows combining the measured data with the information contained in the back-end databases.

1.4. Main Contribution of this Thesis

In this thesis, requirements for a middleware for networked embedded systems also considering the aspect of systems of systems are elaborated and an implementation of the key features is discussed. To show the feasibility of the proposed approach two demonstrators are built, one focusing on a home-automation scenario and one focusing on industrial process monitoring. They will be introduced in Section 1.5 and later on referred to in the corresponding sections. In order to handle the high complexity of distributed embedded systems, the requirements an application and a middleware needs to satisfy are quite high, especially if the systems are hand-crafted without proper tool support.

To allow an easy, save, and convenient way to develop these networked embedded systems, modern software development techniques like model driven software develop-

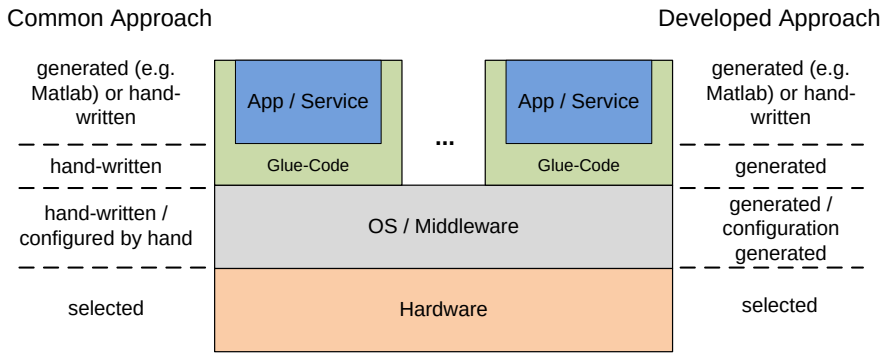


Figure 1.2.: Comparison of Development Approaches: Comparison of the development of networked embedded systems manufactured by using the common approach in relation to the tool-based approach introduced in this thesis

ment and code generation [VSB+13] which are already applied in other application domains needs to be applied to networked embedded systems and, if necessary, adapted for the embedded domain with its special requirements like extra functional aspects. In Figure 1.2 a typical software stack of networked embedded systems is used to compare the common development approach to the key benefits of the approach presented in this thesis.
As soon as techniques like the use of middleware systems, a service oriented architecture or code generation are discussed in the context of networked embedded systems development, similar problem statements are be presented: These are (amongst others) that no efficient code can be generated or that the use of a middleware introduces too much overhead. Although these are quite big challenges, they can be mastered using the approach presented in this thesis. As an introduction for the following chapters, the major challenges are shortly elaborated and a reference is given to the chapter in this thesis, where a solution for the concern is given.

1.4.1. PROVE: SOAs can be Applied to Resource Constraint Devices

The first challenge is about service oriented architectures (SOAs) [DK08] and embedded (resource constraint) systems. As soon as the term SOA is mentioned in conjunction with sensor networks or resource constraint embedded software, people judge the combination as not feasible, because SOA is usually referred to web services [GG+04], Java [Gos00], and XML [BPSM+00]. As a consequence, people tend to think that these techniques are too resource consuming, not deterministically executable, and in the end not practicable for embedded applications at all. But as soon as only the basic concepts and so the benefits of SOA like encapsulation of functionality provided through

well-defined interfaces and the abstraction of the execution location are used, the applicability changes dramatically.
The implementation described in this thesis allows using the benefits of the SOA concept and in addition, maintaining resource consumption due to the tailoring for the embedded domain. The adoption of SOA results in long living and stateful services which are usually implemented in C. The services are instantiated at system configuration or startup time, not for each invocation of a service. The data format is not pure XML-based but realized using an efficient binary protocol and the communication is handled using a middleware tailored to the specific needs of an application. Using this concept, a SOA middleware including all the services can be executed on tiny 8 or 16-bit microcontrollers with not more than 4k of RAM.
A description of these embedded service (eService) and the special requirements and capabilities are elaborated in Chapter 3

1.4.2. PROVE: A Middleware Approach is Feasible for Resource Constraint Embedded Applications

As known from common SOA deployments a suitable middleware needs to be used to deploy services. Having in mind the term middleware, usually heavy weight systems like application servers or systems like J2EE [Joh05] or CORBA [Obj08] are associated. In contrast to these general purpose middleware systems, a middleware for resource constraint embedded systems needs to fulfill certain special requirements like realtime capabilities, resource management, and probably safety features.
The minimum features a middleware needs to provide to justify the potential overhead are mechanisms for communication and data handling. In addition, features like life-cycle management, health monitoring, and fault tolerance might also be an additional requirement depending on the concrete application. To increase development speed and quality and to decrease development cost, a modular middleware is required.

This modularity in combination with the tailoring to the exact application needs provides a suitable way to bring a middleware onto resource constraint devices without producing too much overhead. In order to make such a highly modular and tailor-able system manageable by a developer, tool support is required to do the tailoring process. As a result, the middleware can be applied to very tiny and resource constraint system like an 8-bit microcontroller as well as on state-of-the-art PC hardware. Of course there will not be all high-end middleware capabilities available on small nodes, but the basic set of functionality can be deployed as shown in Chapter 5.

1.4.3. PROVE: MDA is an Efficient Solution for Resource Constraint Systems

When having a look at the Object Management Group (OMG) standard for a model driven architecture (MDA) [MM03], the suggested steps seem to introduce a lot of overhead and complexity into the development process due to the different and highly complex models required for development. These are e.g. the platform independent model (PIM) and the platform specific model (PSM). The PSM is also the basis for code generation. Based on information about the destination platform, a transformation needs to be specified to make a PSM out of a PIM. Basically this seems a good idea, but due to the fact that this approach needs to be generic to be applicable for all kinds of devices, systems and applications, the complexity and overhead is quite high.
In the approach elaborated in this thesis, the application field is restricted to a subset of similar domains and so a special domain specific language (DSL) can be defined for system modeling. This reduces the complexity for the developer, allows optimizing the development process and finally becomes much easier to apply than the general purpose OMG MDA approach. In addition, tailored tools can be developed based on this DSL. Although the approach is more lightweight, the benefits proposed by MDA like portability, cross-platform interoperability, platform independence, and increased productivity in comparison to hand-crafted code are still available. The restriction to a certain field of domains reduces the design space and allows implementing flexible code generators which produce very efficient code. The proposed approach is shown in Chapter 6.

1.5. Demonstrators and Fields of Application

To show the feasibility and prove the applicability of the approach presented in this thesis, two different demonstrators are used showing different application scenarios. The eSOA demonstrator will be mainly used to show the applicability of the eSOA approach in the area of home and building automation. The Multifunk demonstrator focuses on process monitoring in industrial environments and will be used to show the behavior of the system. The system modeling and assembly are for both demonstrators are done with the SensorLab development tool described in Section 6.4.4 which implements the results of this thesis.

1.5.1. Real-time and Fields of Application

Before the application scenarios are detailed, the degree of real-time capability of the demonstrators and the developed system are summarized in this paragraph. In general the following three different levels of real-time [Kop11] requirements for applications are considered:

In the **hard real-time** domain, tasks have a fixed deadline and missing a deadline can cause a critical failure of the whole system and possibly human injury. Application areas for hard real-time systems are for example avionics and the automotive industry. Because of the regulations in this area and the potentially high risk caused by a failure, a migration or reconfiguration of such systems should only performed using a save state where no critical behavior can occur.
In the **soft real-time** domain, deadlines have to be considered as well, but missing these deadlines is not as critical as in the hard real-time domain. In soft real-time applications, deadlines should not be missed, but failures to do so will only decrease the service quality of the applications e.g., slow and uncomfortable response times at a user interface.
The third domain of applications only needs **real world awareness**. Applications which consist of sensors, actuators, and application logic where inputs are processed and used to control actuators are considered real world aware. Fields for these applications can be pure sensor networks and also sensor actuator networks where a non time critical interaction of the network and the environment is targeted. Possible application fields are home and building automation, process automation in industry and environmental monitoring to name some.

In this thesis, the main focus is on applications with real world awareness or soft real-time requirements.

1.5.2. eSOA Demonstrator

In the eSOA[1] project, wireless and wired, networked embedded systems are considered in a home and building automation scenario. The demonstrator is depicted in Figure 1.3 and shows the smart home scenario where energy consumption of a smart home was reduced using intelligent control. The technical goal was to get an end user programmable system for these scenarios. To achieve this, the eSOA system needs to be flexible, cheap, easy to program, and easy to interconnect with external system. In addition, the end user programming paradigm needs to be supported by the provided tool chain. To provide a low cost hardware platform, only resource constraint systems

[1] http://www6.in.tum.de/Main/ResearchEsoa

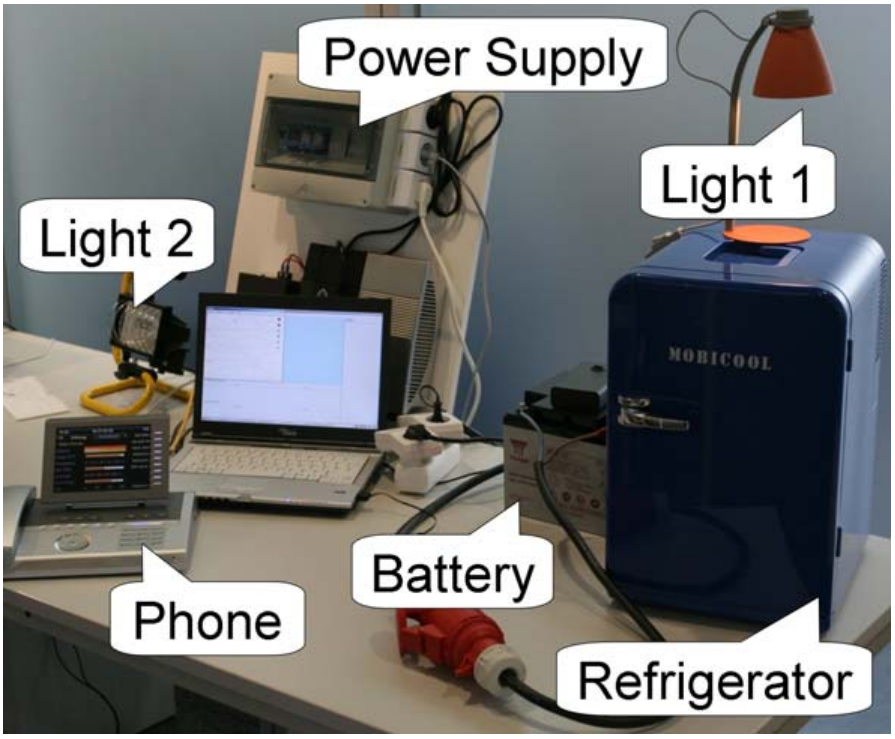


Figure 1.3.: Smart Home Demonstrator: Optimization of Energy Cost

could be used. For the project, these are the MicaZ and the tMote-Sky platform in combination with the TinyOS [LMP+05] operating system. The MicaZ platform is equipped with an 8-bit AVR microcontroller, several I/O pins and some ADCs. For communication, an IEEE 802.15.4[2] based protocol is used. The tMote-Sky platform is based on a 16-bit TI MSP 430 microcontroller and also uses an IEEE 802.15.4 based protocol for wireless communication. To provide an Ethernet-based gateway for easy integration in already existing IT environments, a FritzBox with the Freez operating system is used.
To make software development simpler, an adapted service oriented paradigm for communication is used. Based on these hardware devices, a middleware was developed abstracting from concrete hardware by only providing services for the capabilities available on the devices. In addition, the devices can provide software services housing the control logic for a concrete application. Using the FritzBox, several external devices are connected to the scenario, e.g. a mobile phone, a web-service enabled stationary phone, and services from the Internet.
Considering these internal and external services and devices a highly distributed and heterogeneous system is formed, where communication is performed in a data centric

[2]http://www.ieee802.org/15/pub/TG4.html

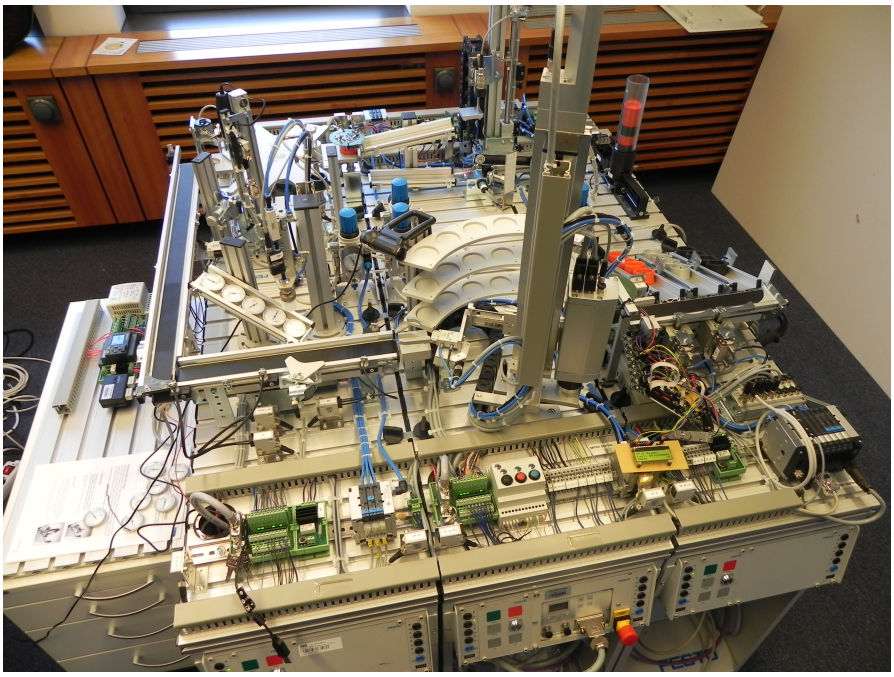


Figure 1.4.: Industry Automation Demonstrator: Work Piece Tracking

way using eServices and web services. The software architecture and additional details are elaborated in Chapter 5.

1.5.3. MultiFunk Demonstrator

In contrast to the eSOA project where also control tasks were involved, the Multifunk[3] project does not focus on control tasks but on data recording in industrial applications. In detail, the goal of the Multifunk project and the corresponding demonstrator depicted in Figure 1.4 is industrial process data acquisition and storage. The demonstrators consists of multiple production stations which are controlled by PLCs [Dum76, Eri96] and handle work pieces. Using the data provided by the PLCs and additional sensors like RFID [Wan06], temperature, and pressure sensors, the process is monitored and the data is stored in a database.

Similar to the eSOA project, the system consists of networked embedded devices, gateways and a database for data storage. To build a flexible and adaptable system, the requirements are modeled using a domain specific modeling tool. Based on this model, which introduces a high level of abstraction, the users can assemble the sensor network based on a library of pre-defined components and operations.

[3] http://www.multi-funk.de

In this project, the basic characteristics of the eSOA middleware, like the data centric and the component-based approach evolved to a more advanced middleware, the CHROMOSOME[4] middleware [SGB+13], where the end user programming is also a required feature.

1.6. Structure of this Thesis

This thesis consists of seven chapters. In Chapter 2 the technical background is elaborated in combination with relevant related work. The contributions adapting a SOA for resource constraint networked embedded systems are presented in Chapter 3. This is followed by the illustration of the model-driven development approach for networked embedded systems in Chapter 4 which provides the framework and the process to unite the contributions in the areas model-driven development, SOA, and middleware. The runtime system and middleware aspects providing the execution container for the services are targeted in Chapter 5. Based on the requirements for networked embedded systems and the specification developed in Chapters 3, 4, and 5 the core part of the presented development process, the transformations assembling the different aspects as well as performing the deployment generation are defined in Chapter 6 as a foundation for the extensive code generation. During these transformation phase formal methods are applied at different steps to check the validity of the assembled system as well as the suitability of the deployment. Finally, the thesis is concluded in Chapter 7, where the contributions are summarized and further work is identified.

In the Appendix A a detailed view of the models derived from the formal specification is given including a discussion of the most important model elements.

[4] http://chromosome.fortiss.org/

CHAPTER 2

Technical Background

Contents

In this chapter, technical background information regarding the topics discussed in this thesis is provided. The information in this background chapter is intended to be an introduction to the topics and challenges, to technical constraints, and to the tooling employed to implement the research concepts of this thesis. The structure of this chapter is aligned to the three main contributions of the thesis. Frist, an introduction to middleware systems in general and to according related work is presented in Section 2.1 and Section 2.2. In Section 2.3 and Section 2.4, the fundamentals of model-driven development including the relevant related work is discussed. This is followed by the presentation of topics relevant for live cycle management in the context of service oriented networked embedded systems. Finally, this chapter is concluded by an introduction to the formal description of the system provided in Section 2.6 as a basis for the formal discussion of the process with its bits and pieces.

2.1. Middleware: Challenges

Since the development and use of middleware systems is nothing new to the IT domain, there are already a lot of concepts, implementations, and products available addressing different aspects of the challenges introduced in the following paragraphs. Although it is an already proved concept to employ a middleware for many applications especially when having in mind, that at least 50-60 percent of a software system is usually developed for communication, error, and exception handling [Geh92] the embedded systems domain needs to be considered separately.

2.1.1. Middleware Fundamentals

During the early days of embedded systems, the focus was mostly on control applications reading and processing local sensor input and calculating output for local actuators. Over the last twenty years, classical embedded systems employed for the control of planes, cars, trains, and industrial settings started to change.

As communication and data transfer between different distributed systems became a key feature, the system complexity was increased by magnitudes. Due to this fact, a great amount of code needs to be written to cover all the extra-functional aspects for the communication infrastructure [CE00].

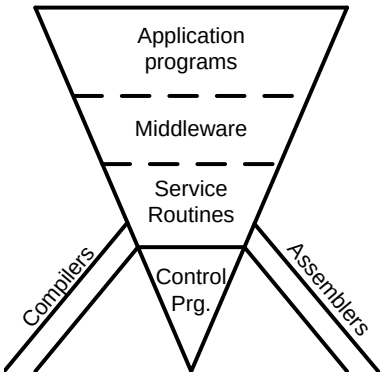


Figure 2.1.: Agapeyeff's Inverted Pyramid [NR69]

A well-established solution of this problem is an application and platform specific middleware providing a well-defined interface for applications. The separation of system and application logic helps to split the software development and lowers the complexity as well as the burden to re-use software. This was already realized in 1969 at a NATO software engineering conference where the term middleware was introduced [NR69] as

a separation between system service routines and the application programs as depicted in Figure 2.1.

2.1.2. Discussion of Middleware Challenges

Due to the long tradition for middleware systems in common software development, there are challenges known a middleware has to handle. In addition to those already well known and solved challenges, there are some additional challenges to focus on when developing a middleware suitable for resource constraint networked embedded systems. In the following paragraphs, a selection of challenges also summarized in [HM06, RKM02, RFC] is discussed in detail:

2.1.2.1. Managing Limited Power and Resources

As networked embedded systems are more and more employed for applications having not even been targeted several years ago, cost is always a critical factor. A common way to reduce cost for a deployment is to shrink a system to its optimal size in the sense of computing power, energy consumption, and space. This hardware tailoring also needs to be reflected by the middleware in order to be applicable to a certain hardware device. This usually leads to customized run-times which cannot be re-used for different applications in an efficient way. For wireless and battery powered devices power consumption is a further challenge in addition to the resource constraints with respect to processing power and memory consumption. Both aspects, the tailoring for constraint processing resources as well as for very limited energy need to be considered during design time by providing a modular middleware including tools for tailoring.

2.1.2.2. Scalability, Mobility, and Dynamic Network Topology

Beside the resource constraints, a middleware for networked embedded system has to focus on mobility and dynamic network topology changes. Even if already deployed network infrastructure like WiFi or Ethernet is available, it is not sufficient to simply use this infrastructure without an additional layer as these networks usually evolve over time and so contain different devices with different networking capabilities. The goal for the middleware is to bridge the gap between different networking technologies beginning from simple RS232 links up to ZigBee [All06] and Ethernet. Depending on the available hardware, different services need to be implemented by the middleware to provide a uniform communication layer for applications. This communication layer also has to handle mobile nodes, changing infrastructure, and additional nodes

joining over time. Even the merge of two distinct deployments can be a scenario the middleware has to deal with.

2.1.2.3. Heterogenety

Heterogenety is introduced to the deployment by the communication infrastructure as well as by different devices. In the past, mostly proprietary networks were deployed in industry where most of them were only connected by a customized bridge to the outside world. A transparent interaction of heterogeneous devices and techniques was not available and not intended. Today as the embedded networks are growing, a consequence is, that these heterogeneous devices need to be seamlessly integrated in process control applications as well as in data warehouse applications. In addition, also legacy systems need to be addressable form the state-of-the-art deployments. To face this challenge, the goal for the middleware is to provide a simple technique to integrate legacy devices as well as being easily adaptable to different devices and systems.

2.1.2.4. Real-World Integration

The seamless integration of embedded networks, process control, and monitoring applications with the remaining corporate infrastructure is a key feature for new deployments. The key benefit is, that beginning from production up to high level business applications, the knowledge can be shared seamlessly if needed. Usually, the transfer from the production to the management layer is implemented using data export functionality. If changes on the low level systems are necessary, these are performed manually. The data derived in the management systems usually cannot be used directly by the embedded devices. As these different systems evolve to a system of systems to increase efficiency and data re-use, a seamless integration is required bottom up and top down. To provide this horizontal and vertical interconnection of (IT-)systems and the real world is a further goal for a middleware employed for networked embedded systems.

2.1.2.5. Application Knowledge

In order to provide good optimizations of network communication, resource usage, and routing, application knowledge is needed. This information can be the knowledge about the amount of data which will be transported in the network as well as the estimated link loads to make sure quality of service[1] requirements can be fulfilled.

[1] QoS properties are, e.g.: timing, bandwidth, footprint

Additionally, communication can be optimized if the amount of data as well as the precision of the data is known. The goal and challenge for the middleware is, to provide a formal notion for the application characteristics which is then used to tailor the system for an application without harming the generic features for a new deployment.

2.1.2.6. Data Aggregation

In order to save as much bandwidth and energy as possible, recorded data can be aggregated on the way to consumers. The aggregation here is a trade of between precision and efficiency. Due to intelligent aggregation methods, sufficient precision can be preserved while the amount of data needed can be minimized. This could be e.g., temperatures measured in a room where 10 sensors are placed but where only the mean temperature of the room is needed by an application. In this case, the data of all these sensors can be aggregated and only one value needs to be sent to the consumer. The goal for the middleware here is, to provide methods or endpoints where these aggregation methods can be integrated without explicitly being placed there by a user.

2.1.2.7. Quality of Service / Non Functional Properties

An additional difference to of-the-shelf middleware systems from the IT domain is the consideration of QoS and non functional properties (NFP)[2] (NFPs) [CNYM00]. These are employed for network transport, for scheduling / execution of applications, and for memory consumption. For a reliable operation of (especially) distributed applications, it is important, that the middleware can provide guarantees in the sense of timing, bandwidth consumption as well as separation to shield applications in case one application misbehaves and e.g., floods the network with messages. These properties need to be taken into account for a single node as well as for a distributed deployment of applications. The goal for a middleware is to provide means of QoS and NFPs. In addition, it is required that the overall system QoS / NFPs can be derived from the application requirements supplied by the user. To assure a reliable operation, the QoS constraints need to be monitored during run-time accordingly.

2.1.2.8. Security

As different systems and entities start to converge or closely work together, means of security, authentication, and authorization need to be provided for a reliable and safe operation. Assuring security in these distributed and heterogeneous systems cannot be

[2] Sometimes also referred to as extra-functional properties.

considered and implemented afterwards at one single point. All the aspects contributing to security need to be considered a priori and need to be a fundamental part or the middleware. The goal is here to unite efficiency for resource constraint systems and security by identifying a perfect tradeoff between both goals for each deployment.

2.2. Middleware: Related Work

In this section, selected related work in the area of middleware systems is presented and discussed based on the challenges elaborated in Section 2.1. The related work is divided into two major parts each presenting a selection of products or projects representing different domains or technologies.

The first part provides a selection of commercial of-the-shelf middleware products widely employed for business applications. The second part presents a selection of work mostly targeting the embedded domain. In contrast to the first part, mostly research-oriented work is considered. Finally the related work and the results are summarized.

2.2.1. Established General Purpose Middleware Implementations

In this paragraph CORBA, .NET, and J2EE will be discussed as representatives for the most relevant middleware concepts for IT systems. All of them have in common, that there are well established products available implementing the standards as well as there are many years of experience available using these systems. Up front needs to be mentioned, that these systems do not really target the market for networked embedded systems but provide many of the features desired for them.

2.2.1.1. CORBA

Common Object Request Broker Architecture (CORBA) [Obj08] is a middleware standard provided by the OMG to enable software applications to communicate with each other. Due to the encapsulation of the communication location transparency is provided and so the applications can be executed distributed without explicit adaptions. To increase usability as well as cross-platform and language usability, the interfaces are described using an interface definition language (IDL) [Sie00] which is then used as a basis to generate implementations for the interfaces for different programming languages like C, C++ and Java. In addition, the endianness of different platforms is also handled within the CORBA stack transparent for the user.

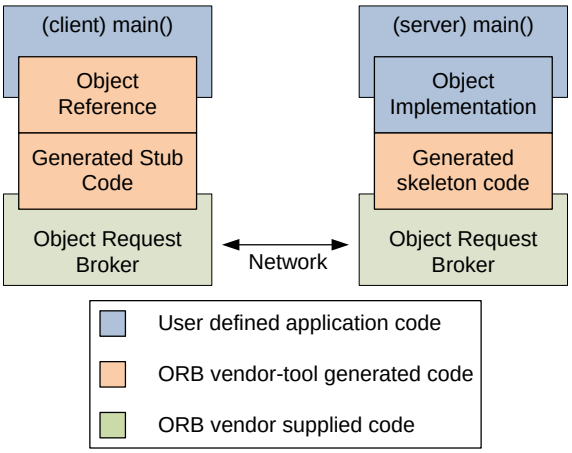


Figure 2.2.: OMG Object Request Broker: Communication Overview [Wik]

To interconnect applications, an object request broker (ORB) is employed as depicted in Figure 2.2 to which all the applications connect to and which takes care of the message forwarding. The data is then forwarded to the receivers connected to the ORB. The great benefit of this approach is, to provide a generic communication scheme between different distributed software modules as soon as a network connection is available.

2.2.1.2. .NET Remoting

A similar approach as proposed by OMG with CORBA is followed by the .NET Remoting Services [Box03] developed my Microsoft. .NET provides a language independent definition of object interfaces as well as a common type system with a mapping to different platforms including marshalling and de-marshalling. The communication is performed using .NET Remoting, a proprietary TCP-based network protocol.

2.2.1.3. J2EE

The Java platform, enterprise edition (J2EE) [Joh05] is a specification of a software architecture to execute distributed Java-based applications. To execute J2EE applications, a distinct run-time environment is required to house the core applications, a so called application server. This server provides the essential services for the interconnection of distributed software modules, like orchestration, persistence, transaction management, security and live cycle management. In the business environment, it is one of the most common architectures and was used by Microsoft as an inspiration for the .NET archi-

tecture. The basic key of J2EE is that applications executed on different nodes connect to one application server, search for information about applications in the directory and then connect to corresponding software modules. Basic services like security are provided by the platform.

2.2.1.4. General Purpose Middleware Assessment

As all three different IT middleware approaches have similar capabilities, they are not examined separately. In the following paragraphs, the basic capabilities presented in Section 2.1 are discussed and the suitability for an embedded approach is elaborated. It is obvious that all of them are perfectly suitable for business applications on standard hardware but lack support for resource constraint networked embedded systems, especially for managing limited power and resources as elaborated in the assessment in the following paragraphs.

Managing Limited Power and Resources

Due to the fact of the generic communication layer, the marshalling, and the remote service calls an overhead is created and so the execution time increases and the determinism decreases. A further drawback is that calls to local software modules are handled in the same way as the calls to remote software modules to comply with the location transparency. This leads to a big overhead, even for local communication. Although there are optimized implementations available like Java 2 micro edition (J2ME) [KT03], where light weight components can be executed on resource constraint devices, the application server is still a quite heavyweight component which needs sufficient processing power [TZL08] and therefore only networks using a powerful centralized infrastructure can benefit from this solution.

Scalability, Mobility, and Dynamic Network Topology

As CORBA as well as the two other implementations rely on a basic network infrastructure. The capabilities to handle this challenge mostly depend on the underlying infrastructure and protocols. Given the basic communication infrastructure provides this service, the middlewares employ location transparent execution of software and so support for dynamic environments is available.

Heterogeneity

Support for different platforms and programming languages is given due to the standard and the various implementations for different operating systems (OSs) and hardware platforms. Targeting network embedded systems, the required support for a full operation systems and high processing power limits the heterogeneity to a limited number of platforms.

Real-World Integration

Real world integration is given due to the fact, that CORBA as well as the other two candidates mostly target business applications and so can be easily connected to these systems. The connection to the embedded world is usually achieved by employing gateways to translate between field level devices and business applications due to the resource requirements. With J2ME and .NET micro framework, the developers have started to target resource constraint devices but still have to rely on a more powerful backend.

Application Knowledge and Data Aggregation

J2EE as well as the other two candidates are basically designed agnostic to application knowledge. So features like data aggregation of application specific optimizations are not targeted by the platform. The platform features are limited to more general ones like queues and storage of persistent data. If data aggregation and fusion are required, this would be handled by a dedicated user application.

Quality of Service / Non Functional Properties

QoS is only targeted by methods like priority queues which do not provide deterministic timing for software module invocation and data processing. One big issue in this context is e.g., the ORB used in CORBA which interconnects requests to the corresponding software modules. To comply with these drawbacks, Realtime CORBA [FWDC+00] was introduced, which provides - in contrast to the standard implementations - a real-time capable ORB and a more efficient implementation to increase determinism.

Security

Security is considered in all three candidates du to its applications in business critical and open systems. Supported features are e.g., encryption of transport data, authorization and authentication of users.

2.2.2. Middleware and Run-time Systems Suitable for the Embedded Domain

The middleware systems described in the past section, which provide many of the required capabilities summarized in Section 2.1, all lack the applicability for resource constraint networked embedded systems. In addition, most of the available concepts and implementations are not capable of realtime systems. In the following paragraphs, middleware systems will be presented and discussed which target networked embedded systems. As some of them target really small devices, the separation between run-time system, operating system and middleware blurs. In contrast to the section discussing off-the-shelf middleware solutions based on the challenges presented in Section 2.1, the following approaches are only discussed on a subset of the challenges due to the research character of most of them.

2.2.2.1. RUNES

RUNES is a component-based middleware ranging from small resource constraint sensor nodes up to high performance desktop PCs. It provides a run-time reconfigurable modularized system consisting of a middleware kernel and services. The middleware consists of two major parts. The foundation is a language-independent, component-based programming model that is sufficiently minimal to run on any of the devices typically found in networked embedded environments. On top of this foundation layer, the middleware functionality is implemented by different, self-contained modules providing the functionality. By composing these modules, the middleware can be individually assembled for each deployment. [CCG+07, CCM+05, CGL+06]

2.2.2.2. TeenyLIME

TeenyLIME is a middleware framework to develop real world sensor network applications. With the middleware, the developer can use data of neighboring nodes with the integrated shared memory approach providing a high level abstraction of a tuple space. Although this increases the development complexity, TeenyLIME provides

an abstraction layer for the developer to mitigate the increased complexity and enhance development speed and quality. This approach also enables in-network-based calculations and reduces e.g. latency in comparison to solutions with a centralized sink. [CMMP07, CMMP06]

2.2.2.3. TinyOS

TinyOS [LMP+05] is currently the most common system for sensor networks in the academic world. It Provides a rich library for supported hardware, comes with its own run-time system and is programmed in NesC, an extended version of the programming language C which supports the means of modules and interfaces [GLC07]. Based on the interconnection of the specified interfaces and mapped interrupt routines, a single ANSI C file is generated reflecting the configuration. Based on this file, a static image is assembled and then flashed to the node. Deployment can be done using direct connections to each node or by one of the available boot loader applications tailored for TinyOS. The deployment itself is not part of the TinyOS system per default.

