POLICY-BASED MANAGEMENT OF MEDICAL DEVICES AND APPLICATIONS

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

ANNA LITVINA

Dortmund

2018

Tag der mündlichen Prüfung:	24.09.2019
Dekan:	Prof. Dr.-Ing. Gernot A. Fink
Gutachter:	Prof. Dr. Heiko Krumm
	Prof. Dr.-Ing. Andreas Hein

Contents

Chapter 1

Introduction

The significance of the medical and health care sector continues to grow nowadays. The German Federal Statistical Office reported that expenditure on health in Germany amounted to 344.2 billion euros or 4,213 euros per inhabitant in 2015. This figure represented 11.3% of the GDP (gross domestic product) and for the fourth year in a row increased more rapidly than the GDP itself.[1]

At the same time, the IT pervades all the aspects of our lives. This ubiquity of the technology also spreads into the medical field. Thereby, the technology's advance changes the medical sector. In his work, the practicing cardiologist Topol states [Top13]:

> "This is a new era of medicine, in which each person can be near fully defined at the individual level, instead of how we (have previously) practiced medicine at a population level. We are each unique human beings, but until now there was no way to determine a relevant metric like blood pressure around the clock while a person is sleeping, or at work, or in the midst of an emotional upheaval. This represents the next frontier of the digital revolution, finally getting to the most important but heretofore insulated domain: preserving our health."

Thus, we face a turnaround in the medical care from the treatment of the population to the treatment of individuals nowadays. The pervasiveness of the IT has the corresponding power to accomplish this and leads to a more specific, personalized medical care, tailored to the individual requirements. The application field makes high demands on the data and functional safety as well as the timeliness of the used sophisticated devices and systems. It dictates an adaptable and flexible system behavior, which always stays strictly predefined and is precisely predictable at any time. Another important aspect to consider is that the common end user is not necessarily technically oriented and requires often additional assistance and supervision. In order to support these characteristics and qualities of the systems, sophisticated automated technical management is needed.

This work presents an advanced form of system management which combines the established approach of policy-based management and the innovative model-based management technique and applies it within the medical application field. The thesis statement is that the introduced management approach can support the development and dependable behavior of medical devices and systems.

The approach includes two main phases: design and runtime. During the design phase the managed system is modeled on three abstraction levels with different degree of technical precision: from the abstract to the technical one. The process is supported by the modeling MoBaSeC tool which assists the user with its advanced visualizing drap-and-drop functions. Afterward, the user defines the specific requirements and constraints. The backend functions of the tool refine the defined requirements and constraints into the management policies, the instrumentals of the management system.

[1] https://www.destatis.de/EN/FactsFigures/SocietyState/Health/HealthExpenditure/HealthExpenditure.html (January 2018)

During the runtime phase the lightweight autonomic management system accomplishes the control loop by enforcing the generated management policies. Due to the formulation of the policies on the management variables, the management stays lightweight: the triggering events are changes of the status variable values, the conditions are expressions on the status variable values, the management actions are limited to value assignments of the configuration variables. This management infrastructure allows a low impact of the management system on the managed system.

The two main contributions of the work can be marked out:

1. A general ***metamodel structure*** has been proposed. It forms the basis for the developed management approach and is used for the tool-supported development of system models during the design phase.

   This general metamodel is specialized for the medical application domain. The concretized metamodel integrates the domain knowledge and allows to include the domain-specific constraints and requirements into the modeling process.

2. A collection of ***policy derivation patterns*** has been elaborated. The patterns include evaluation, control and refinement patterns which support the automated process of the refinement of abstract constraints and requirements into the concrete technical policies and configurations used during the runtime by the management.

The thesis covers the work conducted in cooperation between TU Dortmund and MATERNA GmbH. The working group on the automated technical management has taken part in a row of ITEA research projects, such as OSAMI[2], BaaS[3] and MEDOLUTION[4]. A series of joint articles and conference proceedings has been published. The author's contribution focusing the policy refinement has been partly introduced at the IEEE International Symposiums on Policies for Distributed Systems and Networks (POLICY) in 2010 and 2011 by two conference papers and a demonstrator: "Policy-Based Management for Resource-Constrained Devices and Systems" [DKK+10b] and "Tool-Supported Refinement of High-Level Requirements and Constraints into Low-Level Policies" [DKK+11b]. The author's work on the technical policy-based runtime management have been integrated into the conference papers published in the scope of the proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA) - "Lightweight Policy-Based Management of Quality-Assured, Device-Based Service Systems" [DKK+10a] and of the 16th IEEE Conference on Emerging Technologies & Factory Automation (ETFA) - "Adaptive and Reliable Binding in Ambient Service Systems" [DKK+11a] presented at the Workshops on Service Oriented Architectures in Converging Networked Environments (SOCNE) in 2010 and 2011.

The thesis is structured as follows. The related work on the policy- and model-based management of medical systems is presented in Chapter 2. Chapter 3 sets the scene for the presented approach and explains the application domain specifics. Chapters 4, 5 and 6 provide a basis for the work with the fundamentals of the technical management and the policy- and model-based approaches. The runtime management system is introduced in Chapter 7. The core of the work is depicted in Chapter 8: the main modeling approach including the metamodel applicable to the medical domain is introduced. Chapter 9 presents the elaborated policy derivation patterns used within the refinement process. The application of the approach to the cardiological use case is described in Chapter 10. The work is evaluated in Chapter 11. Chapter 12 concludes the work.

[2]OSAMI (Open Source Ambient Intelligence Commons) https://itea3.org/project/osami-commons.html
[3]BaaS (Building as a Service) https://itea3.org/project/baas.html
[4]Medolution (Medical Care Evolution) https://itea3.org/project/medolution.html

Chapter 2

Related Work

This chapter sets the scene for the thesis by introducing the related work regarding the two approaches to the technical management: the policy-based management and model-based management. We focus primarily on the researches applied to and evaluated for the medical application field.

2.1 Policy-based Management of Medical Systems

The approach of the policy-based management has been applied in the medical application field repeatedly. Thereby, the usage of policies was one of the following: role-based access control, wireless (body) sensor network management and application workflow management. In the first case, the policies are used usually as rules restricting the access to the resources (mostly medical data). In the second case, the policies are applied to the network management of the (body) sensor networks. In the third case, the policies control the application flow by explicitly stating the choices in the behavior of the system. The following sections provide an overview of the relevant research projects conducted during the last years.

2.1.1 AMUSE

The following section is based primarily on [LDS+08] and [KTP+07]. The project *Autonomic Management of Ubiquitous Systems for e-Health (AMUSE)* was carried out during 2004 – 2007 as a joint collaboration between the University of Glasgow and Imperial College, London. The main focus of the work was the developing of an architecture for autonomic management of ubiquitous computing environments, in particular in the e-Health sector. In this field the body sensor networks which enable health monitoring at home arouse a special interest. Formed by low-power on-body and implantable sensors using wireless communications they interact with processing units (e.g. PDAs, mobile phones) as well as with the fixed network infrastructure. This allows continuous medical monitoring, automatic alerting in case of patient's critical condition and possible also direct intervention through the actuators (e.g. defibrillators, pacemakers, insulin pumps). By such systems a high value is set on the autonomic management on account of lacking user's knowledge and experience or constricted ability to configure and administrate the used devices. The developed techniques cater for the runtime extensibility of the component topology, adaptivity of the components to the current situation and self-configuring due to the changes of context or in requirements.

The Self-Managed Cell (SMC) was proposed as a policy-driven architectural pattern for implementing autonomic ubiquitous systems (Figure 2.1). A SMC manages a set of homogeneous components (i.e. managed resources) uniformly using *resource adapters*. The communication with the resources is therefore independent from the used communication protocol and the resource interfaces. The common *event bus* provides the interaction with

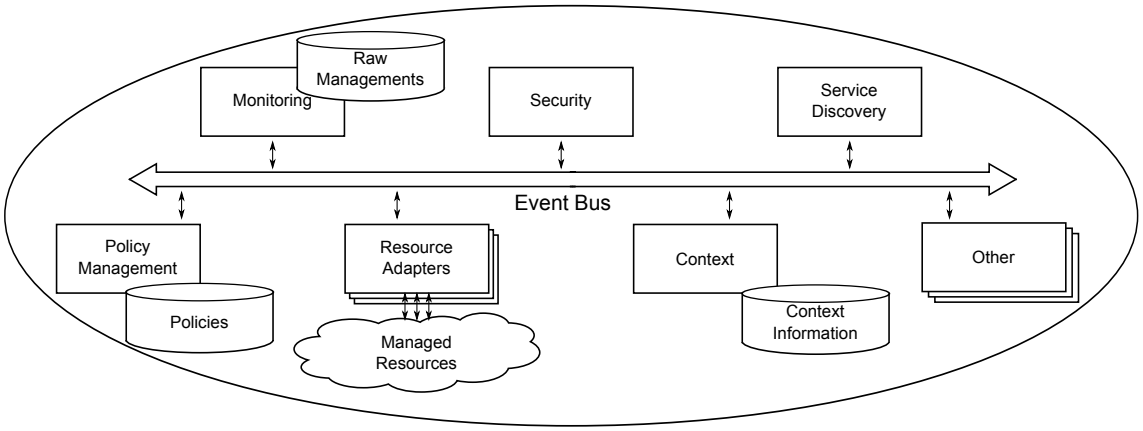

Figure 2.1: Self-Managed Cell (SMC) Architectural Pattern. Adapted from [LDS+08]

the offered services by using a router to forward event notifications from the event publishers to the subscribers. This approach permits to decouple the services, so that the sender does not know the listeners of the event. The advantage of this is that the new services could be added more comfortably without interrupting the others. Furthermore the concurrent and independent response of multiple services to the same event is facilitated. As well as the communication overhead could be lowered by transmitting only the measured data which exceeds the specified threshold. Self-management and adaptation are performed by means of the *policy service* that conducts a basic feed-back control loop. On changing in the state of the managed objects the corresponding reconfiguration actions in form of events are forwarded to the event bus. Which actions are to be executed is a subject to *obligation policies*. These are represented by means of *event-condition-action rules*. The *authorization policies* define which actions may be performed on which resources. As an implementation of the *policy service* the authors present Ponder2 [TDLS09] the successor of Ponder [DDLS00], a policy definition language and toolkit developed at Imperial College, London. The Ponder2 compounds a general-purpose object management system with a *domain service* providing a hierarchy for the managed objects, an *obligation policy interpreter* for handling the *obligation policies*, a *command interpreter* performing invocations on the managed objects and an *authorization enforcement* supporting fine grained authorizations for the managed objects. The detecting new devices or other SMCs is a task of the *discovery service*. It is responsible for generating the corresponding *component-detected* and *component-left* events as well as for distinguishing between the transient disconnections and permanent device departures. For managing more complex environments several SMCs could be composed or collaborate with each other. The composing SMCs allows to manage more smart diagnostic devices which manage in their turn their own resources. The interaction of multiple SMCs permits scenarios where new policies from other SMCs are to be loaded or updated. The requirements of one SMC for interacting with another is defined within its *mission* which is a group of policies determining the communication behavior with the other SMC.

2.1.2 CareGrid

This section is based on [RDD+07b], [RDD07a] and [SBM07]. The project *Autonomous Trust Domains for Healthcare Applications (CareGrid)* running since 2005 is a collaboration between the Imperial College, London and the University of Cambridge. The main aim of the project is developing a middleware for supporting decisions based on trust, privacy,

security and context models in a healthcare application domain. A targeted framework should include an architecture, which would consist of diverse services and support their interaction and administration. The integration of the *national electronic health record (EHR)* service is to be supported. The framework is policy-based, providing a mechanism for controlling access to the medical data and dynamic adaptation of the system. Monitoring and archiving functions, which comprise also system reliability and performance monitoring, are of a special interest.

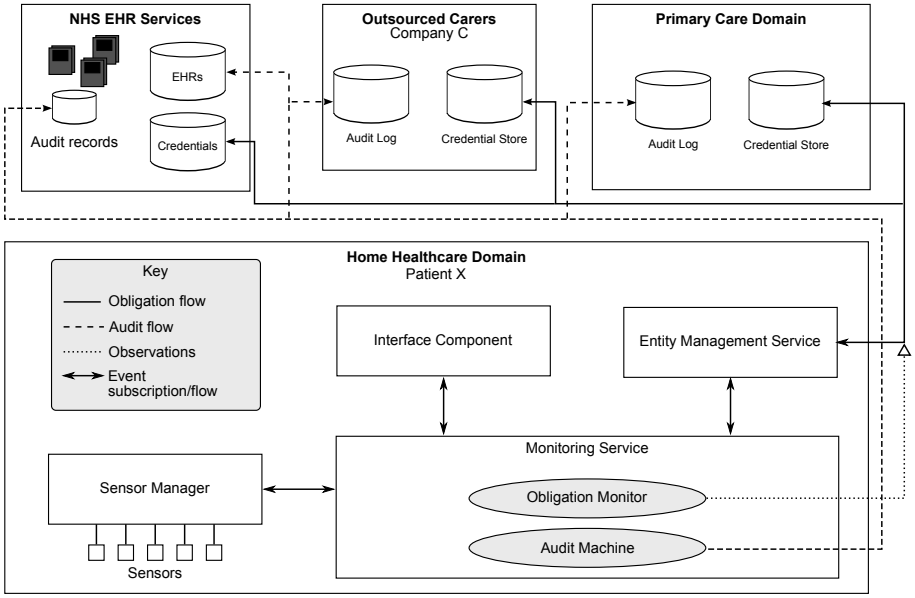

Figure 2.2: CareGrid Architecture. Adapted from [SBM07]

The home healthcare domain interacts with various domains, including hospitals, homecare providers, specialists, surgeries, social care providers and many others (Figure 2.2). The coordinating domains (e.g., the *National Health Service (NHS)*) ensure the compliance of the provided services and the collaboration of the domains. The *primary care domain*, such as hospital, creates a home-based patient care environment. Diverse other domains could provide specific services. To act as an authority for validating and verifying entities, domains require a *credential store*.

