Lightweight Policy-Based Management of Quality-Assured, Device-Based Service Systems

Oliver Dohndorf, Jan Krüger
and Heiko Krumm
TU Dortmund University
(dohndorf,krueger,krumm)@ls4.cs.tu-dortmund.de

Christoph Fiehe, Anna Litvina, Ingo Lück
and Franz-Josef Stewing
MATERNA Information & Communications
(cfiehe,alitvina,ilueck,fstewing)@materna.de

Abstract

Intelligent connected devices become a more vital part of our lives. In contrast to prior years, today embedded systems and devices are loosely coupled and cooperate with each other according to changing objectives. Following the service-oriented architectural style, the approach of so-called device-based service systems emerges. The challenge is to build and manage these systems in a reliable, secure, and efficient manner. New and innovative solutions are needed regarding the problems of interoperability, maintainability, as well as automated configuration and management. In our work, these requirements are met by means of policies which are modeled at different levels of abstraction in a design phase and efficiently enforced at runtime by a lightweight management system.

1 Introduction

Transferring the conception of service orientation to embedded systems and devices leads to device-based service systems which advantageously can be applied in various domains, for example home entertainment, logistics or healthcare. But due to the usage of heterogeneous hard- and software the realization of flexible distributed service systems is a major challenge. In the device world a variety of different communication protocols and interfaces are used, while upcoming applications demand for openness and flexible cooperation capabilities. Thus, standards gain in significance. The systems are in mutual interaction with each other and with their environment. On the one hand, a flexible and adaptive system configuration is needed. On the other hand, a predictable and reliable system behavior is required. Therefore, the systems have to adapt to changing conditions and requirements and have to correct malfunctions and failures at runtime. In the healthcare domain, for example, the patient's medical needs and preferences as well as his health state change over time. Accordingly, a comprising management system is needed to provide the necessary adaptation and fault tolerance mechanisms. Nevertheless, the management system needs to be light-weighted, since many devices have only limited resources and computational power.

We present an appropriate approach for the management of device-based service systems. Technically, we refer to the *OSGi Framework* [9] defined by the OSGi Alliance. The OSGi Framework provides the desired modularity through software components, called *bundles*, that offer their functionality to other bundles in the form of services. Conceptionally, our approach relies on the model-based management paradigm which combines management policies and policy hierarchies with a layered system model [4]. Management policies build an additional control level above the programming code. Policies "govern the choices in behaviour of a system" [2], thus, performing configuration, adaptation and correction functions. According to the approach of policy hierarchies [12], a distinction is made between abstract high-level policies and technical low-level policies. At runtime, only the low-level policies are present within the system and can be enforced efficiently. At design time, the high-level policies are defined with comprehensive tool support. The policy tool allows the interactive graphical modeling of systems and abstract policies, the policy analysis and the automated refinement of the abstract policies into the technical low-level policies. Moreover, it deploys the low-level policies. They are translated to and deployed as executable Java bytecode which directly checks status conditions, computes appropriate configuration changes, and performs them. This paper concentrates on the new application domain "device-based medical systems" and the newly developed lightweight runtime management system.

In the sequel, Section 2 gives a short overview of existing system management concepts. Section 3 introduces the related work. The system structure forming the basis of our approach is described in Section 4. Sections 5 and 6 depict the policy-based management during the design and runtime phase. We demonstrate the applicability of our solution in Section 7. Finally, Section 8 concludes the paper.

2 Technical Management

The technical management of systems and networks is mainly influenced by the following standards.

Open System Interconnection Management The first standardizations were undertaken by the *International Organization for Standardization* (ISO) during the definition of the OSI Reference Model [5]. It specifies five functional areas for the system management: fault, configuration, accounting, performance and security management. *Management objects* provide an OSI management view of a managed resource. They have attributes and corresponding management operations. The management objects of a managed resource are part of its *Management Information Base* (MIB) which structures the objects and provides a uniform view of the management-relevant information.

Web-Based Enterprise Management WBEM [1] is a platform and resource independent standard of the *Distributed Management Task Force* (DMTF) which defines a common model for the management of distributed systems. A part of WBEM is the *Common Information Model* (CIM), an object-oriented information model for a uniform description of management information and functions. It allows the representation of physical and logical objects. WBEM follows the object-oriented approach and represents all manageable resources by objects which are aggregated to classes.