2.2.2.4. SOS and Contiki

SOS and Contiki, both focus on dynamic, reconfigurable networked embedded systems. Form their approaches, they are quite similar and so they are both discussed in this paragraph.
SOS is an operating system specially designed and implemented for small and resource constraint sensor systems which consists of a basic system kernel which can be extended with additional software modules at run-time. The application modules can be attached or detached form the basic system. The deployment of new components is already integrated in this system. [HKS+05b, HKS+05a]
Contiki is an operating system targeting embedded systems. Similar to TinyOS, it provides hardware abstraction and drivers for many different platforms ranging from high performance devices to resource constraint sensor devices. This system consists of a system kernel housing the basic functionality like SOS. Additional modules can be added to the system during compile-time to tailor the system to user needs. The communication between software modules is performed based on messages and so follows the classical and well established micro kernel approach. To increase dynamic system adaptability, software modules can also be dynamically added during run-time. They communicate with the remaining software modules by sending and receiving pre-defined messages. In contrast to these dynamically attached modules which can be removed and replaced during run-time, the statically attached modules attached

during compile-time are fixed. [Dun06, DGV04] Although this approach provides enhanced efficiency for minor runtime updates due to the dynamic modules, significant changes can usually only be performed by a complete firmware update.

2.2.2.5. ROS, OpenRTM, and DDS

Robotics operating system (ROS) is a middleware stack developed by Willow Garage to simplifiy robotics research. It consists of Linux as operating system and the communication middleware itself uses a similar concept as data distribution service (DDS) [PCI+05] for communication. The applications are assembled using a whole set of toolboxes which contain software modules to control hardware, to do image processing, and to perform simulations. [QCG+09] In DDS data is routed through a network using QoS-aware communication protocols. The sender and receiver are interconnected using *topics*, a common understanding of a data type. Although the OMG DDS approach is heavyweight because of using CORBA as a backbone, there is also an efficient implementation provided by RTI [KPC08] capable of hard realtime and safety. A similar approach to ROS is targeted by OpenRTM [ASK08], a middleware for robotics applications developed by AIST. It basically works similar to CORBA with enhanced assurances in the area of the ORB. It provides a good abstraction layer for development, but is hardly usable on resource constraint hard real-time systems.

2.2.3. Embedded Middleware Assessment

In the following paragraphs, the presented middleware approaches for the embedded domain are elaborated discussed based on the same requirements established in Section 2.1 used for the general purpose solutions. Due to their academic origin, not all requirements are considered by each project, and so only a subset is discussed for each implementation.

Managing Limited Power and Resources

RUNES as well as TeenyLIME and TinyOS were designed up front to be executable on small embedded (wireless) devices. Due to the modular character, the systems can be tailored to build applications on different platforms, although the focus is not on the support of a big variety of systems of different scale and to integrate them into a single, heterogeneous system. Similar to TeenyLIME, SOS, and Contiki were developed for low cost and dynamic sensor network applications but also provide support for being executed on Linux PCs. This focus requires the support for resource constraint hard-

ware beginning form 8-bit controllers with less than 4k of RAM as well as modularity to target different architectures. The development focus of ROS is a little bit different and lies on simplifying the development of robotic applications by using operating system mechanisms for communication and scheduling as well as an IP connection for data communication which results in higher resource demands in comparison to the other systems under consideration. (RTI-)DDS, also relying on a suitable network infrastructure, provides support for devices with small footprint. The provided ressources for processing, networking, and storage directly affect the available featureset of DDS.

Scalability, Mobility, and Dynamic Network Topology

TeenyLIME, SOS, and Contiki are designed to operate dynamic sensor networks including the extension of already deployed networks by new nodes as well as by new software modules. These features are also available in TinyOS except the extension of the network by new software modules during run-time as long as no additional execution layer is installed like a virtual machine [LC02]. As ROS and OpenRTM are both using Linux, the addition of new software modules and nodes during run-time is supported.

Heterogeneity

Due to the modular construction of the RUNES system, it can be easily adapted for different applications, systems, and deployments which can also be employed to provide support for heterogeneous platforms and applications. The sensor network operating systems TeenyLIME, SOS, Contiki, and TinyOS provide a hardware abstraction layer to support multiple different devices. Although the number of supported platforms varies depending on the selected representative, all of them can be considered as providing at least basic support for heterogeneous deployments.

Real-World Integration

All of the introduced middleware applications provide different means of interconnection to the network from external devices, but none used established or standardized approaches like WS. This results in the fact, that for most deployment, a suitable connector to external software needs to be hand-crafted and can - in most cases - not be re-used.

Application Knowledge and Data Aggregation

Inspired by a simple sensing only scenario, data aggregation can be performed in SOS, Contiki, and TinyOS deployments by small software modules deployed in a tree network. Using this approach, data reduction is performed on the way to the root element to guarantee equal distribution of communication among the nodes and to prevent node depletion in case of battery powered nodes if they are near the root element. Based on the rule-set specified in the software modules, values of interest can be additionally forwarded to the root element. For TinyOS the programming can be done using TinyDB [MFHH05], a SQL-like querying language which is taken as a basis to automatically assemble the monitoring application and provides data aggregation points within the network. Similar approaches are available for SOS and Contiki but are not within the core of the middleware.

Quality of Service / Non Functional Properties

One of the key requirements of RUNES as a flexible middleware for networked embedded systems is the support of deterministic execution and system behavior. Although RUNES does not provide planning and verifications tools to assemble the system, verify its behavior, and generate the configuration for the system and the network, it is capable of executing real-time applications. The remaining representatives mostly focus on executing sensor network applications and so only provide limited support for hard real-time applications without the addition of alternative schedulers for execution and communication like the DRAND [RWMX06] providing TDMA[3] communication for TinyOS.

Security

SOS is one of the few sensor network middleware approaches in academia having a look at security. Although the implemented mechanisms are not as hard and elaborate as in the business off-the-shelf products, SOS provides basic security features at reasonable cost for resource constraint systems e.g., authentication of new nodes and different keying concepts for symmetrical encryption. In the area of TinyOS there is independent research like SPINS [PST+02] focusing on adding security to sensor networks considering the special requirements in this domain. For Contiki, the main focus is on dynamic systems, reconfiguration and software deployments during run-time without extended considerations of security features.

[3] time division multiple access

	Implementation					
Target	General Purpose	Embedded				Hybrid
Challenge	Corba, .NET, J2EE	RUNES	TeenyLIME	SOS, Contiki	TinyOS	ROS, DDS, OpenRTM
Managing limited power & resources	✗	✓	✓	✓	✓	✗
Scalability, mobility dyn. topology	✓	✌	✓	✓	✌	✌
Heterogeneity	✓	✓		✓	✌	✌
Real-world integration	✓	✗	✗	✗	✗	✗
Application knowledge	✗			✗	✗	✗
Data aggregation	✗			✌	✌	✗
Quality of service / NFPs	✗	✓	✗	✌	✌	✗
Security	✓	✗	✗	✌	✌	?

Table 2.1.: Overview Middleware Challenges and Implementations: ✓ Supported, ✗ Not Supported, ✌ Considered, Empty: No Information Available

2.2.4. Summary of Related Work

As presented in this section, there are plenty of middleware implementations targeting business and academia. The overview summarized in Table 5.1 also shows, that single requirements are covered by many solutions, but none of them satisfies all requirements. Especially the requirements targeting resource constraint networked embedded systems like QoS and flexibility of deployments (e.g. location transparency). Basically was also shown, that the simplification of software development for the application developers is a key aspect, the configuration and tailoring of the middleware itself is mostly not a focus and only seldom supported by intelligent tools. In addition, the tailoring of the middleware depending on the application area and deployment is not available.

Based on the requirements and the related work discussed in this section, a middleware for resource constraint networked embedded systems was developed within this thesis. The details of the middleware as well as the features implementing a subset of the requirements are discussed in Chapter 5.

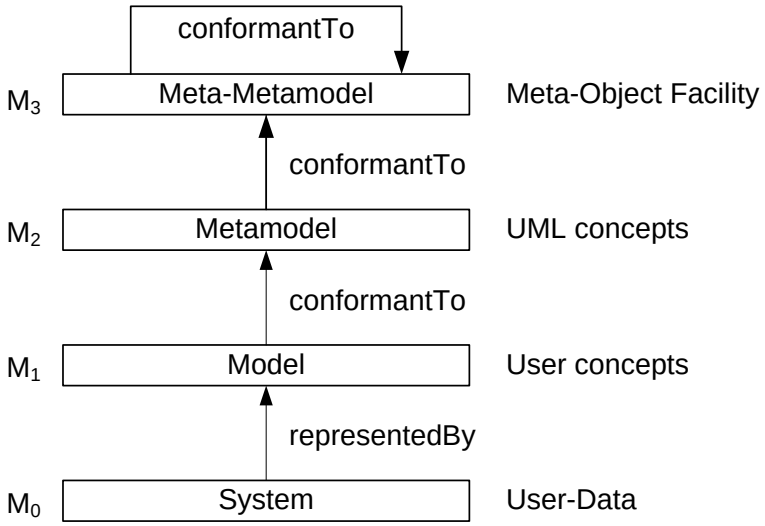


Figure 2.3.: Classic Model Hierarchy [AK03] by OMG

2.3. Model-Driven Development: Fundamentals

In this Section an introduction to model-driven software development in general and the involved components is given. The central part in the model-driven approach is the model. A widely used definition is as follows:

> *A model of a system is a description or specification of that system and its environment for some certain purpose. A model is often presented as a combination of drawings and text. The text may be in a modeling language or in a natural language. [MM03]*

Based on this broad definition of a model as an introduction, formal aspects will be discussed in this section to elaborate a precise definition of the term *model* for the context of this thesis.

2.3.1. Models and their Hierarchy

When using a model, the formal structure, its parts, and the behavior needs to be well defined. This is usually done by a meta-model, where a concrete model is always an instance of its meta-model. It can be compared to XML [BPSM+00], where a XML schema is the meta-model to a XML document. Based on this relationship, a model hierarchy can be established [AK03] as proposed by OMG and depicted in Figure 2.3.

In this hierarchy, each level is characterized as an instance of the level above (except the top level which is self-contained) and can be described as follows: In the bottom level (M0), the real data is represented, which will be processed in the application. This is the layer that represents the real word. In the next level (M1), a representation (model) of this real world is stored. This is the layer used by a developer to create a representation of the target application using a model. To describe the capabilities of the model at level M1, a model is required to represent the modeling options. This is done by the so called meta-model (M2). The name meta-model was selected because it is a model of the model at level M1. Finally, there also needs to be a model of the information stored at M2. This model, a meta-meta-model from the perspective of M0, is in contrast to all other models no instance of a further level. It is self-contained. The model at level M3 is also referred as Meta-Object Facility (MOF) [Obj02, Obj10] in literature. As there now might be a better understanding of a model, its meta-model and their relationship, the next paragraph will introduce mode-driven development.

2.3.2. Model-Driven Development (MDD)

Depending on the field people work in, the meaning of model driven development might differ. The definition provided by OMG for their model driven architecture (MDA) is quite generic and does not explicitly state that the models should be directly used to produce a system (e.g., by code generation):

> *MDA is an approach to system development, which increases the power of models in that work. It is model-driven because it provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance, and modification. [MM03]*

For the approach presented in this thesis where a major focus is on extensive code generation, the following definition provided by Mellor et al. fits best:

> *Model-driven development is simply the notion that we can construct a model of a system that we can then transform into the real thing. [MCF03]*

In this thesis, an approach is presented, which uses different models to represent different aspects of the system under consideration. Based on this representation several checks and transformations are employed to create a final model, the production model, which is used as a source for the template-based code generation. The result of

the generation process is the source code for the complete system including compilation and deployment rules for the participating devices.

2.4. Model-Driven Development: Related Work

The discussion of selected related work in this section is structured as follows: First, different programming paradigms and so levels of abstractions are discussed beginning with the macro programming approach focusing on higher levels of abstraction and its different occurrence to the approaches focusing on local behavior as both levels can be addressed using model-driven software development. In the second part of the related work, concrete model-driven development frameworks, methodologies, and tools as well as their relation to the work presented in this thesis are discussed. The section is concluded by a summary of the related work.

2.4.1. Programming Paradigms and Abstraction Levels

Depending on the application field and the experience and knowledge of the developers, there are different programming paradigms established for networked embedded systems. System developers only focusing on the assembly of applications based on already developed components (e.g., software modules, drivers, and algorithms) are targeted by a macro programming approach. The components assembled by the system developer are provided within toolbox like bundles by different other groups of developers focusing on local aspects of components on the local node hardware.

2.4.1.1. Global Behavior (Macro Programming)

In the macro programming approach, users specify the global behavior of an application [AJG07, NW04, BK07]. A definition of such a global behavior could be very simple: I want the sensor network to measure temperature at all connected sensors once a hour and transmit the data to a central sever where they can be logged. In this case, the user / programmer does not want to focus on how the data is measured, how it is transmitted, aggregated, and stored. Only the result, the temperature saved to a database, is what the user is interested in. Using this paradigm allows users to focus on the key task of a system by specifying some parameters and not to develop sensor network applications from scratch.

Aspect Oriented Programming

One development paradigm supporting this style of macro programming is aspect oriented programming (AOP) [HCG, HC02] where an application is assembled with tool support after the required aspects (pieces of functionality and code) are selected using an abstract definition and requirements and properties are defined. One implementation for AOP is to query sensor networks in a SQL-like query language as it is done in TinyDB [MFHH05, MHH02]. In this approach, users define which aspects and parameters (of which sensor or aggregated values) are of interest. They can also specify the acquisition rate and how data needs to be aggregated. This provides a high level abstraction of the underlying hardware and network infrastructure where the algorithms for the involved nodes are generated tool-based.

Service Oriented Development

The second approach to mention here is the service oriented approach, where the developer builds an applications based on independently developed and executed services [ÖEL+06]. One popular implementation of a SOA can be found in the Internet domain, the so called web services. In contrast to a component-based approach, the coupling in a service-based approach is much more loosely and services can be exchanged by new ones during run-time. In the component-based approach, the coupling is much tighter.

2.4.1.2. Node-centric / Local Behavior

In contrast to the global behavior, where only global effects are defined, the local approach is focused on more fine grained pieces of the system [RDT07]. This can on the one hand be the behavior of a single sensor, a control loop, or a basic algorithm but on the other hand also the behavior of a group of locally adjacent nodes [WSBC04]. In contrast to the developer only having a global view of the system, a developer focusing on local behavior needs to have a more detailed knowledge of the platform, the involved hardware and the algorithms. The developers mainly focusing on local behavior of components usually provide the bits and pieces for a whole application. These pieces can then be employed by other developers not that experienced within a certain domain or technical detail to assemble a sensor actuator network from scratch.

Component-Based Development

Similar to the AOP for global behavior is the component-based development (CBD) of systems with focus on local behavior. As stated by Brown [Bro00], "[...] component-based development (CBD) is application development primarily carried out by composing previously developed software." That means, that in this approach already developed components are re-used as their interaction is specified by the user. Developing in this paradigm increases modularity as components implementing specific behavior provide a well-defined interface and are implemented with respect to flexible assembly and replacement. The implementation is in most cases done using a middleware with well-defined interfaces, where the components can be docked into to form a system. Depending on the implementation, the developer can use the components like a toolbox to build their application [CCM+05, ZWJ+07]. In some cases, even the middleware itself is implemented component-based to increase tailor ability.

2.4.1.3. Summary

Using the separation of local and global concerns, the development can be easily split up for different groups of developers as long as there is a framework and tool support to coordinate and integrate the different bits and pieces into one system. The approach developed within this thesis uses this aspect to support different developer groups and is discussed in Section 4.3. Thereby, the local behavior is needed to add support for the desired hardware and software functionality and the macro programming perspective implemented using tailored web services [SBS+09] makes it much faster and more easy to develop an application even if no expert and in-depth knowledge of the platform is available.

2.4.2. Established Model-Driven Approaches

Most of the related work done can be split into different categories using the aspects which are represented by the tools. The range goes from applications which only consider functional aspects like MatLab / Simulink, to tools focusing on extra-functional aspects. Finally there are also approaches, where functional as well as non-functional aspects are considered.

The different approaches are discussed and the differences to the approach proposed in this thesis are examined. The first approach in this section is the well-known OMG model driven architecture (MDA), which describes elementary parts of systems and

models and can be used to target functional and non-functional aspects, followed by a classical tool focusing on functional aspects. Finally approaches only focusing on-extra functional aspects will be discussed. In the conclusion of this section, the consequences of this work for this thesis are summarized and discussed. The current state-of-the-art is taken as a starting point for the following section, where involved meta-models and models are elaborated.

2.4.3. OMG Model Driven Architecture (MDA)

Having a close look at the progress in software development and the changing requirements and business needs, the OMG realized in 2001, that using models in a software development process can be very useful [Béz01]. To get a flexible and generic solution, the well proven concept of separating the specification of a system from the concrete implementation and capabilities of the underlying platform was employed. This provides the users with the following advantages [MM03]: portability, interoperability, and reusability.

Portability

The portability of a certain system is guaranteed, because it is specified independently of the underlying platform. Platforms in general are also exactly specified. By introducing a transformation from the platform independent specification to a platform specific one, the system can be executed on many different platforms without change to the system itself. This transformation can be seen as an implementation directive for the platform independent descriptions for a specific platform.

Interoperability

By specifying the behavior of an application independently of the potential implementation, the interoperability is guaranteed, as long as e.g., the interaction and communication parts of the system are transformed in the same way to platform specific code.

Reusability

As systems are specified independently of their targeted implementation, they tend to be more generic and so the reuse of them on another platform or in another context is much easier. In best case, the developer only needs to adopt the platform mapping for his application.

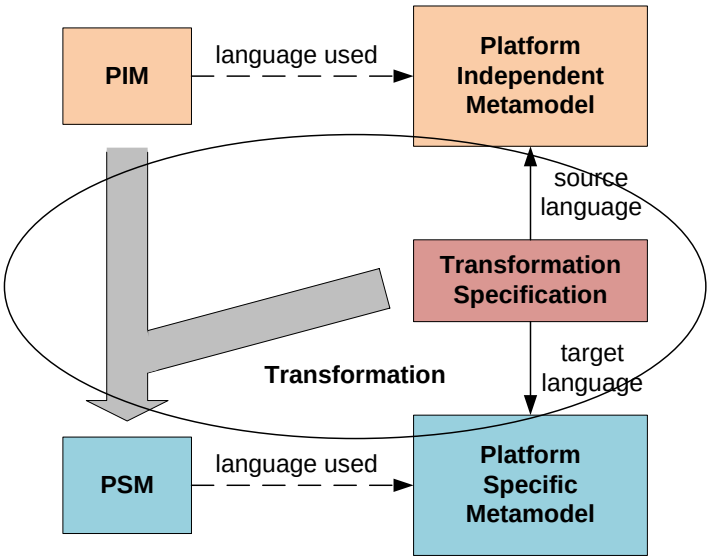


Figure 2.4.: Relation of PIM to PSM [MM03]

2.4.3.1. Approach Proposed by MDA

As already summarized at the beginning of this sections, the MDA approach uses a platform independent representation of a system to describe the basic behavior and structure of the system. This platform independent representation is called platform independent model (PIM). This PIM needs to be transformed to a platform specific representation, the platform specific model (PSM) as depicted in Figure 2.4.

To fulfill the proposed goals of portability, interoperability, and reusability for many different domains, the PIM and PSM need to be quite generic and complex. In addition, the transformation or mapping between them can also be very complex and so the return of investment may be far in the future. In addition, the development process can be quite fragile in the sense that the tooling might change over time. This estimation is also affirmed by a publication of Seitert et al. which states:

> *Using the MDA approach might yield interesting gains with code generation and flexibility regarding the deployment platform, but it does not provide a stable environment for engineering long-lived applications. [SBB04]*

2.4.3.2. Differences from the Proposed Approach to MDA

In comparison to the approach defined by OMG, the approach proposed in this thesis starts form a different direction. There is no generic PIM and no PSM. The proposed approach employs different aspect models each covering a subset of the entire system which are used by the developers to assemble the system. Using these aspect models in combination with a suitable development process as elaborated in Section 4.3 and a flexible code generation to produce a tailored middleware provides the foundation for an efficient system assembly supporting different user groups. However, the approach presented in this thesis somehow uses the same idea as proposed by the MDA: separation of concern and stepwise refinement for a platform. Due to the restriction to certain domains and selected platforms, the overhead and the complexity can be kept at a lower level.

2.4.4. MatLab / Simulink

In contrast to the MDA discussed above, MatLab / Simulink [SN93] is not only a definition of a philosophy or framework of rules a developer should comply with. It is a powerful tool to design, develop, and built complex embedded system. In contrast to the MDA, the focus of Matlab/Simulink is only on functional aspects. This approach is sufficient for many applications, especially for prototyping. There, a fast result is necessary, to get a good estimation of the power of the algorithms and the boundary conditions.

2.4.4.1. Approach Proposed by MatLab / Simulink

When developing an application using MatLab / Simulink, the development progress can be quite high due to many pre-assembled components and functions in the provided tool boxes. Using this approach, the developer can easily focus on his task and does not have to consider many meta-tasks. In addition, the model created and refined during the development process, can be easily used to test an application on different input data. Based on this implementation, code for different platforms can be generated and later on used on the target device. As the transformation of the models to code is considered as error-free, much testing effort can be done on the model level. The drawback of this approach is that only the functional aspects of software are targeted. The developer can examine delays, timing, and so on as long as only one node or program is considered. As soon as there are different components on one system or

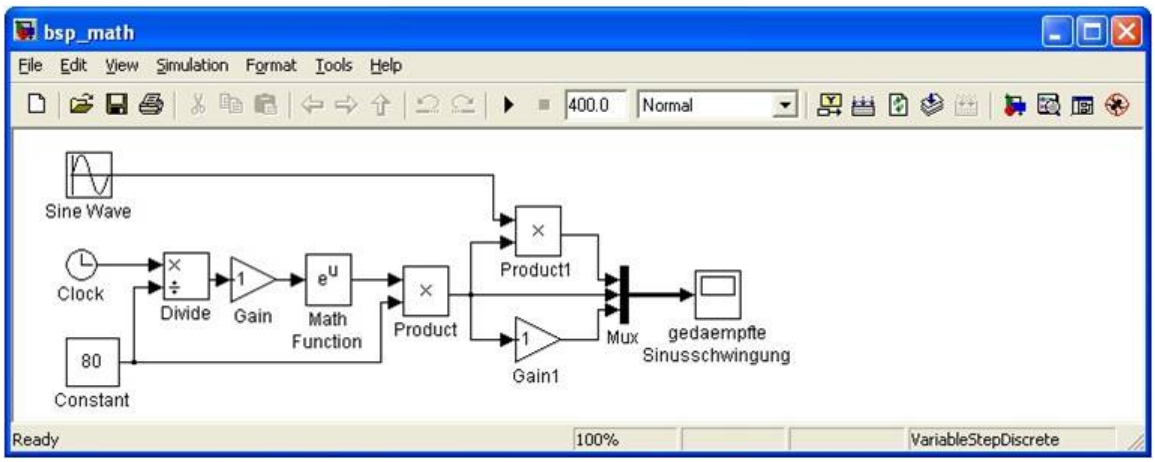


Figure 2.5.: MatLab / Simulink Modeling View

a distributed system is involved, there are no mechanisms to describe or estimate the behavior within this tool.

2.4.4.2. Differences form the Proposed Approach to MatLab / Simulink

In contrast to the approach proposed in this thesis, MatLab / Simulink targets functional aspects of software, which are not considered as part of this work. A beneficial approach is to combine both tools and use MatLab / Simulink to model and generate the application code for a distributed application. These code fragments can then be easily included in a service as application logic and used within the SensorLab tool.

2.4.5. Component Synthesis with Model Integrated Computing (CoSMIC)

CoSMIC[LTGS03, GSL+03] is a model-driven middleware development approach. Similar to the approach presented in this thesis, it is also focusing on a customized middleware layer and a toolchain to develop networked embedded systems.

2.4.5.1. Approach Proposed by CoSMIC

The aspects covered of CoSMIC are middleware tailoring as well as QoS-aware communication and scheduling. The developed toolchain consists of many independent steps providing extension points to refine the model or to add specific, new aspects.

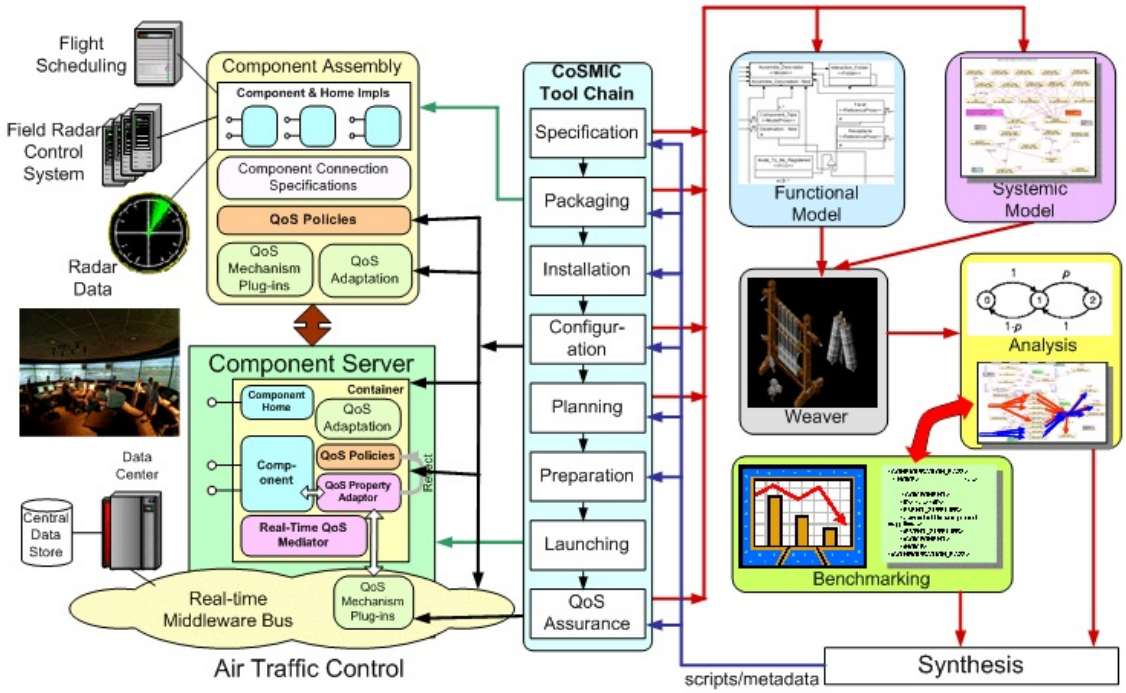


Figure 2.6.: Toolchain Overview of CoSMIC [Com]

An overview of the toolchain is depicted in Figure 2.6.

2.4.5.2. Differences form the Proposed Approach to CoSMIC

In contrast to the approach presented in this thesis, CoSMIC does not support the separation of concern for different developer groups and means of stepwise development to create toolboxes for later use. An additional difference is the resource requirement. According to the application scenarios and case studies, much more powerful hardware is required to execute CoSMIC applications.

2.4.6. Model Integrated Development of Embedded Software (MiDoES)

The approach described in model-integrated development of embedded software [KSLB03] is based on a piercing model-driven approach. The embedded system itself as well as the environment is described using models. In detail, models are used to describe the hardware architecture aspect, the signal-flow aspect, and the environment aspect.

2.4.6.1. Approach Proposed by MiDoES

To allow a simple but precise focus during modeling, the involved models are domain specific. In addition, the models employed are descriptive enough to be used for formal analysis and verification during design-time. To support code generation, the employed models also house enough information to produce platform specific code.
A similar approach is employed by "Model-based Middleware for Embedded Systems" [STV04] as well as by "Automated Middleware QoS Configuration Techniques using Model Transformations" [KG08], and "A Green Family: Generating Publish / Subscribe Middleware Configurations" [BSB05] where the model driven approach is employed to assemble or configure a system. To avoid the overhead of developing a distinctive analysis toolbox for all domains supported, the models are all related by the underlying meta-model.

2.4.6.2. Differences form the Proposed Approach to MiDoES

In contrast to the approach presented in this thesis, this approach focuses mostly on meta-models, the required infrastructure to develop them and editors to actually build models out of the meta-models. Developing a configurable and tailorable middleware including a framework for different user groups using a model-driven approach is not the concern of this related work.

2.4.7. Summary of Related Work

In this section a relevant selection of related work for model-driven / model-based development was presented and discussed with the result, that different approaches are already available to develop new systems and applications. Considering the basic ideas, the OMG MDA provides the best foundation for networked embedded systems but introduces a high overhead. Within this thesis, an OMG MDA inspired model-driven development approach will be elaborated in detail, which unites the high level of abstraction of using models and a low overhead for code generation.

2.5. Life Cycle Management

Having in mind the long time periods of up to 30 years an industrial sensor network is supposed to work, it is clear that the life cycle of such a system has to be considered before deployment. It is quite common, that already deployed installations are extended with further sensors to improve accuracy, that some sensors are replaced by

better ones over time, or that there are different scenarios and workloads the network has to master than it was planned for.

A main aspect for a sensor network considering life cycle management is to provide the ability to customize the whole system at run-time and to adapt it to new challenges after deployment. It can also be crucial to fix software bugs which are discovered after the initial deployment. To provide this ability, it is necessary to also provide an update mechanism for the deployed system which allows a remote update without physical access to the nodes.

2.5.1. Key Aspects and Challenges for a Flexible Sensor Network

To get a flexible and adaptable system, it is crucial to provide as many options for system configuration and update as possible. These options can be split into two different types, system configuration and system update.

2.5.1.1. System Configuration

The term system configuration stands for all configuration and optimization tasks which can be performed without updating the application executed at a specific node or the whole network. Such configurations can be e.g., changes in the configuration of a routing component to force a different path avoiding specific nodes to prohibit energy depletion. These changes could also affect core run-time system components by changing the wirings of involved applications to optimize the message flow or the change behavior.

2.5.1.2. System / Component Update

In contrast to changes in the configuration, where only pre-installed software components are reconfigured or (de/) activated, by updating the system, functionality can be added, removed, or even moved. Having in mind, that changes in the run-time system can affect the whole network, they have to be considered wisely. For example if the network stack of an already deployed sensor network needs to be replaced, it is crucial that this is done in a certain order to prevent nodes form being cut of the network without the possibility to get the new version of the component. A further aspect which needs to be considered is that there needs to be a roll back strategy in case of an update failure or similar problems during or after the update process.

2.5.1.3. Update Challenges

A hard requirement to allow a necessary system upgrade is a technique to deploy new software to the sensor network. Depending on the requirements and size of the sensor network, the deployment strategy can be optimized for a given scenario. When having a closer look at the different possible scenarios, three criteria can be elaborated:

The first one is the system heterogeneity. If the network only consists of nodes executing the same software image, the deployment can be efficiently realized using a flooding mechanism minimizing network traffic. In contrast to such an uniform software image, an update for a network where every node needs its customized image (e.g., because of heterogeneous hardware) can be very resource consuming.

The second aspect to focus on when updating sensor networks is the part of the system which needs an update. This can on the one hand be an application executed on top of a run-time or infrastructure component which only affects the application, not the components provided by the hardware platform e.g., communication, routing, or encryption. On the other hand, an update for the whole run-time, which also affects the infrastructure components, needs careful planning and an option to restore the old version of the system.

The third aspect to consider is the network connectivity if wireless nodes are involved. To master a loosely connected network where nodes or subnets can be disconnected for hours or days need special treatment in the update process. The update process needs to handle the case when nodes are reconnected to the network after the system update was successfully finished for all nodes when the update was initialized.

2.5.2. Related Work

In the following paragraphs, related work for providing a foundation for the live cycle of networked embedded systems is elaborated in two groups. The first group discusses the approach of introducing additional infrastructure to interact with the network and the second group discusses pure software based approaches to maintain a network.

2.5.2.1. Infrastructure-Based Updates

A common approach do deploy new software to a network especially in test setups is a Deployment Support Network (DSN) [DBK+07]. Using this approach, an additional network needs to be deployed to which all nodes are connected instead of the re-use of the already deployed network infrastructure itself. Using a second, redundant network introduces benefits as well as drawbacks. One of the most important benefits is

that a rollback is easily possible using the DSN, even if the whole network is not available because of a bug in the firmware or a broken update. The drawback for a DSN are of course the high costs involved installing a second network covering the whole deployment area which makes this approach only suitable for test labs or academic deployments.

2.5.2.2. Software Based Updates

In contrast to the infrastructure-based approach most suitable for lab deployments, re-using the network infrastructure by introducing an additional software layer e.g., during boot up is the preferred approach for real world deployments. In the following paragraphs, different occurrences of this approach are presented.

Code / Image Propagation and Bootloading

The most common approach in resource constraint embedded networks to update the system is replacing the firmware by an updated version [BS06]. Depending on the concrete application field and underlying system a variety of different solutions are available. Considering networks especially in the area of TinyOS Deluge [HC04], XNP [Inc03], MNP [KW05], MOAP [SHE03], Trickle [LPCS], and Infuse [Aru04] are well known. They all provide the user with functionality to deploy new images to (selected) nodes or even a distinct image for each node in the network. Depending on the reuse of images among different nodes, bandwidth and energy optimized data transfer modes like multicast are available. Even for time triggered networks (TDMA) a solution is available based on Infuse [Aru04].