The developed architecture for a home healthcare domain include following components: *sensor manager*, *interface component*, *entity management service* and *monitoring service*. All the aspects concerning the use of sensors including sensor discovery, failure detection, stream management and data capture belong to the tasks of the *sensor manager*. The *sensor manager* is also in charge of the evaluation of the captured data and the invocation of the appropriate responding actions. The user interface and the interface for access by devices is provided through the *interface component*. The *entity management service* tracks the devices and services within the domain and defines the privileges according to the actual policies. The core of the infrastructure is the *monitoring service* which observes all the interactions between components passing through a monitoring pipeline. The *monitoring service* offers two components: *audit machine* and *obligation monitor*. The first is responsible for transferring relevant information to various audit logs (e.g., to the electronic health record) according to the defined policies. The *obligation monitor* launches compensatory actions in case of a failure in obligation fulfillment. It also informs the credential services about the performance of the system.

The proposed framework is policy driven both at the system-level (e.g., defining events, actions and domains) and the user-level (e.g., defining thresholds for relevant parameters). Ponder2 [TDLS09] was used as a policy language supporting obligation (*event-condition-action rules*) and authorization policies. The entities to which policies apply are organized in hierarchical domains of *managed objects*. The *managed objects* are associated with a set of data, which is used by authentication, authorization and obligations. The domain hierarchy for each component of the system is maintained by the local policy interpreter. For execution of actions on *managed objects* from the external domains proxies are created by the local interpreter and inserted in the local domain structure. The developed conflict resolution strategy used statically and dynamically is a subject of [RDD07a].

2.1.3 MATCH

The section is based mainly on [WDT+06] and [WT08]. The ongoing project *Mobilizing Advanced Technologies for Care at Home (MATCH)* is running during the years 2005 – 2009 as a collaboration among the universities of Stirling as a lead partner, Glasgow, Edinburgh and Dundee. The main aim of the project is to develop advanced technologies in support of social and health care at home, particularly in the area of home network services, lifestyle monitoring, speech communication and multimodal interfaces. OSGi (Knopflerfish Framework Implementation) has been selected as an ideal technology for the implementation as a vendor-neutral, device independent approach to service provision. The management of home networks is to be accomplished using policies which allow multiple stakeholders to configure the system behavior. The use of ontologies enhances the discovery of services and the use of policies managing these services.

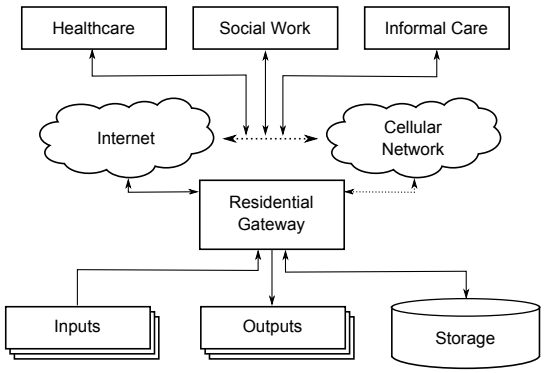

Figure 2.3: MATCH System Architecture. Adapted from [WDT+06]

The proposed architecture (Figure 2.3) involves OSGi *residential gateway* embedding the home services and device control. The sensors (e.g. physical devices, logical or user-oriented data sources) provide the *inputs* of the system. The *outputs* invoke the actuators which could also be in their term physical, logical or relating to user. The link to the outside world is usually via a broadband connection to Internet or a direct link to a cellular network. The captured and in the *storage* saved information could be forwarded to the care providers (e.g. *healthcare centers*, *social work departments* and *informal carers*).

The authors have developed a *service ontology stack* which organizes ontologies of multiple abstract levels. The base ontology provide descriptions for such general concepts like vendor, location, service type and environment. Each of the concepts in its turn is specified within the more specific ontology. Upon this the developers can design their own service type specific ontology which allows a more precise semantically-based description of the

provided services. To provide semantic service discovery a special OSGi bundle has been implemented using Jena2 Semantic Web Toolkit [McB02] and Protege [Sta]. The bundle collaborates with the OSGi service registry storing service ontology descriptions, reasoning about them and answers to the queries (Figure 2.4).

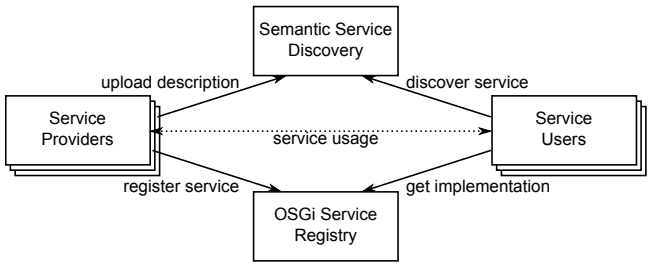

Figure 2.4: MATCH Semantic Service Discovery. Adapted from [WDT+06]

A *policy-based management system* resides on the *residential gateway*. The task of the management system is to manage the involved devices and services by means of predefined policies expressed as *trigger-condition-action rules* (Figure 2.5). These are formulated in a language APPEL (ACCENT Project Policy Environment and Language) [Com] in form of XML documents. A web-based *policy wizard* was developed for the remote policy edition and creation. Domain-specific knowledge of concepts and relationship of policies is integrated using an *ontology server*, a system called POPPET (Policy Ontology-Parsing Program – Extensible Translation). The *policy store* is used as a repository for holding user profiles, the system configuration and state. The latter two allow the policies to refer to abstract terms and to be interpreted depending on context. The interaction with the *policy system* is a task of the *home server*. The communication is performed by sending and receiving events. The *home server* notifies the *policy server* about a triggering event. The *policy server* selects the corresponding policies, evaluates them and responses accordingly. Conflict handling is performed by means of high-level resolution policies which are triggered by a conflicting actions and conduct the resolution according to some given high-level criterion [WT07].

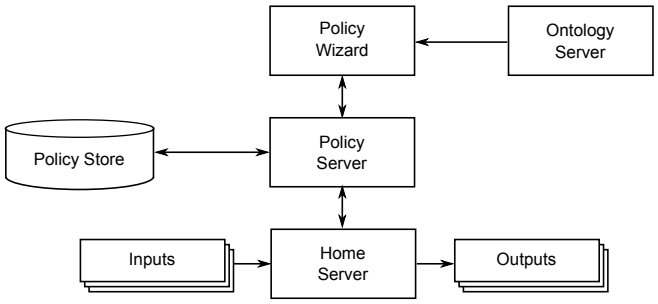

Figure 2.5: MATCH Policy System Architecture. Adapted from [WDT+06]

2.2 Model-based Management of Medical Systems

Usage of models in order to support medical systems varies in its application area. Thus, e.g. models are suitable for supporting the process or workflow management system in health care [BSE12], [LR07]. Another example of application area is model-based medical decision support for diagnosis and/or prognosis assisting the medical staff in their work

[AKP+11]. This methodology, however, is based on a combination of structural and stochastic modeling which is not subject of interest in the present work. Models can also be used for engineering of medical systems [PHAB12], [BGFV11], [RJJZ10] providing a basis for fast prototyping, testing, safety verification.

2.2.1 Model-supported Process Management

In [BSE12], [BE13] a model-supported process management of medical systems is presented. Figure 2.6 illustrates the context of model usage within the medical domain [BE13]. The

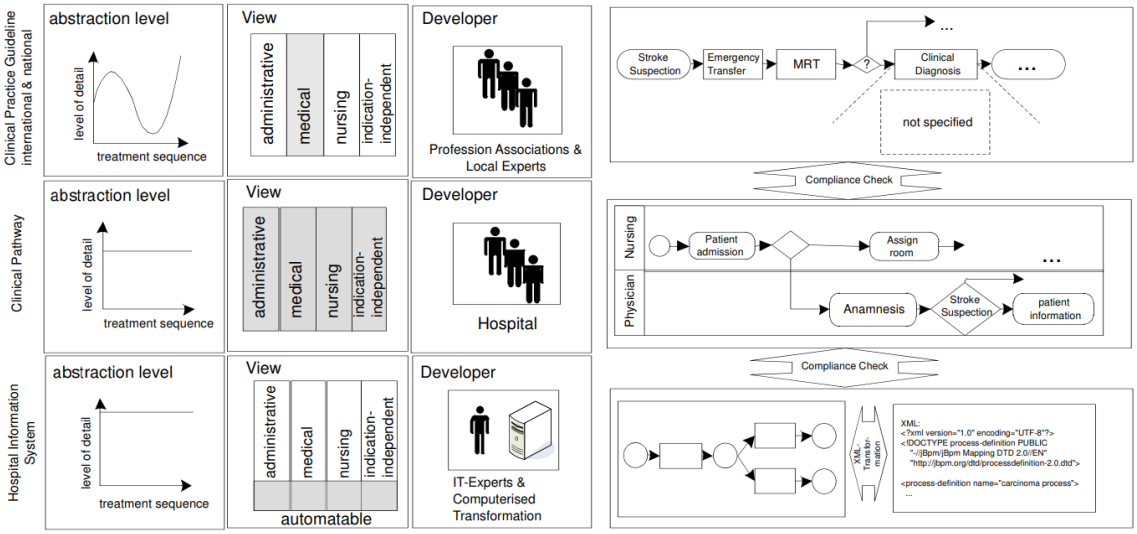

Figure 2.6: Model-based Management of Medical Systems. Reprinted from [BE13]

authors introduce a framework that reflects the link of medical treatment instructions to the context of hospital organizations. The current findings of the medical practice are reflected in the clinical practice guidelines published by the corresponding authorities (upper layer). The guidelines lead to clinical pathways applied within the concrete local clinical structure (middle layer). In order to "to reflect the arrangements of the clinical pathways and to support and standardize the decision-making of the physician as well as the planning of treatment" [BSE12], a hospital information system is used (bottom layer). The authors focus in their research on modeling the clinical pathways, since they allow to "represent the current medical scientific knowledge in combination with institution-specific facts and prepare it as best practices, e.g. a handbook for treatment in a human understandable form".

The authors propose claim, that the long-term quality of care and therefore the patient satisfaction are to be achieved by integrating elaborately modeled and planned patients pathway models. A dedicated management system is used in order to support the user. The management process involves modeling, planning and execution phases. The modeling phase covers the basic work on the analysis and picturing the treatment processes. The organization specific parameters are considered during the planning phase, whereas instantiation of pathways models for individual patients is done during the execution phase. Moreover, a management "cockpit" is supposed to give an opportunity to query the pathway instances in real-time as well as to keep track of the patients individual ways.

2.2.2 SPES2020/SPES XT

The research project Software Platform Embedded Systems 2020 (SPES) 2020 has been carried out during 2008–2012 and its follow-up project SPES XT during 2012–2015 as joint collaborations between 21 industrial and academical partners under the scientific direction of Technical University of Munich. Within the projects a seamless model-based development of safety-critical systems was researched in order to provide for validation and verification of requirements, simulation, verification, as well as virtual integration testing. The research results have been applied to and tested for several application domains. In [PHAB12] the authors report on dramatically increased verification and validation efforts in the field of medical systems over the last years. Thus, amongst others the SPES project has addressed the medical domain.

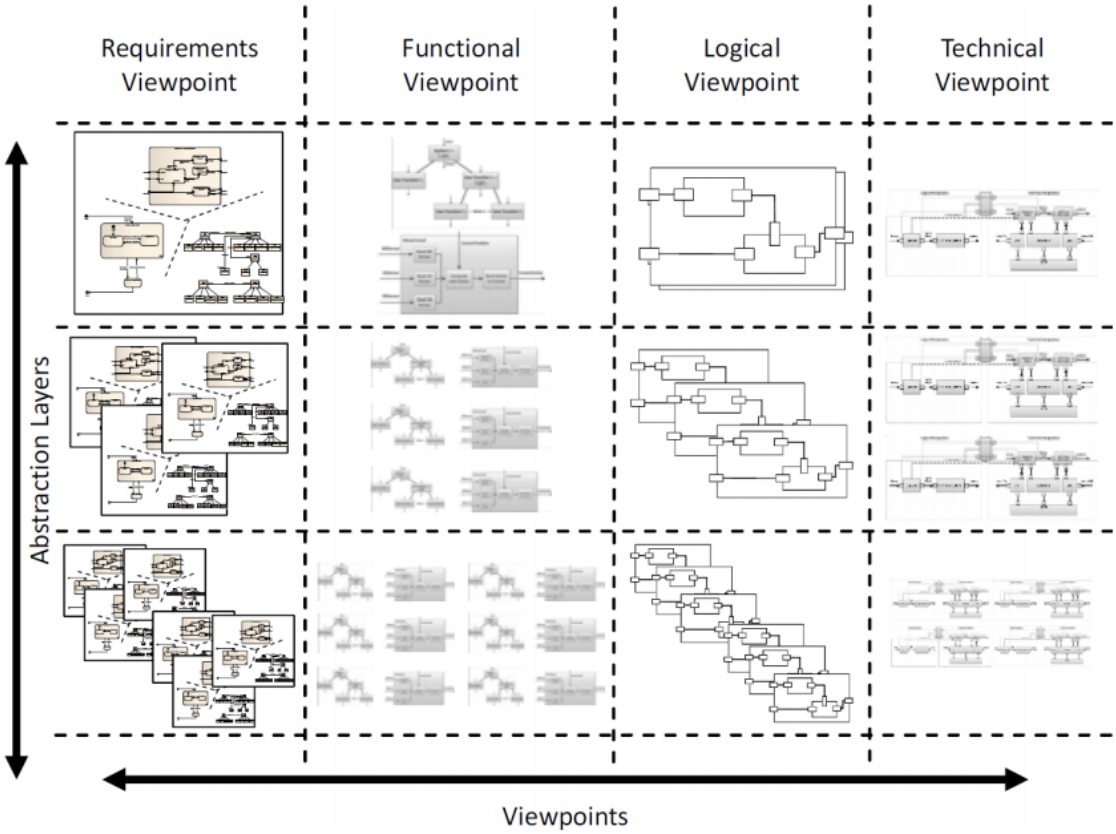

Figure 2.7: SPES Modelling Design Space. Reprinted from [PHAB12]

The proposed SPES Modeling Framework (Figure 2.7) adheres strictly to the principles of stakeholder concerns, hierarchical decomposition, seamless model-based engineering, separation between problem and solution as well as logical and technical solution, and consideration of crosscutting system properties during the development process. Two fundamental concepts of viewpoints and abstraction layers forming a two-dimensional design space are promoted.