Web Services Distributed Management WSDM [6] was developed by the *Organization for Advancement of Structured Information Standards* (OASIS) for the management of service-oriented architectures. WSDM is a platform and manufacturer independent approach which allows the definition of management interfaces for resources. The WSDM specifications are based on Web services and address primarily two domains: *Management Using Web Services* (MUWS) specifies the management of resources by means of Web services; *Management of Web Services* (MOWS) defines a model for the management of Web services.

3 Related Work

The *Service Component Architecture* (SCA) [7] specifies a framework for development of applications and systems according to the SOA principles. The smallest unit in the SCA is a *component*. A component implements a specific business logic and provides access to its functions via external interfaces. Their description depends on the technology used for realizing the service. If a component depends on services of other components, these services must be defined explicitly by means of so-called *references*. *Bindings* describe the protocols which are required for the data communication between a reference and a service. The reference is bound logically to the service through a *wire*. Furthermore, component properties can be defined. They are read during the initialization process and assigned correspondingly. A set of components can be aggregated to a *composite* and a set of composites to a domain. This allows the grouping of components to logical units which are required in order to provide a certain business-function.

The *WS-Policy* [11] specification defines how Web services can advertise their requirements, capabilities, and general characteristics concerning security, quality of service, and other non-functional aspects. Any entity in a Web services based system can expose a policy in order to convey its operation conditions. Given the policy assertions of a service, a client is able to decide whether to use the service or not. Since a policy is an unordered collection of alternatives each providing a valid configuration for interaction with the service, the client is free to choose any alternative. The service, in contrast, is bound to a single configuration for the interaction. Preferences between alternatives in a given context as well as their value or suitability are beyond the scope of the WS-Policy specification.

4 System Structure

A system consists of a set of components. In our approach, software components are of particular interest. Furthermore, associations of different types (e.g., ”located on”, ”consists of”, ”depends on”, ”belongs to”) exist between components. Dynamic associations between clients and services are called *bindings*.

4.1 Software Components

A software component implements domain-specific application logic and offers access to its functionality in the form of services and is realized as an OSGi bundle. The internal structure of a component is depicted in Figure 1. The component developer implements the actual application functions and provides *(1)* or uses *(2)* an unlimited number of services according to the SOA principles. In order to be manageable, the component must specify the state space which is visible to the management. For this reason, the component developer defines a MIB scheme and declares a corresponding set of *management variables*. The management variables can be decomposed into two groups: *status variables* and *configuration variables*. Status variables describe the management-relevant state of the component, whereas configuration variables affect the component's behavior. Status variables are written by the application logic and read by the management, configuration variables are written by the management and read by the application logic.

The management variables of a component are only accessible via services of its *component manager*. The component manager itself is a component in the above sense, responsible for the management of its corresponding components. Each component is associated with exactly one component manager, whereas a component manager can be responsible

for several components. The component manager provides some management services which can be used within the application logic: a *configuration service* offering access to the management variables of the component, a *policy service* allowing to make decisions based on policy evaluations with respect to the current system state, and a *binding service* allowing to establish and release bindings to other components.

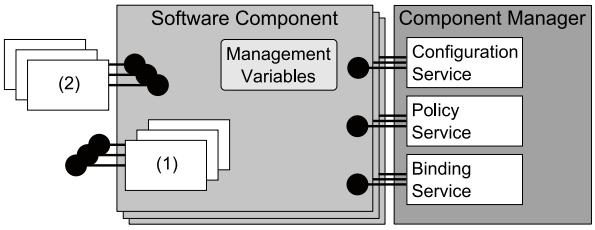

Figure 1: Component Structure

4.2 Bindings

The association between two components, one acting as service requester (client), the other as service provider (server) is called binding. A binding has a status (e.g., requested, established, broken). A binding is established between so-called *endpoints*. An endpoint has properties (e.g., the address, security requirements, protocol parameters). The handling of bindings is unified and supported by the binding service.