Modular System Updates During Runtime

When considering flexible and reconfigurable middleware systems, applications need to be added during run-time to the networks or to distinct nodes. Therefor it is usually not useful to deploy new images to many nodes if only one application or service needs to be replaced or updated. Having in mind the dynamic capabilities of modern operating systems, networked embedded systems also need to provide at least basic capabilities for software deployments during run-time. Depending on the underlying hardware and communication resources, different academic solutions are already available providing support for updates like Imapa [LM03], ZebraNet [LSZM04], Contiki [Dun06], and SOS [HKS+05b]. ZebraNet e.g., is based on a middleware system developed for animal monitoring. A ZebraNet application consists of small modules

which can be loaded at run-time. The idea of this tool was that animals will always meet at areas where water resources are available. So an update would be injected on one sensor node and then, over time, propagated through the network form node to node (animal to animal). Using this way, even remote sensor nodes have a high probability of getting updates from time to time.

System Inspection

Similar to the idea of updating distinct software modules across the network, installing new monitors for debugging like declarative tracepoints [CAS+08] and in-field-maintenance framework [CS08] across the distributed system is also an application for updates at run-time.

2.5.3. Summary of Related Work

Considering the discussed applications fields as well as the key aspects in combination with the presented related work, it is obvious that a solid foundation for further work is already available for reuse as an infrastructure. The challenge however is to use these basic features in combination with the modularization of the middleware (Section 5) and the applications to provide a toolbox for new and flexible deployments. As most of the work done in this thesis uses TinyOS as a basic layer, only software updates at nodes level are considered. However for the reconfiguration of deployed software (triggered by the user or by addition of new services or nodes) especially services no system updates are necessary and so the restriction to mostly using TinyOS is no disadvantage.

2.6. Formal Notions

The System discussed in this thesis is specified by four basic parts, the $HardwareDescription$, the $ServiceDescription$, the $NetworkDescription$, and the $ApplicationDescription$. The hardware description specifies the different hardware classes available for specifying the system, the service description specifies the available services including their properties and interfaces. The networking infrastructure (e.g. communication media and links) is specified in the network description. The details of these basic parts will be discussed in detail in the following chapters.

Based on these basic building blocks, the application consisting of hardware, services, and their interaction is specified using the application description. In the following

three chapters, single aspects of the development methodology will be discussed including their contribution to the system specification.

In this thesis a specification is given based on n-tuples representing the components, configurations and variables of the system. The n-th element of a tuple $T(a_1, a_2, ..., a_n)$ can be selected using the projection $\Pi_n T = a_n$. For a more convenient access to an element a_n of a tuple, the notation $T.a_n$ is used in this thesis.

In the next chapters of this thesis, a formal system specification is elaborated bit by bit to provide the software artifacts required for building networked embedded systems. The formal system specification also provides the foundation for the meta-models and modes employed for code generation.

2.7. Summary of Technical Background

In this chapter, the technical background and relevant related work for the topics elaborated in this thesis were presented. The major aspects discussed were the challenges for networked embedded systems middleware solutions providing features like scalability, mobility, and heterogeneity as well as the according related work in relation to the identified challenges. An introduction to model-driven development, the challenges and the related work done in this area was the second topic elaborated in this chapter. The third topic discussed in this chapter was the the life cycle management applied to networked embedded systems considered in this thesis.

Based on this technical foundation, the contributions of this thesis will be elaborated in the next chapters.

CHAPTER 3

Service Oriented Architecture and Embedded Systems

Contents

In this chapter, the question regarding a suitable software architecture for heterogeneous, resource constraint, and distributed embedded systems is discussed. The service oriented architecture is identified as a solution and the required adoptions making the concept well known from WSs suitable for embedded applications are elaborated. In addition, the interconnection of embedded SOA (eSOA) and the corporate SOA world is discussed and a generic transformation mechanism is provided. Finally, the suitability of the approach is shown using the eSOA demonstrator before the contributions of this chapter are summarized.

3.1. eSOA: A Service Oriented Architecture for Embedded Systems

The basic idea of using (web) services is stated in a W3C working group note as follows:

> *(Web) services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. [GG+04]*

Having in mind the challenges introduced in Section 1.3, two major ones (heterogeneous infrastructures and run-time adaptability) are inherently supported by the SOA pattern but usually also come with a big overhead when implemented equally to the well-known solution used for the Internet domain, the web services. In order to make the SOA suitable for embedded systems three major adoptions need to be done in comparison to web services and will be discussed in Section 3.1.2. As a foundation to this discussion, an introduction to SOA is given in the following section.

3.1.1. Introduction to SOA

A practical perspective on the design and implementation of service-oriented solutions is presented in [BDJ07]. Basically a SOA consists of services which communicate with each other. A service is a function that is well-defined, self-contained, and does not depend on the context or state of other services. Based on this description, a service can also be seen an implementation of the actor concept introduced by Agha et al. [AH87] in the 1980s.

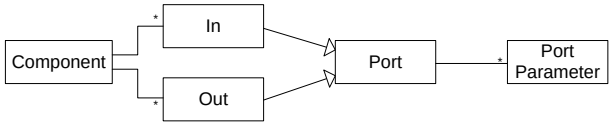


Figure 3.1.: Component Model: Components, Ports, and Parameter

Providing a high level of abstraction using well-defined interfaces, so called ports (see Figure 3.1) hides the implementation details of the components from the user performing the application assembly. Additionally, treating sensors and actuators as services allows dealing with the dynamics of the underlying network. Newly added devices provide services which can be automatically discovered and (semi-)automatically integrated into existing applications or used to build new applications.

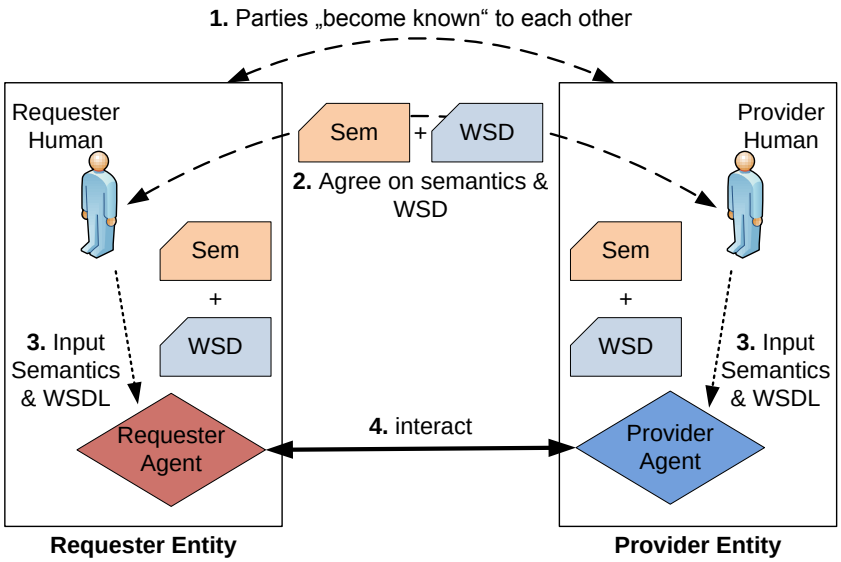


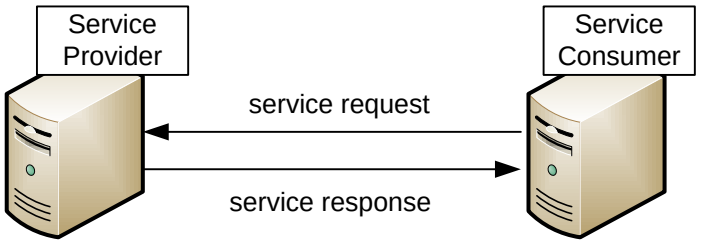
Figure 3.2.: (Web) Service Interconnection and Interaction [GG+04]

A simplified description of service interaction is depicted in Figure 3.2 detailing the different steps to interconnect two services based on their interface description (WSDL) [C+01]. The communication between services follows in most cases a request response scheme (see Figure 3.3(a)) and can be simple, where one service processes data provided by a second service. A complex communication pattern consists of a couple of services, where one service utilizes other services to fulfil his own task.

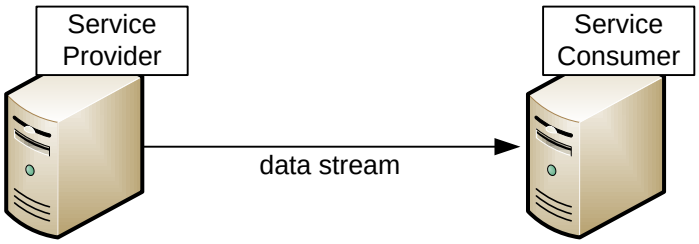
The benefits of SOAs known from traditional application fields such as enterprise service architectures can be translated to embedded network applications. The decomposition of applications into loosely coupled software modules provides high flexibility, re-usability, and extensibility and simultaneously eases the coexistence of different applications. Another benefit is the possibility to integrate services from various hard- and software vendors in a seamless way. Furthermore, due to the high abstraction level, application domain knowledge is sufficient to intuitively understand the functionality of services and to install and (re-)configure applications in the network.

3.1.2. Adoptions of SOA for Embedded Systems

As SOA is a development paradigm, there can be many different implementations and so SOA is usually treated as an equivalent to web service widely used in the Internet do-



(a) Web-Service Communication Example



(b) eSOA: Stream-based Communication

Figure 3.3.: Comparison of Web Service Communication and Embedded Service Communication

main. The basic idea of web services also comes with stateless services, an invocation-based life cycle and dynamic data routing trough the network or Internet. Having in mind resource constraint devices and deterministic behavior, dynamic invocations and data routing suggest a contradiction and so an adaptation of SOA for the embedded domain, an adapted SOA [SBS+09] is necessary to master the challenges discussed in Section 1.3. This approach then provides the advantages of SOA like flexibility and re-usability by loosely coupled software modules as well as the resource efficiency required for the deployment to constraint embedded devices. The limited usability of web services for the embedded domain is no new issue. To overcome the drawback of the heavy-weight software stack required by web services, DPWS [DM09] was in-

troduced by OASIS. In contrast to application servers housing web services, DPWS only provides the bare minimum to execute and discover services, but sticks to XML for communication. Additional contributions to lower execution and communication overhead were made by Moritz et al. [MTSG10, MZP+09]. In contrast to the approach presented in this thesis, DPWS as well as the WS4D [ZMTG10] initiative do not tailor the means of a service for the challenges introduced by networked embedded system. By lowering the execution and communication overhead of web services to make them suitable for devices they only target the symptoms, not the core issues like the service life cycle and the stream-based communication widely used in embedded systems.

In the following paragraphs, the key adoptions of web services and the resulting architecture are discussed based on the main qualities:

Adoption: Data Stream based Communication

While traditional SOAs are based on a request / response message pattern, control applications running on embedded networks are typically stream-based as depicted in Figure 3.3(b). Data is acquired periodically at the sensors and then pushed to connected services. These services produce new data based on the received input which is consecutively pushed to the next service in the processing chain and finally provided to an actuator service.

Adoption: Service Life Cycle

The typical life cycle of service instances in the web service domain is rather short-lived. Individual instances have a lifetime that ranges from some seconds for simple processing services, over some minutes or hours for services involved in web transactions like online shopping. All of them have in common, that they are invoked for each call and terminated after the call was processed. In contrast to the common approach eServices are invoked at system startup time and terminated prior to system shutdown. In the meantime, they are available for data processing. This adoption dramatically reduces the overhead required for processing an event and is perfectly suitable for the stream-based communication employed in embedded networks.

Adoption: Resource Constraints

Services used in the embedded domain are often executed on small devices with severely limited storage and processing capabilities. That requires efficient message formats for the communication between these devices. This can be achieved by a clear

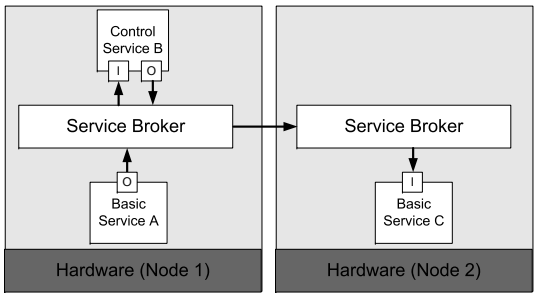


Figure 3.4.: Simplified View: SOA for Embedded Systems

separation between the description of the data, which is specified only once, and the transmission format, which contains solely the raw data payload in a binary representation like EXI [SK08] which is employed in this thesis. Additionally, the services and the middleware executed on top have to be designed in an efficient way to facilitate the execution on small sized devices.

Result: Embedded SOA: eSOA

Services with these special characteristics will be called embedded services eServices in the following. The corresponding SOA consisting of these services is called eSOA. Embedded network applications based on eSOA consist of a set of connected eServices. Each eService can either produce data (sensor eService), process data (logic eService), or consume data (actuator eService) as depicted in Figure 3.4. The available hardware is abstracted as hardware services, which may be an actuator or sensor eService. Logic services on the other hand do not depend on an underlying hardware and are executed on programmable control devices. The communication between services is based on data streams. A data stream consists of a sequence of data packets and connects an output port of an eService with an input port of another eService. In other words, eServices can be seen as operators on data streams.

3.2. Formal Service Specification

In the following, a formal specification of the core part of a SOA, the services is proposed. A service s_i $(s, InPorts, OutPorts, Inst, sResources)$ is described using the service name s, the input and output ports $InPorts$ and $OutPorts$ ($Ports = InPorts \cup OutPorts$) and an instance description $Inst$. The $Inst$ specifies, additional information available to tailor a service using configuration parameters. The resource requirements

of a service are specified by the $sResources$ tuple. To determine if a port p is part of a service s, the notation $partOfService(s, p)$ is used. This function evaluates to $true$ if the port p is part of the given service s. To determine, if a port p is an output or input port, the notion $direction(p)$ is used, where the result is $input$ for an input port and $output$ for an output port. All the services specified are located in the $ServiceDescription$.

$$\begin{aligned} ServiceDescription &= \bigcup_{service\ s_i} s_i \\ service &: (s, InPorts, OutPorts, Inst, sResources) \\ Ports &= \bigcup_{port\ p_i} p_i \\ port &: (p, PortParams, PortID) \\ InPorts &\subseteq \{ip \in Ports | direction(ip) = input\} \\ OutPorts &\subseteq \{op \in Ports | direction(op) = output\} \\ Inst &\in SpecificServiceInstance = \{\emptyset, TimerCounterService\} \end{aligned} \tag{3.1}$$

Each resource requirement ($sResources$) of a service is specified in the context of the execution environment for which a service is available using the tuple $(t, os, pl, sRAM)$ for e.g., the $sRAM$ requirement. Each entry is specified by the device type t, the operating system os and the programming language pl. The $sFLASH$, $sEEPROM$ and $sWCET$ requirements are specified equally. Within this thesis only a tight selection of combinations is implemented: $(os, pl) \in \{(TinyOS, NesC), (Linux, C)\}$

$$\begin{aligned} sResources &: (SRAM, SFLASH, SEEPROM, SWCET) \\ SRAM &= \bigcup(t, os, pl, sRAM) \\ SFLASH &= \bigcup(t, os, pl, sFLASH) \\ SEEPROM &= \bigcup(t, os, pl, sEEPROM) \\ SWCET &= \bigcup(t, os, pl, sWCET) \end{aligned} \tag{3.2}$$

As the interface of a service, a port $(p, PortParams, PortID)$ is described using the port name p, port parameters $PortParams$ and a port ID $PortID$.

$$\begin{aligned} PortParams &: (paramName, paramType, paramRep) \\ PortID &\in N_0; p_a, p_b \in Ports; p_a.PortID \neq p_b.PortID \Rightarrow p_a \neq p_b \end{aligned} \tag{3.3}$$

The port parameter $(paramName, paramType, paramRep)$ is used to specify the signature of a port using the parameter name $paramName$, the parameter data type $paramType$ and the operating system specific representation $paramRep$. The transformation $T_{platform}$ is performed during the transformation steps prior to code generation elaborated in Chapter 6.

$$\begin{aligned} paramType \in & \{char, uchar, bool, int16, int32, int64, uint16, \\ & uint32, uint64, float32, float64\} \\ paramRep \in & SysType : \forall pt \in paramType \; \exists pr \in paramRep : \\ & pr = T_{platform}(pt) \end{aligned} \tag{3.4}$$

3.3. Interaction of Embedded Networks with the Internet

Currently, a lot of research is done to create web service interfaces between field level devices and enterprise systems [dDCK+06, KBDSS07]. This trend will most likely continue, especially because of the envisioned Internet of things (IoT), which aims at integrating all kinds of embedded devices via the Internet.

3.3.1. Interconnection Challenges

The upcoming challenge for application developers is the integration of both worlds, web services on the one side and embedded services on the other side. Real-time awareness[1] for manufacturing or logistics is growing in importance. A break in information exchange between the embedded world and the business back end is not acceptable anymore. Failures and delays on the device level have to be reported fast, in order to allow the timely execution of compensatory actions. Another example is highly flexible production environments, which have to be (re-)configurable from back end services to reduce downtimes and support on-demand production.
This leads to the following four different interaction scenarios.

[1] In this context, the expression "real-time" does not imply hard timing constraints as known from the embedded world, but should be read as "data should be supplied in a timely manner". This is due to the reason that the web services consuming data from the embedded networks do not provide real-time guarantees at all.

Continuous Interaction with the Embedded Network

In this scenario, an external web service continuously interacts with one or more services in the embedded network, e.g., to retrieve measurement values or to submit externally acquired data. In order to keep the communication overhead low and to support non-periodic interactions, the communication between services is managed via subscriptions, i.e., a web service developer subscribes to the output of an eService or announces data submissions to the input of an eService. The management of these data subscriptions can be done with established technologies like WS-Eventing[W3C], but have not been considered in more detail in this thesis.

Ad-hoc Interaction with the Embedded Network

In contrast to the previous scenario, the interaction between services is not planned beforehand via subscriptions, but occurs dynamically. RPC-style web service invocations are an example for this kind of interactions, e.g., in order to retrieve the current measurement value of a sensor, an external service could invoke a *getData* method on an eService.

Continuous Interaction with External Web Services

In this scenario, a developer from the embedded domain wants to retrieve data from or submit data to an external web service on a repeating basis. This interaction has to support the stream-based paradigm used in the embedded network, i.e., to submit data to the external service the developer routes a stream to the web service, to receive data he routes a stream from the web service to the eService.

Ad-hoc interaction with external Web services

The last interaction mode is not meaningful for data-centric services as used in the embedded domain. In the embedded world, applications are installed by connecting the services running on the individual nodes. As the individual services have no knowledge about the concrete wiring, reconfigurations of the application are only triggered by end-users (typically in the web service domain) or the middleware, but not by the eServices.

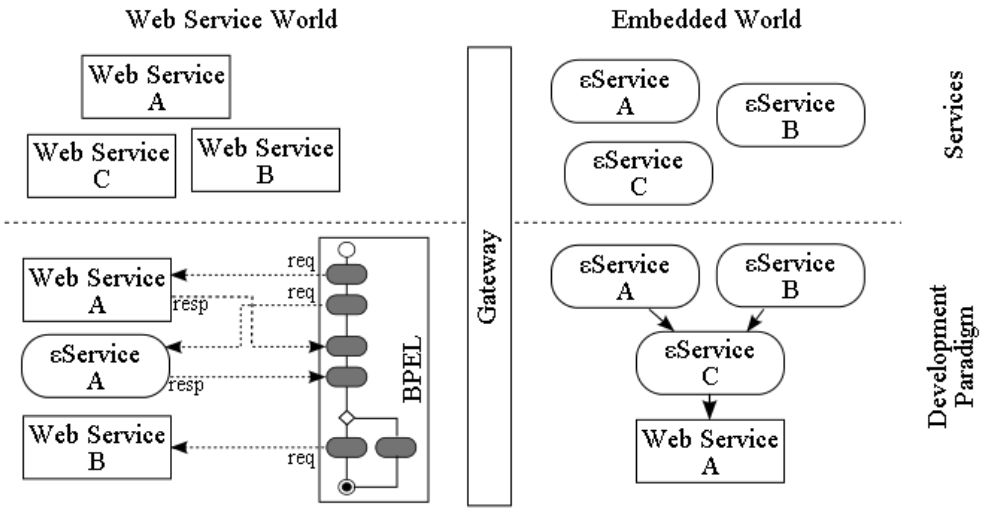


Figure 3.5.: Web Services and Embedded Services - Two Views

3.3.2. Integration of Both Worlds

To solve the issues discussed in the section above, in this thesis the following solution is suggested: The integration has to be performed in two ways, as shown in Figure 3.5. A developer familiar with web service technologies should be able to interact with services from the embedded world just like he would interact with any other web service. On the one hand, if a business process is modeled using Business Process Execution Language (BPEL) [ACD+03] (as depicted in the lower left part of Figure 3.5), the process designer should be able to use eServices to acquire or submit information to field level devices. On the other hand, a developer familiar with application development for embedded networks should have access to services in the enterprise back-end in the same manner as he accesses other eServices. E.g., if data has to be transmitted to a back end web service, it should be sufficient to route the corresponding data stream to the remote service (as depicted in the lower right part of Figure 3.5).

The service gateway introduced is the mediator between the two worlds: it translates messages to facilitate communication between services in both worlds and provides an abstraction layer that supports both of the above mentioned views.

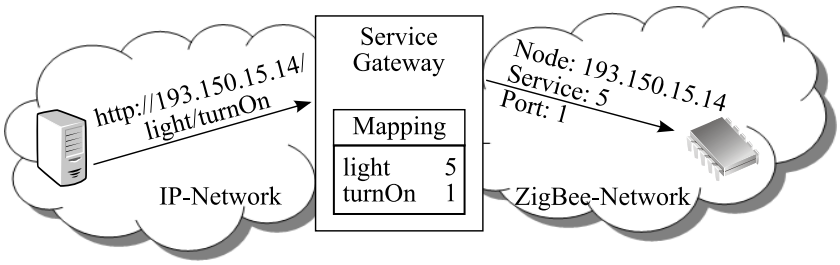


Figure 3.6.: Web Service Bridge interconnecting Embedded and Corporate SOAs

3.3.3. Web Service Bridge

A mechanism to interconnect the two worlds as elaborated is the web service gateway shown in Figure 3.6 which was also presented in [BSS+09]. The gateway mediates between the web service world and the embedded world. Devices from the embedded world are assigned virtual IP-addresses. Web service calls targeted at these addresses are intercepted by the gateway and translated into the message and addressing format used in the embedded network. The same holds for outgoing messages, which are translated to Simple Object Access Protocol (SOAP) messages.

By implementing a lightweight gateway, QoS-requirements can be satisfied if they are supported by all underlying networks and protocols. A single point of failure can be avoided by using multiple gateways and an appropriate balancing mechanism. The performance impact of this solution is hard to quantify since it depends on the concrete hardware. Even if it is possible to run a web service client on all embedded devices, our gateway based solution may outperform the web service solution, since processing overhead can be shifted from resource constrained devices to more powerful gateways. In addition, caching effects can be employed using a generic bridge understanding the traffic semantics.

If a developer wants to access an external web service from the embedded world, the Service Gateway creates a virtual embedded service representing this web service. The virtual service's in- and outputs are created according to the Web Services Description Language (WSDL) [C+01] description of the web service. For continuous interaction, the *One-way* and *Notification* WSDL port types are supported. A one-way port in a WSDL specifies a port, which only receives messages. The virtual service will therefore possess a corresponding input. Analogously, an output is created for every notification port of the WSDL. The correlation between these ports is stored in an internal mapping table in the Service Gateway. From the view of the embedded network, the virtual service is offered by the node hosting the Service Gateway. In order to send data to

the external web service, an embedded service can send data to the input of the virtual service running on the gateway node. The arriving messages are intercepted by the Service Gateway and converted to a SOAP call. The destination web service is determined with the mapping table and the message is forwarded to its destination in the web service world. Incoming SOAP messages are treated analogously. They are intercepted by the Service Gateway and converted to embedded network messages. These messages are injected to the network, as if the output of the virtual service created them.

The Service Gateway does not directly support ad-hoc interaction with external web services because it violates the data-centric processing paradigm in the sensor network. Many benefits of data-centric systems, like free placement of services, splitting and reusing of data streams, etc. are only achievable if the individual services operating on a data stream are implemented "locally", i.e., produce their outputs solely depending on the data received. An ad-hoc interaction would require the service to decide which external web service it should address, which violates this paradigm. If the ad-hoc interaction is needed anyway, it can be mimicked by installing temporary data streams for the duration of the invocation. The message exchange in this case is the same as in the continuous interaction scenario.

In order to make an embedded service accessible from the web service world, a WSDL generator creates a WSDL document describing the eService's interfaces. It will contain a notification type port for every output of the service and a one-way port for every input of the service. Analogously to the interaction with external web services, the correlation between these ports is added to a mapping table. Additionally, the newly generated WSDL is made available through a UDDI[2] [UDD] based discovery interface, which allows users form the web service world to search for specific embedded services. The message exchange in the continuous interaction mode is the same as described in the previous paragraphs.

The support for ad-hoc interactions requires mediation between the pull-based request/response invocation scheme in the web service domain and the push-based communication paradigm in the embedded world. In this case, the Service Gateway will install a caching service and extend the WSDL with a "getter" method for the corresponding output. The caching service has two inputs and one output. The data input is connected with the output of the target service. The caching service will always store the latest data received at this input. If a message is sent to the second input, the trigger input, the caching service will send the stored data from the cache output. The last measurement produced by the target service is therefore pullable via a call to the trigger input. If an embedded device supports on demand data acquisition, i.e., data ac-

[2]Universal Description, Discovery and Integration (UDDI)

quisition can be triggered via submission of a message, the cache service is not needed. Upon the arrival of a request the Service Gateway will trigger the measurement at the target service and send the reply to the web service.

In the example in Figure 3.6, the incoming call for IP-address "193.150.15.14" is converted to a sensor network address - in this case a ZigBee address. For this sample application, our ZigBee nodes are addressed by IP but support for different addressing schemes can be added easily. Based on a mapping table, the service address "light/turnOn" is translated to a service and port identifier. This mapping table is automatically generated by the bridge whenever embedded services are made available as web services. At this point, the bridge generates a WSDL description for the embedded service and updates the mapping table. It is important to note that this approach does not contradict the different communication schemes of web service SOA and eSOA. Ad-hoc messages from the web service world are intercepted at the bridge similar to sensor events. In the following, the message is forwarded using pre-defined (static) connections to the targeted service component.

Following the presented appraoch, a seamless integration of the embedded and the web service SOA world is possible without using hand-crafted transformations for each message. This approach also lowers the burden to extend the service bridge by additional messages.

3.4. Integration of Semantic Information and an Ontology to eSOA

As already indicated in the description of the web service bridge, the web service interface generated by the bridge should provide an intuitive access point to the embedded world. Because the users of this interface will be domain experts and no embedded network programmers, it is important to provide an interface that describes a service in terms of the application domain. This is for example done by using domain specific terms for the identification of services, such as "light" instead of the technical addresses. In order to create these domain specific interfaces fully automated and to ensure that a combination of services is meaningful, services in the eSOA platform possess meta-data information. This meta-data describes the in- and outputs of a service with respect to their technical characteristics, data types, data rates, etc., and the kind of data that is produced or consumed by the service. The latter information is based on a domain specific taxonomy. During the generation of the WSDL, this taxonomy is used to create descriptive names for the web service interfaces. Note that this information

can also be used to ease the discovery of services. Often a user will not know the exact address of an embedded device, but can provide some semantic information that allows determining which device should be accessed. In this example a user could issue a request like "turn on the light in room 4". In this case, the semantic information about the location of an embedded device (which can be attached during its installation) and the fact that the device must have an input that allows modifying "light" can be used to determine the address of the device. This discovery interface can be realized with existing web service technologies like UDDI [UDD].

3.5. Migration Scenarios and the Derived Workflow

As sensor networks tend to be used for a long period of time in industry deployments, additional devices or even additional sub-networks often need to be deployed to adapt the network to new challenges emerging over time.

Having in mind that a deployment is considered an assembly of services distributed over the network forming applications, support for new applications can be provided by installing new services on existing nodes or by adding new nodes. Adding these new services to already existing applications or replacing services in an already deployed application is a critical task because the interaction of different applications have to be taken into account. A second challenging task in these networks is to move already deployed services to different nodes to improve reliability or resource utilization across the network. For both tasks, the state of a service needs to be taken into account for the transitions. In the following, the term state comprises all (configuration and run-time) information locally stored in a service necessary to process incoming data. Here one big advantage of the service paradigm is that the state can only be changed by data received from ports.

In the following the reconfiguration scenarios relevant for this thesis are elaborated.

3.5.1. Extension of already Deployed Applications

Extending an already deployed application by a new, additional service is the easiest update scenario. In this scenario, the new service requires data already provided by one service of the old application. It is only necessary to deploy the new service to the network and to connect its input ports to some of the output ports of the already deployed application. Before deploying the service, possible changes in network utilization and resource consumption need to be considered. For this, the same workflow can be used as for the initial service placement, but with fixed placements for the already deployed

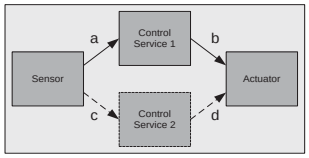


Figure 3.7.: Simple Application Containing a Sensor, two Control Services and an Actuator

ones. The deployment can be done at run-time without harming the already deployed application if the new service can be instantiated at run-time (depending on the platform), or if the new service is deployed to a node not involved in any application until now.

3.5.2. Service Migration without Explicit State Transfer

A second scenario for service migration is the replacement of an already deployed service by a new one. First only a migration where no inner state of the service needs to be transferred is considered, either because the service is stateless or because the new service can automatically recover the internal state. This could be the case for very simple services like data converters or basic logic operators and for those services, which can acquire the state over time just by listening to input data[3] like services calculating the average of the last x values. Afterwards, this scenario will be extend for state full services.
The first step for the migration or replacement [4] of a service is to deploy the new service to an adequate node. As already mentioned for the first scenario, the placement can be done using the already available tools. A very simple application is depicted in Figure 3.7. This applications consists of a source service (Sensor), a control service (*Control Service 1*) and a destination service (Actuator). The *Control Service 2* is the service which replaces the *Control Service 1*.

Stateless Services

For stateless services, the migration is almost completed at this point. The only remaining task is to remove the data paths connected to the old service and add connections for the new one. Usually, connections from a source service which provides the data to the service, and further connections from the output ports of the service to all data subscribers exist.

[3]This interface has to be implemented by the service developer.
[4]A migration can also be seen as a replacement of a service by a new one of the same kind.

The best way to perform this task is to add the new connections for the newly deployed service beginning from the sink side to the service (Figure 3.7, connection c) and, after that, from the new service to his data recipients (Figure 3.7, connection d). The removal of the connections involving the *Control Service 1* is done vice versa (Figure 3.7, connection b and a).

Stateful Services

For statefull services which can acquire the current state only by listening to the data flow and therefor do not provide an interface for explicit state migration, additional tasks need to be performed before the migration is complete. The first steps are the same as for stateless services until the connections from the new service to the subscribers are added. Before these connections can be configured (Figure 3.7, connection d), the reconfiguration process has to be a halted until the correct internal state of the new service (*Control Service 2*) is acquired. After this service is up to date, the remaining data paths can be added from this service to all the subscribers. At the subscribing nodes, the reconfiguration (removal of the old data paths related to the old service (connection b) and the addition of the new data paths for the new service (connection d)) needs to performed in a transactional way. If a message arrives in this very short transaction phase, it has to be buffered to avoid possible application misbehavior.

3.5.3. Service Migration with Explicit State Transfer

The approach already described for stateless services, can be extended to handle the migration of statefull services which require an explicit state transfer. Handling the explicit state transfer also comes with coordination challenges for the reconfiguration of data paths. As shown for the case without explicit state transfer, the data paths need to be adapted in a coordinated way to integrate the newly deployed service.
To guarantee a seamless and consistent migration, it is important to add the data paths from all sources to the new service in exactly the moment where the state transfer begins. After that point in time, the old service does not receive further data until the state transfer is completed. For the new service receiving the state, it is important to not start processing of inputs before the state transmission was completed. If the application is e.g., a climate control system in building automation and some temperatures measured are lost (assuming a high enough data acquisition rate) it will not harm the application. If data loss in an application with very rare events or a user interaction e.g., a user pushing a button is considered, the application and the real world could get

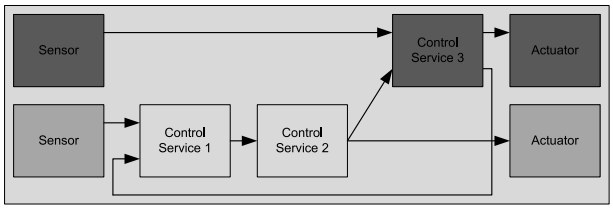


Figure 3.8.: Multiple Applications with Overlapping Services

inconsistent without buffering the messages.
If the service to migrate is involved in many different applications, all requirements of the involved applications need to be taken into account. For example some of the involved services might tolerate the data loss while others do not, some can tolerate downtime while others cannot. In the worst case all applications related to a service which needs to be migrated need to be stopped.
According to the challenges elaborated for the service migration in this paragraph a workflow is derived and presented in the next paragraph.