A viewpoint concept follows the notion of the IEEE Standard 1471 "Recommended Practice for Architectural Description of Software-Intensive Systems" [IEE00] and is regarded as s a template or pattern for the development of individual views on the system: requirements, functional, logical, and technical viewpoint.

Requirements Viewpoint Requirements engineering process is supported by means of the requirements viewpoint. Users, stakeholders as well as external systems are regarded

as the system context which provides specific requirements and goals the system is supposed to satisfy.

Functional Viewpoint The functional viewpoint provides a formal specification and abstract realization of the functions which are to be offered by the system. The functions described in this viewpoint specify the system behavior and stem from the system requirements defined in the requirements viewpoint. A hierarchical decomposition of functions into sub-functions as well as interactions between (sub-)functions are presented.

Logical Viewpoint The logical viewpoint provides a structural decomposition of the system independently from the technological aspects. The logical components of the system and their relationship in form of logical channels are described. The components form a platform independent model, which abstracts from the underlying hardware solution and specifies realization of the system functions defined in the functional viewpoint.

Technical Viewpoint The realm of the technical viewpoint includes the platform-specific representation of the system and its components. The view addresses modeling of resources (e.g. storage, memory, bandwidth), schedulers (e.g. virtual computational elements load balancing, hardware and network resource scheduling) and tasks which specify the physical architecture of the system. The technical viewpoint is intended to specify for the logical components on which hardware resources they are executed as well as how the subsystems are physically organized. Resource consumption and redundancy in matters of timing and safety as essential aspects are addressed within this viewpoint.

System elements can be modeled on different abstraction levels. Abstraction layers are user defined, i.e. application domain specific ("Supersystem", "System", "Subsystem", and "Hardware/- Software Component"). In order to allow tracing of refinements, mappings between the different abstraction layers can be used.

As a proof of concept a study case from the healthcare domain has been described demonstrating the application of the approach to engineering of an extended care system comprising body area network devices, a VAD, and a telematics system.

Also in [HR08], the model-based design is claimed to support the development of such critical systems like medical. Its well-founded methodology is reported to provide a solid basis for the tool support.

Chapter 3

Medical Domain

The contemporary practice of medical and health care relies on extensive usage of information technologies. Due to the sensitivity of the application domain the stringent regulatory procedures are applied to the used devices and applications. Whereas the scope and content of laws, regulations and norms as well as certification and inspection procedures vary, they address some general application domain specific issues:

Safeguard clause The clinical condition and safety of patients (or other persons) should not be compromised. Associated risks should be eliminated or reduced as far as possible. Adequate protection measures including alarms, if necessary, should be taken. Users are to be informed of the residual risks in case any shortcomings of the protection measures exist.

Standard conformity The modern healthcare delivery relies on the usage of applications and devices which are distributed across multiple organization, provided by different vendors, based on diverse technologies and handle with all sorts of data. To enable interoperability at the highest level, it is essential to resort to existing standards. They address device connectivity, communication as well as personal health information exchange issues.

Information security Patient confidentiality is one of the main principles in medical ethics according to which any information revealed by a patient to a healthcare provider is strictly private, unless the patient gives a consent to disclose it to a third party or it can be justified by law. Therefore, access to patient data by applications and devices is reflected in guarantee of the common information security fundamentals: confidentiality, integrity and availability of data.

Accountability Accountability is a basic component which means obligation of the parties to justify and take responsibilities for their activities. In the light of the application domain it entails processes and procedures to provide for professional competence, legal and ethical conduct, financial performance, adequacy of access, public health promotion, and community benefit [EE96].

Quality Assurance Attaining the highest performance and safety level is essential for the healthcare services. Thus, comprehensive quality management systems for medical devices and systems are inevitable. Market entering implies a set of regulatory procedures which refer to the quality assurance. So, the manufacturers must demonstrate that their product does what it is supposed to do and is able to demonstrably meet the medical claim.

The effective use of information technology and management systems not only advances but also transforms the healthcare sector. The mission of the Health Information and Management Systems Society (HIMSS), a non-profit international organization, is to lead this process. It aims to promote information and management systems' contributions to the medical domain and in doing so to improve the quality, safety, access, and cost-effectiveness of patient care.

Integrating the Healthcare Enterprise (IHE) is a non-profit initiative by healthcare professionals and industry to improve the process of sharing information between IT systems

used in healthcare. Focusing on standardizing and harmonizing the existing standards, it aims for the primary objective of optimal patient care. IHE publishes, expands annually and maintains IHE Technical Frameworks which define specific implementations of established standards. It is supposed to achieve effective system integration and to facilitate the entire sector. Several healthcare domains are specified (e.g., radiology, cardiology, patient care devices), within these domains they describe the main workflows from practice, e.g. patient admission, registration, examination, data acquiring, recording, etc. (Figure 3.1). IHE defines integration profiles with regard to a specific clinical task or medical condition. They specify the main actors involved, the information to be exchanged between systems and actions to take place on certain events.

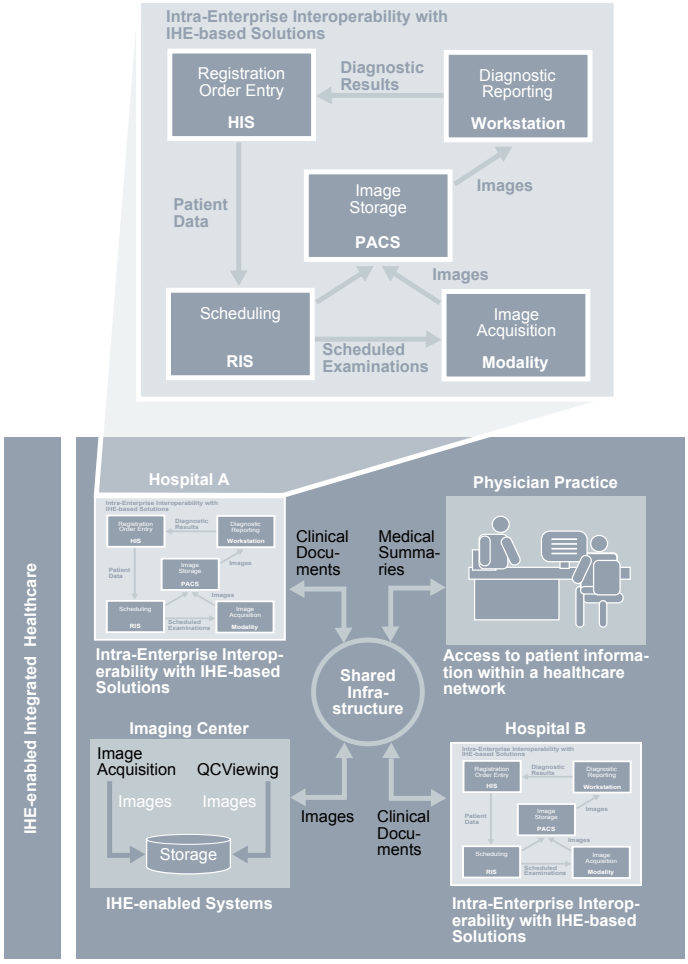

Figure 3.1: Medical Domain by Integrating the Healthcare Enterprise. Adapted from [Sie10]

IHE claims that adopting their approach would result in the following benefits: reduction of medical errors, lower costs, more efficient workflow, better informed medical decisions, faster results for patient and clinician, etc. [CPW+01].

3.1 Medical Domain Actors

3.1.1 Medical Service Consumer

Patient (from Latin *patiens* - "one who suffers") is a central actor in the medical domain, a person who receives a health service, in most cases a treatment from a health professional.

Usually, a patient refers to a medical service in case of an illness, what means his health condition does not allow to define him as healthy. World Health Organization (WHO) has formulated the mode of being healthy in a broader sense as "a state of complete physical, mental, and social well-being and not merely the absence of disease or infirmity." This formulation in fact arises some critics especially concerning the usage of the word "complete" and lack of concrete metrics. However it stays most widely accepted since its publication in 1946.

A *disease* is referred as any abnormal condition impairing functions of the patient's body. Commonly, it is associated with dysfunction of organism and goes together with certain symptoms and signs. The causes of a disease can be of different etiology. So, diseases can result from the influence of exogenous (external) factors (e.g., infection, radiation, trauma), from endogenous (internal) factors (e.g., autoimmune breakdown, genetic disorder) or their combination.

Due to the fact that a disease (it's treatment or prevention) is primary reason of medical service consumption, it is common to classify patients on the basis of their diseases. ICD-10, the 10th revision of the International Classification of Diseases and Related Health Problems published by WHO, provides a hierarchical classification system comprising chapters, blocks of categories, categories and subcategories. The classification is of practical nature and strikes a balance between classification based on "etiology of diseases, anatomical site, circumstances of onset, etc." [Wor10b] It has become the international standard diagnostic classification for health management purposes, because it permits to carry out "systematic recording analysis, interpretation and comparison of mortality and morbidity data collected in different countries or areas and at different times". Provided alphanumeric code of diseases and health problems supports convenient data storage, retrieval and analysis.

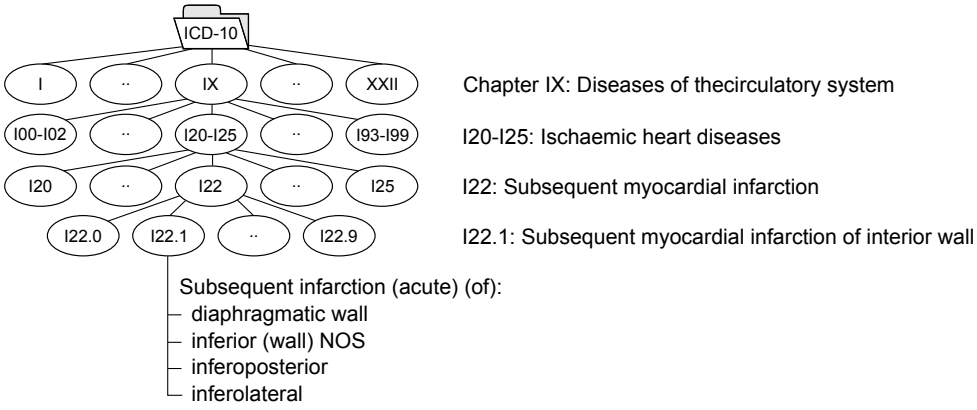

Figure 3.2: ICD-10 Classification Example

Figure 3.2 shows an example of ICD-10 based classification of a patient suffering from a subsequent myocardial infarction of inferior wall. The corresponding ICD-10 code `I22.1` can be taken from classification hierarchy.

3.1.2 Medical Service Provider

Provision of a health service is typically accomplished by the cooperation of multiple stakeholders belonging to the *health workforce*. Based on the International Standard Classification of Occupations (ISCO, 2008), WHO suggests a classification system defining the main occupations in this domain and ordering them into a hierarchical structure: health professionals, health associate professionals, personal care workers in health services, health

management and support personnel, and other health service providers not elsewhere classified.

Health professionals "study, advise on or provide preventive, curative, rehabilitative and promotional health services based on an extensive body of theoretical and factual knowledge in diagnosis and treatment of disease and other health problems. They may conduct research on human disorders and illnesses and ways of treating them, and supervise other workers. The knowledge and skills required are usually obtained as the result of study at a higher educational institution in a health-related field for a period of 3-6 years leading to the award of a first degree or higher qualification." [Wor10a] This group of professionals include generalist medical doctors (e.g., physician), specialist medical doctors (e.g., surgeon, cardiologist, neurologist), nursing professionals (e.g., clinical nurse), midwifery professionals, traditional and complementary medicine professionals (e.g., homeopath), paramedical practitioners (e.g., feldsher), dentists, pharmacists, environmental and occupational health and hygiene professionals (e.g., occupational hygienist, radiation protection adviser), physiotherapists, dieticians and nutritionists, audiologists and speech therapists, optometrists and ophthalmic opticians.