Bindings are dynamic. They are established at runtime and allow, depending on their properties, a transparent substitution. That supports a flexible and fault tolerant configuration of systems. Applications, however, usually need reliable operation phases. In such a phase, a client application needs stable bindings with a set of services which are used in combination in order to support the current task. We manage the bindings of these services according to two principles: *ensemble* [10] and *lease* [3]. The set of services needed for one task is called ensemble. Ensembles support the comfortable implementation of logical service groups. The services of an ensemble are allocated as a whole. The binding is performed atomically using a two-phase allocation protocol. The ensemble allocation is based on leases, where a lease is a time-limited contract between a client and service supporting stability as well as flexibility. During the time period of a lease, the contract can only be released by a client or due to exceptional conditions. After expiration of a lease, services may be reconfigured and subjects of change.

5 Model-Based Management

We adopt the concept of model-based management, developed in [4]. It allows the automated step-by-step policy refinement based on a model which consists of three hierarchically organized layers. Each layer models the system at a different level of abstraction. The policies attached to each layer apply to the system elements on this layer.

Model-based management can be divided into two phases. In the *runtime phase* technical low-level policies are enforced efficiently by the runtime management system. In the *design phase*, the policy tool is used. The system is modeled interactively on the three layers. The highest layer is complemented by the definition of the abstract high-level policies. The tool checks consistency criteria and automatically refines the abstract policies to technical low-level policies. The additional information needed for policy refinement is supplied by the layered system model.

5.1 Model and Abstraction Layers

The system is modeled on the three layers by self-contained and independent models, i.e., there are three models of the same system differing in the degree of abstraction. Moreover, refinement relations exist associating elements of adjacent models. The three layers are:

- *Use Cases & Aspects*: On the top layer, the technical details are in the background. The existing use cases with their aspects relating to non-functional requirements and quality criteria of the system are modeled.
- *Services & Domains*: On the middle layer, the system is represented from a service-oriented point of view comprising clients, services and dependency relations. The elements are assigned to so-called domains.
- *Components & Devices*: On the bottom layer, the actual software components and devices of the system are represented. These are the elements which exist at runtime and are subject to the enforcement of the technical low-level policies.

5.2 Policy Refinement

The policy refinement is a step-by-step derivation of technical low-level policies from abstract high-level policies. The derivation is carried out according to the given refinement relations. A refinement relation associates a model element with the corresponding elements of the next lower layer. For example, an abstract function modeled on the ”Use Case & Aspects” level has a refinement relation to those services of the ”Services & Domains” level that are necessary to provide this abstract function. The level of abstraction decreases with each of the two refinement steps, so that finally the low-level policies refer only to management variables. For instance, an abstract security requirement for high confidentiality is mapped to the protection requirements of the corresponding services resulting in the specific binding requirements for cryptographic methods and minimal key length in the low-level policies. The relation between

the model layers and the refinement from *declarative* to *imperative policies* is depicted in Figure 2. Formally, policy refinement is defined as a function from the domain of "Use Cases & Aspects" to the range of "Services & Domains" and from "Services & Domains" to "Components & Devices", respectively. The function has homomorphic properties and preserves the refinement relations.

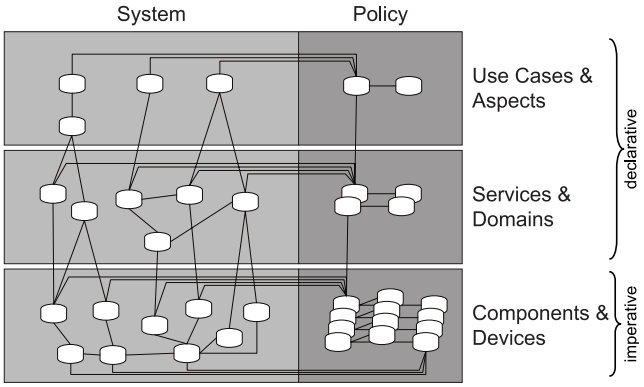

Figure 2: Levels of Abstraction and Policy Refinement

The policy tool used for modeling and refinement is called *MoBaSeC* (Model Based Service Configuration) [4]. It is developed in cooperation between TU Dortmund University and MATERNA.