3.5.3.1. Migration Workflow

To reduce complexity, the migration of services is implemented as a stepwise approach. The migration starts with the creation of an instance of the eService at the destination node. The way this instance is created depends on the runtime and the underlying system infrastructure.
After a new instance is created at the destination host, the internal state needs to be transferred to the new instance. The state transfer is split up into four phases namely *serialization*, *state transfer*, *de-serialization* and *reconfiguration*.
In the process of state transfer two components are involved in addition to the source and destination service. These components, namely the $MigrationCoordinator$ which initiates the migration, and the $MigrationFacilities$, which actually performs the local operations, necessary for the migration at the nodes. An example scenario is depicted in Figure 3.9. In this scenario, the service instance x is moved from the node B to node C (x'). All management extensions for the middleware namely the application repository, the facilities for network management, and the $MigrationCoordinator$ are located at node A in this example. The interaction of services is stored in the application repository. Information related to the network is stored in the network management, which subscribes statistics from the nodes (using the middleware) in the network and processes them to provide e.g., utilization statistics. Although information is gathered by the system, a migration is only triggered by the user / administrator.

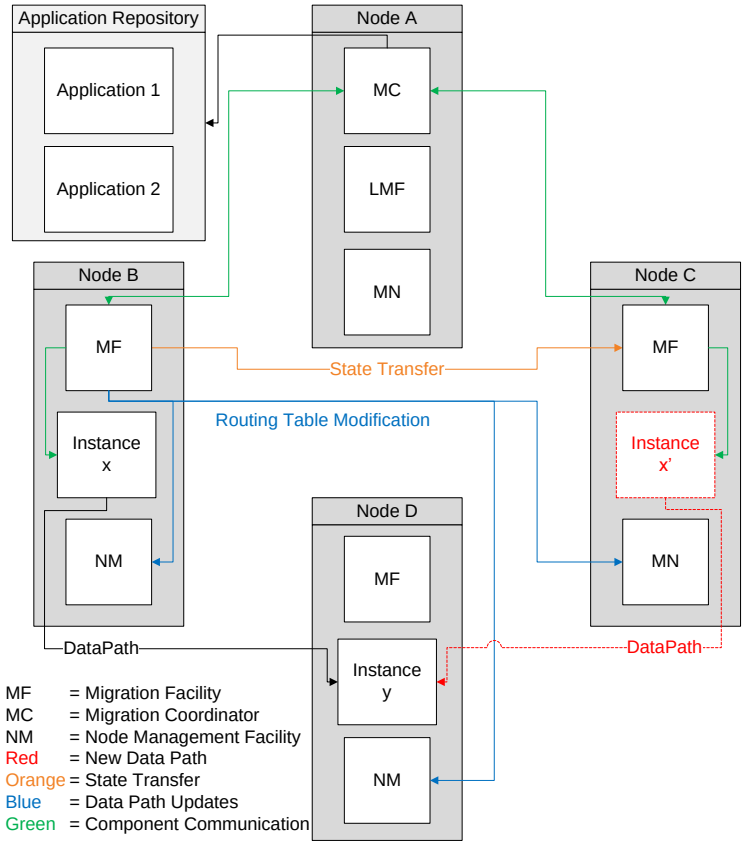


Figure 3.9.: Migration Scenario

At the beginning of the process, the source service serializes its inner state and provides it to the migration facility located on the source node. The migration facility then transfers the state data to the corresponding migration facility on the destination node. The migration facility finally provides the data to the destination service using the migration interface where the data is de-serialized.

After transferring the internal state and starting the new service, connections using the old service need to be replaced by connections using the new service. The last step finally is to decommission the old service.

Both components, the Migration Coordinator and the Migration Facilities are generic and interact with the services using predefined interfaces. In the middleware, the software parts responsible for the migration are split into two different kinds. The first one is the *Migration Coordinator*, which is the centralized part, and the *Migration Facilities*, which need to be installed at least on the nodes involved in the migration process. In the following, the components will be explained in more detail.

Migration Coordinator

The migration coordinator is responsible to coordinate the migration according to network and application needs. To fulfill this task, the migration coordinator has in-depth knowledge of the applications, services, requirements, and the data paths of the network. It gathers this information from the network management facility installed on one or, if a distributed implementation is used, on several nodes in the network.
When a migration is triggered by the user or by a monitoring agent, the coordinator first checks if the source and destination nodes are available in the network and if the requested service is installed on the source node. Using the meta-data dictionary located in the network management facility, the information is gathered if the destination node can handle the service and if the state migration is supported by the service or not. In case of problems, the user gets notified and the process is stopped here.
If all pre-conditions for the migration are met, the migration coordinator triggers the instantiation of the destination service instance at the destination node. The instantiation itself is done by a middleware component and can, in worst case, require to overwrite the complete software image on the destination. This can of course effect the remaining applications being executed on the node or transmitting data using this node as a hub.

Migration Facility

In contrast to the migration coordinator which is a centralized component, a migration facility is located at each networked node which provides support for service migration. The migration facility can perform two different tasks according to the responsibility in the migration process. The migration facility on the source node is responsible for checking if the requested source service is available and if it implements the required migration interface. If the interface is implemented, the migration facility requests the service state. This state information is then stored and, according to the information provided by the migration coordinator, sent to the migration facility at the destination node.
The migration facility at the destination node receives the system state. It checks if the destination service is instantiated properly and transfers the state using the migration interface. Finally it starts the new service and notifies the migration facility. For monitoring purposes, this message is also forwarded to the migration coordinator.
To finish the migration process, the data paths reconfiguration is triggered by the mi-

gration facility at the source node. If the reconfiguration was performed properly, the application can be resumed.

3.6. Suitability of SOA for Embedded Applications

In this chapter, a SOA inspired software architecture for distributed embedded systems was presented including a service bridge to interconnect the embedded to the corporate world. The suitability of this approach, considering the challenges a distributed embedded architecture is facing (see Section 1.3), is elaborated in the following paragraphs.

Heterogeneity

The heterogeneity of hardware and devices is supported due to the encapsulation of application logic into services as well as by the introduction of services for employed hardware devices (e.g., digital I/O service). Using the underlying middleware for service interaction and data transport, applications can be developed agnostic of the underlying execution environment. Using SOA, this challenge can be perfectly mastered.

Distributed and Reconfigurable Architecture

Distributed applications as well as reconfiguration are a sole feature of SOA as services are loosely coupled and provide functionality which is then assembled to an application by a process definition e.g. using BPEL [ACD+03] or by explicitly calling services. Reconfiguration is additionally supported by using a (message) broker[5] to decouple the services and so to introduce a data centric, stream based communication.

Resource Limitations

The resource limitations are a well employed argument against SOA for embedded systems. Using the main contribution of this chapter, the tailoring of SOA for the embedded domain by making service state full and long living, this argument becomes invalid. The presented approach provides the benefits of SOA (especially decoupling and re-use) and sufficient resource efficiency for embedded devices.

[5]The broker is a middleware component and will be introduced in Section 5.4.4

Bridging

Horizontal and vertical integration of field level devices to corporate systems already using web services become more and more important. This interconnection can be perfectly handled using the eSOA approach in combination with a generic web service bridge as presented in Section 3.3. This combination provided a simple and robust solution for a tight and cost efficient coupling of both worlds.

End-User Programming

Using SOA provides a perfect level of abstraction for application assembly as end-users can employ already developed services like a toolbox for new applications. This toolbox based approach has already been proven suitable by tools like MatLab / Simulink [SN93] for application logic. By employing this approach using the benefits of decoupling provided by SOA in combination with a suitable tooling and an integrated development process as introduced in this thesis, support for end-user programming can easily be provided.

Error Detection and Recovery

Using SOA is no contradiction for in system error detection and recovery as long as the execution environment provides support for these features. Although basic features can be provided by using dedicated services implementing error detection as well as by services triggering a reconfiguration, enhanced safety features were not investigated in the evaluation of this approach and are considered as future work.

Overall Result

Summing up the results in comparison to the requirements and challenges introduced at the beginning of this thesis, the claim that SOA is suitable for embedded applications is sustainable. The next building block crucial for deploying SOA to embedded systems is a suitable middleware providing an execution environment and the communication infrastructure for the services. In the next Chapter, a middleware for eSOA is proposed and the key components are elaborated.

3.7. Summary and Contributions

In this chapter, the basic challenges for distributed applications were summarized and the SOA was identified as a suitable solution to fulfill these requirements. In order to comply with the additional challenges for the embedded domain like resource constraints, an adapted notion of SOA for the embedded domain eSOA was introduced. The contributions are an adopted SOA for embedded devices eSOA including the notion of embedded services eServices, their formal specification and their interactions as well as the generic interconnection of eServices to the standardized web service world using a web service bridge performing a translation based on the semantic description of data and ports. These results provide a substantial contribution to the eSOA demonstrator introduced in Section 1.5.2.

CHAPTER 4

A Model Driven Approach for Embedded SOA

Contents

A key factor for reducing complexity is the separation of concerns regarding the different contributors to a networked embedded system. These contributors can be distinct by their different areas of expertise and so one possibility is to partition them according to their contributions into different groups. Uniting these contribution to one system is a major challenge and enabled by the domain specific model-driven development process including extensive code generation for networked embedded systems elaborated in this chapter.
The remainder of this chapter is structured as follows. At the beginning the motivation for model-driven development (MDD) in the area of networked embedded systems is presented as an introduction and followed by the elaboration of the requirements on the MDD approach in Section 4.2. Based on these, a development process is derived and detailed in Section 4.3.

4.1. Separation of Concerns for Reduced Complexity

Developing distributed software systems is a complex, time consuming and expensive task, especially if system size and so complexity grows and if software development is done in the well-known way by writing source code by hand with a low level of abstraction. To guarantee good software quality, additional effort to coding needs to be spent for testing and code reviews which are proven to be at least as time-consuming and expensive as coding itself [Enc03]. To cope with this problem, an efficient to use and flexible graphical development tool is a suitable way to increase productivity [BH94] by lifting the level of abstraction and guiding the developer trough the development process. In addition, it is shown in [AVT06], that using the model driven approach improves clarity and validity of specifications as well as the reusability of the knowledge encoded in models.

One possibility to implement this level of abstraction is the model driven development paradigm in conjunction with extensive code generation [SSBG03, TG06] where the employed models represent a DSL targeting a specific domain or field of application. How important model driven development and the corresponding DSLs are considered for the future economic success of the European Union is shown by the following statement taken form an EU commission report:

> *The introduction of domain specific programming techniques must be supported by the introduction of appropriate tools for using these techniques. With the shift from implementation towards design, the use of different programming languages and different software platforms, it becomes important to intensify the research for dedicated tool chains supporting a seamless model driven development process. This seems to be one of the most important points for Europe to succeed in the global competition. [HKM+05]*

Taking into account the definitions presented in Section 1.2, networked embedded systems provide a challenge for a whole group of expters on different fields of expertise. Usually domain experts, control engineers, computer scientists and electrical engineers contribute to the development. To unite the efforts spent by these expert groups a suitable development process including tool support is almost mandatory.

This development tools needs to provide a high level of abstraction as well as a detailed enough description to do code generation for resource constraint networked embedded system. Additionally, a clearly structured development process is required supporting the separation of concerns introduced by the employment of different expert groups.

This separation is achieved in this thesis by using separate (aspect) models and views customized for different groups of stakeholders. These aspects are then automatically combined to one model prior to system validation and code generation.

The solution employed in this thesis for this seperation of concerns during the development of networked embedded systems using the model-driven approach is to introduce separate models for the networks, the hardware platforms, the software modules as well as one model representing the deployment under consideration [SBK09]. By employing domain specific models integrated in a domain specific development tool a developer can focus on his domain and describe the system and its applications according to his needs without explicitly considering all the aspects already considered by another expert. The task of developing a system converges from writing source code to modeling which provides the advantage of a higher level of abstraction where ever possible. This model-driven approach makes software development much easier [BKK+11] and also helps to lower the burden for new developers.

In the development process presented in this thesis in Section 4.3, the user is guided through the development and so errors can be detected in an early stage. This error detection mechanism can be provided by model validation. A detailed view on the models and validation is given in Chapter 6. In the phase prior to code generation system validity can also be proved by integrating verification techniques [MGT+10] e.g., for timing constraints. Based on the models for networks, hardware platforms, software modules and the application, system tests can be generated automatically. In combination with user-supplied tests, the whole system can be tested systematically to prove e.g., code quality.

In addition to the eased system development, the models created for development are also suitable as a part of the documentation. The direct re-use of the models as documentation guarantees, that the documentation is always in sync with the system under development.

4.2. Requirements on the MDD Approach

Before the development process is presented in more detail in the following sections, a set of key requirements on model-driven development of networked embedded systems will be elaborated in this section. As there are many more requirements then the choice discussed in the following paragraph, the selection is limited to the most important ones in relation to the process and tools developed in this thesis.

Extendable Models

As models and their corresponding meta-models represent the bits and pieces applications consist of, it is obvious that they need to be flexible and extendable. The models can be seen as components similar to the components in a component container structure [ABPG05]. The probability that a developer can think of all possible use cases especially for future applications is quite low, so there needs to be a mechanism that the basis of the tool can be easily extended to comply with the arising requirements.

Extendable Code Generation

As the code generation needs to provide code for different platforms, where all target platforms cannot be known up front, it is obvious, that the code generator needs to be easily extendable. The generator also needs to provide the possibility to be extended, even by an end-user to add new or adapted implementations of different aspects. In addition, as soon as meta-models are allowed to change, the code generator also needs to be updated to use the changed input data.

Separation of Concerns

Usually there are many different groups of experts involved in developing networked embedded systems. Each of these groups has different requirements on the tool as well as different expectations of the functionality and the level of detail. As a consequence the software system has to be assembled based on software artifacts whereas direct dependencies between these artifacts needs to be at a minimal level to guarantee maintainability [AK03]. To provide proper support for all these groups and their different software artifacts, the tool needs to provide the capability to support all of them at their level of detail by providing mechanisms to support an integrated development process.

Models Need to Allow Precise Definition of Application(s)

As all aspects of a networked embedded system are defined using the provided models, they need to allow a precise definition of the behavior. All requirements need to be specified precisely and implemented according to the specification. The specification is not allowed to have margin for ambiguity. A language often used to model systems is UML [Obj07], but it does not provide enough information for extensive code generation [SBEJ04].

Validation of Model Input

Testing software targeting especially embedded devices is an important and expensive task in the development process. Research done by IBM shows, that even if 50% of the effort is spent on testing, only 44% of the designs realize 20% of both, features and performance expectations [Enc03]. Beside unit tests, integration tests, and system tests, there is an additional method to increase the quality of a system built based on models - model validation. The tool needs to support tests performed before code generation to detect conflicting models or configurations. Introducing these tests in an early development phase helps to detect errors prior to the first integration and system tests.

4.3. Distinct Developer Groups United by the Development Process

Developing distributed embedded systems is often split into different groups of developers each having expertise in a distinct field. When providing a tool for networked embedded system, it is a key requirement to take this development paradigm into account in addition to the requirements discussed as technical background in Section 4.2. A high level overview of the process developed in this thesis is depicted in Figure 4.1 showing the involved groups of developers as well as the process steps.

In the following paragraphs, this process is elaborated in detail by discussing the different phases of the process and by presenting the different involved groups of developers in detail. Finally, the presented process is summarized.

4.3.1. Development Process Overview

To implement the requirements mentioned in the previous section, a suitable development process needs to be defined. In Figure 4.2 the development process developed in this thesis is depicted. It structures the development into three main phases. In the first phase (green box) the basic meta-models and the implementation of the middleware as well as the development tool is provided by the *Platform Specialists*.

Based on the meta-models for the services, a model is instantiated and employed by the *Domain Experts* in the second phase (blue box) to describe the services they want to implement. By using a first code generation step, templates for these services are generated. The domain experts then implement the application logic within the generated service templates by hand or by using tools like MatLab / Simulink. After testing the

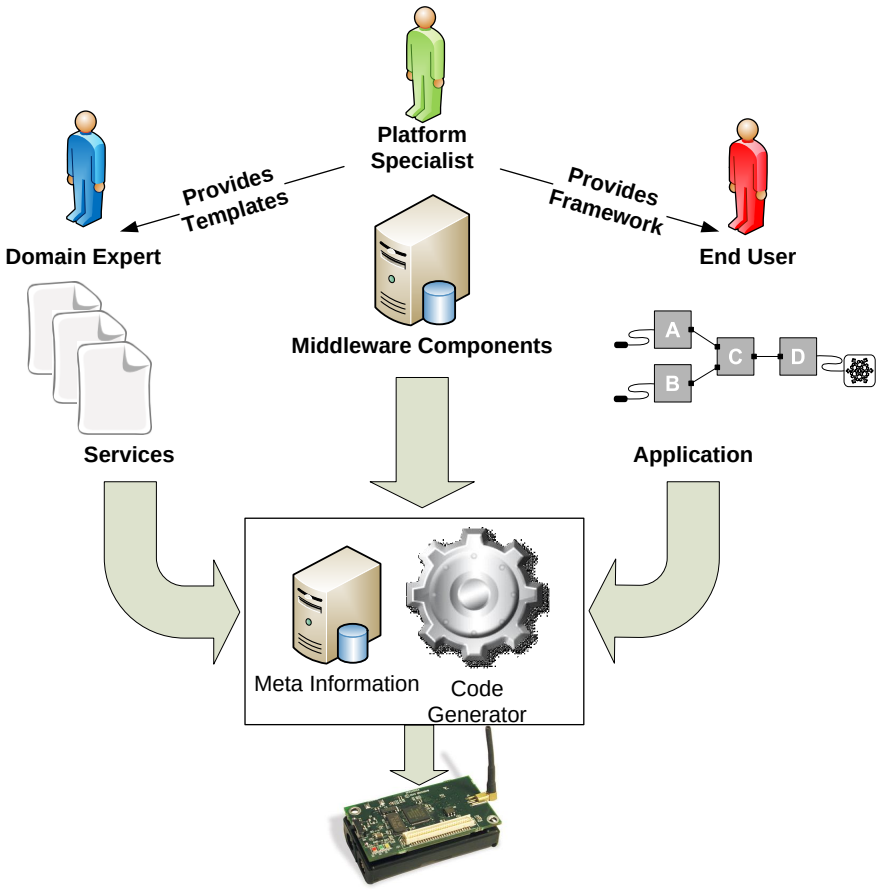


Figure 4.1.: High Level Overview of Development Process and User Groups

implementation, these service templates become part of the development tool. They can be employed by the end-users simply by selecting them form a toolbox similar to well-known MatLab toolboxes.

In the third phase (red box), *End-users / Installers* use the development tool to model the system they want to assemble consisting of the hardware, the network, and the services including their interconnections. Based on this description (model) and the code templates for the services provided by the domain experts as well as the templates provided by the platform specialists, the code for all nodes including a tailored middleware is generated. To deploy the newly created application to the network, the end-users only have to flash the binaries to the hardware using batch-scripts also provided as a result of the generation phase.

The separation of concerns during the development process is directly reflected by the different groups of users and developers involved in system assembly as well as by the

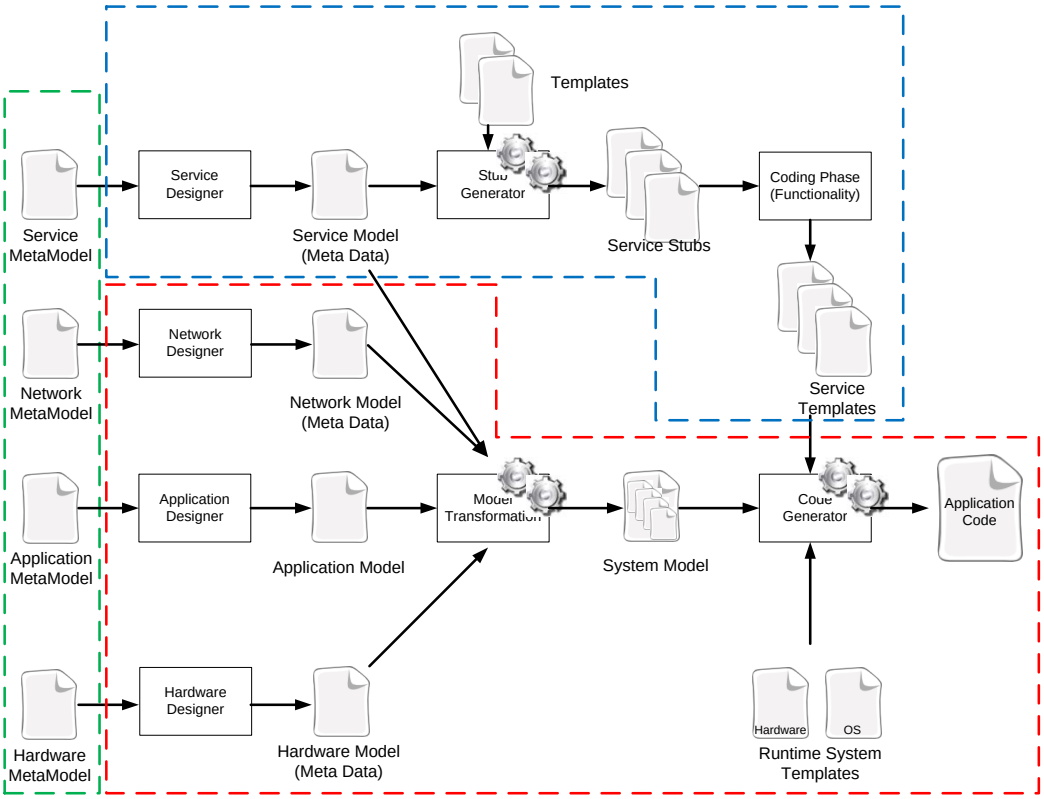


Figure 4.2.: Meta-Models, Models and Processes Separated into Phases

models used to describe a system. In the following, the different user groups having expert knowledge in the different development steps are introduced in more detail.

4.3.2. Platform Specialist

The *Platform Specialists* are the tool and infrastructure maintainer also providing the generator and the tailorable 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 [BSS+08]. The templates are implemented, maintained and extended by the 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. To lower the burden for function development, they also provide a template-mechanism to provide service templates according to the specification given by the *Domain Experts*. Due to the expandability of the employed code generator, new platforms can easily be supported by adding new templates or modifying existing ones.

Figure 4.3.: Device Driver Services provided by Platform Specialist

Beside the contributions to tooling and middleware, Platform Specialists also provide basic services for convenient hardware access. Basic services reflect the software instances to access sensors and actuators provided by the hardware as well as drivers for additional extensions like communication interfaces. The basic services abstract all implementation details and allow a black box usage of the hardware by the other developer groups.

A simple example is depicted in Figure 4.3 where the software components for a shutter und two push buttons are modeled using the SensorLab development tool. The service *ShutterHardwareService* here represents a basic service connected to the shutter actuator where the service *DoublePushButtonHardwareService* represents the sensor. The actuator service provides input ports to control the actuator where the sensor service has ports to provide the current sensor value.

4.3.3. Domain Experts

The Domain Experts have in-depth knowledge of a specific domain and implement the required functionality for this domain. This functionality is encapsulated in so called logic services that are later on used to assemble the applications. By using the infrastructure provided by the platform experts, especially the templating-mechansim providing the domain experts templates for their applications according to their interface specifications, they can focus on developing their functionality.

In the home automation domain for example, a building block is a heating / air conditioner control service or the shutter mentioned in the previous paragraph. Since basic services are provided 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 application logic. The Domain Expert has expert knowledge in

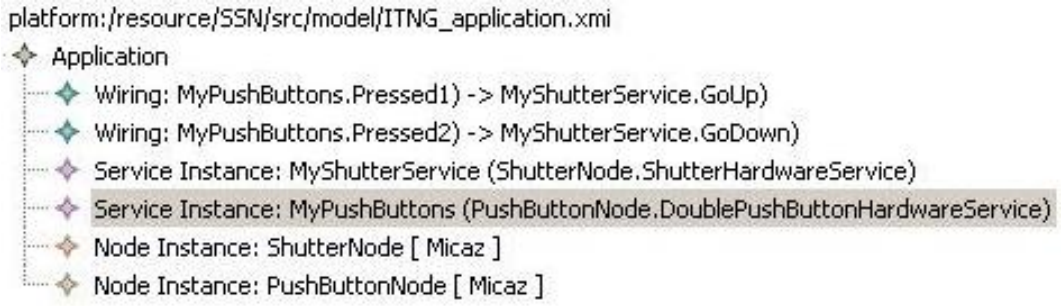


Figure 4.4.: Application assembled by End-User

his domain (e.g., climate control in buildings) and implements the pure application logic using e.g., ANSI C [HS91].

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 can be specified based on a domain specific ontology to have a common understanding if there is a standard available in the target domain e.g., [FSSF04]. In addition, it is possible to specify constraints like measurement resolution and minimal sampling rates.

4.3.4. End-User / Installer

Using the basic services provided by Platform Specialists representing the hardware in combination with the logic services provided by one or more Domain Experts, the End-User / Installer assembles the services in the same way he installs and wires the hardware components. After the installation of the hardware, the application is assembled and launched. This is done by and end-user with full tool support. A very simple example would be the control of a shutter. The installer selects a shutter control application from the toolbox capable of all the features he has in mind.

In the next step the end-user selects, 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, a central building control system can also be connected to the deployment to assure, that all shutters can be opened centrally in case of a tempest. The selection of services can be based on the specification of the interface and a textual description. Most important, implementation details are completely encapsulated by this approach.

The End User imports the pre-implemented basic services representing the hardware

(Figure 4.3), performs a configuration of the components, and builds his application by interconnecting (wiring) the involved services as depicted in Figure 4.4. After the generation step, the application is ready for deployment.

4.4. Summary and Contributions

The separation of concerns, the requirements on the MDD approach as well as different groups of experts involved in developing networked embedded systems including a suitable development process have been presented in this chapter. The elaborated development process provides the separation of concerns as required by the different user groups involved and is suitable to be represented within a development tool (SensorLab). The feasibility was shown on a small example which was designed as a part of the demonstrator introduced in Section 1.5.2. As the first part of the formal system specification was already introduced in Chapter 3, the remainder of this thesis elaborates the remaining building blocks contributing to the development, namely a tailor-able middleware housing the services and the tool itself combining the specification aspects and providing the code generation and system assembly.

CHAPTER 5

Middleware for Resource Constraint Heterogeneous Embedded Devices

Contents

In Chapter 3 was shown that SOA is perfectly applicable for embedded networks. It is common knowledge, that executing services as part of a SOA requires a middleware as underlying software platform. The requirements for a middleware housing embedded services on resource constraint devices are elaborated in this chapter and the key software components are derived. The design principles therefore are: modularity, tailorability and efficiency. As a reminder mostly for the middleware challenges, Table 5.1 summarizes the results elaborated in the background chapter.

The system architecture is presented in Section 5.1 and components of the middleware are discussed in detail in Section 5.4. As a formal foundation, the specification of applications is given in Section 5.5.

	Implementation					
Target	General Purpose	Embedded				Hybrid
Challenge	Corba, .NET, J2EE	RUNES	TeenyLIME	SOS, Contiki	TinyOS	ROS, DDS, OpenRTM
Managing limited power & resources	✗	✓	✓	✓	✓	✗
Scalability, mobility dyn. topology	✓	✌	✓	✓	✌	✌
Heterogeneity	✓	✓		✓	✌	✌
Real-world integration	✓	✗	✗	✗	✗	✗
Application knowledge	✗			✗	✗	✗
Data aggregation	✗			✌	✌	✗
Quality of service / NFPs	✗	✓	✗	✌	✌	✗
Security	✓	✗	✗	✌	✌	?

Table 5.1.: Overview Middleware Challenges and Implementations: ✓ Supported, ✗ Not Supported, ✌ Considered, Empty: No Information Available

5.1. Proposed Middleware Architecture

The general architecture is depicted in Figure 5.1. Similar to CORBA [Obj08], well defined interfaces for the application components are provided to access the middleware. In contrast to CORBA, the application container is tailored for a specific application and hardware using code generation as described in Chapter 6.

As the system is expected to be heterogeneous concerning the computational power and memory capabilities, the nodes can take over different roles within the network. Resource constrained nodes can be used to perform simple interactions with the environment like sensing or actuating. More powerful nodes can control the whole network, optimize the data flow, and trigger application changes.

The middleware provides a container that allows an easy combination of the services realizing the application functionality. Regarding these services, we distinguish two different kinds. An application or logic service realizes a control function of the application. The functionality can be implemented independently of the underlying hardware. Therefore, these components can be placed within the distributed system according to e.g., performance criteria and QoS. In contrast, a hardware interaction service realizes

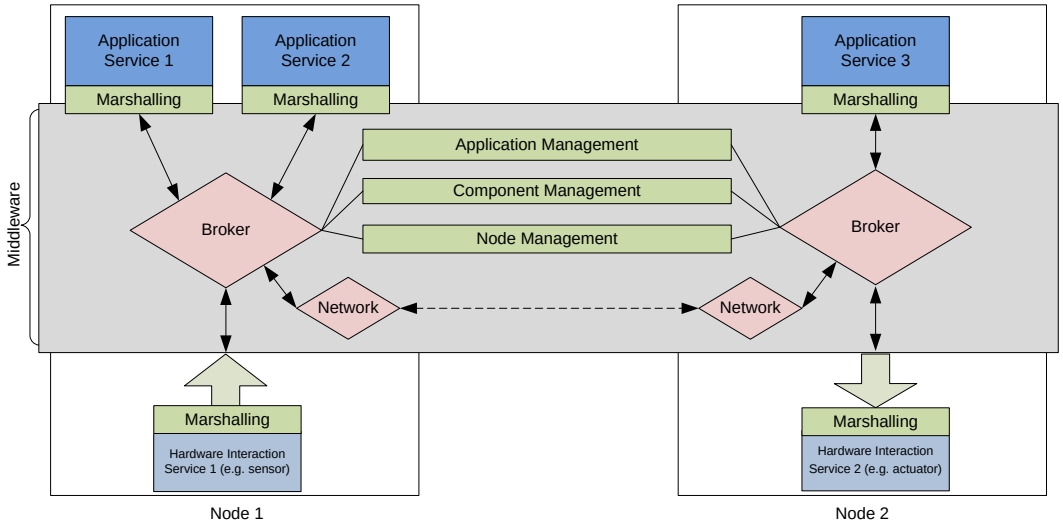


Figure 5.1.: eSOAMiddleware Architecture

the hardware access, e.g., sensing or actuating, and must be implemented hardware dependent.

The middleware realizes the interaction of these components and can be seen as intelligent glue code. In contrast to the operating systems such as TinyOS [Tin] or SOS [HKS+05b], which are very often considered to be a middleware themselves, the presented approach has to be seen at a higher level. In particular, it offers services related to the distributed execution of sensor applications such as routing, node failure management and QoS.

To master the challenges of a distributed embedded system, it is not only enough to adapt the SOA for these systems but it is also necessary to develop an architecture suitable for the system requirements and also as open and modular as possible to support the easy extendibility requested by the users. An overview of the architecture can also be found in [BSS+08].

5.2. Management Facilities and Application Services

Beside the application services housing the application logic, a systems usually consists of additional service required for the management of a single node or of the whole deployment. These management components (e.g., Node Management) are also depicted in Figure 5.1 and are called facilities in the following. Equal to application services, these management facilities need to share information and communicate with each other or external tools and additionally with the middleware itself by using distinct

interfaces. Having this similarity in mind, the employment of the same communication interface and mechanisms for facilities as for services is obvious. In order to keep a system stable even in an overload situation, a distinction between data flow (services) and control flow (facilities) is required in order to prioritise control messages. In addition to the communication mechanisms employed for the services which forward the messages based on their sender information, the facilities also have the possiblity to directly adress the destination of their messages by using the sink adress.

5.3. Communication and Execution Semantics

Beside the basic architecture overview, a system needs additionally to be described by its communication and exectution semantic. These semantics are discussed in the following paragraphs.