The delivery of high-quality healthcare is attendant on the sound relationship between patient and medical service provider belonging in most cases to this group.

Health associate professionals "perform technical and practical tasks to support diagnosis and treatment of illness, disease, injuries and impairments, and to support implementation of health care, treatment and referral plans usually established by medical, nursing and other health professionals. Appropriate formal qualifications are often an essential requirement for entry to these occupations; in some cases relevant work experience and prolonged on-the-job training may substitute for the formal education." To these professionals belong such occupations as medical imaging and therapeutic equipment technicians (e.g., magnetic resonance imaging technologist, sonographer), medical and pathology laboratory technicians, pharmaceutical technicians and assistants, ambulance workers, medical records and health information technicians (e.g., medical records analyst, disease registry technician).

Personal care workers "provide direct personal care services in health care and residential settings, assist with health care procedures, and perform a variety of other tasks of a simple and routine nature for the provision of health services. These occupations typically require relatively advanced literacy and numeracy skills, a high level of manual dexterity, and good interpersonal communication skills." The group comprises health care assistants (e.g., hospital nursing aide) and home-based personal care workers (e.g. home nursing aide, home care aide).

Health management and support personnel "include a wide range of other types of health systems personnel, such as health service managers, health economists, health policy lawyers, biomedical engineers, medical physicists, clinical psychologists, social workers, medical secretaries, ambulance drivers, building maintenance staff, and other general management, professional, technical, administrative and support staff." This group of occupations include health service managers (e.g., health facility administrator, clinical director), life science professionals (e.g., bacteriologist, water quality analyst), social work and counseling professionals (e.g., clinical social worker), non-health professionals not elsewhere classified (e.g., health policy analyst, health statistician, safety engineer, software developer), medical secretaries, clerical support workers, non-health technicians and associate professionals not elsewhere classified (e.g., bookkeeper, computer network technician, data entry supervisor, fitness instructor, forensic science technician, health insurance claims officer).

The classification also specifies such health service providers as armed forces occupations (e.g., veteran hospital nursing aide), medical student intern, hospital volunteers etc.

As reported by the German Federal Statistical Office the number of persons employed in the health sector up to about 4.8 million in 2010 in Germany. That means, that about one in nine of all persons employed, worked in the health sector. Compared with the previous year there was an increase in employment of 1.9%. [1]

Taking into account the entire public health sector *health care organizations* take action as actors within medical domain. They comprise hospitals, clinics, prevention and rehabilitation facilities, medical practices, ambulances, rescue services, pharmacies, medical supply stores, scientific organizations etc. *Public health authorities* (e.g., medical associations, health departments, health ministries, federal offices for statistics, disease registers) play an important role in this sector, also. Worth mentioning are such stakeholders as *insurance companies* (health, long-term care, annuity, accident), *pharmaceutical* and *medical technology manufacturers*, *medical equipment vendors*, *medical data processing centers*, etc.

In [KB11] they address the problem of competing interests between "actors" in healthcare. The authors describe the health arena as a complex selfmanaging formation affected by the conflict of interests of the main stakeholders: individuals are motivated to enhance their health, enterprises target for profit but are controlled by political and organizational forces, whereas government acts as a main healthcare provider in the economic sense and as a people protector (Figure 3.3).

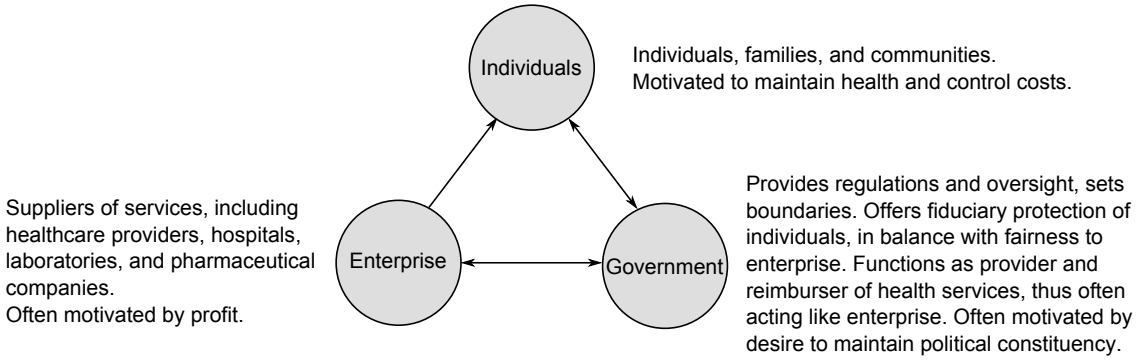

Figure 3.3: "Triangle of Actors" on the Health Arena. Adapted from [KB11]

3.2 Medical Assets and Specifics

Provision of medical and health care services supposes in most cases a treatment of a patient or preventing measures conducted by a health professional. A *treatment*, or *therapy*, implies application of remedies for a disease or injury. Normally, a disease is indicated by a patient in form of *symptoms*. They form a "subjective evidence of disease or physical disturbance observed by the patient". On the contrary, a *sign* is "an objective evidence of disease especially as observed and interpreted by the physician rather than by the patient or lay observer."

The major indicators of body function, which indicate the presence of life, are usually referred as *vital signs*. These measures include e.g., temperature, respiratory rate, heart beat (pulse), and blood pressure. Observing, measuring and monitoring of these parameters enables the assessment of patient's body function. There are exist some normal ranges of measurements of vital signs which can vary with patient's age, gender, medical condition, time of day, ambient temperature, activity level, etc.

[1] https://www.destatis.de/EN/FactsFigures/SocietyState/Health/HealthPersonnel/HealthPersonnel.html (Jan. 2013)

The process of identifying a disease from its symptoms and signs is called *diagnostic procedure*. The decision reached by this process is referred to as a *diagnostic opinion* (or *diagnosis*). Diagnostic procedure is in most cases a complex cognitive process which demands *anamnesis*, i.e. recollection and accumulation of data concerning the patient case history and background (e.g., family, environment), *physical examination*, i.e. body investigation to determine its state of health, and *diagnostic tests* (e.g., radiologic, microbiologic, genetic, electrodiagnostic, blood tests). In the process of *differential diagnosis* the probability of one disease versus the one of the others is weighted in order to determine the disease which causes the symptoms. Often, *multimorbidity* is present, i.e. several medical conditions coexist.

An essential component of any treatment is a thoroughly worked out *treatment plan*. In most cases, treatment planning is a collaborative process between the patient and the physician. Treatment options are negotiated in order to select an appropriate treatment model and approach. Depending on chosen treatment objectives and methods, a manageable treatment plan is specified. Later on, in the process of treatment it is to be reevaluated and revisioned, if required.

Patient's medical information is held in a *medical record*, a chronologically written account of patient's health history including his illnesses, injuries, complaints, medical conditions, allergies, findings, the results of diagnostic tests, treatments, and medications and therapeutic procedures. It also contains patient's personal data such as environment, profession, family health information. Medical records are legal documents which are usually created and written by physicians or in hospitals and ambulatory environments. In case they made in a computerized form, they are referred as *electronic medical records*.

Sharing electronic medical records across single medical care provider boundaries allows new forms of medical and health care.

3.3 Medical Devices

Devices used within the medical domain vary greatly in their complexity and art. They range from contrivances with no intended medical purpose like electronic components, computers, communication networks to specific medical equipment like electrocardiographs, sonographic machines and human microchip implants. To ensure high quality and safety is essential in today's medical device industry. Innovations and inventions are introduced daily, whereas the process of clearance, premarket approval and control stays strictly regulated. Companies who design, manufacture and assemble medical components and devices must face the challenge of not only ensuring high performance requirements but also of reducing production time. Several standards exist which regulate quality management systems for medical devices, e.g. ISO 13485 [ISO03], ISO 14971 [ISO07].

At European level it is intended to harmonize the laws and legislative procedures relating to medical devices with the ultimate objective to ensure the patient's safety and to support the innovation and the competitiveness of this sector at the same time. Three directives form the core legal framework: Directive 93/42/EEC [Eur93] comprises essential requirements, quality assurance system, examination, verification, conformity to the type, clinical evaluation and marking of medical devices in general. Directive 90/385/EEC [Eur90] focuses on active implantable medical devices. In vitro diagnostic medical devices are subject to Directive 98/79/EC [Eur98]. The directives were amended by Directive 2007/47/EC [Eur07] providing last technical revision of the documents. In the United States Food and Drugs Association's Center for Devices and Radiological Health (CDRH) regulates the procedures relating to manufacture, repackage, relabel, and/or import of

medical devices sold in the country. Premarket notification process 510(K) is regulated by the Part 807 Subpart E and the premarket approval process by the Part 814 of the Title 21 of Code of Federal Regulations (CFR) [U.S12].

3.3.1 Definition

Taking into consideration the variety and diversity of devices used in the medical scene, it is not easy to define a medical device. Multiple definitions have been formulated by several countries and organizations. European Union Legal Framework defines in Directive 2007/47/EC [Eur07] *medical device* as "any instrument, apparatus, appliance, software, material or other article, whether used alone or in combination, including the software intended by its manufacturer to be used specifically for diagnostic and/or therapeutic purposes and necessary for its proper application, intended by the manufacturer to be used for human beings for the purpose of: diagnosis, prevention, monitoring, treatment or alleviation of disease, diagnosis, monitoring, treatment, alleviation of or compensation for an injury or handicap, investigation, replacement or modification of the anatomy or of a physiological process, control of conception, and which does not achieve its principal intended action in or on the human body by pharmacological, immunological or metabolic means, but which may be assisted in its function by such means." This definition covers a great multitude of products from a simple thermometer to a sophisticated medical robot.

Further, Directive 90/385/EEC points out *active medical devices* defining them as "any medical device relying for its functioning on a source of electrical energy or any source of power other than that directly generated by the human body or gravity". At the same time, medical devices which are intended to transmit energy, substances or other elements between an active medical device and the patient without any significant change, are not considered to be active medical devices [Eur90].

Considering the intended use of a device, it is common to distinguish *active device for diagnosis*, "any active medical device, whether used alone or in combination with other medical devices, to supply information for detecting, diagnosing, monitoring or treating physiological conditions, states of health, illnesses or congenital deformities". As well as *devices intended for clinical investigation*, what means "any device intended for use by a duly qualified medical practitioner when conducting clinical investigations" in clinical environment [Eur93].

Active implantable medical devices arise special interest and are defined as "any active medical device which is intended to be totally or partially introduced, surgically or medically, into the human body or by medical intervention into a natural orifice, and which is intended to remain after the procedure".

In vitro diagnostic medical devices have become indispensable by providing diagnostic measures. They are defined as "any medical device which is a reagent, reagent product, calibrator, control material, kit, instrument, apparatus, equipment, or system, whether used alone or in combination, intended by the manufacturer to be used in vitro for the examination of specimens, including blood and tissue donations, derived from the human body, solely or principally for the purpose of providing information: concerning a physiological or pathological state, or concerning a congenital abnormality, or to determine the safety and compatibility with potential recipients, or to monitor therapeutic measures".

Devices can be intended to be used once only for a single patient (*single use device*) or not.

3.3.2 Classification

An urge to classify existing medical devices is recognized by legislative authorities in many countries and regions. Current classifications take into consideration such facts like design complexity, usage characteristics, level of potential hazard connected with use or misuse. Thereby it is aimed to achieve that rigorous conformity assessment procedures are applied in an economically feasible, justifiable in practice and at the same time transparent way.

For example, European Commision for Health and Consumer has presented set of guidelines relating to questions of application of EU Directives on medical devices. In Annex IX of the Council Directive 93/42/EEC [Eur93] they propose a 'risk-based' medical device classification system which uses such criteria like e.g. duration of contact with the patient, degree of invasiveness, part of the body affected by the use of the device, diagnostic impact. The classification system comprises a set of complex rules which are used to decide within which class devices fall: I (Is and Im), II (IIa and IIb), III. Table 3.1 demonstrates an example of a classification rule concerning active medical devices used in diagnostics (e.g., magnetic resonance imaging equipment).

Rule#10	**Examples**
Active devices intended for diagnosis are in Class IIa: - if they are intended to supply energy which will be absorbed by the human body, except for devices used to illuminate the patient's body, in the visible spectrum,	- Magnetic resonance equipment - Pulp testers - Evoked response stimulators - Diagnostic ultrasound
- if they are intended to image in vivo distribution of radiopharmaceuticals,	- Gamma cameras - Positron emission tomography and single photon emission computer tomography
- if they are intended to allow direct diagnosis or monitoring of vital physiological processes,	- Electrocardiographs - Electroencephalographs - Cardioscopes with or without pacing pulse indicators - Electronic thermometers - Electronic stethoscopes - Electronic blood pressure measuring equipment.
unless they are specifically intended for monitoring of vital physiological parameters, where the nature of variations is such that it could result in immediate danger to the patient, for instance variations in cardiac performance, respiration, activity of CNS in which case they are in Class IIb.	- Intensive care monitoring and alarm devices (e.g. blood pressure, temperature, oxygen saturation) - Biological sensors - Blood gas analysers used in open heart surgery - Cardioscopes - Apnoea monitors, including apnoea monitors in home care
Active devices intended to emit ionizing radiation and intended for diagnostic and therapeutic interventional radiology including devices which control or monitor such devices, or which directly influence their performance, are in Class IIb.	- Diagnostic X-ray sources

Table 3.1: Rule 10 - Active Devices for Diagnosis. Adapted from [Eur10]

3.4 Medical Software

Software plays an increasingly important role in medical healthcare and finds nowadays widespread acceptance in all medical application areas. Rapid evolution in the technology provides a great challenge for regulatory bodies regulating the qualification and classification criteria.