6 Policy-Based Runtime Management

The runtime management is carried out by the component managers which enforce the derived low-level policies. The component managers are in charge of monitoring, controlling, and configuring their corresponding components. The allocation of the components to their manager as well as the assignment of configurations and low-level policies to the components is planned in the design phase by MoBaSeC.

6.1 Policy Types

Low-level policies are defined on management variables, event parameters, operations, and constants and have the form of conditions, expressions and variable assignments. At runtime, they exist as efficiently executable byte code. The corresponding classes are generated by MoBaSeC through a backend function. We distinguish between the following policy types:

Policy Expression A policy expression is an expression on management variables, constants, and operations. It is evaluated on demand and allows to estimate the current system state. According to the result, a component can react by adapting its program flow. For instance, an application requests the evaluation of the system stability which depends on the current state of its components.

Policy Condition A policy condition is a special case of a policy expression. The return data type is restricted to Boolean. For example, an application requests the decision, whether to stop the operation or not in dependence of the current system state.

Policy Rule A policy rule represents a simple event-condition-action rule specifying the actions to be performed when a certain event occurs. The actions are restricted to variable assignments. Thus, the reactive management can be performed in order to achieve a reliable and adaptive system behavior. For example, the operating mode of a component can be adjusted as a reaction to an increasing resource consumption.

Binding Requirements A binding requirement defines a non-functional requirement for a client-server connection. It is defined as a Boolean expression on management variables. For its establishment, the binding requirements of the client's and server's endpoints are intersected. The result specifies the configuration space of this connection. For instance, the binding requirements of the client's endpoint determine that outgoing communication channels must be encrypted with a key length of 128 or 256 bit. The binding requirements of the server's endpoint prescribe the encryption with a key length of 256 bit. According to the intersection result the encryption key length of the communication channel is 256 bit.

6.2 Binding Management

The binding management is responsible for establishing, monitoring, and releasing ensembles. The establishment of a binding comprises four phases: in the first phase, functional compatible binding partners are searched, in the second phase, partners are calculated which are suitable in compliance with the predefined binding requirements, in the third phase, the quality criteria are compared and finally in the fourth phase, the actual binding is established resulting in the configuration of both endpoints.

For the reason of quality assurance, the binding management monitors all established bindings. If the agreed quality criteria are violated, the binding management performs corrective actions. Such an action can be a transparent substitution of the communication partner. If a binding is released, the binding management notifies the service, which in turn can release allocated resources.

When a client requests an ensemble, the binding management establishes the necessary bindings to the ensemble members. During operation, if any connection violates the agreed quality criteria, the binding management tries to reconfigure the ensemble. Failure of the reconfiguration process results in an exceptional condition, leading to the premature release of bindings.