Execution Semantic

The middleware elaborated in this thesis basically employes a data-driven execution semantic for the services, where a service is executed every time data is delivered to a port of the service. After the data is processed, the service can also provide data on its output ports. The data delivery to all destinations is then performed by the middleware. For services interacting with the hardware (basic services), this execution semantics is not sufficient anymore. Additionaly, the execution of these basic services can be triggered by registered hardware interrupts and by registered timers. The ouput data of these basic services is submitted in the same way as for application services.

Communication Semantic

The basic idea for the communication using the elaborated middleware is to form an overlay network as an abstraction for different networking standards and hardware. This allows an uniform adressing of the nodes in the network without considering the underlying communication infrastructure. For the data provided and consumed by the services the behavior is as follows:

All data provided by services is forwarded to its destination by the middleware. In contrast to many well-established approaches, where the destination is listed in the message, the middleware elaborated in this thesis employs the source address (source addressing) for message routing. The destination of all messages is initially determined during the configuration phase of the system and is reflected by a forwarding table

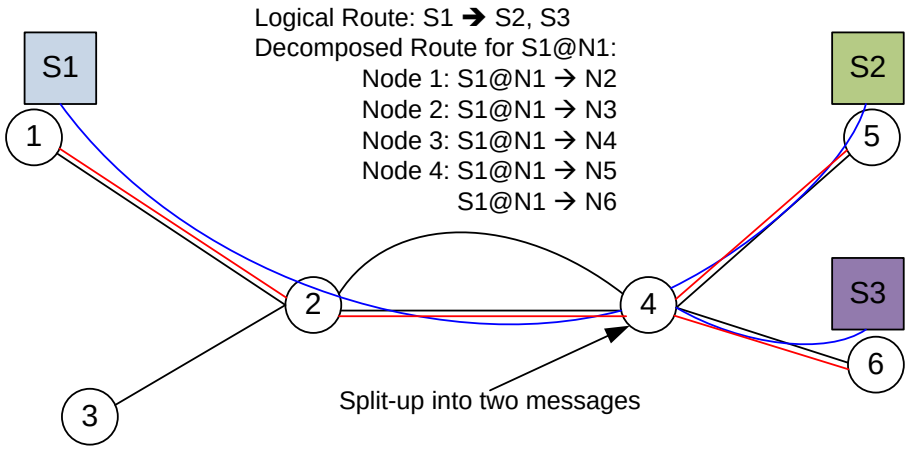


Figure 5.2.: Source-Based Routing: Network consisting of six nodes housing services $S1$, $S2$ and $S3$ where the same data is transmitted from $S1$ to $S2$ and $S3$. The data route is represented by the blue line and shows that the message is duplicated at the latest common node on the path.

stored on each involved node. Using this approach, multicast messages can easily be created agnostic of the underlying communication infrastructure. If necessary, the messages traveling on their pre-configured communication path can be duplicated on the latest common node and then be forwarded to multiple receivers. A simple example is depicted in Figure 5.2 and shows the physical links (red), logical route (blue) and the forwarding rules for a message sent from service $S1$ on node $N1$ to services $S2$ and $S3$. The rule $S1@N1 \rightarrow N2$ indicates, that the message originating from service $S1$ on node $N1$ needs to be forwarded to $N2$. In order to support dynamic reconfiguration, the forwarding rules can be updated during runtime.

Beside the data flow, also control flow messages need to be delivered to their destination. In contrast to the data messages, control messages (mostly generated by facilities) are tagged to allow distinction as well as prioritization during message processing and use the desination (sink-based) for adressing.

5.4. Selected Middleware Components

In this section a selection of the core middleware components are discussed in more detail. These are the Node Management, the Component Management, the Application Management, the Broker, the Marshaller, and the Network Service. Beside these selected components, the middleware consists of a number of additional components

known from various middleware implementations. The selection for these sections is performed according to their relevance to for eSOA and to the tailoring done to the middleware using the code generation.

5.4.1. Node Management

The first middleware component discussed in more detail is the node management. This distributed service is used to collect status information and capabilities of all nodes in the sensor network. The capabilities of the network comprise the available sensors and actuators, the provided communication media as well as processing power and storage capabilities. In addition, run-time data like battery status, free memory or hardware failures must be monitored. This information can be used to optimize the configuration of the application. Furthermore, the status information can be used for maintenance to identify nodes with heavy load or low energy resource at an early stage and to make arrangements to replace these nodes or their battery.
To gather all these information, it is essential for the whole system that each node announces its presence and keeps the state up to date. A node failure can be detected and reported by neighbor nodes due to the fact that communication to a lost node is not possible anymore. The core features provided by the node management are:

- Announcement of the node to the central management facility (if available) and to its neighbors
- Announcement of alive signal
- Announcement of health status (e.g., battery, load)
- Neighborhood detection

For resource constrained nodes in the network, a passive version of the node management is sufficient. It is passive in the sense that they provide information about the hosting node but do not collect information about other nodes. More powerful nodes execute active versions of the node management that gather the forwarded information and report changes to the application management.

5.4.2. Component Management

The component management provides information about all components available for the entire sensor network. We differentiate between application services and hardware interaction services or basic services. Hardware interaction services are offered on each node with attached dedicated hardware devices. In comparison to basic services, it is possible to locate application services on an arbitrary node in the network.

To acquire an optimal service placement in the sensor network, the application management service needs in-depth knowledge of all interfaces, the provided functionality and resources requirements (memory consumption, required processor time) of each component. An example how this information is used to assemble a system is given in Section 6.3. This information is stored, maintained, and provided by the component management. Different application components may realize a similar functionality. Based on the description of these components, the application manager can choose an adequate component based on the available devices and QoS constraints. Having in mind the formal system specification, this information is specified in the service description introduced in Definition 3.1.

5.4.3. Application Management

The application management component handles the configuration of the application. The configuration depends on the set of available nodes and their status, the set of software components, the topology, and QoS requirements. Application components can be placed intelligently within the distributed system to minimize network load or to balance the load on the different processors. If for example an average value out of a set of redundant sensor results is used at a remote controller, the application component computing this average value should be placed close to the sensors. A new configuration can be obtained by moving the affected software components and updating the routing. The latter is done by reconfiguring the broker detailed in the following paragraph. The information describing the applications is specified in the ApplicationDescription specified in Definition 5.3.

5.4.4. Broker

The component realizing the broker must be implemented as a local service on each node. The task of this component is to realize the routing at the level of application logic. The routing table of the broker is maintained by the application management to guarantee an optimal routing. All messages consumed and/or produced on a specific node need to pass the broker. It is the task of the broker to decide to which components on which nodes the message will be forwarded. This information is represented using the wirings specified in Definition 5.3. To determine if a message is delivered locally, the wiring information in combination with the service instances and their location is employed to generate the broker routing configuration. For local message delivery $outputService.sl = inputService.sl$ must be fulfilled. In contrast to messages delivered to local services, the messages for non-local services need to be sent over network to

the destination node. The message including routing information such as the receiver, security and reliability requirements is delegated to the network service for further processing.

5.4.5. Marshaller

Depending on the underlying communication infrastructure and the power resources available, different marshalling methods are available. To target communication resource constraint devices, EXI [SK08], a binary XML representation is employed. To enable efficient en-/decoding, the software module is tailored for an application. Based on the port specification of a service, the marshaller is generated using the $paramType$ and $paramRep$ information introduced in Definition 3.4. The tailoring of the marshaller e.g., for EXI saves the overhead and complexity to parse the data during run-time using a schema definition. As this component is generated for and shipped with each service as a glue code, is does not introduce additional burdens for extending or reconfiguring the system.

5.4.6. Network Service

The network service is used to communicate with other nodes in the sensor network regardless of the concrete communication medium available. In order to increase efficiency and reduce overhead, the capabilities of this service are tailored with respect to the infrastructure and the data during the code generation phase. Using the information available in the system specification, the network service can be tailored to application needs and infrastructure as well as kept generic to support extendibility. The calculation of the optimal data route considering QoS information is performed according to the approach detailed in Section 6.2.6 during the tailoring of the system. In the specification, this information is represented using the network and hardware description provided in Definition 5.1 and Definition 5.2.

This leads to a quite optimal performance without the need of any manual adaptations. From an implementation perspective, the network service provides the end-to-end routing by forwarding the message to the next neighbor on the route and so introducing an overlay network by applying the appropriate communication protocols for the hardware layer e.g., ZigBee [All06] or Reliable UDP [BK05]. To achieve secure communication, message de- and encryption can be activated in this service to transparently get a secure communication layer for message transport. For better efficiency and because of the resource constraints, only critical messages are encrypted.

5.5. Formal Specification

In this section, the formal system specification is extended by the means of network and hardware. These two, together with the specification of services already introduced in Section 3.2 provide the foundation for the specification of applications discussed at the end of this section.

5.5.1. Formal Network Specification

The $NetworkDescription$ consists of a set of $channel$. The channels (N, m, e, r, l) are defined by the Node instances set N, the communication medium m, the encryption algorithm e, the reliability probability r and the average latency l. The channels c represent the overlay network deployed on the physical node links and abstracts from different hardware. An example with two channels is depicted in Figure 5.3 where node $N2$ and $N4$ are equipped with two network interfaces each to interconnect both channels.

$$\begin{aligned} NetworkDescription = & \bigcup_{channel\ c_i} c_i \\ channel : & (N, m, e, r, l, QoS) \\ & N \subseteq NodeInstances \\ & m \in \{ZigBee, RS232, Ethernet\_UDP\} \\ & e \in \{\emptyset, CaesarCipher\} \\ & r : 0 < r < 1 \\ & l \in N \end{aligned} \tag{5.1}$$

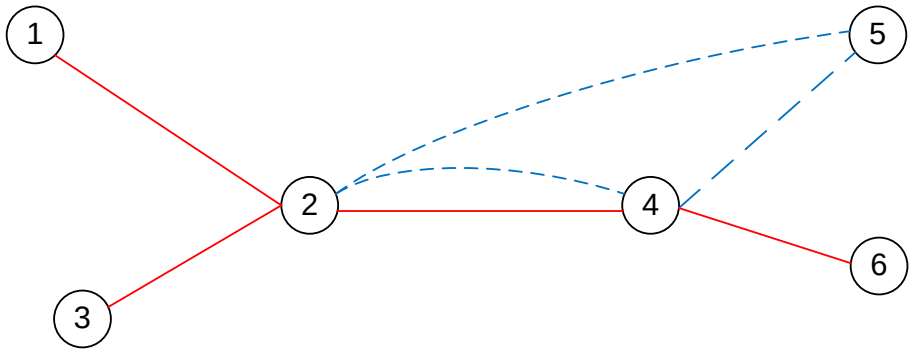


Figure 5.3.: Network Consisting of Six Nodes and Two Channels (red and blue).

5.5.2. Formal Hardware Specification

$$
\begin{aligned}
HardwareDescription = & \bigcup_{nodeClass\ nc_i} nc_i \\
nodeClass : & (t, a, r, f, e, NodeDevice, m) \\
& t \in \{TMote, MicaZ, PC\} \\
& a \in \{AVR, MST430, x86\} \\
& r \in N \\
& f \in N \\
& e \in N_0 \\
& NodeDevice \in \{MDA510, RFID\} \\
& m \subseteq \{ZigBee, RS232, Ethernet\_UDP\}
\end{aligned}
\tag{5.2}
$$

As the $NetworkDescription$ wires node instances, the hardware has to be specified up front as a template for the node instances using the $HardwareDescription$ of a system S consisting of a set of $nodeClass$. A node class $(t, a, r, f, e, NodeDevice, m)$ is defined by a node type description t, an architecture description a, a RAM size r, a FLASH size f, an EEPROM size e, an device identifier $NodeDevice$ for additional devices (e.g., a sensor cluster or a RFID reader), and a communication medium m. The supported communication medium then provides the basis to interconnect a group of nodes using a channel as specified in Definition 5.1.

5.5.3. Formal Application Specification

Based on the specification of services introduced in Section 2.6 as a foundation, an application is assembled. This assembly is specified using the $ApplicationDescription$. In this thesis, an application consists of the involved nodes, the services, and their interconnections as specified in the $ApplicationDescription$ which consists of the following tuple ($NodeInstances$, $ServiceInstances$, $Wiring$).

$$
\begin{aligned}
ApplicationDescription : & (NodeInstances, ServiceInstances, Wirings) \\
NodeInstances = & \bigcup_{nodeInstances\ n_i} n_i \\
ServiceInstances = & \bigcup_{serviceInstance\ s_i} s_i \\
Wirings = & \bigcup_{wiring\ w_i} w_i
\end{aligned}
\tag{5.3}
$$

$$\begin{aligned} nodeInstance &: (n, t, D, os, pl, nodeID) \\ serviceInstance &: (si, s, sl, serviceID) \\ wiring &: (inputPort, inputService, outputPort, outputService, encr, QoS) \end{aligned} \tag{5.4}$$

A NodeInstance $(n, t, D, os, pl, nodeID)$ is specified by the node name n, the node class t, a set of additional node devices d, by the platform os, the programming language pl, and a node unique id $nodeID$.

$$\begin{aligned} nodeInstance &: (n, t, D, os, pl, nodeID) \\ t &\in HardwareDescription \\ D &\subseteq NodeDevice \\ os &\in \{TinyOS, Linux, Windows\} \\ pl &\in \{NesC, C, JAVA\} \\ nodeID &\in N_0; n_a, n_b \in NodeInstances; \\ & n_a.nodeID \neq n_b.nodeID \Rightarrow n_a \neq n_b \end{aligned} \tag{5.5}$$

A ServiceInstance $(si, s, sl, serviceID)$ is specified by the service instance name si, the instantiated service s, the deployment location sl, and a node unique service id $serviceID$.

$$\begin{aligned} serviceInstances &: (si, s, sl, serviceID) \\ s &\in ServiceDescription \\ sl &\in NodeInstances \\ serviceID &\in N_0 \end{aligned} \tag{5.6}$$

The $wiring(inputPort, inputService, outputPort, outputService, encr, QoS)$ describes the interconnection of service instances agnostic of their locations by refering to the $inputPort$ of the $inputService$ and the $outputPort$ of the $outputService$. In addition, the communication is characterized using enc to specify if encryption is required and which QoS parameters specified in the QoS set should be applied.

$$
\begin{aligned}
wiring : (&inputPort, inputService, outputPort,\\
&outputService, encr, QoS)\\
inputService &\in ServiceInstances\\
inputPort &\in inputService.InPorts\\
outputService &\in ServiceInstances\\
outputPort &\in outputService.OutPorts\\
encr &\in \{\emptyset, CaesarCipher\}\\
QoS &\subseteq \{reliable, unreliable, CRC\}
\end{aligned}
\tag{5.7}
$$

5.6. Summary and Contributions

In this capter, the proposed middleware providing an execution environment for the embedded services was presented. The major aspects discussed were the execution semantics, the communication paradigm as well as the distinction between management facilities and services housing the application logic. Based on this architectural overview, a selection of core components of the middleware have been presented in more detail, namely the Node-, Application- and Component-Management as well as the Broker and the network serivce providing the interconnection of local and remote nodes. Finally a formal specification of the system aspects hardware and network communication were elaborated as a detailed definition of the introduced middleware components.

The contributions of this capter chapter are twofold: First it was shown, how a flexible middleware for resource constraint networked embedded system can look like and which major components are essential for the middleware. The second contribution is the formal specification of the hardware and networking aspects of a networked embedded system as input for the final system assembly and validation presented in the next chapter.

CHAPTER 6

Model Driven Software Development and Code Generation

Contents

Based on the idea of information decoupling and separation of concerns introduced by the presented development process, a formal specification of a part of the system was given in Chapter 3 and Chapter 5.

Using the specification as a foundation, EMF models [SBMP08] are derived implementing this specification. An example of one aspect model derived from the specification is given in Section 6.1. The remaining models are detailed in Appendix A.

One major part of the system assembly, the model transformations implementing the system manufacturing are described in Section 6.2. These transformation steps can in principle be extended by additional (also external) tools, e.g., by automated service placement as introduced in Section 6.3. The code generation (or model-to-text transformation) providing the source code as well as the build files for the whole deployment is presented in Section 6.4. Finally, the contributions of this chapter are summarized and evaluated in Section 6.5.

6.1. Derived Meta-Models and Models

The whole model-driven design process is based on suitable meta-models and models [MSUW02]. The meta-models are a representation of a DSL and help to focus on dedicated aspects during development. By introducing DSLs to allow a precise formulation of problems and their solutions, some authors claim that productivity can be increased by a factor of four [Wig01].

In this section the employed meta-model for the development process is introduced and the key aspects are discussed. To increase flexibility and maintainability, the monolithic meta-model is split into four different meta-models. These are the hardware, service, network and application meta-model. Based on these meta-models, a fifth meta-model, the production meta-model is created, which is basically a concatenation of the four meta-models introduced above extended by additional information required for code generation. This information is calculated during the model-to-model (M2M) transformation phase described in Section 6.2.

6.1.1. Implementation

Based on the formal specification of the System S $(HardwareDescription$, $ServiceDescription$, $NetworkDescription$, $ApplicationDescription)$, an implementation is provided as an example how this formal description is transformed into a development tool. The basic idea is, to transform the specification into a meta-model as a foundation for modeling the system. Based on this input data, further transformations are performed to tailor the user input for the targeted system and finally the code is generated for the targeted platforms. To discuss the single transformation steps, the formal specification as well as a tool dependent implementation is presented.

6.1.2. Example: Hardware Meta-Model

The hardware meta-model depicted in Figure 6.1 is one of the basic models of the developed system. It is used to describe the involved hardware classes. Based on this meta-model, a hardware model is instantiated to describe the involved device types for a specific deployment. In case, additional information is required to sufficiently describe a deployment, the meta-model and hence the model can be easily extended.
The structure of the hardware meta-model is elaborated in the following paragraphs.

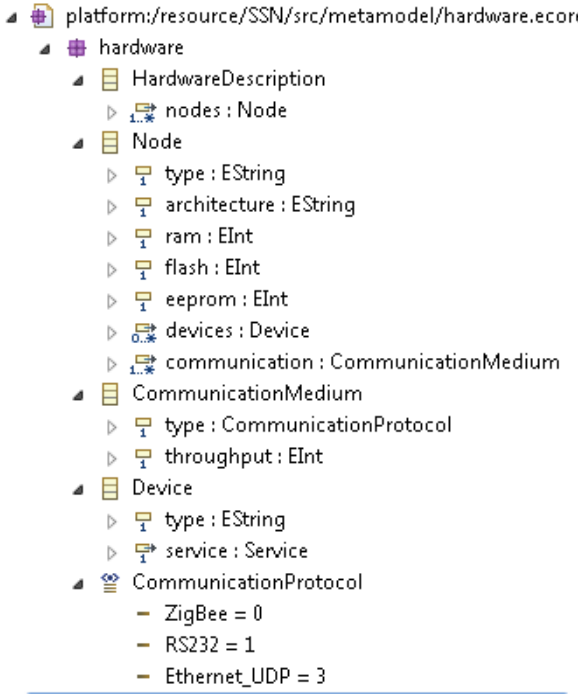


Figure 6.1.: SensorLab Hardware Meta Model

Hardware Description

The hardware is described using a *Hardware Description* class, which itself consists of *Nodes*. This class is used as a container for the remaining parts of the meta-model.

Nodes

The nodes described in this meta-model are no real instances, they represent the different types of nodes, which are characterized using several attributes like *RAM* and *ROM* for the available RAM and ROM. To make the nodes suitable for networked embedded systems, all of them require at least one communication interface which is represented by the *Communication Medium* class.

Communication Medium

The communication medium describes communication between at least two nodes. In the initial version, communication using RS232, ZigBee, and Ethernet is supported. This selection is represented by the *type* parameter. Based on the communication type (or technique), reasonable parameters for the underlying communication medium can

be suggested. The most important characteristic is the *throughput*. Throughput is specified using an integer value and is later on used to check the feasibility of a configured application in the sense of communication requirements.

Device

Many devices used in the embedded domain provide interfaces to extend their functionality by additional hardware, which is probably not known form the beginning of a design phase. Such interfaces are e.g., TWI [Sem00b], SPI [Sem00a], or a simple UART [Osb80]. To connect devices attached using these interfaces, device drivers are necessary to bring the functionality to the runtime environment. To model this case, the *Device* class is used. It consists of a verbal description of the attached device (the attribute *type*) and the attribute *service*, which represents a service implementing the software functionality (device driver) of the device.

6.2. Model-to-Model Transformation

In model-driven development, the basic tasks performed on models is the creation and adaption as well as the transformation of models as stated by Jouault et al.:

> *In the context of Model Driven Engineering, models are the main development artifacts and model transformations are among the most important operations applied to models. [JABK08]*

Transformation here means to read data from a model, process the information and finally write the resulting information to a new or adapted model. This can be done in order to convert model data into a different representation (metamodel) [ABGR10] or to calculate additional information based on the data stored in the source model(s) [KBSK10] as it is done in this thesis.

In this Section, first the employment of M2M transformation in this thesis is elaborated followed by the different steps of the M2M transformation process developed within this thesis. In paragraph 6.2.4 an example how the development process can be extended e.g., by the service placement framework introduced in Section 6.3 is discussed. At the end of this section, system validation checks are introduced as well as the technique employed to implement the specified checks.

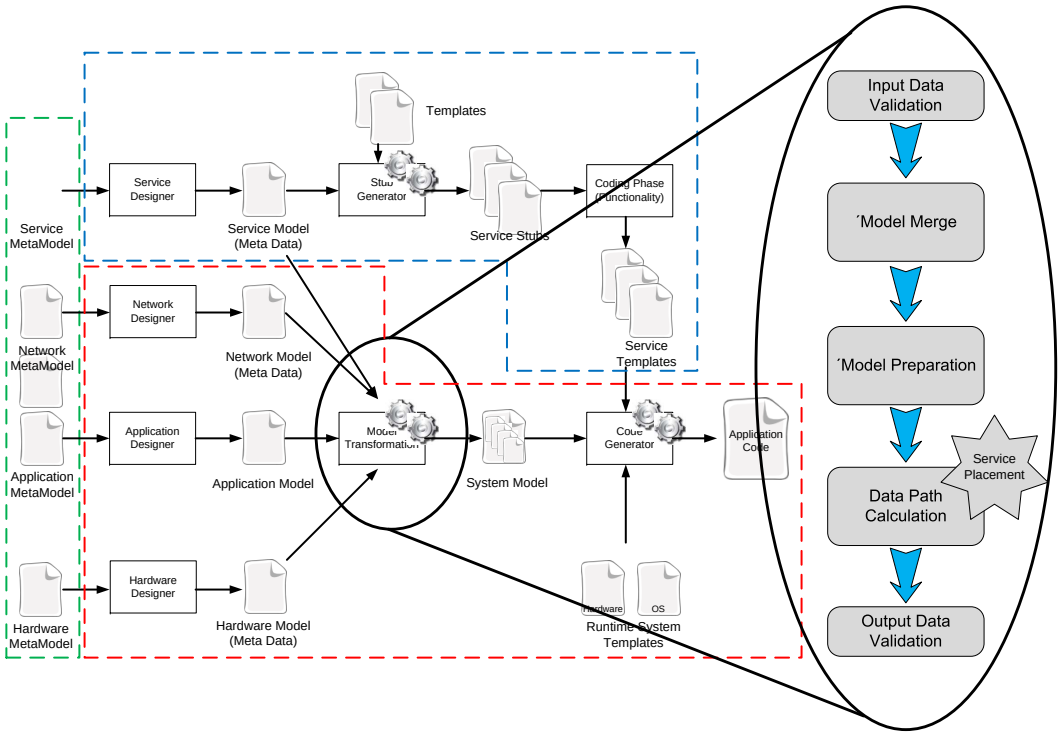


Figure 6.2.: Model-to-Model Transformation: Process Steps

6.2.1. Employment of M2M in this Thesis

Basically, the model transformation $S \rightarrow S'$ done in this thesis is twofold:
First, the static information artifacts are transferred from the models holding the user input to the production model to make the handling during the following transformation steps more convenient.
Second, the concatenated model is transformed into the production model by unification of ports, services, and nodes by assigning them scope wide unique IDs and by calculating the network topology based on the specified information. A detailed description of the steps in the transformation is given in the followings paragraphs and is depicted in Figure 6.2. As model transformations can be quite complex, they become a crucial part of the software development process and have a need for extensive testing. One solution on this is to generate the tests based on the meta-models as described in [BFS+06].

6.2.2. Formal Production System Specification

Based on the already discussed specification of the system S, a production specification S' $(HardwareDescription', ServiceDescription', NetworkDescription',$

```
3/*************************************************************************
4 Application Model Checks
5*************************************************************************
6
7/** Ensure that Node name is unique **/
8context NodeInstance ERROR "Node name '" + name +
9    "' is used more than once!" :
10    ((Application)eContainer).nodeInstance.notExists(
11        node | node != this && node.name == name);
12
13/** Ensure that Output Service of wiring is set **/
14context Wiring ERROR "Wiring Error: Output Service not set!" :
15    ((Application)eContainer).wiring.notExists(
16        wiring | wiring == this && wiring.outputService.name == null);
17
18/** Ensure that Input Service of wiring is set **/
19context Wiring ERROR "Wiring Error: Input Service not set!" :
20    ((Application)eContainer).wiring.notExists(
21        wiring | wiring == this && wiring.inputService.name == null);
22
23/** Ensure that Output Port of wiring is set **/
24context Wiring ERROR "Wiring Error: Output Port of Service '" +
25    outputService.name + "'is null!" :
26    ((Application)eContainer).wiring.notExists(
27        wiring | wiring == this && wiring.output.name == null);
28
```

Figure 6.3.: Check Example for Application Model Performing Basic Sanity Checks

$ApplicationDescription'$, $SystemConfiguration$) is derived which is enriched by information required to assemble the system. The transformations $S \rightarrow S'$ including the transformation steps are discussed in the following paragraphs. Each transformation step shown in Figure 6.2 is discussed based on its contribution to the development process.

6.2.3. Step 1: Validation of Input Data

Before the modeled aspects are considered for further processing, the validity of the input data is checked using rules specified using the Check [Effa] language which is part of the employed modeling framework. The validation starts by employing simple checks for unique naming of element identifiers and proceeds to checks, if wired services have compatible interfaces. An excerpt of the check rules is depicted in Figure 6.3. Additional checks can be easily added during development time, if previously undetected modeling or transformation errors are discovered. The approach here is

```
3 extension extend::references;
4
5 create production::Application this
6     extendApp_Instances(application::Application app):
7     serviceInstance.addAll(app.serviceInstance.transformSInstance())->
8     nodeInstance.addAll(app.nodeInstance.transformNInstance());
9
10
11 Void extendApp_Wiring(
12     production::Application ret, application::Application app):
13         ret.wiring.addAll(app.wiring.transformWiring());
14
15 create production::Wiring this
16     transformWiring(application::Wiring w):
17     setInputService(w.inputService.getReferencedObject())->
18     setInput(w.input.getReferencedObject()) ->
19     setOutput(w.output.getReferencedObject())->
20     setOutputService(w.outputService.getReferencedObject());
```

Figure 6.4.: Model to Model Transformation: Application Model Content is Copied to Production Model and References are Resolved

similar to unit testing, where an additional test is added to verify that a discovered bug is fixed [GC11].

6.2.4. Step 2: Merge into Production Model

$$
\begin{aligned}
S \rightarrow S' : & \\
HardwareDescription' &= M_{hd}(HardwareDescription) \\
ServiceDescription' &= M_{sd}(ServiceDescription) \\
NetworkDescription' &= M_{nd}(NetworkDescription) \\
ApplicationDescription' &= M_{ad}(ApplicationDescription)
\end{aligned}
\tag{6.1}
$$

After the validity of the input data is checked in the initial step, the model elements are merged into the production model by a transformation M. The transformation M therefore consists of four aspect model specific transformations $M_{\{hd,sd,nd,ad\}}$. The merge transformations are implemented as copy operations of the data from the source models (the aspect models) to the production model. In this step, references between the different aspect models are resolved, to make the production model self-contained. Technically, this step is performed using the *xtend-Language* [VG07] extended by Java functions mostly for resolving dependencies. An excerpt of the transformation is depicted in Figure 6.4 showing the part of the transformation performing the data copy

and the resolving of references by calling external Java functions (*e.g., getReferencedObject()*).

After the model merge, basic preparations followed by the service placement and data path calculations are be performed. If services have no node assigned, the service placement framework described in Section 6.3 is used to find a suitable placement. As soon as a placement is calculated, the data path calculation can be performed in the same way as if the services had been placed by hand.

6.2.5. Step 3: Preparation of the Production Model

After all model data is stored inside the production model, it is self-contained and basic preparations for the following steps can be performed. One of these steps is providing the model elements with system wide unique IDs for the nodes, the services, the ports, and the communication channels, as long as these have not previously been assigned by the user. In order to allow backwards compatibility and easy extendibility, the IDs can also be propagated back to the aspect models to make sure they are kept the same for further deployment iterations.

Formal description

The calculation and assignment of unique IDs is performed in the $T_{unification}$ step which consists of transformations for $nodeID$ (T_{un_nodeID}), for $serviceID$ ($T_{un\_serviceID}$), and by the transformation for $portID$ (T_{un_portID}).

$$T_{unification} = (T_{un\_portID} \times T_{un\_serviceID} \times T_{un\_nodeID})(S') \tag{6.2}$$

First, the unification T_{un_nodeID} is performed for the $nodeID$. If the node ID was user-defined ($\neq 0$), the value is kept, otherwise a unique $nodeID$ is generated based on the following two rules.

$$\forall n \in nodeInstance | n.nodeID = 0 : n.nodeID = getNextNodeID() \tag{6.3}$$

$$\forall n_1 \in nodeInstance \quad \forall n_2 \in nodeInstance : \\ n_1 \neq n_2 \Rightarrow n_1.nodeID \neq n_2.nodeID \tag{6.4}$$

The second step is to assign each $serviceInstance$ an unique $serviceID$ in the scope

of a node by $T_{un_serviceID}$. Equally to the $nodeID$, a system generated $serviceID$ is assigned, if no $serviceID$ is user-defined. The assignment is performed according to the following two rules:

$$\forall s \in serviceInstance | s.serviceID = 0 : s.serviceID = getNextServiceID(s.sl) \quad (6.5)$$

$$\forall s_1 \in serviceInstance \quad \forall s_2 \in serviceInstance : \\ s_1 \neq s_2 \wedge s_1.sl = s_2.sl \Rightarrow s_1.serviceID \neq n_2.serviceID \quad (6.6)$$

Similar to the $nodeID$ and $serviceID$ each input and output port p is assigned an unique ID by T_{un_portID}. This ID is unique within the scope of a service and the port direction.

6.2.6. Step 4: Calculation of Data Paths Throughout the Network

Based on the communication channels and the participating nodes and their network interfaces, a network graph is generated as a foundation for the data paths which are calculated for each wiring interconnecting two services. Depending on the network topology and medium, gateways are identified and finally routing table entries for all participating nodes are generated. This is done for each wiring consecutively. To represent the different communication media and their resource utilization, the links between nodes can be charged with weights and so the cheapest link is selected. Additionally, the reliability attribute describing each communication channel is also used to provide an estimation of the end-to-end communication reliability.

The additional information calculated during this phase is then stored in the production model and its extensions for further use within the transformation as well as for code generation. This can also be used as an input for external analysis and verification tools by simply adding an export functions to write the data to the specific format understood by the external tool. As an example, a description of the services, their interfaces and interconnections is exported using the JSON[1] format.

6.2.6.1. Formal Description

In the following paragraphs, basic terms and definitions are introduced which are necessary to describe the steps performed to map the applications to the given network.

[1] JavaScript Object Notation is a lightweight data-interchange format and here employed to provide a description of the service interfaces to external applications.

Representation of a Network Graph

Definition: A network graph represents the communication network of the considered system and is basically described by the tuple (N, E) where N are the network nodes and E are communication links between network nodes. An edge $e \in E$ $(n_i, n_j, w, channel)$ is defined by the connected nodes n_i, n_j, by the weight w of a link and by the communication channel the edge is part of.