Independent of the regulatory body, software which falls under the jurisdiction of medical devices is also qualified (and therefore regulated) as a medical device. The variety of regulations concerning medical devices has been introduced in Section 3.3. In addition to the above mentioned device definitions, the regulatory bodies also published specific guidelines in relation to software qualification and classification. Thus, International Medical Device Regulatory Forum (IMDRF) released a finalized a set of definitions for device software [Int15]. This document defines "software as a medical device" (SaMD), as "software intended to be used for one or more medical purposes that perform these purposes without being part of a hardware medical device". Similarly, the guideline MEDDEV 2.1/6 [Eur16] published by the European Commission refers to "standalone software" for medical device qualification. It strikes out, that primarily the intended purpose as described by the manufacturer of the product is relevant for the qualification.

3.4.1 Medical Device Software

In [Asi14] the main forms of medical device software are presented. They classify medical software broadly into three main forms:

Part of (IVD) Medical Device This form includes software which drives a medical device or is intended to influence the use of a device directly. In the most cases this refers to embedded software. It can be incorporated as a component or part of accessory of a medical device. Thereby the intended usage together with an in-vitro diagnostic medical device supposes the software to fall under the scope of the IVD Directive [Eur98]. E.g. imaging software in diagnostic ultrasound system, software in ECG monitor, software in pacemaker, mobile software that controls insulin pump delivery rate.

Medical Device Accessory This form includes software which is intended to be an accessory of a medical device, including the device operating or controlling software. Accessories are intended in general to support, supplement, and/or augment the performance of one or more parent devices, being built in or installed separately. E.g. Software that is intended for the analysis and interpretation of raw data transmitted from an MRT device.

Standalone Medical Device Software This form includes software and applications that are not a component of a medical device. Such software is in general placed on the market separately from the related devices. In most cases, they are intended to be used for the purpose of process and analysis of gathered medical information in order to assist diagnosis and treatment. E.g. Treatment or operation planning software, data analysis software for the purpose of directly aiding in the treatment or diagnosis of a patient.

Thus, the above mentioned groups include software able to perform their intended medical purpose "without being embedded in a hardware medical device or being dependent on specific or proprietary medical purpose hardware", i.e. software falling under the definition of SaMD [Asi14]. Such software is capable of running on general purpose, not necessary medical, computing platforms. It is also to mention, that "not being part of" supposes that the hardware medical device itself does not need the software to achieve its intended purpose. They can be used in combination with medical devices, other SaMD as well as general purpose software.

Further, IMDRF defines two main major factors that provide adequate description of the intended use of SaMD. Firstly, the significance of the information provided by the SaMD to the healthcare decision. Secondly, the state of the healthcare situation or condition in which the SaMD is used or may lead to.

Speaking of the intended use of the information provided by SaMD IMDRF distinguishes in [Int15] the following:

- To treat or to diagnose a disease or condition
- To drive clinical management
- To inform clinical management

Using the data provided by SaMD for treatment and diagnosis means that immediate or short-term actions are done in order to treat, mitigate or even prevent a disease or condition by using other medical devices, medicinal products or other means of providing therapy to a human body. It also involves actions intended to diagnose, screen, detect a disease, condition or its trend using sensors, data, or other information from other hardware or software devices.

In order to drive clinical management, the information provided by SaMD is used to aid in treatment and diagnosis, as well as in identification of early signs of a disease or condition in order to guide the next necessary treatment and diagnostic interventions. This will be achieved by providing enhanced support to safe and effective use of medicinal products and medical devices, applying sophisticated analysis of relevant information in order to predict risk of a disease or support in confirming a final diagnosis.

The information provided by SaMD in order to inform clinical management does not trigger an immediate or short-term actions to inform of options for treatment, diagnosis, prevention, or mitigation of a disease and condition, or to provide clinical information by aggregating relevant disease, condition, drugs, medical devices, and population data.

Concerning the state of the healthcare situation or condition which define the intended use of the information provided by SaMD IMDRF distinguishes the following groups [Int15]:

- Critical situation or condition. That means that accurate and timely actions are vital in order to avoid death, long-term disability, impairment of a patient or to reduce impact on public health.
- Serious situation or condition. That means that accurate and timely actions are of vital importance in order to avoid unnecessary interventions or long-term irreversible consequences on a patient's health condition or public health.
- Non-serious situation or condition. That means that accurate and timely actions are important but not critical for interventions to mitigate long term irreversible consequences on a patient's health condition or public health.

	Treatment & Diagnosis	**Drive clinical management**	**Inform clinical management**
Critical	IV	III	II
Serious	III	II	I
Non-serious	II	I	I

Table 3.2: Software as Medical Device Categorization. Adapted from [Int15]

Based on the above explained criteria, IMDRF defines four categories (I, II, III, IV) of SaMD, in relative significance to each other. The category IV has the highest level of impact on the patient or public health (e.g. software used to perform analysis of cerebrospinal fluid spectroscopy data to diagnose tuberculosis meningitis or viral meningitis), the category I - the lowest one (e.g. software which analyzes images, movement of the eye or other information to guide next diagnostic action of astigmatism).

A software may comprise a number of applications, each of them may be correlated with a module which can have a medical purpose or not. According to the European MEDDEV 2.1/6 [Eur16], the modules which are subject to the medical device Directive must comply with the requirements of the medical device Directive and must carry the appropriate CE marking. The non-medical device modules are not subject to the requirements for medical devices. The manufacturer is obligated to identify the boundaries and the interfaces of the different modules according to the intended use.

3.4.2 Fields of Application

The fields of application of software in healthcare and medical area vary in their diversity.

Hospital Information Systems, Workflow Management Systems

A Hospital Information Systems (HIS) as well as Hospital Workflow Management Systems are supposed to support the process of patient management in the clinic [HWAB04]. Typically they are intended to replace paper records in keeping and checking patient information as well as automate the main clinic workflow. Such systems assist in patient admission and discharge, in scheduling patient appointments and managing the electronic medical records. Another focus is the support of the insurance and billing processes in the hospital. In general, such systems are not qualified as SaMD.

Decision Support Software

A *Decision Support Software (DSS)* is intended to support healthcare professionals with recommendations for diagnosis, prognosis, monitoring and treatment of individual patients. They provide exclusively diagnostic references and suggestions, whereas the ultimate clinical decisions are to be made by the therapist himself. In the core, such tools incorporate medical knowledge databases and algorithms with patient specific data. This software is qualified in the most cases as SaMD.

Information Systems

An *information system* is intended to store, archive and transfer medical and patient data. Often they incorporate additional modules. Thus, electronic patient record systems can be intended to store and transfer electronic patient records. Their purpose is to archive all kinds of documents and data related to a specific patient as explained in Section 3.2.

Another example of an information system is a *Clinical Information System (CIS)* intended for e.g. intensive care units to store and transfer patient information generated in association with the patient's intensive care treatment. Such systems can contain information such as patient identification, vital intensive care parameters and other documented clinical observations.

A *Radiological Information System (RIS)* is intended to be used by radiology departments as a database in order to store and transfer radiological images and patient in-

formation [DHTM06], [HWAB04]. In general, such systems include functions for patient identification and scheduling, store examination results and imaging identification details.

Similarly to RIS, a *Picture Archive Communication System (PACS)* are intended to provide economical storage, rapid retrieval of images, as well as simultaneous access to images at multiple sites [DHTM06]. Such systems often incorporate different modules , such as an image acquisition device (an interface to the scanner), a data management system which controls the data flow, an image storage system, a display module with the corresponding user interface, and a post-processing tool used to interpret raw images.

Communication Systems

General purpose communication systems are often used in the healthcare sector. Thus, email systems, mobile telecommunication systems, video communication systems, and paging etc. are used in order to transfer both non-medical as well as medical information (e.g. prescriptions, referrals, images, patient records). Such system are not qualified in their own right as SaMD, however may be used with other as SaMD qualified modules.

A *telemedicine system* is intended to support monitoring and/or delivery of healthcare service to patients located remote from the healthcare professional. As a special case, a *telemonitoring system* is used in order to enable an ongoing assessment of patient's physiological and/or context data at home or outdoors. A *telesurgery system* is intended to conduct a surgical intervention from a remote location. Such systems fall under the Medical Devices Directives. Another example, is a *video appointment software* which is intended to support remote consultations between healthcare professionals and patients and does not fall under the Medical Devices Directives.

In Vitro Diagnostic (IVD) Software

An *in vitro diagnostic (IVD)* software is intended to be used in or with IVD devices [Eur98]. Thus, a *Laboratory Information Systems (LIS)* is applied in laboratory based point of care supporting the process of in vitro examination of specimens derived from the human body. Besides the pre-analytical functions of ordering and sorting test samples, these systems provide for the management and validation of obtained information as well as connection to the analytic instruments. The post-analytical functions include communication of laboratory results, statistics and reporting to external systems.

Software for General Fitness, Health, Wellness Management

The purpose of *software for general fitness, health and wellness management* is to provide sources of information by supplying health professionals and patients with general health advice. It is intended for individuals for gaining or maintaining general fitness, health or wellness by logging, recording, tracking and/or making decisions or behavioral suggestions. They have no purpose in neither diagnosis nor in the cure, mitigation, treatment, or prevention of a disease or health condition. In the most cases such software is not regulated by Medical Devices Directives.

Chapter 4

Technical Management

The complexity and size of contemporary IT environment has been growing tremendously during the last years [RC07]. Today's distributed systems are extremely large-scale, highly dynamic and device-rich. The price for it though is a sophisticated system management, which has to answer wide-ranging, complex, and diverse system requirements.

4.1 Paradigms and Fundamentals

In order to provide a review of the research field, we need to introduce the main paradigms and fundamentals of the overall complex of *management*. According to [HAN98],

> "The management of networked systems comprises all the measures necessary to ensure the effective and efficient operation of a system and its resources pursuant to an organization's goal".

Thus, the management aims to provide the services and applications of a system with a desired quality level as well as to assure availability and a rapid, flexible deployment of resources. In doing so, not only technical tools but also personnel, procedures, and programs affecting different levels of objects (resources, services, applications) are incorporated.

It is common to distinguish between the following levels of management: *network*, *systems*, and *applications*. *Network management* concentrates on the communication network and its components, *systems management* emphasizes the resources of the end system, whereas *applications management* is responsible for the management of distributed applications and services. The overall complex of management is, however, more comprehensive (Figure 4.1). Thus, design and maintenance of enterprise-wide distributed data is the focus of the *information management*. The task of *enterprise (or business) management* includes financial, personnel, and production measures covering enterprise-wide aspects and establishing guiding principles for IT infrastructure, operating services, associated services, and data. These principles produce the conditions for the lower-level management.

Modern distributed systems are heterogeneous and complex require *integrated management* solutions [HAN98]. They strive first of all for an integral approach to different aspects (management level, functional area, global databases, open programming and user interfaces, etc.). *Isolated approaches* (be that e.g. in terms of the functional area or the vendor) tend to be cost-unjustified considering the heterogeneity and complexity of the management environment. While following the integrated approach, the needs and requirements of all the stakeholders are to be taken into account and fairly satisfied, so that an optimal level of efficiency can be achieved while using a synergy effect and best use of all resources. An integrated management environment presumes that communication between relevant components rests upon shared protocols. It also requires managed elements to expose the management-relevant information in a manner which is vendor-independent and accessible via well-defined interfaces.

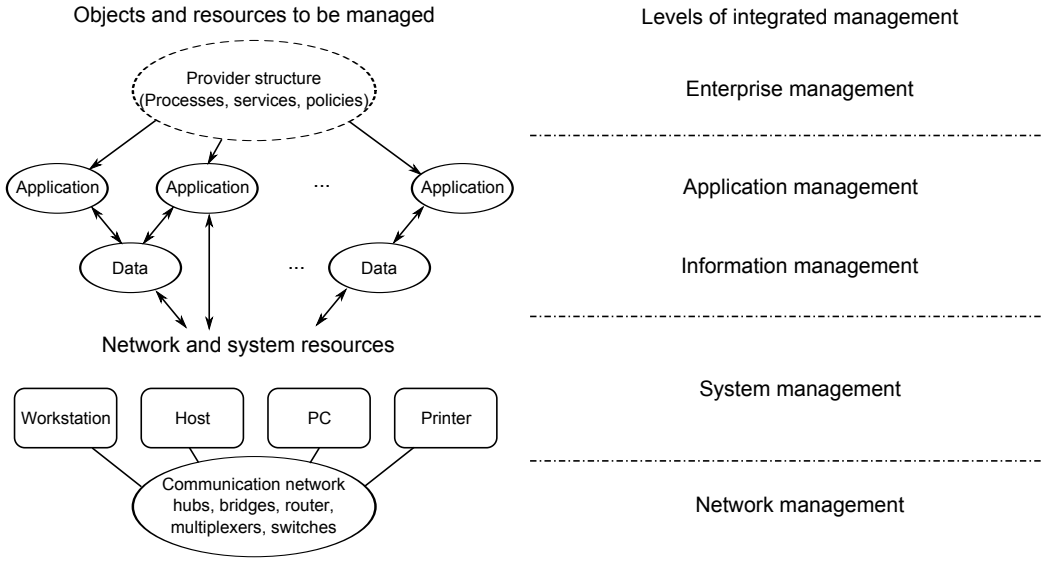

Figure 4.1: Levels of Management. Adapted from [HAN98]

Management architecture provides a framework for management-relevant standards. The most important forms of contemporary management architectures are presented by e.g. OSI and TMN, Internet, CORBA management architectures, DMTF Desktop Management Interface, and web-based architectures as Java Management API and web-based enterprise management(WBEM). It is common to distinguish between four key models essential for definition of a management platform [HAN98]:

- *Information model* controls the methods for modeling and description of managed resources.
- *Communication model* defines access to the managed resources and communication protocols.
- *Functional model* structures the overall management functionality.
- *Organization model* specifies management domains, partitions management realm, defines roles, responsibilities, and cooperation forms.