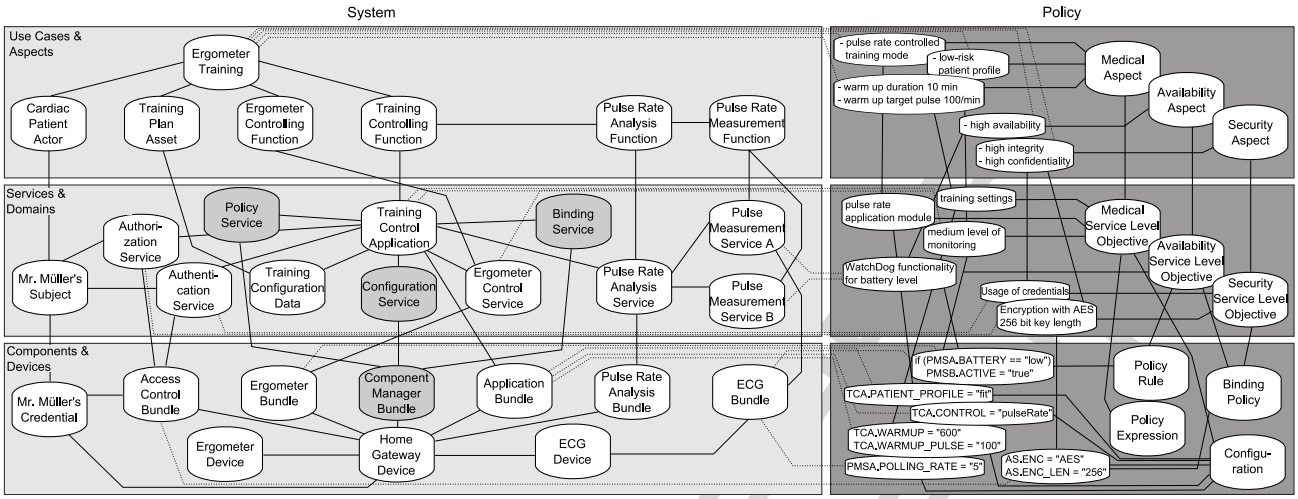

Figure 3: Application Example

7 Application Example

We demonstrate the applicability of our approach by means of a simple example taken from the home healthcare domain. In the scenario, a cardiac patient goes through a rehabilitation training program using an ergometer at home. For that purpose, the hospital supplies the patient with the necessary equipment including an ergometer with a display, medical sensors and a home gateway. It connects the home equipment with the hospital servers. The home gateway and intelligent devices serve as execution platforms for the software components. The training is controlled locally by the home system. On demand the patient can be telemedically monitored by a medical supervisor at the hospital.

The ergometer target load and the training parameters are adjusted in accordance with the patient's training plan. The assistant software system monitors the patient's vital signs, provides him with auxiliary information, and controls the training process. Besides the compliance with all functional requirements, a set of non-functional requirements have to be met in order to assure the predefined quality criteria of the system. These criteria relate to different aspects of the systems: costs, energy, security, reliability, and application domain's specifics. For instance, because the system handles sensible information like the patient's medical and personal data, it must be guaranteed that all protection objectives like confidentiality and integrity are met. It has also to be considered that the user of the system is a person with special needs: elderly and diseased. For this reason, specific usability requirements for the user interface exist concerning ergonomics and user friendliness. Moreover, the costs caused by the system have to be kept under control. The user's preferences about the expenses he is willing to bear have to be taken into account.

Figure 3 shows an excerpt from the system as it is planned and modeled during the design phase. The left column comprises the system itself, the policies are depicted in the right column. According to the scenario, the main use case is the *Ergometer Training* which is presented on the top layer "Use Cases & Aspects" and conducted according to the *Training Plan*. The layer also includes the central actor of the use case, the *Cardiac Patient*. The primary functions that are carried out within this use case are *Ergometer Controlling* and *Training Controlling*. Furthermore, some additional functions like *Pulse Rate Analysis* and *Pulse Rate Measurement* are involved. Among the high-level constraints which are provided for this self-contained model, there are abstract requirements concerning aspects like security, availability, and application domain. For example, the system must ensure high integrity, confidentiality, and availability. Regarding the application domain issue, there are also some medical specific use case requirements, for instance, the training plan provided by the supervisor. Besides the patient's personal profile indicating whether the patient belongs to a certain risk group or not, the training plan includes all parameter settings for conducting the training.

On the layer "Services & Domains", all services and client applications are modeled which are necessary to provide the functions defined in the upper layer. The *Training Control Application* controls the training's process by using an ensemble of the services *Ergometer Control* and *Pulse Rate Analysis* as well as the abstract resource *Training Configuration Data*. The availability issue is addressed by introducing two services for carrying out the pulse measurement. The patient's identity is represented in the model by his subject. It is used by the *Authentication Service* and the *Authorization Service* which are required in order to meet the security and confidentiality requirements. The

domain-specific requirements are reflected in the training and monitoring parameter settings which are used to configure the application and the corresponding services. The management-specific services are represented by the *Configuration Service*, the *Binding Service*, and the *Policy Service*.

The bottom layer "Components & Domains" contains all software components, resources, and devices that exist at runtime. The devices are the *Ergometer Device*, the *ECG Device*, and the *Home Gateway*. The software components in the form of OSGi bundles are the *Ergometer Bundle*, the *Application Bundle*, the *Pulse Rate Analysis Bundle*, and the *ECG Bundle* all running on their corresponding devices. The ECG bundle encapsulates the ECG device and provides a pulse measurement service. Another pulse measurement service is hosted by the Ergometer bundle encapsulating the ergometer. If one of the services fails, it can be substituted by the other one. The subject-related credential is stored on the home gateway and is used by the *Access Control Bundle*. The runtime management is performed by the *Component Manager Bundle* which hosts the management-specific services and is executed on the home gateway.