Representation of Node Reachability

The notion $reachable(n_i, n_j)$ provides, if node n_i can reach node n_j and is defined in the following paragraph:
$sameChannel(n_i, n_j) = \{\exists c | c \in NetworkDescription : n_i \in c.N \wedge n_j \in c.N\}$

$$reachable(n_i, n_j) = \begin{cases} \text{if } sameChannel(n_i, n_j) \neq \emptyset, & 1 \\ \text{if } reachable(n_i, n_k) \wedge reachable(n_k, n_j), & 1 \\ \text{if } otherwise, & 0 \end{cases} \tag{6.7}$$

Based on Definition 6.7 for each node $n \in NodeInstances$ a $ReachabilitySet_n$ $(n, Nodes)$ is calculated where n is the node and $Nodes$ is the set of nodes which are reachable from node n. The same technique is used to calculate a $path(n_i, n_j)$ which in this context is a sequence of channels interconnecting a sequence of nodes where the sequence starts at node n_j and ends at node n_j.

$$\begin{aligned} Network &= \bigcup_{n \in NodeInstances} ReachabilitySet_n \\ ReachabilitySet_n &: (n, Nodes) \\ n &\in NodeInstances \\ Nodes &= \bigcup_{n_j \in NodeInstances} reachable(n, n_j) = 1 \end{aligned} \tag{6.8}$$

Representation of a Routing Graph

Definition: A routing graph represents the data channels between services on one or on different nodes and is also described by (N, E) where the edges E (n_i, n_j, w) represent the interconnection of two nodes n_i, $n_j \in N$ and the wiring w under inspection.

Representation of the Forwarding Rule Set

Based on the information stored in the routing graph and in the network graph, a rule set $R\ (n, w, n_h, channel)$ for package forwarding is created for each node and each wiring passing a given node.

$$RuleSet = NetworkPathCalculation(NetworkDescription, ApplicationDescription) \tag{6.9}$$

The forwarding rule here consists of the node n under inspection, the wiring w, the next node n_h to forward the data to and of the communication channel $channel$ to use.

6.2.6.2. Network Path Calculation

To generate a network configuration based on the input information specified by the user, different approaches are possible depending on the quality requirements on the results. For a best effort system, the reachability of two interconnected services needs to be given. For these systems, a solution outlined in paragraph 6.2.6.3 is sufficient, but as soon as QoS requirements need to be fulfilled, more sophisticated calculations need to be performed as outlined in paragraph 6.2.6.4. The basic idea is always aligned to the following approach where based on the applications (services and their wiring) and the network information a rule set R for data forwarding is generated. This rule set also influences the code generation performed for the core components of the middleware presented in Section 5.4, especially the network service.

6.2.6.3. Simple Network Path Calculation

For simple network configurations a fast result for a networking configuration is calculated with low effort by only taking the reachability information into account. This routing rule set generation is performed within the the same process step where the $Network$ set is calculated. As first step a network graph (N, E) is derived from the channel information specified in the $NetworkDescription$:

$$
\begin{aligned}
N &= \bigcup_{c \in NetworkDescription} \{\forall(n) | n \in c.N \wedge n \notin N : n\} \\
E &= \bigcup_{c \in NetworkDescription} EdgesInChannel(c) \\
EdgesInChannel(c) &: \{\forall(n_1, n_2) | n_1 \in c.N \wedge n_2 \in c.N \wedge n_1 \neq n_2 : (n_1, n_2, weight(c), c)\} \\
weight(c) &= \begin{cases} \text{if } c.m = Ethernet\_UDP, & 1 \\ \text{if } c.m = RS232, & 2 \\ \text{if } c.m = ZigBee, & 3 \end{cases}
\end{aligned}
\tag{6.10}
$$

By adjusting the weight function, different communication technologies can be preferred. The simplest case here is to set the weight equally to 1 for all communication media. Using the Floyd-Warshall [CLRS09] algorithm, the shortest path between all nodes is calculated as follows:

$$
\begin{aligned}
D &= d_{ij}^{(k)} \\
d_{ij}^{(k)} &= \begin{cases} \text{if } k = 0, & weight_{ij} \\ \text{if } k > 0, & min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) \end{cases} \\
weight_{ij} &= min(e.weight), e \in E \wedge e.n_i = n_i \wedge e.n_j = n_j \\
& k : is\ an\ intermediate\ vertex\ of\ path\ p : n_i \rightarrow^{p_1} n_k \rightarrow^{p_2} n_j
\end{aligned}
\tag{6.11}
$$

The matrix D represents the shortest path weights and is used to construct the predecessor matrix π accordig to [CLRS09].

Based on this matrix π, the next network hop can be easily derived. The distances and next hop information are stored during this calculation in the $Distances$ set $(n, n_j, n_h, d, channel)$, where n is the inspected node, n_j is the destination node, n_h is the next hop, d the number of hops to the destination and $channel$ the communication channel to use. This information is queried as follows:

$$\begin{aligned} nextHop(n, n_j) &= \{d | d \in Distances \wedge d.n = n \wedge d.n_j = n_j \wedge d.d \neq \infty : d.n_h\} \\ hopCount(n, n_j) &= \{d | d \in Distances \wedge d.n = n \wedge d.n_j = n_j : n.d\} \end{aligned} \tag{6.12}$$

In this context, the function $nextHop(n, n_j)$ returns the next hop (node) on the path from node n to node n_j. $hopCount(n, n_j)$ provides the number of remaining hops to node n_j.

$$\begin{aligned} Distances = & \bigcup_{n \in NodeInstances} (n, n_j, n_h, d, channel) \\ n \in & \; NodeInstances \\ n_j \in & \; NodeInstances \\ n_h = & \begin{cases} \text{if } reachable(n, n_j) \neq 0, & nextHop(n, n_j) \\ \text{if } otherwise, & 0 \end{cases} \\ d = & \begin{cases} \text{if } reachable(n, n_j) \neq 0, & hopCount(n, n_j) \\ \text{if } otherwise, & 0 \end{cases} \\ channel = & \begin{cases} \text{if } reachable(n, n_j) \neq 0, & nextHop_c(n, n_j) \\ \text{if } otherwise, & 0 \end{cases} \end{aligned} \tag{6.13}$$

A simple example of an application consisting of a network, several services and their interconnection is given in Figure 6.5. The network consists of five nodes with the following physical edges: $E = \{(1,2),(3,2),(2,4),(2,4)',(4,5)\}$. To these nodes, six services are deployed with the following wirings represented by the blue lines in the Figure (ports are not depicted for a compact representation): $W = \{(S1,S2),(S3,S2),(S2,S4),(S3,S4),(S5,S4),(S4,S6)\}$. Based on this input data and the locations of the services, one feasible data communication path is calculated and represented in the Figure by the red lines. The result of this calculation is then transformed into the rule set for each node to forward data accordingly.
For node 4 the forwarding rule is $(4,(S4,S6),5,(4,5))$ which states that on node 4, the data received for wiring $(S4,S6)$ needs to be forwarded to node 5 using the communication channel $(4,5)$. This representation is a basic compromise which allows configuring static distributed communication systems agnostic of the QoS requirements and so forms a foundation layer even for more complex network QoS calculations as discussed in the next paragraph.

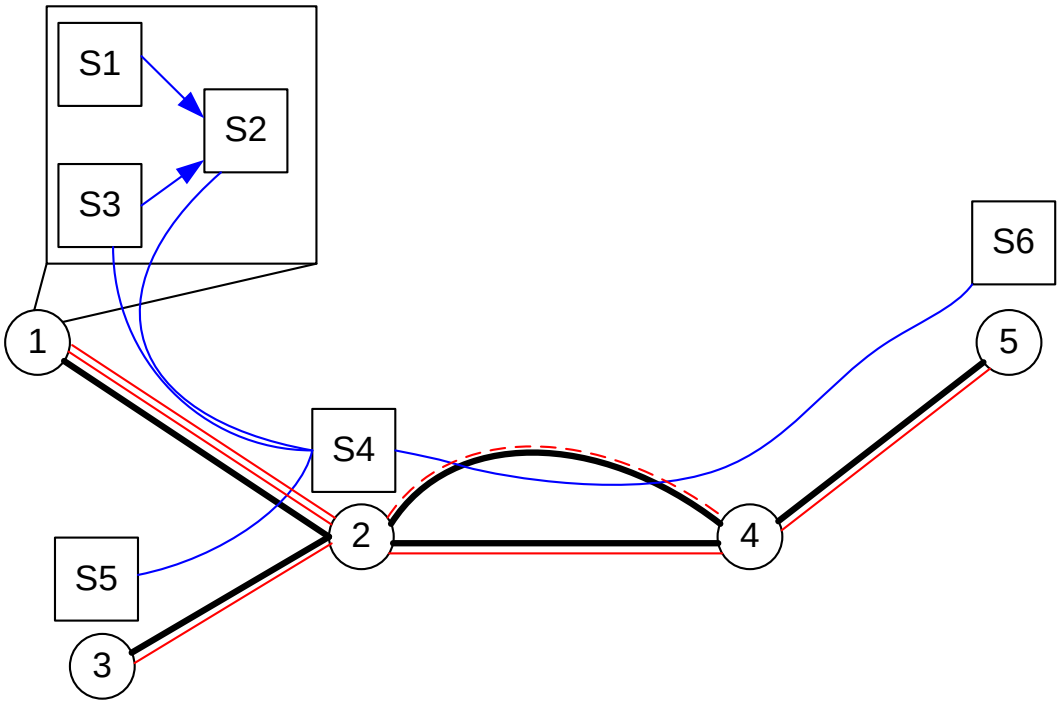


Figure 6.5.: Simple Network Routing Example: Network consisting of five nodes housing six services with their logical (blue) data paths. The physical data paths are represented by the red edges where the dashed red edge identifies an alternative solution for the edge between node 2 and node 4 . The network interconnections are represented by the black edges between the nodes.

6.2.6.4. Complex Network Path Calculation

Taking into account network deployments, where time critical applications are executed or where reliable (in the sense of bandwidth and packet loss probability) communication is required, further aspects of a communication channel need to be examined. The basic work to formally describe communication goes back to Shannon et al. [Sha01]. By adding constraints to the communication / route planning algorithms, the calculation of the data paths can be seen as a classical network optimization problem where already many different solutions and protocols exist for real-time communication in multihop networks [KSF94]. Also taking into account distinct QoS parameters even in wireless network as proposed by e.g., Felemban et al. [FLE06] has already been done.

The results of these calculations always are network graphs where communication paths are aligned to nodes and their links considering the QoS parameters of the links and the wirings. The result is then mapped to a rule set for each node as discussed

for the case of simple network paths and so a basis for the initial network and system configuration is provided.

6.2.7. Step 5: Validation of the Model before Code Generation

Before the code generation is done an additional validation step can be performed where resource allocation, interconnections, and QoS parameters are checked. This is also the step, where additional tools and verification steps can be plugged in to check or prove the feasibility of a deployment.

$$\begin{aligned} validSystemConfiguration = & \forall n \in NodeInstances : validDeployment(n) \wedge \\ & \forall w \in Wirings : validWiring(w) \wedge \\ & \forall c \in Channels : sufficientBandwidth(c) \end{aligned} \tag{6.14}$$

A subset of the checks is discussed in more detail the next paragraphs:

All Nodes Can Execute their Assigned Services

The first check is used to assure, that the assigned services can be executed on the selected nodes. This is validated by the $validDeployment$ function checking the resource parameters Flash, sRAM, EEPROM and the validation of the schedule based on the worst case execution time budgets.

$$\begin{aligned} validDeployment(n) = & validFlash(n) \wedge \\ & validSRAM(n) \wedge \\ & validEEPROM(n) \wedge \\ & validSchedule(n) \end{aligned} \tag{6.15}$$

Each of these validation criteria must be satisfied for a valid system configuration. The criteria $validFlash$ is presented in the following paragraph as an example.

$$\forall n \in NodeInstances : validFlash(n) \tag{6.16}$$

Where $validFlash(n)$ is defined as follows:

$$validFalsh(n) = (\sum_{si \in serviceInstances | si.sl = n} flashRequirement(si)) < n.t.f \tag{6.17}$$

$$\begin{aligned} flashRequirement(si) = (sd \in ServiceDescription | si.s = sd) \wedge \\ (flash \in sd.SFLASH | flash.t = si.sl.t \wedge \\ flash.os = si.sl.os \wedge \\ flash.pl = si.sl.pl) : \\ flash.sFLASH \end{aligned} \tag{6.18}$$

All Wirings Valid: Services Reach Each Other

As wirings represent the logical link between services, the first property to check is whether the wired services and the nodes housing them are directly or indirectly connected via network:

$$\forall w \in Wirings : validWiring(w) \tag{6.19}$$

Where $validWiring(w)$ is defined as follows:

$$\begin{aligned} validWiring(w) : reachable(w.inputService.sl, w.outputService.sl) \wedge \\ signatureEqual(w.inputPort, w.outputPort) \end{aligned} \tag{6.20}$$

Beside the pure reachability, matching service signatures need also to be assured. This check is performed by the $signatureEqual$ rule.

Network Bandwidth Sufficient

The third property which needs to be checked prior to deployment is, if the network bandwidth is sufficient for the transferred data. To check this property each rule set $R \in RuleSet$ has to be inspected for each channel c and each wiring w, to assure, that the required network bandwidth of a channel c is sufficient for all wirings using this channel. In the implementation provided within this thesis, this property is not validated.

6.2.7.1. Implementation

The validation rules described are implemented using the Check language. Experience during development and use of the tool showed, that it is very useful to do a final check on a set of selected key parameters before code generation, although the input of the transformations are already checked. This second validation step helps to decrease the time required to localize errors probably only becoming clearly visible after the code generation or errors produced by the code generator which otherwise would only appear at compile or deployment time. Here the same strategy of testing is recommended as suggested in paragraph 6.2.3.

6.3. Automated Service Placement

In contrast to a completely manual configured deployment it also can be desired to calculate the placement of the services automatically during the deployment after the services requiring direct hardware acces (basic services) have been placed by the user. When considering complex systems, services requiring direct hardware access have shown during the last years to be quite few in comparison to the number of services without direct hardware access.
This directly leads to the idea to optimize the placement of all these independent services to e.g., minimize the expensive network bandwidth in (wireless) sensor networks or to maximize node lifetime by distributing the services over as many nodes as possible [SSB+09].
With increasing number of nodes, the manual deployment gets time consuming as well as quite complex, and so users do not want to take care of each and every service and it's placement as long as it does not reduce overall system performance. This is exactly the case, where an automated mechanism for service placement should be employed.

6.3.1. Preparations for Service Placement

To automatically place services on nodes requires on the one hand a description of the services to be placed and on the other hand a description of the network and the nodes on which the services should be placed. This information can be gathered from the the system specification and models introduced in the past sections. Based on the provided information, a suitable representation for the service placement framework is calculated.
For the network, a graph interconnecting all nodes and all communication channels

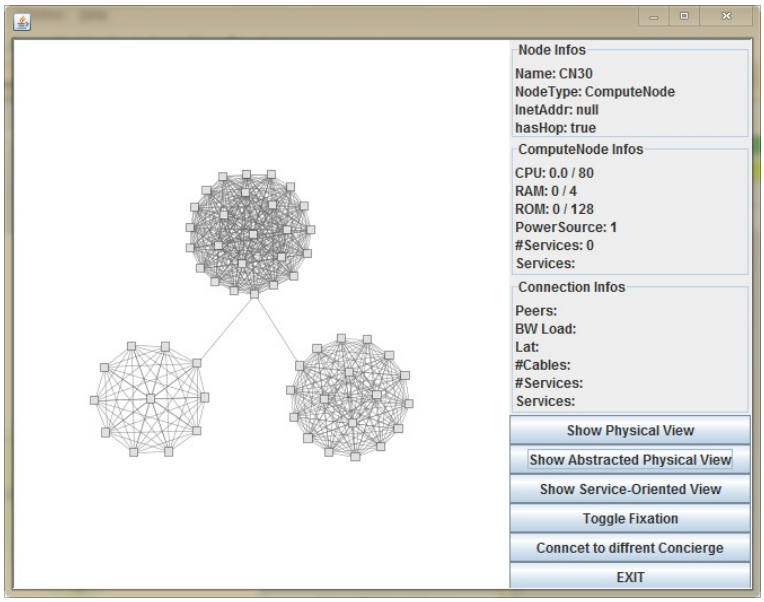


Figure 6.6.: Service Placement: Abstract Network View [Kul11]

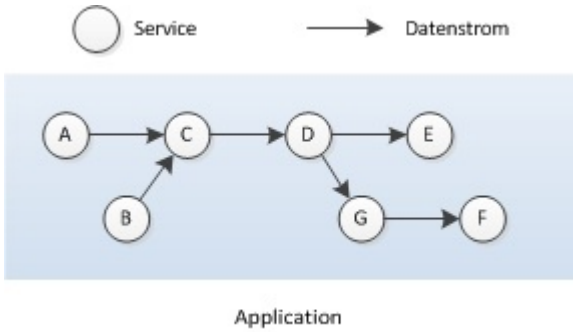


Figure 6.7.: Service Placement: Chain of Services [Kul11]

is calculated as depicted in Figure 6.6. In case of a wireless network using the same frequency or channel, all nodes are directly connected to all other nodes.

An application described in the application model is reduced to the logical chain of services (see Figure 6.7). As in most deployments, not all services are related to each other and so distinct applications can be extracted form the service chains. These chains form the basis to calculate the service placement calculation.

Depending of the optimization goal, different placement algorithms can be suitable to generate a service placements. In general, an optimization goal needs to be defined based on different evaluation metrics. In the following paragraph, different evaluation metrics are discussed.

6.3.2. Placement Metrics

A prerequisite to calculate service placements are metrics. They are used to evaluate a given service placement based on given criteria and allow comparing different placements. The information to calculate these metrics must be provided to the optimization framework before placement calculation. In best case, all this information is already available in the models used as a basis for the application assembly. Alternatively, this information can be made available inside the optimization framework and linked to the model elements during the preparation phase of the optimization. In the following paragraphs, some evaluation metrics for the service placements are presented.

Hop Count

A very simple metrics is the hop count. It represents the number of hops a data set needs to be transferred to execute an application. Without taking into account different cost functions for network communication, the hop count can be used to reduce network communication.

Memory Utilization

For the metrics RAM utilization, the consumed RAM of the service(s) allocated to a node is considered. Depending on the concrete optimization goal, the average or min. / max. allocation can be taken into account. Due to the fact, that all services have to allocate their memory statically (or at least during initialization), this metrics can be easily determine with low tooling effort.

CPU Utilization

In contrast to the RAM utilization, the CPU utilization can only be determined using complex tooling or by WCET estimation. In addition, even if the sum of the required CPU resources could in principle be provided by a node, the time of the execution could bring the system into an overload if all resources are requested at the same time. To simplify the use of this metrics, the CPU resources are over-estimated and a timely execution is assumed in this thesis.

Data Volume

A refinement of the hop count metrics is the data volume metrics. Here, the data consumed and provided by a service is accounted to the network links the data needs to be transmitted. As a result (if taken as an optimization criteria), services interconnected with high bandwidth requirements are allocated on nodes close by or (preferably) on a single node.

Link Utilization

An additional refinement to the hop count and the data volume is the link utilization. Based on the data volume accounted to a link and the overall bandwidth of the physical connection, the link utilization is calculated by dividing the data volume by the overall bandwidth. This factor provides the information of the link utilization and, in case it is greater than one, that the link is overloaded and the calculated placement is not feasible. In order to avoid networking problems in event triggered wireless networks an utilization much below 50% should be targeted.

Combined Metrics

In order to optimize against a combination of these criteria, the quality Q of a placement can be calculated by combining each single metrics with a custom weight.

$$\begin{aligned} Q &= \sum_{x=0}^{x=n} (w_x * M_x); \\ \sum_{x=0}^{x=n} w_x &= 1 \end{aligned} \tag{6.21}$$

Depending on the weight, the importance of the metrics for the overall placement quality can be specified.

6.3.3. Placement Algorithms

A detailed discussion of suitable algorithms and their performance was already published by Scholz et al. [SSB+10]. In this paragraph, only the basic ideas of the different appraoches and the results are summarized.

The task of the algorithms presented in the following paragraph is to determine an optimal placement, i.e., a placement with as little costs as possible, based on a user supplied weighting function for the metrics presented in the previous section and information about the hardware characteristics and the application requirements. The optimization problem of distributing services to nodes can be efficiently mapped to the bin packing problem. The task is to distribute n services with resource demands $d_1 \dots d_n$ to m nodes with resource capacities $c_1 \dots c_m$ in a way that avoids overload situations. The problem is therefore NP hard. For small networks (< 10 nodes) and a small number of services (< 20 services), a solution based on a simple enumeration of all possible combinations is possible.

For larger problem instances, heuristic solutions have to be applied. In the following three optimization algorithms will be presented: an approach based on ant colony optimization, an approach based on simulated annealing, and an approach based on genetic programming.

6.3.3.1. Ant Colony Optimization

It is very difficult to apply the ant colony optimization [Dor06] algorithm to the problem of mapping services to nodes. The reason for this is that the service placement problem exhibits no optimal substructure in many cases. If one service is assigned to a node, this decision may influences other service assignments, because the assigned service will increase the resource utilization on the node and the used communication links. As a consequence, adding a single new service to an optimal placement may require a massive reorganization of the already assigned services in order to meet all resource constraints. Mapped to the ant colony optimization algorithm this leads to the following problem: even if a fairly good "path", i.e., a placement with low costs, is found for a subset of services, this information can not be re-used in subsequent runs. The assignment of other services can change the resource utilization on the nodes used in this subset and therefore render the solution invalid.

6.3.3.2. Simulated Annealing

The simulated annealing [VLA87] based solution is intended to be used on a central management node in the network that possesses global knowledge about the network topology, hardware characteristics and service requirements. This requirement is perfectly aligned with inteded use as part of the system development tool as there all information about nodes, services, and network are available.

The algorithm aims at finding a global solution to the optimization problem, i.e., it will

move already placed services in the network if a new application should be installed and requires already occupied resources. These reorganizations come at a cost, because services have to be migrated between nodes and the corresponding applications will cease to work during the migration process. To provide a good trade-off between the migration costs and the long time savings of a new placement, the algorithm creates a list of placements containing different levels of reorganization, which can be used by the user to select an appropriate placement. This is done by running the placement optimization multiple times with different restrictions for the placement of services, e.g., restricting all installed services to the node they are executed on will result in a scenario with no reorganization.

6.3.3.3. Genetic Programming

Results show, that genetic programming [Koz92] seems to be the most suitable strategy. As mutation function the neighborhood function already used in the simulated annealing approach is employed, i.e., to mutate a genome one service is moved to a randomly chosen new node. If elitist selection is applied, i.e., the currently best genome is always preserved in the gene pool, genetic programming is capable of finding the optimal solution even if there are a lot of sub-optimal solutions with small cost differences.

6.3.3.4. Related Work

Regarding related work with respect to optimal placement of services/aggregators most work deals with sensor networks that perform monitoring tasks [MFHH05, YG02, PLS+06, Bon03]. In such systems, applications/queries can be organized in a tree-like structure. In contrast to this related work, the solution presented in this thesis targets sensor actuator networks with its special needs. It allows optimizing applications that are not centered around a dedicated sink node and it allows a global optimization of embedded networks that takes into account interferences between multiple simultaneously executed applications.

6.3.3.5. Results

As the topic service placement is no direct contribution to this thesis, a detailed evaluation of the topic will not be discussed in this section. The quantitative results and the evaluation of the different approaches are already published by Scholz [Sch11]. The qualitative result concerning this thesis is, that the service placement can be employed

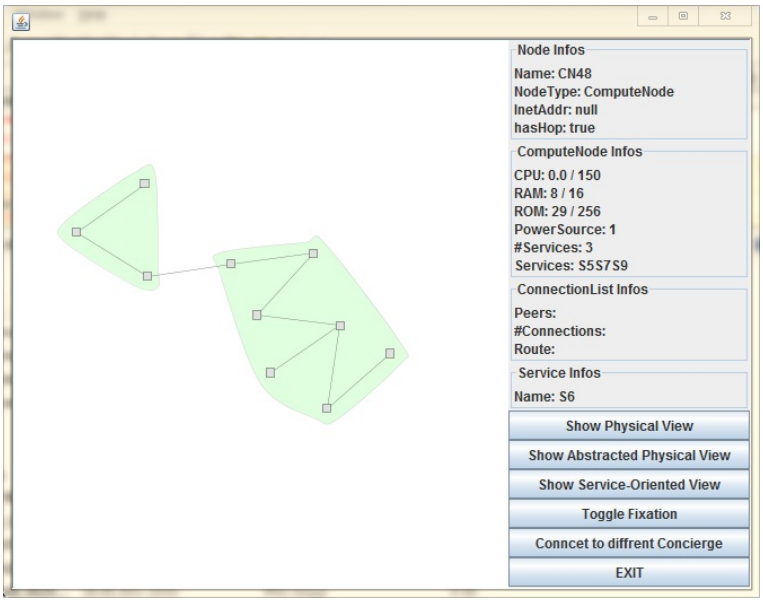


Figure 6.8.: Service Placement: Service and Node View

to automatically calculate an application configuration for a given set of nodes, services, wirings, and network connections.

6.3.4. Integration of Placement Results into the Models

After a service placement is successfully calculated, the result can be visualized as depicted in Figure 6.8 where each node is a service, the edge between services is a communication relationship and the greenish area represents a computation node. In the depicted example, each node houses more than one service, which reduces the overall network utilization. The results of the placement can be integrated into the production model during the model transformation phase. Potential inconsistencies and resource-constraint violations are detected in the model validation step performed in step five of the deployment process and detailed in Section 6.2.7.

From the outside view, the placement step is transparent for all other transformation steps; there is no difference whether the placement is gnerated by the user or by the development tool. After the validation of the generated setting, the production model is employed for the code generation discussed in the next section.

6.4. Code Generation and Tooling

Using models to gain a better understanding of the system under development is a well-established approach. They help to refine and understand a specification and can also be used to verify that the modeled system is feasible based on certain assumptions. The next step to increase the use of a model is to employ it to generate source code without additional user interaction. The code generation can be considered as a further transformation step in the context of model transformations, where the target is no model but source code for a programming language. This transformation is then called model-to-text (M2T) instead of M2M transformation [ONG+05].

In this section, the requirements on the code generation process will be discussed and different types of code generation frameworks will be presented. The different approaches will be discussed and the approach selected for the implementation in this thesis will be presented. This is followed by a short overview of the development tool and its features, which have already being discussed in the past sections.

6.4.1. Code Generation Requirements

As already elaborated in Section 4.2 one requirement of code generation is extendability. As code generation or M2T is a fundamental part of the OMG MDA [MM03], there was already a lot of work done raising requirements for this task as stated in [ONA04] and [Gro07]. The six basic properties of a code generation language considered in this thesis are the following:

Structuring

A code generator needs to implement mechanisms which allow modularizing the code generation modules to reduce complexity and to increase maintainability. This is especially important as soon as more than one generation target needs to be handled.

Control Mechanisms

In order to use different modules and handle different aspects of a model or a platform, control flow mechanisms like branches (if and loop) are required to allow a flexible and maintainable backend implementation.

Mix of Tool Code and Output Text

In order to conveniently use the code generation language to describe the output data, it is useful to be able to mix the generator statements into the output text as well as to directly call service functions within the template. A code generation framework needs to provide this capability to allow a fast and easy development and maintenance of the code generators.

Services Methods

Implementing M2T transformation implies handling data structures like strings or complex data types. In order to allow convenient handling of these data structures, according service methods like string concatenation, transformation to lower or upper case and comparison methods are necessary. For efficient development, these methods should be seamlessly integrated into the code generator framework.

Ease of Use

To provide all the functionality derived from the requirements, a transformation / code generation language is a feasible way to implement these methods. In order to guarantee that the language is easy to use and easy to learn, it should be employed like well-known languages and behave as expected and known from other tools.

Expressiveness

In order to provide all the desired functions without employing external plugins written in a different language, the language needs to provide enough expressiveness to formulate the tasks. The expressiveness always is a tradeoff between flexibility, ease of use and the demands of the user.

6.4.2. Code Generation Techniques

As code generation is the direct implication of fully using model-driven approaches, there are many different implementations as MOF2Text [Gro07], MOFScript [Old06, Old06], Jamda [Boo], Velocity [SvVB02], XTEND [Effb], and TCS [JBK06], all targeting different aspects. All of them have certain advantages or disadvantages in respect to the targeted application. To give a short overview, a selection is discussed in the following paragraph based on the employed approach.

6.4.2.1. Visitor-based Approach

A very simple approach to generate code out of a model is the visitor mechanism. In this approach, an internal representation of the model is traversed by a visitor [VHJG95] collecting all the information required for code generation. This visitor is also the source for the generated code, which is written to files while or after the model traversing. A tool implementing this approach is the Jamda [Boo] framework. It provides an application programming interface (API) to manipulate the model and uses so called CodeWriters to traverse the model for code generation. Using this approach (based on an already available tool or by implementing the model by hand) provides a simple way for doing code generation. The main drawback here is that all the generation logic, the traversal und even the output needs to be implemented by hand and so needs to be changed as soon as the model or the desired output changes.

6.4.2.2. Pattern Substitution-based Generation

In contrast to the visitor-based approach, the pattern substitution based approach already provides the framework to traverse the model and collect all information required for code generation. The actual code generation is here done by replacing preknown and pre-defined key words by model elements using pattern matching. An implementation of a pattern substitution-based program transformation is described by Visser [Vis04]. This approach is in general quite simple to setup and easy to debug, but has one major drawback, its limited flexibility. As soon as more complex information needs to be gathered or as soon as the code generation cannot be limited to simply replacing key words, this approach is not suitable anymore.

6.4.2.3. Template-based Code Generation

A consistent enhancement to the pattern substitution-based approach is the template-based [BFVY96] approach. In this approach, a template is provided by the user which is then filled by the code generator. Depending on the complexity of the template, the approach can be as simple as a pattern substitution-based generation, but also much more flexible and complex. There are many different implementations available using this pattern. One implementation of this approach is the Velocity [SvVB02], another one is the XTEND language [Effb]. This is also the code generation technique employed in this thesis to enable SensorLab to generate application and configuration code.

6.4.2.4. Textual Concrete Syntax

The Textual Concrete Syntax (TCS) [JBK06] is a DSL to bridge the gap between models and text. Based on the specification written in TCS, a link between a meta-model and a set of keywords and symbols is given which provides the relation between models and a grammar. Based on this specification, text can be generated from models and vice versa. Although the way back from the text to the model can be a great benefit if needed, it requires a tight coupling between the meta-model and the generated text to allow a distinct mapping between both. When only focusing on the model-to-text transformation, this can become a restriction for the code generation by introducing a tighter coupling then necessary.

6.4.3. Code Generator in SensorLab

To implement the code generator for this thesis, a template-based [BFVY96] approach is employed. The XTEND language [Effb] satisfies the requirements stated in paragraph 6.4.1. It allows a good structure by modularization, implements control flow by statements like *foreach* and conditional branches like *if*. There are built in methods for string handling as well as doing calculations. Statements of the code generator can be written next to code templates and variables to guarantee a good ease of use. The required expressiveness of the language is sufficient to implement the code generation of SensorLab. However, complex transformations and pre-processing of model data was mostly done inside one of the model transformation steps elaborated in Section 6.2 to keep a clear separation between model-to-model and model-to-text transformations. The template-based approach offers not only the possibility to adjust some parameters of the template for a generation run, but also to generate strongly application dependent components of the middleware like a routing table of the broker.
Templates can be used to solve certain aspects of the run-time system or to combine the results of different templates to form the middleware. Most templates are platform dependent in the sense that they offer a solution only for a certain combination of hardware and operating system.

As already mentioned, developing a code generator from scratch does not make sense, as it is time consuming and complex depending on the functionality required so an off-the-shelf framework was used for the code generator in SensorLab called openArchitectureWare[2] [VSK05]. openArchitectureWare provides for these problems a special template language, call *XPand*. XPand offers the statements *DEFINE* to declare a new code generation function and *EXPAND* to call other generation functions during the

[2] http://www.openarchitectureware.org/

```
≪FOREACH app.componentInstance AS ci≫≪IF ci.node==n≫
  Main.StdControl ->≪ci.name≫C.StdControl;
  BrokerC.≪ci.name≫ ->≪ci.name≫C;
  ≪ENDIF-≫
≪ENDFOREACH-≫
```

(a) Code Template using XTEND Language

```
Main.StdControl ->OnOffLEDC.StdControl;
BrokerC.OnOffLED ->OnOffLEDC;

Main.StdControl ->LightClapServiceC.StdControl;
BrokerC.LightClapService ->LightClapServiceC;

Main.StdControl ->SoundSensorC.StdControl;
BrokerC.SoundSensor ->SoundSensorC;
```

(b) Code Generated from Template

Figure 6.9.: Code Generation: From Template to Code

code generation. openArchitectureWare also allows polymorphism as one element to select adequate templates.
To specify the control flow of the code generation, the commands *FOR/FOREACH* and *IF/ELSE* can be used. The *FOREACH* statement is used to generate code for each object of a certain type that is declared within the model. Finally, the commands *FILE* and *ENDFILE* allow the management of the generated files. The code generation process is then rather simple. The adaptation of the templates to the model is performed using a technique similar to preprocessor macros. Text sequences between the different XPand commands are directly copied to the generated files and variables allow the access to objects and their attributes.
Figure 6.9(a) shows a simple template that illustrates the basic concept. The template realizes the generation of links between the components on one node and its broker in TinyOS 1.x. The required information can be retrieved from the model. The generated code is depicted in Figure 6.9(b).