Serving solely as a framework, a specified management architecture has to be realized in order to gain a practical use. Under the term *management platform* we understand implementations of management architectures done on the basis of standardized programming and service interfaces. Platforms act as open carrier systems for management applications. Management applications then again can use other management systems, tools or managed resources (Figure 4.2).

Aiming for an integrated approach, a management platform must address the following issues on the vendor-independent basis:

- What information and resources are relevant for managing (information model).
- How to describe the managed information and resources (information model).
- What management-relevant communication flows exist (communication model).
- What protocols should be used to communicate for the management purposes (communication model).

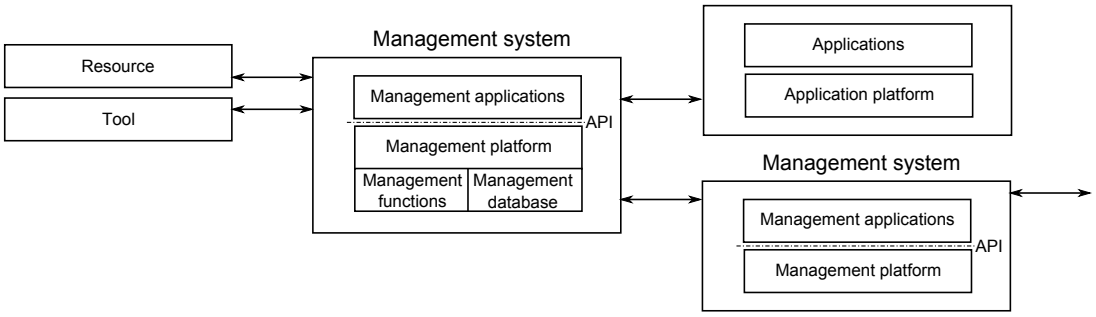

Figure 4.2: Management Platforms. Adapted from [HAN98]

- What functions the management has to perform (functional model).
- How the management is organized and structured with regard to overall management policies (organization model).
- What type of user interaction is desired (operational model).
- What are the operational specifications of the management system (operational model).

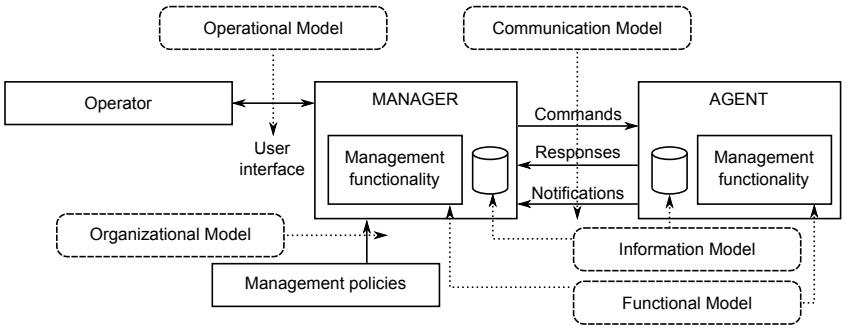

Figure 4.3: Management Architecture Models. Adapted from [Pat09]

Figure 4.3 demonstrates how these aspects mirror the above mentioned models within the overall management structure involving *managing (manager)*, and *managed (agent) entities* [Pat09], [HAN98].

4.1.1 Information Model

In order to solve management tasks successfully, a common view of managed resources and the information they provide is essential. In heterogeneous dynamic environments, this common view can be achieved by means of an information model. Under this model we understand a common formal framework used to describe managed resources and structure management-relevant information. Thereby, it is important that the level of abstraction is sufficient to provide a scalable and homogeneous view regardless of what kind of resource it is, where it is located, or how it is accessed.

Characteristics of the managed resources (physical or logical) accessible by management operations are typically represented by *managed objects*. Thus, the information model must establish basis for managed objects, their naming, state, behavior, access model, relation to each other, and operations on them. The notion of managed objects can incorporate dynamic, generic, and composed concepts. For example, apart from network components

and system resources and devices which are straight forward to be identified as managed objects, such abstract terms like virtual networks, connections, set up locations, domains can be handled as managed objects.

Management information base (MIB) represents a conceptual repository of management information, or speaking more generally a collection of managed objects describing a particular management domain. Different modeling approaches can be used. Thus, OSI information model [ISO93] just like CORBA object model follows the object-oriented approach, DMI uses, on the contrary, the data type approach (MIF). The notation for description of management information is also prescribed by the information model. E.g., OSI and Internet management use ASN.1 syntax whereas CORBA implies usage of IDL.

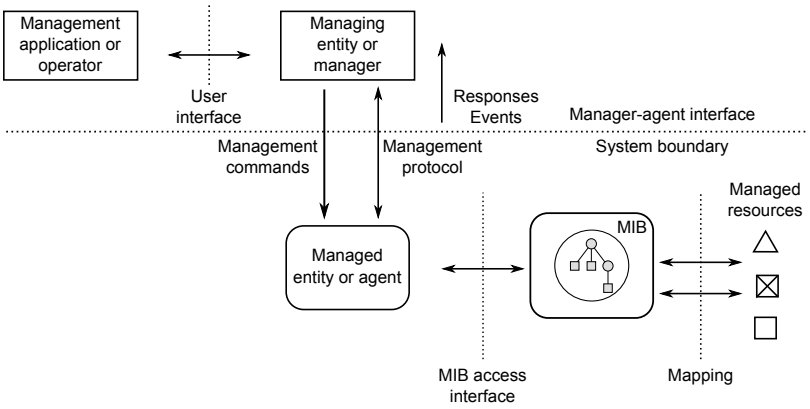

Figure 4.4: Information Model. Adapted from [HAN98]

Figure 4.4 demonstrates the common management process. Thus, management implies accessing managed objects by means of management commands. Transmission of the commands to corresponding managed entities is done via management protocols. As MIB access is standardized, the mapping of managed objects to resources is a local matter.

The identification of management-relevant information is not a trivial task. The selection of management objects and their characteristics as well as their abstraction level determine functional scope of the management system. We speak about bottom-up approach, if deriving management-relevant abstractions we proceed from given protocols, components, product specifications. In other words, one tries to answer the question "What information is available?". On the contrary, top-down approach proceeds from requirements of management applications, users or operators and derives relevant management information, i.e. answering the question "What information is required?".

4.1.2 Organization model

The organization model of a management architecture defines actors, their roles and how they cooperate with each other.

Cooperation models

Two main cooperation models are typical for management architectures: *manager-agent* and *peer-to-peer*. The *manager-agent* model adopts the common client-server paradigm. The manager requests the client to perform a certain operation. The client responses with the result of the operation. Suppose, the monitoring task is performed. In this case, the operation corresponds to a read access to the relevant managed objects. The manager (client) requests the agent (server) to perform a read operation. The response

of the agent includes the result of the read operation. In case of the controlling task, the operation corresponds to the write access to the certain managed objects. Depending on how operation semantic has been defined by the information model, the effect of the access will be returned by the agent to the manager (Figure 4.4). The client-server paradigm implies an *asymmetric* or *hierarchical* cooperation form between the management entities. The manager-agent model is adopted by e.g. OSI and Internet management architectures. The roles are not compulsory permanently assigned to the components. In practice, they can be dynamically reassigned depending on the current task.

Another cooperation form is *peer-to-peer* model which is based on the reciprocal relationship between management entities (peers). It is a completely *symmetric* cooperation form, which means that the flexible mutual operation performing and information exchange takes place. The peer-to-peer model is used by the CORBA management architecture.

Topological forms

In general, management entities are in many-to-many relationship to each other. A manager is responsible as a rule for a group of managing agents (*domain*). At the same time a managing agent can be assigned to several managers at the same time. Thus, concerning the topology of the management there is a variety of possible forms [Pav98], [HAN98] (Figure 4.5) .

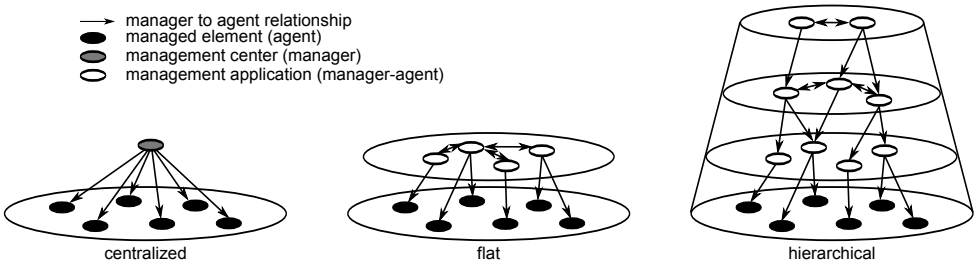

Figure 4.5: Management Topology Forms. Adapted from [Pav98]

The topological form can be characterized according to multiple aspects. The first one to mention is centralization. Assume a fully centralized management form, where a single manager is responsible for a row of managing agents. Its weakness is concentration of all the functions in one place, lack of scalability because of limitation on the number of managed resources as well as vulnerability because of the single point of failure. A significant advantage of this management form is, however, its simplicity. Alternatively, decentralized forms which employ multiple managers can be used. Thus, the managed resources are assigned according to a certain criteria to responsible managers. For example, these criteria can be their physical location, type, functionality, organization affiliation. Thus, each manager has its own allocated domain and is responsible solely for the resources of this group.

Another aspect is a hierarchical layout of the manager topology. We speak about a flat form, in case there is no relationship between distinct managers or they are of equal rank (peer). Otherwise, it is a hierarchical form, where subordinate relationship between managers exists. They speak about *manager of managers* concept. Being logically centralized, the hierarchical form still provides distributed control capabilities.

Manager affiliation is also a characterizing factor for the topological form. Thus, each managing agent can be assigned to a distinct manager or to the multiple managers (*network of managers*). In the latter case managers may be responsible for different management areas, e.g. security, performance, accounting. The main benefit of this form is that the

splitting of management areas improves the robustness of the management system. On the contrary, access to the same resources requires from the management system support of conflict detection and resolution.

	Centralized	Distinct manager affiliation	Intramanager coordination	Hierarchy support
Central management	+	-	-	-
Multipoint control	-	-	-	-
Multicenter control	-	-	+	-
Hierarchical management	+	-	+	+
Network of manager	-	+	+	-

Table 4.1: Topology Forms

Table 4.1 summarizes the above mentioned topological forms with regard to their properties. So, *central management* form evolves a single management system a set of managed resources. *Multipoint control* form partitions the resources into disjoint subsets which are controlled by separate management systems.

A general management paradigm bases on the *delegation* of tasks to other entities. Delegation is a process of "transferring power, authority, accountability, and responsibility" [MFZH99]. Also the technical management incorporates the delegation notion. The topological form walks along with the delegation direction used within the management system. *Vertical delegation* is typical of hierarchical paradigms. The two main kinds of vertical delegation are *downward* and *upward delegation*. Downward delegation supposes that a manager at level N delegates tasks to a subordinate at level N+1. On the contrary, in case a subordinate at level N+1 is out of due or is not capable of decision making, he delegates his tasks upwards to his manager at level N. *Horizontal delegation* is typical of cooperative paradigms (flat forms) where peers of equal rank cooperate with each other [MFZH99].

4.1.3 Communication model

Controlling and monitoring of distributed resources assumes exchange of information. Regardless of the cooperation model, management entities are supposed to exchange data while transmitting management operations and manipulating management data. The communication model of a management architecture specifies the protocol for exchanging management information between managing and managed entities. Thereby, three main tasks are covered by the communication model:

Controlling In order to operate on a resource, a corresponding control information must be transferred to the responsible managed entity. The communication model defines the protocol for transferring a control command from the managing to the managed entity and the corresponding answer from the managed to the managing entity, if applicable. Within the controlling task the communication is initiated by the managing entity.

Monitoring Monitoring resources implies regular status queries performed by the responsible managing entities. Similar to the controlling task, the monitoring task involves

the managing entity to initiate communication with managed entities and actively request desired information in a way specified by the communication model.

Notifications Notifications are used to asynchronously inform the managing entity about important events on the part of the system comprising the relevant resources. By contrast to the monitoring and controlling tasks, communication is performed in a one-way mode. The communication model defines the protocol for exchanging of event messages.

The communication model can also specify mechanisms to build subsets of services and protocols to perform the above mentioned tasks, i.e. profiles. Specifying protocols involves naturally defining syntax and semantics of corresponding data structures including protocol data units, data exchange formats, etc. The embedding of the protocols into the global architecture or protocol hierarchy can be *inbound* or *outbound*. Inbound embedding means that management communication takes place over the same transport network as the normal communication. Outbound embedding presumes usage of a logically separate transport network for the management communication.