In this example, the high-level policies determining the system behavior abstractly are the requirements for high availability, high integrity, and high confidentiality. On the next lower level, the availability requirements are met by representing a service type through several service instances, so that in case of any failure, for example, if the battery charge is low, a service can be transparently substituted by another one. This fact involves the usage of a watch dog that detects these exceptional situations. On the lowest level, the availability requirement corresponds directly to a set of low-level policy rules determining that if the *Pulse Measurement Service A* reports a low battery charge, the *Pulse Measurement Service B* has to be activated immediately by setting the appropriate configuration variables. In order to meet the security requirements "high integrity" and "high confidentiality", credentials and specific encryption algorithms with corresponding key lengths are configured. The medical aspect is described by the patient's training plan which contains the training mode, the target ergometer load, and other medical data. These specifications determine the concrete values of the configuration variables for thresholds, training settings, polling rates, and the level of monitoring. The runtime management system comprises 46 kB bytecode for the component management bundle, including 12 kB bytecode representing the low-level policies.

8 Summary and Outlook

In this paper, we have introduced our approach of policy-based management developed within the OSAmI[1] project [8]. It divides management into a design and a runtime phase. The design phase utilizes a system model representing components and their associations on three hierarchically organized abstraction layers. In order to achieve a reliable and adaptive system behavior, the management resorts to policies. During design, they are specified as abstract high-level policies and allow the concise definition of scopes, preferences, and weights. The policy tool MoBaSeC automatically refines them to technical low-level policies which are defined directly in the context of runtime management using management variables, event parameters, operations, and constants. The low-level policies can be divided into policy expressions, conditions, rules, and binding requirements.

Within the OSAmI project, a more sophisticated e-Health demonstrator is under development. The introduced management approach will be validated on the basis of this demonstrator, too. Moreover, it is planned to apply the management approach to an industrial automation scenario.

[1] Funded by the German Federal Ministry of Education and Research

References

[1] Distributed Management Task Force. *Web-Based Enterprise Management (WBEM)*, 2009. http://www.dmtf.org/standards/wbem.

[2] N. Dulay, E. Lupu, M. Sloman, and N. Damianou. A Policy Deployment Model for the Ponder Language. In *Proc. IEEE/IFIP International Symposium on Integrated Network Management (IM2001)*, pages 14–18, Seattle, May 2001.

[3] C. G. Gray and D. R. Cheriton. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In *Proc. of the 12th ACM Symposium on Operating System Principles (SOSP '89)*, pages 202–210. ACM Press, 1989.

[4] S. Illner, H. Krumm, I. Lück, et al. Model-based management of embedded service systems – an applied approach. In *Proc. of the 20th Int. Conf. on Advanced Information Networking and Applications (AINA '06)*, pages 519–523. IEEE Computer Society, 2006.

[5] ISO. ISO/IEC 7498-4: Information Processing Systems – Open Systems Interconnection – Basic Reference Model – Part 4: Management Framework, 1989.

[6] OASIS Web Services Distributed Management (WSDM) TC. *WSDM 1.1 OASIS Standard Specifications*, 2006.

[7] Open SOA Collaboration (OSOA). *SCA Service Component Architecture: Assembly Model Specification*, 2007.

[8] OSAMI-D Consortium. OSAmI: Open Source AMbient Intelligence. http://www.osami-commons.org, 2009.

[9] OSGi Alliance. *OSGi Service Platform Core Specification – Release 4, Version 4.1*, 2007.

[10] A. Pohl, H. Krumm, F. Holland, et al. Service-orientation and flexible service binding in distributed automation and control systems. In *Proc. of the 22nd Int. Conf. on Advanced Information Networking and Applications (AINA'08)*, pages 1393–1398. IEEE Computer Society, 2008.

[11] A. S. Vedamuthu, D. Orchard, F. Hirsch, et al. *Web Services Policy 1.5 - Framework*, September 2007.

[12] R. Wies. Policies in network and system management – formal definition and architecture. *Journal of Network and System Management*, 2(1):63–83, 1994.