6.4.4. Development Tool: SensorLab

The SensorLab development tool depicted in Figure 6.10 integrates the development process as well as the models and transformations discussed in this chapter. It is used to generate a run-time system based on the environment and applications specified in the models and based on the service oriented middleware presented in Chapter 5.

In the central pane of Figure 6.10, all wirings (interconnections between services) for the eSOA demonstrator are shown. On the right pane the service repository is depicted

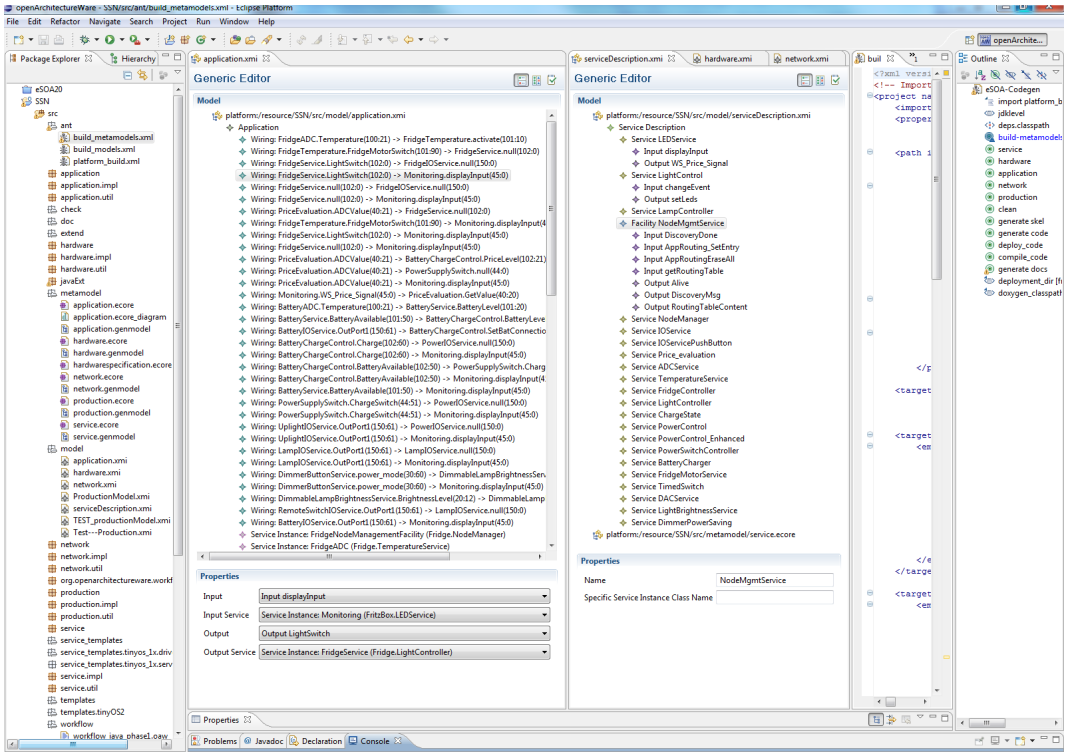


Figure 6.10.: SensorLab Development Tool - Main View

with the NodeManagement service in focus showing the input and output ports. On the left pane, the tree structure housing the meta-models, models, templates, and workflow files is shown.

The development tool is based on Eclipse and uses the Eclipse Modeling Framework (EMF) [SBMP08] for the modeling part. The development workflow is implemented using the open architectureware framework and executed using ANT [HLRV03] scripts. The output of the code generation is a structured folder tree housing static as well as generated files e.g., for the TinyOS software platform. Using the ANT scripts, the source code is copied to build folder and then compiled for the target platform. If the nodes are directly connected to the development workstation, the images can be flashed to the respective node identified by its unique node ID provided by the manufacturer.

6.5. Summary and Contribution

In this section, the presented approach and the developed tool will be evaluated based on the key requirements presented at the beginning of this chapter. For this purpose, the requirements will be discussed one by one in the context of the presented approach. The result of this evaluation is then summarized and the contributions are pointed out.

6.5.1. Realization of the Key Requirements

As already mentioned in Section 4.2 the approach elaborated in this thesis is based on few key requirements targeting model-driven development. The realization of these requirements by the presented development process will be discussed in the following paragraphs.

Extendable Models

The first requirements is that (meta-)models need to be extendable to target new or changed applications. In contrast to well-established tools like MatLab / Simulink where the meta-model can be generic and hence static considering the supported (mathematical) operators, deployments for embedded systems can not be tailored based on a generic view of the system. This leads to the necessity of specialized meta-models to describe the concrete hardware. As all hardware variants can not be knows during design-time of the development tool, the (meta-)models need to be extendable. This requirement is satisfied as the meta-models can be easily extended by new elements. As long as the new elements do not require a change of the production model, the changes only affect the transformation and the aspect model representing the element in question. As soon as the changes are system wide, the aspect model as well as the transformation, the production model and the code generation need to be adapted. Although these changes have system wide effects, many of these changes can be applied using suitable tooling as proposed by Kainz et al. [KBK12, KBK11].

Extendable Code Generation

As a consequence of extendable models and changing generation targets, the code generation also needs to be extendable. This requirements is satisfied by using the off-the-shelf code generator based on the EXPAND language which allows a simple extension

of already developed templates as well as a simple addition of new templates if completely new targets need to be targeted.

Separation of Concerns

The next requirement is also somehow related to the flexibility and maintainability claimed by the first requirements. The separation of concerns is realized using the different aspect models representing the different aspects of the system. As the combination of these aspects into a single representation is done automatically, additional aspect models can be easily added to the development tool without changing the already available aspect models.

Models Need to Allow Precise Definition of Application

As the main goal of the models is to be the basis for code generation, they need to be precise enough to represent the desired system behavior. As the tool developed within this thesis does only focus on a certain application domain and does not focus on the generation of application code (which can be easily provided using e.g., MatLab / Simulink), the models are precise enough to describe the system under development. This constraint to extra-functional properties and a dedicated application domain reduces complexity in comparison to the broad OMG MDA approach.

Validation of Model Input

The validation of the models prior to code generation is one key benefit of the model-driven development approach. This is implemented using CHECK rules during the development process. In addition, formal methods can be easily applied during the model transformation phase using external tools.

The result of the evaluation of the presented development process and tool in respect to the key requirements is, that they are all satisfied which confirms, that the approach is suitable for developing networked embedded systems. In addition, due to the flexibility of the presented approach, it can be easily extended to add additional functionality or off-the-shelf tools. One very useful extension would be the addition of a generator for the copy operations within the model transformation to increase maintainability as stated by Kainz et al. [KBK12].

6.5.2. Contributions

In this chapter, a model-driven development process for networked embedded systems was presented, supporting the separation of concerns for different user groups namely the Platform Specialist, the Domain Experts and the End-Users / Installers. The development process was elaborated in detail including the required models, the transformations and the template-based code generation. In addition, an example application (service placement) was presented to show, how the model transformation can be extended by external tools. Beside the functional description, a short overview of the development tool and the employed technologies was given. The suitability of the process in respect to the key requirements on a model-driven development process for networked embedded systems was shown at the end of this chapter.

CHAPTER 7

Conclusion

Contents

Due to decreasing cost for hardware and increasing processing power formerly simple sensing and actuating devices are becoming "smart" and interconnected. They form networked embedded systems and present the developer with a variety of challegnes. This is the area of research where this thesis contributes by **lowering the complexity** for developing and deploying new networked embedded systems by employing **model-driven design and development** techniques including extensive **code generation**. The focus of this work is mostly in the home automation as well as in the process monitoring domain where two demonstrators have been build to show the feasibility of the elaborated approach.

7.1. Summary of Contributions

The contributions of this thesis are threefold. First, an adapted service oriented architecture (eSOA) suitable for resource constraint networked embedded systems is elaborated. To provide an execution environment for eSOA with its services, a tailorable

middleware is defined and elaborated. These efforts to lower the complexity of developing networked embedded systems are brought together by a development process and a suitable model-driven development tool, SensorLab. SensorLab finally employs code generation to provide the user with the source code and the configuration of the system under consideration.

Adapted SOA for Embedded

The first step to lower the burden for new deployments is to increase reuse of available components by introducing a clear separation between application and infrastructure code as well as by introducing well-defined interfaces to interact with applications. To master this challenge, the first contribution of this thesis is the employment of a embedded service oriented architecture (eSOA) by mapping the basic principles of SOA to the embedded domain and by providing a set of adaptions to tailor services for the use in resource constraint networked embedded systems.

Modular and Tailorable Middleware

As a consequence of using services only housing application logic, a middleware or run-time system is required to provide the services with the necessary infrastructure to interact and communicate. The goal of making the middleware **resource efficient** and at the same time **tailorable** for each application is the second contribution of this thesis. This is achieved by providing a modular selection and move the tailoring and configuration effort into a model-driven development tool.

Model-Driven Development Process and Code Generation

The third and final step of reducing the complexity of developing networked embedded systems is to raise the level of abstraction by introducing a (graphical) development tool. Key factor here is, that the developer groups involved in engineering these systems are sufficiently represented. In the work done in this thesis, three different groups involved in the development process are considered. These groups are the *Platform Specialist* providing the infrastructure as well as the tool support, the *Domain Experts* providing the application code and the *End-Users / Installers* assembling the applications. This separation of concerns is especially visible in the models forming the foundation for the deployment validation as well as for the code generation.

7.2. Prove of Applicability

The three contributions of this thesis summarized in the last paragraphs provide the foundation for the application of the concepts in practice. Therefor the implementation of the concepts are available as SensorLab development tool. This tool was employed to assemble the following demonstration scenarios.

The feasibility of this approach was shown using two different demonstrators, one for the home automation domain controlling a smart, energy-aware home and one for the process monitoring domain tracking a selection of parameters of the production plant. For both show cases, the application on the embedded networked nodes was specified and assembled using the SensorLab tool developed as part of this thesis. Thereby, it was also shown by applying the approach to different domains and applications scenarios, that the presented approach is not restricted to a single example application nor to one domain.

7.3. Outlook and Future Work

The approach developed and presented in this thesis showed its applicability for the home automation and process monitoring domain. A consequent continuation of the work is to elaborate on the applicability to additional application scenarios and domains. One application scenario for future work are automotive information and communication (ICT) architectures which are facing challenges due to their increasing complexity [BBD+11].

7.3.1. Mapping of the Approach

In current research, centralized ICT architectures [SCB+13] similar to avionics are considered to provide a solution for the increasing complexity. These centralized architectures present a problem space similar to the work done in this thesis. A system consists of a number of computing nodes forming the centralized platform, of a number of devices providing access to sensors, and actuators and of software modules (services) which can be arbitrarily placed on the centralized platform depending on the safety goals. The mapping of the three contributions of this thesis to automotive ICT architectures could be as follows.

Services for Embedded Devices

The major advantage of using services instead of application functionality interwoven with run-time system code is obvious. A clear separation of concerns and reuse of application code. This can directly be mapped to future automotive architectures, where a standardized, safety-aware run-time system is provided as a execution environment for the applications. Similar to the embedded services described in this thesis, an application component only provides the implementation of a domain specific task like steering or adaptive cruise control and allows the developers to focus on their domain and expertise.

Tailorable and Modular Middleware and Run-time System

To allow the application developers to focus on their expertise in providing the application logic, a suitable and easy to interact run-time system needs to be available. A transfer of the results of this thesis can be an increased applicability of the automotive run-time system by using a code generator to tailor the system. The generator-driven tailoring could be applied to the configuration by automatically deriving an appropriate selection of software modules to assemble the run-time system as well as to tailor selected components for a specific deployment. Both leads to a reduction of complexity and an increase of development speed.

Tooling and Development Process

The major goal to continue the work done in this thesis by transferring the results to the development of automotive run-time systems is to enhance the methodology and the system specification by the means of safety and redundancy. The tooling can then be employed to generate placements of application software modules regarding resource utilization and safety requirements. The models and the derived information can also be employed to assemble and configure the run-time system for a specific hardware platform or deployment.

7.3.2. Summary

The applicability of the approach elaborated within this thesis has already been shown using two demonstrators. The transfer and the mapping of the three key parts of the approach to an additional domain (automotive) can be done as elaborated in this section.

The major benefit of mapping the model-driven approach, especially the tooling support to future automotive run-time systems can be a significant reduction of complexity. A obvious task for future work is to investigate the precise requirements for future automotive run-time systems and derive the necessary extensions of the system specification. Employing the automotive industry with its scale effect of mass production and cost reduction can then be used as a stepping stone for other domains where electronics are now considered as to expensive. Additionally, the integration of formal methods to support the validation and qualification (in safety means) of the system is an interesting point for future research.

APPENDIX A

System Meta-Models and Models

A.1. Hardware Meta-Model

The hardware meta-model depicted in Figure A.1 is one of the basic models of the developed system. It is used to describe the involved hardware classes. Based on this meta-model, a hardware-model is be instantiated to describe the involved device types for a specific setting. In case, additional information is needed, the meta-model and so also the model can be easily extended.

The structure of this meta-model is described in the following paragraphs:

Hardware Description

The hardware is described using a *Hardware Description* class, which itself consists of *Nodes*. This class is used as a container for the remaining parts of the meta-model.

Nodes

The nodes described in this meta-model are no real instances, they represent the different types of nodes, which are characterized using several attributes like *RAM* and *ROM* for the available RAM and ROM. To make the nodes networked embedded systems, all of them have at least one communication interface which is represented by the *Communication Medium* class.

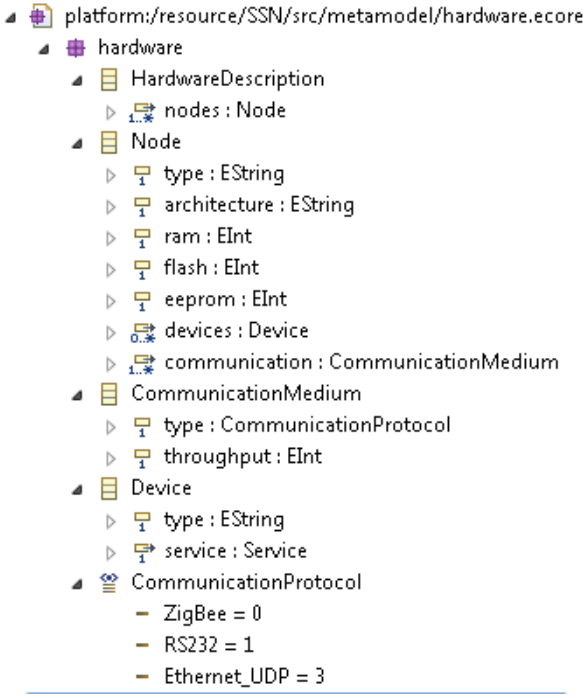


Figure A.1.: SensorLab Hardware Meta Model

Communication Medium

The communication medium describes communication between at least two nodes. In the initial version, communication using RS232, ZigBee, and Ethernet is supported. This selection is represented by the *type* parameter. Based on the communication type (or technique), reasonable parameters for the underlying communication medium can be suggested. The most important characteristic is the *throughput*. Throughput is specified using an integer value and is later on used to check the feasibility of a configured application in the sense of communication requirements.

Device

Many devices used in the embedded domain provide interfaces to extend their functionality by additional hardware, which is probably not known form the beginning of a design phase. Such interfaces are e.g. TWI [Sem00b], SPI [Sem00a] or a simple UART [Osb80]. To connect devices attached using these interfaces, device drivers are necessary to bring the functionality to the runtime environment. To model this case, the *Device* class is used. It consists of a verbal description of the attached device (the

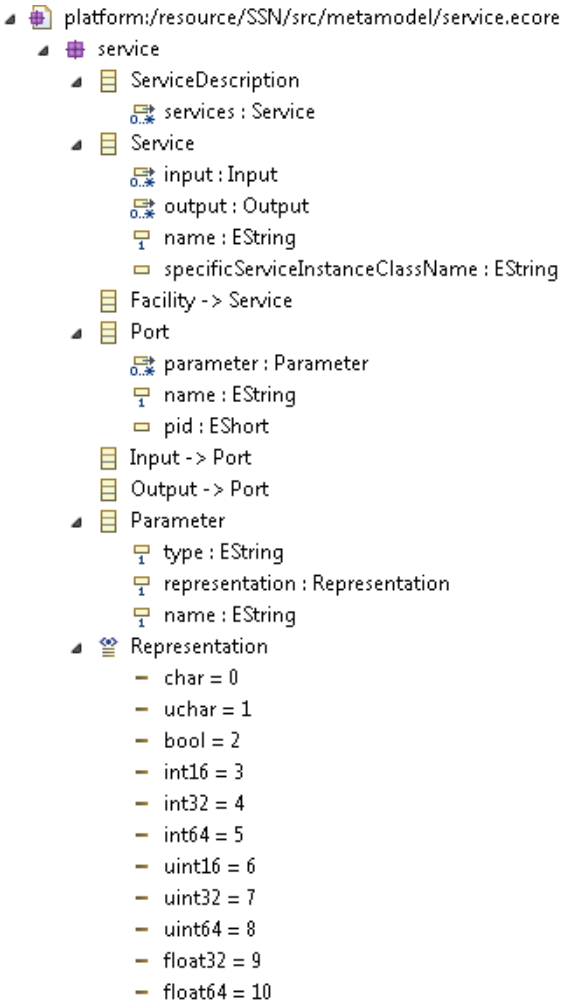


Figure A.2.: SensorLab Service Meta Model

attribute *type*) and the attribute *service*, which represents a service implementing the software functionality (device driver) of the device.

A.2. Service Meta-Model

The second meta-model required to describe an application, is the service meta-model depicted in Figure A.2. It provides the basic structure of a service including the communication interface. For advanced mechanisms like service placement and optimization of the communication topology, additional characteristics e.g. required memory and processing power can be stored in the service model. The basic meta-model is discussed in the following paragraphs:

Service Description

The service meta-model is based on a *Service Description* class, holding the information about all available services and their description. It can be seen as a library of building blocks. In this library, each service is described using several attributes, especially the interfaces.

Service

The *Service* class consists of attributes for service name, for inputs, outputs and an extension point called *specificServiceInstanceClassName* to describe special services, where e.g. service configuration parameters are already available in the model.

Ports

For communication, services use *Ports*. Ports can be inputs as well as outputs. They describe the interface of a service. Each port consists of at least a name, the parameters and their data type. So a port can be considered similar to the signature of a function. In addition, a port has a port id. The port id (pid) can be set by the user during development or is left blank. All unset port ids are configured by the development tool prior to code generation. Based on the settings, the id alignment is done in a pre-defined range. To assure a consistent configuration of port ids over several development cycles, the assigned port id can be stored in the model and reused.

Parameter

The parameters of a port describe the signature in detail. Each parameter consists of a name, the data-type and a representation. The distinction between representation and data-type is necessary, to abstract from different platform and compiler settings. During the development process, the user only specifies the representation and the name of the parameter. Based on the selected configuration (platform, CPU, compiler), the corresponding data-type of the target platform is added during the transformation from the basic models to the production model. This step will be considered in detail in Section 6.2.

A.3. Network Meta-Model

The network meta-model provides the basic types and structures to describe the networking part of the networked embedded systems considered in this thesis. The information provided in this meta model is used to create a representation of a physical network topology including the parameters relevant for service placement described in Section 6.3 and sanity checks discussed in Section 6.3. The representation depicted in Figure A.3 is quite simple, but can be used as a basis for further extensions. The content of the meta-model is discussed in the following paragraphs:

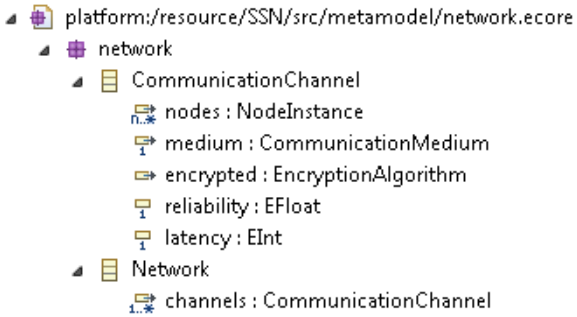


Figure A.3.: SensorLab Network Meta Model

Network

The network class is a container housing all available *Communication Channels*. It is also the point, where additional, non-channel dependent options and information can be added for further extension like a Schedule for time triggered execution when using a TDMA-based communication infrastructure. Information items of this kind can be added during modeling time by the user or during a transformation step later in the process.

Communication Channel

The communication channel represents a set of nodes, communicating with each other on distinct channel. A communication channel is a virtual network associated to a communication medium (e.g. ZigBee, Ethernet) and a physical network interface. Each node has to be connected to at least one communication channel; the number of communication channels themselves is not restricted as well as the number of participating nodes. A node can participate in multiple communication channels with a dedicated interface for each channel. Nodes being part of more than one communica-

tion channel can act as gateways between communication channels. Using more than one communication channel to connect nodes introduces redundancy and basically enables fault tolerance.

To describe a communication channel in more detail, there are several additional parameters. To decide, if a communication channel implements encryption, there is the *encrypted* attribute which provides a selection of implemented encryption algorithms. Based on the physical communication medium a communication channel is using, attributes like latency and reliability can be defined. Latency for example depends on the medium access implemented by the communication medium and the overhead introduced by the communication middleware. For a TDMA medium, a clear upper bound for the communication latency can be given. The reliability is described using a percentage of possibly lost packets. Based on this information, the system configuration can be derived in later steps of the development process as well as configuration and reliability problems can be identified using suitable checks.

A.4. Application Meta-Model

The last meta-model involved in modeling an application scenario (form the users perspective) is the application meta-model. It is used to describe the application under development. In contrast to the service and hardware model, the application model represents real instances of involved devices and services, as the others only act as a kind of repository to store information on supported hardware and available software services.
The application meta-model consists of one major entry, the *Application*, which acts as a container for the remaining description of a scenario. Basically the application meta-model consists of three major parts, the *Node Instance*, the *Service Instance* and the *Wiring*. These will be discussed in the following paragraphs:

Node Instance

A node instance represents a physically available node involved in the modeled application. It is an instance of one piece of hardware described in the hardware meta-model. This instance is identified using the *name* attribute to provide a human readable name to a specific instance. The *nodeType* attribute is used to represent the link between an instance and the corresponding type of a node. Further information like *OS - (Operating System)* and *Programming Language* can be defined. Depending on the selected OS and

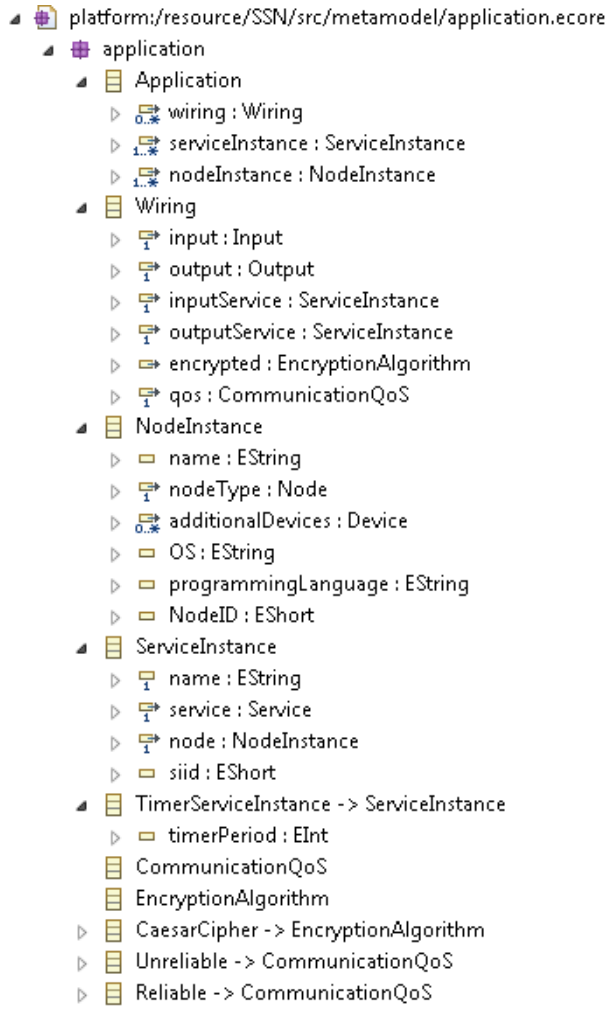


Figure A.4.: SensorLab Application Meta Model

programming language, different code generators are employed. By using the *additionalDevices* attribute e.g. extension boards providing additional hardware interfaces or storage can be modeled and so linked to the node. Finally, the *NodeID* attribute is used to uniquely identify the node in the network. Depending on the application scenario, the NodeID can be selected by the user or by the tool. All IDs which are left blank are automatically set during the model transformation into the production model. The mechanism here is the same as for the port id of a service (see paragraph A.2).

Service Instance

The second type of instance required to describe an application is the service instance. Beside the *name* attribute providing the user an attribute to supply a speaking name

for every instance of a service involved in an application, there is the *service* attribute which represents the link from the service instance to the service described in the service model. The *node* attribute, is used to specify on which node a service is executed on. In principle, a service could be executed on any node in the network as long as there is no hardware directly required by the service and if there is an implementation available for a specific node. In this section, the placement of a service on a node is user defined during modeling time and considered fixed for a deployment. In Section 6.3, an automated service placement framework is discussed to simplify service placement and to increase the service placement quality in the sense of bandwidth and node capacity use. Finally, the *siid* (service instance id) attribute is used to identify the service instance at runtime. This identifier can, as well as the identifier for a node instance, be left blank or set by the user. All unset values will be set by the development tool during the model transformation.

Wiring

The element making a distributed application out of nodes and services represented in their respective models is the *Wiring*. It represents the interconnection between services via their ports, implementing logical *data paths*. To describe a data path, the most important informations are the source and the destination of the data. This information is represented by the *outputService*, *output*, *inputService* and *input* attributes. The (input/output) services are used to identify the services which interact which each other. The input and output attributes identify the interface they interact with each other. Using this notation, the location of a service is not specified. As from application's perspective, only the data streams between different services are relevant to fulfill a task, the location of the services is explicitly avoided in the specification of the wirings. As a consequence, the wiring is static, even if services are moved to different nodes, even into different subnets.

In Adition to these basic attributes required to establish the connection, there are two additional attributes describing the data channel in more detail. The first is the *encrypted* attribute which is used to state if communication needs to be encrypted and how or if the communication can be plain. Depending on this setting, the middleware will take care of the encryption. This attribute can additionally be sued to optimize the middleware by deciding, if e.g. encryption should be part of the middleware at all or not. This decision can be made based on the configured wirings and system wide configuration settings. The last attribute describing the data channel is *QoS*. This is used to model different requirements like timing and reliability.

A.5. Production Meta-Model

Based on the elements included in the meta-models discussed above, the *Production Meta-Model* consists of a concatenation of these elements to increase usability for the following transformations. To keep a maintainable and clear structure, the elements of the already existing meta-models are stored as children of the production class. Additional information calculated during the transformation process, is added as element at the top level (if globally needed) or as child of the according model-element. Additional elements e.g. house elements to describe packet routing or elements for additional unique identifiers (IDs) for certain elements. How the transformation is done based on the source models and which elements need to be added will be discussed in Section 6.2.

A.6. Models, Instances of Meta-Models

Based on these meta models, an instance of each model is created and the system is assembled by the user. The modeled elements are then used as basis for further development steps. As already mentioned for production meta-models, the basic content of the production model (source for code generation) is also filled using the user input in the different aspect models. One example excerpt of a production model ready for code generation is depicted in Figure A.6. It consists of the information about the services supported by the system (*Service Description*: e.g. LEDService to control a bank of LEDs), the underlying network topology including the communication media described in *Network* (e.g. RS232, ZigBee) and the application itself with the instantiated services and their interconnections. Additional information is stored in *Generator Config* where e.g. the generation of debug stubs can be enabled. How the production model is assembled based in the aspect models and which steps need to be performed for this transformation is elaborated in the next section.

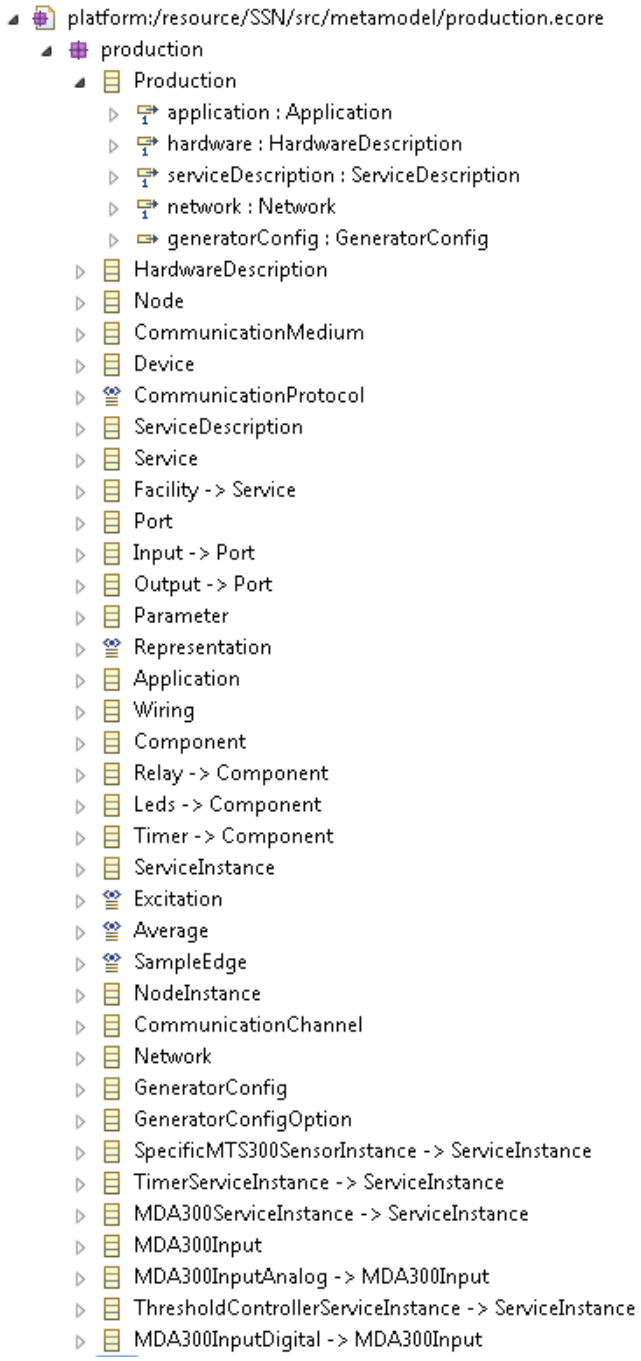


Figure A.5.: SensorLab Production Meta Model

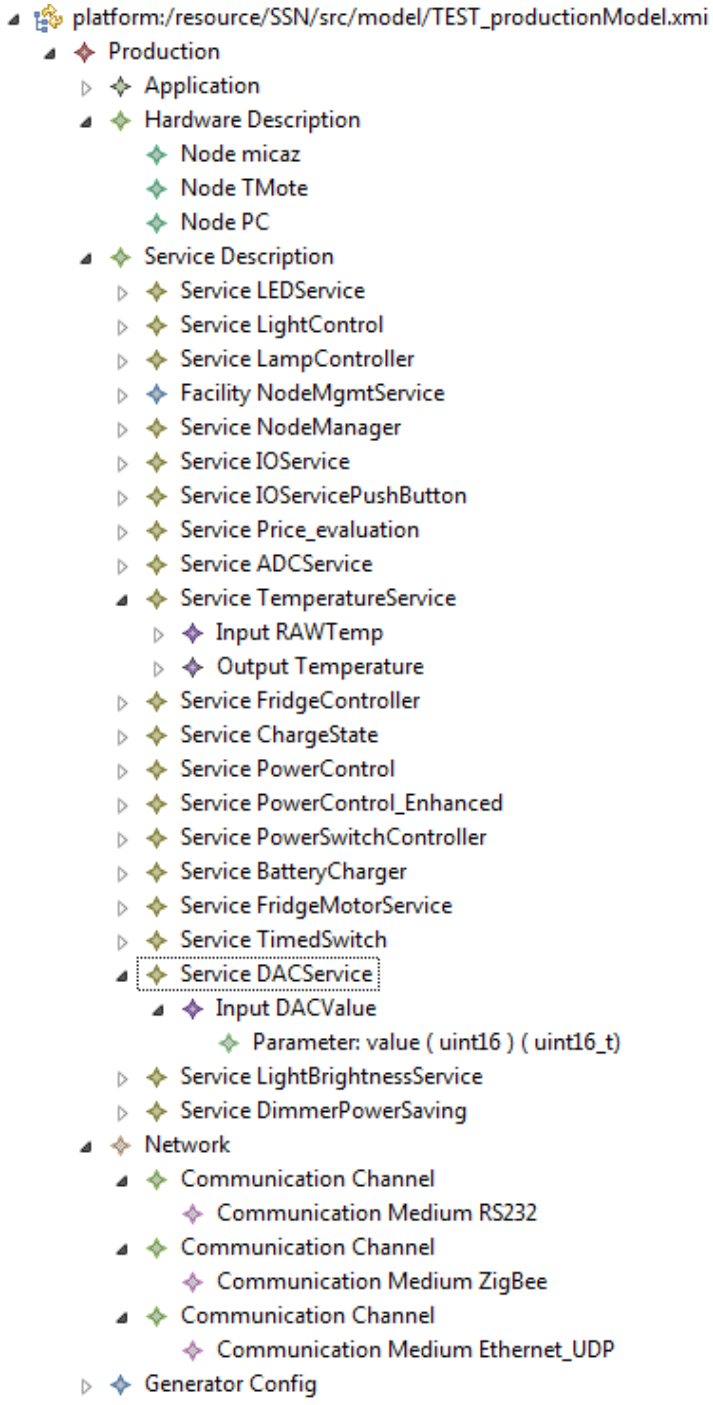


Figure A.6.: SensorLab Production Model Expanded

Bibliography

[ABGR10] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. On challenges of model transformation from UML to Alloy. *Software & Systems Modeling*, 9(1):69–86, 2010.