4.1.4 Functional model

The functional model of a management architecture specifies the structure of management functions performed by the overall management system by means of management applications and tools. Some of the functions can be specified in terms of a dedicated management function area, the others are general enough to span multiple areas.

Specifying management functions comprises the definition of the following aspects:

- Expected management functionality
- Services providing the management functionality
- Management objects needed for the services realization
- Logical subsets of management functions
- Invocation conventions for management functions

Tight liaison with the associated information model is obvious. Thus, the description of management objects and invocation conventions assumes a common understanding relying on the information model. Thereby, invocation conventions include construction of external and internal interfaces for managers and management agents in order to provide the basis for open and expandable platforms. Especially management scalability is reliant on the widespread applicable, flexible configurable, and delegateable management functionality.

Functional decomposition of management allows a modular approach in development of management applications and tools. It must, however, face the issue of a certain overlapping of function areas. The overlap can be caused due to the following points:

Common resources The same resources can be used by multiple function areas. This, for example, may cause challenges originated form concurrent use of objects or conflict of objectives.

Mutual calls Management applications may need to call each other mutually and trigger execution of corresponding actions. Control of call sequence along with coordination of collaboration requires additional efforts by overall management.

Designing a functional model is usually a comprehensive task accomplished in multiple steps by integrating dedicated management aspects. The approach is in most cases to start from the standpoint of function areas and process the aspects in a gradual manner. The next section introduces the fundamental functional areas regarded for the management purposes.

4.2 Management Functional Areas

The ISO/ITU-T joint committee has worked out a Recommendation ITU-T X.700 (09/92) which turned into a widely accepted standard for network and systems management. According to it, the OSI Management Architecture defines in its functional model the following five functional areas: Fault, Configuration, Accounting, Performance, and Security Management (FCAPS) [Int92].

4.2.1 Fault Management

Faults cause the abnormal operation of the system which results in failing the operational objectives or system functions. The nature of the faults can be of persistent or transient matter and the manifestation can be in form of particular events in the operation (e.g., errors, malfunctions, performance issues). The sources of faults are enormous in their variety starting from network components and communication channels to software components and end systems. The faults themselves can be presented as a binary variable signaling about the system state (e.g., "failed", "okay") or as a numeric indicating the measure of fault extent. In the latter case fine-tuned thresholds are to be worked out in order to recognize faulty conditions.

Fault management involves reactive and proactive measures in order to detect, isolate, and eliminate these faults in the system behavior. It is also to mention that faults occur often combined together. Simultaneous handling of multiple faults is unavoidable for effective fault management.

Fault detection relies on periodical checking the system health. In case a fault is detected, a corresponding notification is supposed to take place. Notification conveyance may be done by the faulty component itself or by any other monitoring component. In certain cases not only the management system is to be notified about the detected fault. Also the end user may need to be informed about it. Irrespective of the mode of checking (push or pull mechanism) an intelligent balance between real-time notification and costs related with processing and transport has to be found [SJCC07]. Management tasks providing for the fault detection usually include monitoring system state, maintaining and periodically studying error logs, carrying out diagnostic tests, etc.

The next step after the fault detection is usually the *fault isolation*. Fault isolation includes diagnosing fault causes and tracing their roots up to potential conceptual weaknesses. Once a component fails, a set of fault notifications can be generated and registered by the management system. A strategy is needed in order to separate the actual faulty component from the bulk of possible cascading failures. In other words, the main aim is to localize the root cause, so that point can be found, where a corrective action can take place. This can turn out to be a physical component or a substantial weak point in the system design. Some algorithms and techniques for fault isolation in TCP networks are presented in [Cla82], [KP97]. Among others the following management instrumentals for fault isolation are useful: fault signatures, pattern recognition, and classifiers [Ise06], model-based reasoning techniques [Ise04], [VRYK03], rule-based approaches [KAC+05], [SZ08] etc.

Once the fault is isolated, the next step is to eliminate it. *Fault elimination* includes all sorts of corrective measures to return the system to the normal state. In case the correction of the fault is not possible or takes more time as allowed, procedures are to undertake in order to compensate for the faulty behavior. Management tasks for fault elimination vary in their complexity and nature: e.g. recovery actions can integrate arranging for resets and restarts, providing for redundant components, catering for clearing processed alarms, etc.

To take a step forward in the fault management, a *proactive* handling is needed. Knowing about a critical situation in advance, allows the management system to apply countermeasures for preventing the fault occurrence [PN04], [SLL+08], [CMK05]. If the fault is not to be avoided, the system can at least provide for repair mechanisms or fallback solutions in order to minimize the time-to-repair. A comprehensive work on algorithms used for long- and short-term prediction in systems management is presented in [VAH+02].

4.2.2 Configuration Management

As defined by ITIL, *configuration item* means any artifact that is to be monitored and controlled in order to deliver an IT service [AXE11]. Configuration items may include include physical, logical, as well as conceptual entities like e.g. hard- and software, network and its components, databases, services, SLAs, documents, user data, buildings etc. Irrespective of their dimension, configuration items are regarded as the smallest units which are designated for and treated like a single entity within a *configuration*. Thus, [AXE11] adheres to a service-centered definition of *configuration management*:

> "The process responsible for ensuring that the assets required to deliver services are properly controlled, and that accurate and reliable information about those assets is available when and where it is needed."

The above mentioned Recommendation ITU-T X.700 (09/92) states that the main task of the configuration management is to identify, control, collect data from and provide data in order to prepare, initialize, start, provide for the continuous operation of, as well as terminate services [Int92].

Another definition of the configuration management is suggested by ISO [ISO10]:

> "A discipline applying technical and administrative direction and surveillance to: identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements."

An essential logical partition into *static* and *dynamic* configuration management realms is obvious [KM85], [CCSZ03], [SMM+12]. The scope of responsibility of static configuration management comprises measures for providing initial configuration of relevant components, system starting and stopping procedures, provisioning of components. It also performs inventory tasks, records, reports, and traces system state. These are more or less standard procedures that are independent of the environmental conditions. Typical use cases for the static configuration management comprise e.g. software distribution, network topology setup, access control information management, backup execution, upgrade execution.

Dynamic configuration management implies actions that are context-aware. That means that the management decisions directly depend on the current state of the environment. Therefore a comprehensive planning beforehand is essential. The usual outcome of the changing conditions is that the management system should adjust to them and reconfigure the managed system in order to meet the predefined requirements. Reconfiguration according to the environment establishes a basis for the reactive and taking a step forward for the proactive system behavior. Among typical use cases for the dynamic configuration management are such tasks like network topology reconfiguration, substitution of failed components, network roots establishment at runtime, selection of appropriate services due to the agreed SLAs, etc.

4.2.3 Accounting Management

There are many definitions of *accounting*. Thus, American Accounting Association (AAA) defines it as [Dru92]:

> "The process of identifying, measuring and communicating economic information to permit informed judgments and decisions by users of the information."

In his work, Needles [NP10] provides another definition of the notion of accounting:

> "Accounting is an information system that measures, processes, and communicates financial information about" economic entities.

Inspecting the used terms reveals that the attribute of *economic information*, its *measurement* and *communication* as well as its usage for *making decisions* plays a central role in the accounting process.

Transferring the accounting notion to the technical management, the attribute of economic entities refers to resources and services which can be provided on the commercial basis. Following this, ISO determines *accounting management* as [Int92]:

> A way to identify costs arising from the usage of resources and to establish charges for it.

To follow up the definition of AAA, it is to decide what users of the accounting information exist. Firstly, the resource provider has an overview of what resources have been consumed, by whom, and in what extent. Secondly, the resource user is to be informed about the incurred expenses. In order to enable this, tariff schedules are to be associated with the resources in use. Further, accurate logging of resource usage must take place in order to allocate their usage to the users.

Thereby, cost allocation strategies and procedures should be the subject of predefined *accounting policy*. The accounting management system is to be configured to enforce these policies and not to perform on its own initiative.

The ability of accounting management to report, allocate, and administrate costs brings further benefits about. Thus, it allows a better SLA adherence by means of advanced controlling of accounting limits. Runtime control of currently arising costs and remaining buffer enables speedy reaction and intervention in sufficient time. Provided information about costs and charges, billing procedures as well as income statistics calculation and reporting also benefit through these sophisticated functions [Pat09].

Hegering [HAN98] divides accounting management functions into four main groups of functions: *usage management* functions, *accounting process functions*, *control* functions, and *charging* functions. Usage management functions inclose all sorts of functions related to collection, recording, and maintaining statistics on distributed resource and service usage data based on monitoring and metering. Accounting process functions represent administration, monitoring and surveillance of resource usage data. Control functions include functions for supervision, administration and controlling of any kind artifacts and processes: tariffs, schedules, records, data transfer and storage, etc. Charging functions address calculation of charges, production of bills, contracts and payments processing functions.

Accounting management can be successfully used to leverage planning of resources and services. Its outcomes and deliverables may help with optimization tasks during the system design phase. It is obvious, that accounting management bears a close relationship to service and business management [HAN98], [Jos08].

4.2.4 Performance Management

Speaking of system performance, it is firstly to define what output and effort of the system are to be expected. Thereby the commonly used measure is *quality of service (QoS)*. On the basis of the definition of the Recommendation ITU-T E.800 (09/08) [ITU08] which defines the term of quality of service in terms of telecommunication services, we generalize this notion as:

> The totality of observable and/or measurable characteristics of a service that bear on its ability to satisfy stated and implied needs of the user of the service.

Well defined observable and/or measurable characteristics are referred as parameters. The selection and specification of such QoS parameters is not a trivial task. Once the QoS parameters are defined, they can be used for specifying service level agreements (SLAs) which actually express stated and implied user needs concerning the service. Providing the service with the desired QoS, permanent performance control, as well as subsequent traceability are the next steps to undertake. *Performance management* copes with these task. We define the term of performance management as:

> A process that defines the QoS provided by the system and enforces measures to guarantee and provide evidence that the QoS complies with the agreed SLAs.

Thus, the responsibilities of the performance management include *performance monitoring*, *analysis*, *evaluation*, *reporting* functions [HAN98]. Performance monitoring functions comprise functions for monitoring resources in order to discover existing soft spots and exceeding of limits in performance. Performance analysis functions concentrate on prediction of potential bottlenecks and performance problems by performing required measurements and calculations. Evaluation of recordings and logs of system activity and performance malfunctions adheres to the evaluation functions. Reporting functions include composing and providing all sorts of reports concerning the system performance.

Similar to the accounting management, performance management can be used while planning resources and services of a system. By means of analytical and simulative models, performance management can e.g. test the system behavior, check up new configurations, tune management procedures and instrumentals, etc.

4.2.5 Security Management

Security aspect in the IT addresses mostly two main points: *functional security* and *data security*. Functional security includes system availability and functional correctness, whereas data security includes confidentiality and integrity of data [GKS+85].

The OMG in "CORBAservices Specifications" [OMG02] states that security aims the system to be protected from unauthorized information access or interfering with its operation. These security breaches compromising the system have become numerous and various lately. Thereby they can be of deliberate or accidental nature. The examples of the common threats identified by the CORBA security specification are bypassing security controls, eavesdropping on a communication line, user masquerading, tampering with communication, etc.

The amount of protection and undertaken security measures depends on the value of the assets that are to be protected as well as on the potential treats that are to be expected. Working out an adequate level of protection is a non-trivial task which has become an indispensable part of system planning and design process.

Being a central asset to be protected, information has value only if it is correct and the right people can access it at the right time. Thus, according to [OMG02], security is concerned with the following targets:

Confidentiality Only authorized users gain access to information. Disclosure to unauthorized parties can be viewed as a serious menace to the sensible information and the system itself. Disclosed data can lose in value for the data owner or even turn hazardous. The loss of confidentiality is irreversible. It is to take into account that acquisition of confidential information can be done directly or indirectly. Direct acquisition means unmediated access to the stored, processed or transmitted confidential data whereas indirect access implies retrieval or extraction of confidential data on the basis of other information [FJ02].

Integrity Only authorized and entitled users can modify information in intended way. Protection of information from being modified by unauthorized parties is a key task, since modification of sensible information may cause a serious damage. Similar to the confidentiality, malicious or accidental modification of stored, processed, or transmitted data should be considered. This time, however, provided certain recognition and safeguard mechanisms, a correction of modified data is possible, i.e. it can be reversible [SWZ05].

Accountability Responsibility for security-relevant actions lies with users who undertake these actions. Holding users responsible for their actions allows to achieve security by deterrence in spirit of traditional law practice [JJPR09]. Accountability relies on the after-the-fact verification which presumes widespread logging as well as auditing log records afterward. Non-repudiation as a special case of accountability implies that the actions of a user can be traced uniquely to the user, i.e. in addition to logging of user actions an unambiguous proof of authorship can be provided.

Availability Denial of system usage for authorized users has to be prohibited. It is to ensure that authorized and entitled parties were able to access information when needed. Availability loss can be of malicious or accidental nature. E.g. DDoS attack compromises maliciously the availability where as outage of power causes an accidental loss of system availability.

Security management evolves all sorts of measures and mechanisms in order guarantee and support application of defined security policies. It is pervasive and operates on different abstraction levels starting from physical devices through security services up to abstract organizational arrangements. According to the Recommendation ITU-T E.800 (09/08) [ITU08] its functions include:

- definition and control of security services and mechanisms
- definition, distribution and storage of security-relevant information
- reporting and warning on security-relevant events

Security management measures are also to be found during the system design phase, when corresponding prevention procedures and mechanisms are planned. Similar to other functional areas, a close relationship with configuration management is obvious.