[ABPG05] Colin Atkinson, Christian Bunse, Christian Peper, and Hans-Gerhard Gross. Component-based software development for embedded systems–an introduction. In *Component-Based Software Development for Embedded Systems*, pages 1–7. Springer, 2005.

[ACD+03] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, et al. Business process execution language for web services, 2003.

[AH87] Gul Agha and Carl Hewitt. Actors: A conceptual foundation for concurrent object-oriented programming. 1987.

[AJG07] Asad Awan, Suresh Jagannathan, and Ananth Grama. Macroprogramming heterogeneous sensor networks using COSMOS. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 159–172. ACM, 2007.

[AK03] Colin Atkinson and Thomas Kuhne. Model-driven development: A metamodeling foundation. *Software, IEEE*, 20(5):36–41, 2003.

[All06] ZigBee Alliance. Zigbee specification. *Document 053474r06, Version*, 1, 2006.

[Aru04] Mahesh Umamaheswaran Arumugam. Infuse: a TDMA based reprogramming service for sensor networks. In *Proceedings of the 2nd inter-*

national conference on Embedded networked sensor systems, pages 281–282. ACM, 2004.

[ASK08] Noriaki Ando, Takashi Suehiro, and Tetsuo Kotoku. A software platform for component based rt-system development: OpenRTM-AIST. In *Simulation, Modeling, and Programming for Autonomous Robots*, pages 87–98. Springer, 2008.

[AVT06] Margarida Afonso, Regis Vogel, and Jose Teixeira. From code centric to model centric software engineering: practical case study of MDD infusion in a systems integration company. In *Model-Based Development of Computer-Based Systems and Model-Based Methodologies for Pervasive and Embedded Software, 2006. MBD/MOMPES 2006. Fourth and Third International Workshop on*, pages 10–pp. IEEE, 2006.

[BBD+11] Manuel Bernhard, Christian Buckl, Volkmar Döricht, Marcus Fehling, Ludger Fiege, Helmut von Grolman, Nicolas Ivandic, Christoph Janelle, Cornel Klein, Karl-Josef Kuhn, Christian Patzlaff, Bettina Riedl, Bernhard Schätz, and Christian Stanek. *The Software Car: Information and Communication Technology (ICT) as an Engine for the Electromobility of the Future, Summary of results of the "eCar ICT System Architecture for Electromobility" research project sponsored by the Federal Ministry of Economics and Technology*. ForTISS GmbH, March 2011.

[BDJ07] Alan W Brown, Marc Delbaere, and Simon K Johnston. A practical perspective on the design and implementation of service-oriented solutions. In *Model Driven Engineering Languages and Systems*, pages 390–404. Springer, 2007.

[Béz01] Jean Bézivin. From object composition to model transformation with the mda. In *TOOLS (39)*, pages 350–354, 2001.

[BFS+06] Erwan Brottier, Franck Fleurey, Jim Steel, Benoit Baudry, and Yves Le Traon. Metamodel-based test generation for model transformations: an algorithm and a tool. In *Software Reliability Engineering, 2006. ISSRE'06. 17th International Symposium on*, pages 85–94. IEEE, 2006.

[BFVY96] Frank J. Budinsky, Marilyn A. Finnie, John M. Vlissides, and Patsy S. Yu. Automatic code generation from design patterns. *IBM Systems Journal*, 35(2):151–171, 1996.

[BH94] Ed Baroth and Chris Hartsough. Experience report: Visual programming in the real world. 1994.

[BK05] Tom Bova and Ted Krivoruchka. Reliable UDP protocol. *Available as*

IETF draft from http://www3. ietf. org/proceedings/99mar/ID/draft-ietf-sigtran-reliable-% udp-00. txt, accessed October, 2005.

[BK07] Urs Bischoff and Gerd Kortuem. A state-based programming model and system for wireless sensor networks. In *Pervasive Computing and Communications Workshops, 2007. PerCom Workshops' 07. Fifth Annual IEEE International Conference on*, pages 261–266. IEEE, 2007.

[BKK+11] Manfred Broy, Sascha Kirstan, Helmut Krcmar, Bernhard Schätz, and Jens Zimmermann. What is the benefit of a model-based design of embedded software systems in the car industry? *Emerging Technologies for the Evolution and Maintenance of Software Models. ICI*, 2011.

[Bon03] Boris Jan Bonfils. Adaptive and decentralized operator placement for in-network query processing. In *In IPSN*, pages 47–62, 2003.

[Boo] Paul Boocock. Jamda: The Java Model Driven Architecture, May 2003.

[Box03] Don Box. *Essential .NET: The common language runtime*, volume 1. Addison-Wesley Professional, 2003.

[BPSM+00] Tim Bray, Jean Paoli, C. Michael Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible Markup Language (XML) 1.0, W3C Recommendation. 2000. *The Role of Citizen Cards in e-Government*, 455, 2000.

[Bro00] Alan W Brown. *Large-scale, component-based development*, volume 1. Prentice Hall PTR Englewood Cliffs, 2000.

[BS06] Stephen Brown and Cormac J. Sreenan. Updating software in wireless sensor networks: A survey. *Dept. of Computer Science, National Univ. of Ireland, Maynooth, Tech. Rep*, 2006.

[BSB05] Nelly Bencomo, Thirunavukkarasu Sivaharan, and Gordon Blair. A Green Family: Generating Publish/Subscribe Middleware Configurations. 92:105, 2005.

[BSS+08] Christian Buckl, Stephan Sommer, Andreas Scholz, Alois Knoll, and Alfons Kemper. Generating a tailored middleware for wireless sensor network applications. *Sensor Networks, Ubiquitous, and Trustworthy Computing, International Conference on*, 0:162–169, 2008.

[BSS+09] Christian Buckl, Stephan Sommer, Andreas Scholz, Alois Knoll, Alfons Kemper, Jörg Heuer, and Anton Schmitt. Services to the field: An approach for resource constrained sensor/actor networks. In *The Fourth Workshop on Service Oriented Architectures in Converging Networked Environments (SOCNE 2009) - extended version*. IEEE, 2009.

[C+01] World Wide Web Consortium et al. Web services description language (wsdl) 1.1, 2001.

[CAS+08] Qing Cao, Tarek Abdelzaher, John Stankovic, Kamin Whitehouse, and Liqian Luo. Declarative tracepoints: A programmable and application independent debugging system for wireless sensor networks. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 85–98. ACM New York, NY, USA, 2008.

[CCG+07] Paolo Costa, Geoff Coulson, Richard Gold, Manish Lad, Cecilia Mascolo, Luca Mottola, Gian Pietro Picco, Thirunavukkarasu Sivaharan, Nirmal Weerasinghe, and Stefanos Zachariadis. The RUNES Middleware for Networked Embedded Systems and its Application in a Disaster Management Scenario. *Pervasive Computing and Communications, IEEE International Conference on*, 0:69–78, 2007.

[CCM+05] Paolo Costa, Geoff Coulson, Cecilia Mascolo, Gian Pietro Piccoand, and Stefanos 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.

[CE00] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative programming: methods, tools, and applications*. Addison Wesley, 2000.

[CGL+06] Geoff Coulson, Richard Gold, Manish Lad, Cecilia Mascolo, Luca Mottola, Gian Pietro Picco, and Stefanos Zachariadis. Dynamic Reconfiguration in the RUNES Middleware. In *Mobile Adhoc and Sensor Sysetems (MASS), 2006 IEEE International Conference on*, pages 574–577, 2006.

[CLRS09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms; 3rd ed.* MIT Press, Cambridge, MA, 2009.

[CMMP06] Paolo Costa, Luca Mottola, Amy L Murphy, and Gian Pietro Picco. TeenyLIME: transiently shared tuple space middleware for wireless sensor networks. In *Proceedings of the international workshop on Middleware for sensor networks*, pages 43–48. ACM Press New York, NY, USA, 2006.

[CMMP07] Paolo Costa, Luca Mottola, Amy L Murphy, and Gian Pietro Picco. Programming Wireless Sensor Networks with the TeenyLime Middleware. *LECTURE NOTES IN COMPUTER SCIENCE*, 4834:429, 2007.

[CNYM00] Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. Nonfunctional requirements. *Software Engineering*, 2000.

[Com] Component Synthesis with Model Integrated Computing (CoSMIC).

http://www.dre.vanderbilt.edu/cosmic/html/overview.shtml.

[CS08] Qiuhua Cao and John A. Stankovic. An in-field-maintenance framework for wireless sensor networks. *Lecture Notes in Computer Science*, 5067:457–468, 2008.

[DBK+07] Matthias Dyer, Jan Beutel, Thomas Kalt, Patrice Oehen, Lothar Thiele, Kevin Martin, and Philipp Blum. Deployment support network. In *Wireless Sensor Networks*, pages 195–211. Springer, 2007.

[dDCK+06] Scott de Deugd, Randy Carroll, Kevin E. Kelly, Bill Millett, and Jeffrey Ricker. Soda: Service-oriented device architecture. *IEEE Pervasive Computing*, 5(3):94–C3, 2006.

[DGV04] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. Contiki - A lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors*, volume 2004, 2004.

[DK08] Frank Leymann Dimka Karastoyanova. Service oriented architecture – overview of technologies and standards. *it – Information Technology 50*, 2/2008.

[DM09] Dan Driscoll and Antoine Mensch. Devices profile for web services version 1.1. *OASIS, Mai*, 2009.

[Dor06] Marco Dorigo. *Ant Colony Optimization and Swarm Intelligence: 5th International Workshop, ANTS 2006, Brussels, Belgium, September 4-7, 2006, Proceedings*, volume 4150. Springer-Verlag New York Incorporated, 2006.

[Dum76] Ernst Dummermuth. Programmable logic controller, March 2 1976. US Patent 3,942,158.

[Dun06] Adam Dunkels. The Contiki Operating System. *Web page. Visited Oct*, 24, 2006.

[Effa] Sven Efftinge. OpenArchitectureWare 4.1 Check Validation Language.

[Effb] Sven Efftinge. Xtend language reference, 4.1. *Obtenido de http://www.eclipse. org/gmt/oaw/doc/4.1/r25_extendReference. pdf*.

[Enc03] Vincent Encontre. Testing embedded systems: Do you have the guts for it. *IBM, November*, 2003.

[Eri96] Kelvin T. Erickson. Programmable logic controllers. *Potentials, IEEE*, 15(1):14–17, 1996.

[FLE06] Emad Felemban, Chang-Gun Lee, and Eylem Ekici. MMSPEED: multipath Multi-SPEED protocol for QoS guarantee of reliability and. Timeli-

ness in wireless sensor networks. *Mobile Computing, IEEE Transactions on*, 5(6):738–754, 2006.

[FSSF04] Francesco Furfari, Lorenzo Sommaruga, Claudia Soria, and Roberto Fresco. DomoML: The definition of a standard markup for interoperability of human home interactions. In *Proceedings of the 2nd European Union symposium on Ambient intelligence*, pages 41–44. ACM, 2004.

[FWDC+00] Victor Fay-Wolfe, Lisa C DiPippo, Gregory Cooper, R Johnson, Peter Kortmann, and Bhavani Thuraisingham. Real-time CORBA. *Parallel and Distributed Systems, IEEE Transactions on*, 11(10):1073–1089, 2000.

[GC11] James W. Grenning and Jacquelyn Carter. *Test-driven development for embedded C*. The pragmatic programmers. Pragmatic Bookshelf, Raleigh, N.C., 2011.

[Geh92] Narain H. Gehani. Exceptional C or C with Exceptions. *Software: Practice and Experience*, 22(10):827–848, 1992.

[GG+04] W3C Working Group, W3C Working Group, et al. Web services architecture. *W3C Note*, 2004.

[GLC07] David Gay, Philip Levis, and David Culler. Software design patterns for tinyos. *ACM Transactions on Embedded Computing Systems (TECS)*, 6(4):22, 2007.

[Gos00] James Gosling. *The Java language specification*. Addison-Wesley Professional, 2000.

[Gro07] Object Management Group. MOF Model to Text Transformation Language Language Final Adopted Specification. 2007.

[GSL+03] A.S. Gokhale, D.C. Schmidt, T. Lu, B. Natarajan, and N. Wang. Cosmic: An MDA generative tool for distributed real-time and embedded applications. *Middleware Workshops*, pages 300–306, 2003.

[HC02] F. Hunleth and R.K. Cytron. Footprint and feature management using aspect-oriented programming techniques. In *Proceedings of the joint conference on Languages, compilers and tools for embedded systems: software and compilers for embedded systems*, 2002.

[HC04] J.W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. *ACM Proc. on the 2nd international conference on Embedded Networked Sensor Systems*, 2004.

[HCG] F. Hunleth, R. Cytron, and C. Gill. Building customizable middleware using aspect oriented programming. In *The OOPSLA 2001 Workshop on*

Advanced Separation of Concerns in Object-Oriented Systems.

[HKM+05] Alfred Helmerich, Nora Koch, Luis Mandel, P Braun, P Dornbusch, A Gruler, P Keil, R Leisibach, J Romberg, B Schätz, et al. Study of worldwide trends and r&d programmes in embedded systems in view of maximising the impact of a technology platform in the area. *Final Report for the European Comission, Brussels, Belgium*, 2005.

[HKS+05a] Chih-Chieh Han, Ram Kumar, Roy Shea, Eddie Kohler, and Mani Srivastava. A dynamic operating system for sensor nodes. In *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 163–176. ACM, 2005.

[HKS+05b] Chih-Chieh Han, Ram Kumar, Roy Shea, Eddie Kohler, and Mani Srivastava. SOS: A dynamic operating system for sensor networks. In *Third International Conference on Mobile Systems, Applications, And Services (Mobisys)*, 2005.

[HLRV03] Erik Hatcher, Steve Loughran, Matthew Robinson, and Pavel Vorobiev. *Java development with Ant*. Manning, 2003.

[HM06] Salem Hadim and Nader Mohamed. Middleware: middleware challenges and approaches for wireless sensor networks. *IEEE DISTRIBUTED SYSTEMS ONLINE 1541-4922*, Vol. 7, No. 3, 2006.

[HS91] Samuel P Harbison and Guy L Steele. *C, a reference manual*. Prentice-Hall, Inc., 1991.

[Inc03] C.T. Inc. Mote in-network programming user reference, 2003.

[JABK08] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. ATL: A model transformation tool. *Science of Computer Programming*, 72(1):31–39, 2008.

[JBK06] Frédéric Jouault, Jean Bézivin, and Ivan Kurtev. TCS: A DSL for the specification of textual concrete syntaxes in model engineering. In *Proceedings of the 5th international conference on Generative programming and component engineering*, pages 249–254. ACM, 2006.

[Joh05] Rod Johnson. J2EE development frameworks. *Computer*, 38(1):107–110, 2005.

[KBDSS07] Stamatis Karnouskos, Oliver Baecker, Luciana Moreira Sá De Souza, and Patrik Spiess. 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*,

pages 293–300, Sept. 2007.

[KBK11] Gerd Kainz, Christian Buckl, and Alois Knoll. Automated model-to-metamodel transformations based on the concepts of deep instantiation. *Model Driven Engineering Languages and Systems*, pages 17–31, 2011.

[KBK12] Gerd Kainz, Christian Buckl, and Alois Knoll. A generic approach simplifying model-to-model transformation chains. *Model Driven Engineering Languages and Systems*, pages 579–594, 2012.

[KBSK10] Gerd Kainz, Christian Buckl, Stephan Sommer, and Alois Knoll. Model-to-metamodel transformation for the development of component-based systems. *Model Driven Engineering Languages and Systems*, pages 391–405, 2010.

[KG08] Amogh Kavimandan and Aniruddha Gokhale. Automated Middleware QoS Configuration Techniques using Model Transformations. In *Proceedings of the 14 th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2008), St. Louis, MO, USA*, 2008.

[Kop11] Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications*. Springer, 2011.

[Kot97] Vadim Kotov. *Systems of systems as communicating structures*. Hewlett Packard Laboratories, 1997.

[Koz92] John R Koza. *Genetic Programming: vol. 1, On the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[KPC08] Ki-Jeong Kwon, Choong-Bum Park, and Hoon Choi. DDSS: A Communication Middleware based on the DDS for Mobile and Pervasive Systems. In *Advanced Communication Technology, 2008. ICACT 2008. 10th International Conference on*, volume 2, pages 1364–1369. IEEE, 2008.

[KRU+03] Charles Keating, Ralph Rogers, Resit Unal, David Dryer, Andres Sousa-Poza, Robert Safford, William Peterson, and Ghaith Rabadi. System of systems engineering. *Engineering Management Review, IEEE*, 36(4):62–62, 2003.

[KSF94] DD Kandhlur, Kang G Shin, and Domenico Ferrari. Real-time communication in multihop networks. *Parallel and Distributed Systems, IEEE Transactions on*, 5(10):1044–1056, 1994.

[KSLB03] Gabor Karsai, Janos Sztipanovits, Akos Ledeczi, and Ted Bapty. Model-integrated development of embedded software. *Proceedings of the IEEE*, 91(1):145–164, 2003.

[KT03] James Keogh and By Kim Topley. *J2ME*. Mc Graw-Hill-Osborne, 2003.

[Kul11] Cyrill Kulka. Service Placement in (Wireless) Sensor Networks. Bachelor's thesis (Studienarbeit), Supervisor: Prof. Alois Knoll, Advisor: Stephan Sommer, Robotics and Embedded Systems, Technische Universität München, Germany, 2011.

[KW05] Sandeep S Kulkarni and Limin Wang. MNP: Multihop network reprogramming service for sensor networks. In *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pages 7–16. IEEE, 2005.

[LC02] Philip Levis and David Culler. Maté: A tiny virtual machine for sensor networks. In *ACM Sigplan Notices*, volume 37, pages 85–95. ACM, 2002.

[Lee08] Edward A. Lee. Cyber physical systems: Design challenges. In *International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, May 2008. Invited Paper.

[LM03] Ting Liu and Margaret Martonosi. Impala: A middleware system for managing autonomic, parallel sensor systems. In *Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 107–118. ACM New York, NY, USA, 2003.

[LMP+05] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, et al. TinyOS: An Operating System for Sensor Networks. *Ambient Intelligence*, pages 115–148, 2005.

[LPCS] Philip Alexander Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks.

[LSZM04] Ting Liu, Christopher M Sadler, Pei Zhang, and Margaret Martonosi. Implementing Software on Resource-Constrained Mobile Sensors: Experiences with Impala and ZebraNet. In *Proc. Second Intl. Conf. on Mobile Systems, Applications and Services*, pages 256–269, June 2004.

[LTGS03] Tao Lu, Emre Turkay, Aniruddha Gokhale, and Douglas C Schmidt. CoSMIC: An MDA tool suite for application deployment and configuration. In *Proceedings of the OOPSLA 2003 Workshop on Generative Techniques in the Context of Model Driven Architecture*, 2003.

[Man96] William HJ Manthorpe. The emerging joint system of systems: A systems engineering challenge and opportunity for APL. *Johns Hopkins APL Technical Digest*, 17(3):305, 1996.

[MCF03] Stephen J Mellor, Tony Clark, and Takao Futagami. Model-driven development. *IEEE software*, pages 14–18, 2003.

[MFHH05] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TinyDB: An Acquisitional Query Processing System for Sensor Networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[MGT+10] Ahmed Mekki, Mohamed Ghazel, Armand Toguyeni, et al. Time-constrained systems validation using MDA model transformation. A railway case study. In *Proceedings of the 8th International Conference of Modeling and Simulation (MOSIM'10)*. Citeseer, 2010.

[MHH02] Sam Madden, Joe Hellerstein, and Wei Hong. TinyDB: In-Network Query Processing in TinyOS. *Intel Research, IRB-TR-02-014, October*, 2002.

[MM03] Joaquin Miller and Jishnu Mukerji. Model Driven Architecture (MDA) 1.0. 1 Guide. Object Management Group. *Inc.(June 2003)*, 2003.

[MSUW02] Stephen J Mellor, Kendall Scott, Axel Uhl, and Dirk Weise. Model-driven architecture. *Advances in Object-Oriented Information Systems*, pages 233–239, 2002.

[MTSG10] Guido Moritz, Dirk Timmermann, Regina Stoll, and Frank Golatowski. Encoding and compression for the devices profile for web services. In *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, pages 514–519. IEEE, 2010.

[MZP+09] Guido Moritz, Elmar Zeeb, S Pruter, Frank Golatowski, Dirk Timmermann, and Regina Stoll. Devices profile for web services in wireless sensor networks: adaptations and enhancements. In *Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on*, pages 1–8. IEEE, 2009.

[NR69] Peter Naur and Brian Randell. Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO. 1969.

[NW04] Ryan Newton and Matt Welsh. Region streams: Functional macroprogramming for sensor networks. In *First International Workshop on Data Management for Sensor Networks (DMSN)*, 2004.

[Obj02] Object Management Group. *MetaObjectFacility (MOF) Specification*, 1.4 edition, Apr 2002.

[Obj07] Object Management Group. *OMG Unified Modelling Language Specification*, 2.1.2 edition, Nov 2007.

[Obj08] Object Management Group. Common Object Request Broker Architecture (CORBA) Specification, Version 3.1, Jan 2008.

[Obj10] Object Management Group. *OMG Meta Object Facility (MOF) Core Specification*, 2.4.1 edition, Apr 2010.

[ÖEL+06] Åke Östmark, Jens Eliasson, Per Lindgren, Aart van Halteren, and Lianne Meppelink. An infrastructure for service oriented sensor networks. *Journal of Computers*, 1, 2006.

[Old06] Jon Oldevik. MOFScript Eclipse plug-in: Metamodel-based code generation. In *Eclipse Technology Workshop (EtX) at ECOOP*, volume 2006, 2006.

[ONA04] Jon Oldevik, Tor Neple, and Jan Øyvind Aagedal. Model abstraction versus model to text transformation. *Computer Science at Kent*, page 188, 2004.

[ONG+05] Jon Oldevik, Tor Neple, Roy Grønmo, Jan Aagedal, and Arne-J Berre. Toward standardised model to text transformations. In *Model Driven Architecture–Foundations and Applications*, pages 239–253. Springer, 2005.

[Osb80] Adam Osborne. *An Introduction to Microcomputers (v. 1)*. McGraw-Hill, 1980.

[PCI+05] Gerardo Pardo-Castellote, Real-Time Innovations, et al. OMG data distribution service: Real-time publish/subscribe becomes a standard. *RTC Magazine*, 14, 2005.

[PLS+06] Peter Pietzuch, Jonathan Ledlie, Jeffrey Shneidman, Mema Roussopoulos, Matt Welsh, and Margo Seltzer. Network-aware operator placement for stream-processing systems. In *In ICDE*, 2006.

[PST+02] Adrian Perrig, Robert Szewczyk, JD Tygar, Victor Wen, and David E Culler. Spins: Security protocols for sensor networks. *Wireless networks*, 8(5):521–534, 2002.

[QCG+09] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, volume 3, 2009.

[RDT07] Bartolome Rubio, Manuel Diaz, and Jose M Troya. Programming approaches and challenges for wireless sensor networks. In *Systems and Networks Communications, 2007. ICSNC 2007. Second International Conference on*, pages 36–36. IEEE, 2007.

[RFC] RFC2768, Network Policy and Services:. http://doc.rz.ifi.lmu.de/rfc/rfc2768.html.

[RKM02] Kay Römer, Oliver Kasten, and Friedemann Mattern. Middleware challenges for wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):59–61, 2002.

[RWMX06] Injong Rhee, Ajit Warrier, Jeongki Min, and Lisong Xu. DRAND: distributed randomized TDMA scheduling for wireless ad-hoc networks. In *Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, pages 190–201. ACM, 2006.

[SBB04] Tilman Seifert, Gerd Beneken, and Niko Baehr. Engineering long-lived applications using mda. In *IASTED Conf. on Software Engineering and Applications*, pages 241–246, 2004.

[SBEJ04] C. Snook, M. Butler, A. Edmunds, and I. Johnson. Rigorous development of reusable, domain-specific components, for complex applications. 2004.

[SBK09] Stephan Sommer, Christian Buckl, and Alois Knoll. Developing service oriented sensor/actuator networks using a tailored middleware. In *6th International Conference on Information Technology : New Generations (ITNG 2009)*. IEEE, 2009.

[SBMP08] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: eclipse modeling framework*. Addison-Wesley Professional, 2008.

[SBS+09] Andreas Scholz, Christian Buckl, Stephan Sommer, Alfons Kemper, Alois Knoll an d Jörg Heuer, and Anton Schmitt. eSOA - service oriented architectures adapted for embedded networks. In *Proceedings of the 7th International Conference on Industrial Informatics)*, June 2009.

[SC01] Andrew P Sage and Christopher D Cuppan. On the systems engineering and management of systems of systems and federations of systems. *Information-Knowledge-Systems Management*, 2(4):325–345, 2001.

[SCB+13] Stephan Sommer, Alexander Camek, Klaus Becker, Christian Buckl, Andreas Zirkler, Ludger Fiege, Michael Armbruster, Gernot Spiegelberg, and Alois Knoll. Race: A centralized platform computer based architecture for automotive applications. In *Vehicular Electronics Conference (VEC) and the International Electric Vehicle Conference (IEVC) (VEC/IEVC 2013)*. IEEE, October 2013.

[Sch11] Andreas Scholz. *Adaptive Data Processing in Embedded Networks*. PhD thesis, München, Technische Universität München, Diss., 2011, 2011.

[Sem00a] Freescale Semiconductor. SPI Block Guide. 21, 2000.

[Sem00b] Philips Semiconductors. THE I2C-BUS SPECIFICATION. Technical re-

port, 2000.

[SGB+13] Stephan Sommer, Michael Geisinger, Christian Buckl, Gerd Bauer, and Alois Knoll. Reconfigurable industrial process monitoring using the CHROMOSOME middleware. In *The Fifth International Workshop on Adaptive and Reconfigurable Embedded Systems (APRES 2013)*. ACM, April 2013.

[Sha01] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[SHE03] T. Stathopoulos, J. Heidemann, and D. Estrin. A remote code update mechanism for wireless sensor networks. *University of California, LA, Tech. Rep. CENS-TR-30*, 2003.

[Sie00] Jon Siegel. *CORBA 3 fundamentals and programming*, volume 2. John Wiley & Sons Chichester, 2000.

[SK08] John Schneider and Takuki Kamiya. Efficient XML interchange (EXI) format 1.0. *W3C Working Draft*, 19, 2008.

[SN93] Matlab Simulink and MA Natick. The mathworks. *Inc., Natick, MA*, 1993.

[SSB+09] Stephan Sommer, Andreas Scholz, Christian Buckl, Alfons Kemper, Alois Knoll, Jörg Heuer, and Anton Schmitt. Towards the internet of things: Integration of web services and field level devices. In *International Workshop on the Future Internet of Things and Services Embedded Web Services for Pervasive Devices (at FITS 2009)*, 2009.

[SSB+10] Andreas Scholz, Stephan Sommer, Christian Buckl, Gerd Kainz, Alfons Kemper, Alois Knoll, Jörg Heuer, and Anton Schmitt. Towards an adaptive execution of applications in heterogeneous embedded networks. In *Software Engineering for Sensor Network Applications (SESENA 2010)*. ACM/IEEE, 2010.

[SSBG03] S Shankar Sastry, Janos Sztipanovits, Ruzena Bajcsy, and Helen Gill. Scanning the issue-special issue on modeling and design of embedded software. *Proceedings of the IEEE*, 91(1):3–10, 2003.

[SSH+07] Ferat Sahin, Prasanna Sridhar, Ben Horan, Vikraman Raghavan, and Mo Jamshidi. System of systems approach to threat detection and integration of heterogeneous independently operable systems. In *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, pages 1376–1381. IEEE, 2007.

[Sta08] John A. Stankovic. When sensor and actuator networks cover the world.

ETRI journal, 30(5):627–633, 2008.

[STV04] Chris Salzmann, Martin Thiede, and Markus Völter. Model-based middleware for embedded systems. *GI Jahrestagung (2)*, 51:3–7, 2004.

[SvVB02] Thorsten Sturm, Jesco von Voss, and Marko Boger. Generating code from uml with velocity templates. *«UML» 2002—The Unified Modeling Language*, pages 379–386, 2002.

[TG06] François Terrier and Sébastien Gérard. Mde benefits for distributed, real time and embedded systems. *From Model-Driven Design to Resource Management for Distributed Embedded Systems*, pages 15–24, 2006.

[Tin] TinyOS. http://www.tinyos.net/.

[TZL08] Run-hua TANG, Lu ZHANG, and Wai-xi LIU. Design and development of mobile integration inquiry system based on j2me and j2ee technologies [j]. *Science Technology and Engineering*, 1:022, 2008.

[UDD] UDDI. http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm.

[VG07] Markus Voelter and Iris Groher. Handling variability in model transformations and generators. In *7th OOPSLA Workshop on Domain-Specific Modeling*, 2007.

[VHJG95] John Vlissides, R Helm, R Johnson, and E Gamma. Design patterns: Elements of reusable object-oriented software. *Reading: Addison-Wesley*, 1995.

[Vis04] Eelco Visser. Program transformation with Stratego/XT. *Domain-Specific Program Generation*, pages 315–349, 2004.

[VLA87] Peter JM Van Laarhoven and Emile HL Aarts. *Simulated annealing*. Springer, 1987.

[VSB+13] Markus Völter, Thomas Stahl, Jorn Bettin, Arno Haase, and Simon Helsen. *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2013.

[VSK05] Markus Voelter, Christian Salzmann, and Michael Kircher. *Model Driven Software Development in the Context of Embedded Component Infrastructures*, pages 143–163. 2005.

[W3C] Web Services Eventing (WS-Eventing). http://www.w3.org/Submission/WS-Eventing/.

[Wan06] Roy Want. An introduction to RFID technology. *Pervasive Computing, IEEE*, 5(1):25–33, 2006.

[Wig01] Ulf Wiger. Four-fold Increase in Productivity and Quality-Industrial-

Strength Functional Programming in Telecom-Class Products. *Ericsson Telecom*, 2001.

[Wik] Wikipedia Image: OMG Object Request Broker. http://en.wikipedia.org/wiki/file:orb.svg.

[WSBC04] Kamin Whitehouse, Cory Sharp, Eric Brewer, and David Culler. Hood: A neighborhood abstraction for sensor networks. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 99–110. ACM, 2004.

[YG02] Yong Yao and Johannes Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18, 2002.

[ZMTG10] Elmar Zeeb, Guido Moritz, Dirk Timmermann, and Frank Golatowski. WS4D: Toolkits for networked embedded systems based on the devices profile for web services. In *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, pages 1–8. IEEE, 2010.

[ZWJ+07] Di Zheng, Jun Wang, Yan Jia, Wei-Hong Han, and Peng Zou. Deployment of context-aware component-based applications based on middleware. In *Ubiquitous Intelligence and Computing 4th International Conference, UIC 2007, Hong Kong, China, July 11-13, 2007: Proceedings*, 2007.