4.3 Automated Management Challenges

The complexity of modern distributed systems challenges the automated management solutions in multiple architectural issues. Thus, in [KDR+06] they point out the following aspects, which are to be handled by the management systems: autonomy, scalability, heterogeneity and administrative isolation.

4.3.1 Autonomy

The need of eliminating the human intervention has become a topic of ongoing research during the last decades. In 2001 IBM introduced the term of *autonomic computing* which was used to describe systems that can prove self-management properties under varying and unpredictable conditions [KC03], [IBM05]. The intent of self-management is to free system administrator from details of system operation and maintenance after providing once or at least relatively rarely high-level management objectives or guidelines. Within the vision of IBM, self-managing systems and devices naturally and unremarkably adjust and upgrade to the changing environment without us knowing any details or technicalities of migration or transformation [DSNH10].

Self-management properties include abilities for self-configuration, self-healing, self-optimization, and self-protection [HM08], [KC03].

Self-configuration The dimensions and complexity of modern systems makes the process of initial installation, configuration and setup is a complex, time-consuming and error-prone task. It is targeted that this task is performed by the system itself according to specified high-level goals. That means, the main aims and needs are specified but not how to achieve them. Introducing a new component should be performed transparently for the rest of the system. The component is to be configured in accordance with its environment and the system is to integrate the new component seamlessly.

Self-healing Distributed and heterogeneous nature of systems makes errors and problems during deployment and runtime very likely. Thus, detection and diagnosis of these conditions is inevitable. The system should fix the problem, if possible, or at least handle the situation appropriately. No further harm should occur and the corresponding notification must come about. In terms of self-healing, reactive and proactive behavior is desired. Thus, the hazardous conditions should not only to be handled directly on arising in a timely fashion, but also a certain looking ahead should be striven. Early signs of possible failures are to be recognized and dealt with in advance.

Self-optimization Large and complex systems involve a big amount of resources. The number of tunable parameters which are to be tweaked is tremendous, correspondingly. That is why the use of them should be properly planned and thought through demanding a great expertise and know-how. Optimizing the resource usage is a part of it. Proactive behavior with constant seeking for ways to improve the operation should be brought forward in this case, too. With respect to the defined requirements, automatic monitoring and control of resources should facilitate system performance by seizing every opportunity to make its behavior more efficient.

Self-protection Hazards of the systems originate not only from malicious attacks but also from unintended failures and errors. Despite existing automated common protecting procedures, human intervention in order to protect the system or handle the hazardous condition is often necessary. Thus, the ability of self-protection gains in importance. Self-protection measures include defending against harmful attacks, failures and problems arising from them as well as avoiding such precarious situation in the first place. In order to prevent fault and damage, weak points and vulnerabilities are to be anticipated. Then again proactive features enhance systems substantially.

The above mentioned self-management properties are a core of the autonomy aspect. They help to reduce costs of systems management as well as to improve its efficiency. Autonomic management systems tackle administration complexity that is out of reach of a human administrator be that due to his absence or his inability to manage the situation by hand.

4.3.2 Scalability

Large-scale distributed systems must face the problem of constantly growing number of system components as well as permanently increasing amount of work the systems must deal with. In this context, the issue of *scalability* arises. A rigorous wide-spread definition of the scalability term as a property of a system is, however, absent [Hil90], [DRW06]. In most cases it is application case dependent and needs a concrete definition of specific requirements and metrics.

In [WG06] an extensive analysis has been presented of what is meant by scalability as well as on factors to be considered in order to evaluate system in terms of scalability. Thus, two main uses of the term were identified. The first one is rather informal, commonly-used but at the same time paying no particular attention to the mechanisms or strategies of addressing the case the workload increases beyond a threshold:

> "Scalability is the ability to handle increased workload (without adding resources to a system)."

In fact it actually means for a system "to continue to function with acceptable performance when the workload has been significantly increased" [WA02]. In order to improve the scalability in this sense, one can enhance used algorithms, apply more effective data structure, or just reduce the amount of services the system provides when it is heavily loaded.

The second definition identified in [WG06] addresses more issues:

> "Scalability is the ability to handle increased workload by repeatedly applying a cost-effective strategy for extending a system's capacity."

Thus, a strategy for capacity extending as well as how often a capacity extending can take place are subjects of interest. Improving scalability in this sense means searching for a better strategy for adding capacity or catering for a better cost-effectiveness.

In general, overcoming performance limits by adding resources can be done using the two following approaches:

Vertical scaling In case physical resources are added to an existing node, we speak of scaling vertically, i.e. *scaling up*. The approach requires no changes to the architecture but is limited in the scalability level due to hardware restriction.

Horizontal scaling Additional nodes with physical resources are added to the architecture means that the system scales horizontally, i.e. it scales out. In this case the workload is to be dispatched between additional nodes. Thus, the architecture has to be designed to be extendable in this way as well as a special load scheduler (or load balancer) is required. Horizontal scaling is, therefore, more sophisticated by design but is eventually the one and only way for scaling due to limited hardware capacity of a single machine.

Adopting the general definition described above the scalability of a management system concerns its ability to accommodate the growth of the managed system. Thereby, the growth of the managed system may apply to size, number, and diversity of managed resources.

4.3.3 Heterogeneity

Dealing with large managed systems assumes that the management system should support a wide spectrum of managed objects. The issue of heterogeneity of the managed systems arises. It concerns temporal, geographical, structural as well as vendor specific aspects.

Dynamics The management should be able to operate under the circumstance of high dynamics. Constant changes of the environment as well as the requirements of the

managed system demand for it. Moreover, new objects pop up, the old ones disappear. Similarly, new connections between objects are established and dissolved. Objects can build dynamically new logical groups that are to be managed as an ensemble. At the same time these groups can vanish. Such changes in the infrastructure of the managed system should be mastered by the management system.

Location transparency Managed objects can often be spread noticeably in their geographical position. It is desired that the management system can handle this locational diversity and address distributed managed objects in the same manner as it addresses the local ones. Thus, location transparency is eligible.

Legacy assets The most of the managed resources include diverse devices, applications, and systems, which can vary extremely in their nature. Thus, they are mostly manufactured by different vendors and have as a result some certain specifics. E.g. they can use proprietary protocols, bring no standardized management information along, and have no description of management objects at all. It is natural, that in most cases managed resources have not been tailored to be managed by a dedicated management system. In order to compensate for the by design missing management support, the management system should provide required mechanisms and instrumentals.

4.3.4 Administrative Isolation

Contemporary large-scaled systems are so huge that it often makes sense to arrange their elements into logical groups. From the management point of view, an interest of logical partition of managed resources exists, so that different management views can be implemented [HAN98].

A concept of a *management domain* arises. Thus, a management domain can be defined as:

> "a collection of managed objects which have been explicitly grouped together for the purposes of management" [Slo94].

Being more precise, a management domain can be viewed as a managed object itself which holds references to managed objects belonging to this managed domain. In this case, a managed domain is a parent domain whereas the managed objects are its direct members. In its own turn, a domain as a managed object may be a member of another domain. So, it is called a management subdomain and its members are indirect members of its parent domain. Thus, a notion of hierarchy is brought in.

The criterion of domain membership can vary depending on the purpose for which the domains are defined. For example, geographical location can be used as a characteristics to group management objects into management domains. Another example is the organizational affiliation as a grouping criterion. In this case enterprise's, institution's, and facility's hierarchy or structural distribution can be reflected in the management domain fragmentation. Eventually, domain can be used to address management objects of a a common art or type, e.g. all the routers, software processes, sensors.

Of special interest is ability to express complex structures by using multiple domain membership criteria at the same time. So that advanced filtering for addressing of designated group of management objects is possible, e.g. the software process running on hosts of an operating department belonging to a particular enterprise.

Grouping together for the purpose of management implies ability to apply common management operations or actions for designated sets of management objects. In conjunction with management domain, the *policy* concept is brought in [SM88]. The next chapter provides an overview of the policy-based management field.

Chapter 5

Policy-Based Management

One of the administrative approaches to the technical management which aims to undertake the given endeavor and strive for certain goals and missions is known as *policy-based management*. Being an established and well-accepted solution, policy-based management is referred as a way to maintain order, consistency and uniformity of not only the management targets and objectives but also of the way the management is established itself. Policies represent logic which determines the behavior of the managed system. The operation of computing resources is conducted according to the predefined rules which express this desired behavior. The operation is supposed to be dynamically adaptable in order to allow system reactivity and sometimes proactivity due to constant changes in requirements and conditions at runtime. Due to policies, a new level of autonomic computing has arisen. Thereby the variety of policy-based management approaches is extremely great starting from deeply tied to the application domain and dependent on the field of operation to those completely generic and suitable to be applied in multiple operation environments.

5.1 Historical Perspective

The historical perspective of research efforts on the policy-based management is widely presented in [BA07]. The authors highlight not only the chronological order of events but also concentrate on different functional areas of policy-based management. The roots of the policy-based management paradigm originate from the security models which date from the late 1960's. The focus of research was emphasized on security considerations targeting from the single machines via enterprises through to networked environments. The policies were regarded as abstractly formulated rules which were used to regulate the access control. Within the access control process a selective restriction of access to resources takes place. The decision about granting the permission to access a resource is done according to the predefined security policies. They can be of different form and nature, concerning a variety of criteria as well as means and specifics of security assurance. A scheme for specifying and enforcing the security policies, i.e. a formal representation of security policies and their functions, was provided in that case by the security model. The low-level functions which implement the controls on the soft- and hardware level were defined by the corresponding predefined security mechanisms. By establishing the formal representations and the appropriate mapping between them, a proof of properties concerning the security issues could take place.

The introduction of *role-based access control (RBAC)* model by [FK92] has gained a wide acceptance in the field of security. Its refined and extended model was accepted in 2004 as an ANSI standard ANSI/INCITS 359-2004 [Inf04]. Though the RBAC model does not describes the usage of policies directly, the security model promotes their employment within the role-based access control by providing the basis for its specification and enforcement. Moreover, the security model of RBAC has been adopted by several policy specification languages (e.g., Ponder, XACML).

In the mid 1980's the aspect of policy research drifted to the network and distributed systems management. The research work of D.C. Robinson, J.D. Moffet, and K. Twidle supervised by M. Sloman has proven great promise of policies as a management paradigm. Especially the notion of domains arises in this context [RS88]. The domain model is used to specify and structure policies. Policies represent a set of rules associated with the corresponding domain. Domains adhere to a manager responsible for his sphere of interest. Thus, a scalable and flexible management of large distributed computing systems can be achieved by using subdomains and touching domains. Further, different functions of management like Fault, Configuration, Accounting, Performance, and Security (FCAPS) are addressed by using a uniform interface.

Later on, as the Internet came to the fore, the use of policies for routing and networking has begun. In RFC 1102 [Cla89]the need of policy routing was addressed. The author proposes that selection of routs should be done according to predefined resource policies which refer such considerations like restriction of usage of network resources to certain customer classes or some measures of route. Policy-based routing came into usage with emergence of routing protocols BGP (1989), IDRP (1994) and IDPR (1993). The research field of network policies has turned out to be very promising, so that IETF and DMTF have started working on standardization of policy framework. A policy management architecture for policy-based networking has been worked out, which later on was accepted and used widely apart from network background. So, in 2001 an object-oriented information model for representing policy information Policy Core Information Model (PCIM) was published in RFC 3060 [MESW01]. Extending the basic Common Information Model (CIM) concepts for policy, PCIM describes a "core" model and cannot be applied without domain-specific extensions. It includes the basic concepts of policy groups, rules, conditions, actions, repositories as well as their relationships. In RFC 2748 [DBC+00], IETF has also proposed the Common Open Policy Service (COPS) protocol employing a simple client/server model for supporting policy control over QoS signaling protocols. The research of IETF/DMTF was followed by other researches studying different aspects of policies in IP networks. E.g., alternate policy architectures have been introduced addressing QoS in Differentiated Services (DiffServ) networks.

The usage of policies for network management was continued with the usage for business-driven management. The interest of major industries was drawn to the ability to provide an autonomic management where concrete device configuration and control is guided in accordance with business-level expert guidance performed by humans. Thus, IBM has started an initiative for researching autonomic management for large-scale computing systems and studied policy notion in these terms. Policy Management for Autonomic Computing (PMAC) framework has been developed and was determined to "help developers simplify the automation of IT and business processes and makes applications more self-managing and self-configuring" [IBM05]. The intelligent control loop is implemented in the PMAC architecture as an autonomic manager (AM). AM is responsible for monitoring, analyzing, planning, and executing functions in accordance with predefined policies [Kir05]. Later on, HP Research Labs have started works on a business aware management of policy-based systems and applications. In their research they have addressed the lack of business and service level context to drive policy-based decisions at runtime and proposed a business-driven management framework (BDMF). BDMF supported an interaction mechanism between the more proactive business aware management layer and the underpinning reactive policy-based resource control layer [ASB+06].

5.2 Policy Definition

Considering the historical perspective, the definition of policies has varied a lot depending on the application domain, usage of abstraction levels, form, and management mechanism. In the following, the main concepts and terms arising with policy-based management are presented.

In general, *policies* were defined by Sloman[Slo94] as

> "rules governing the choices in behavior of a system".

Adopting the Sloman's definition, however, allows a certain ambiguity of policies usage. The behavior of a system can be interpreted as application flow and policies in this case would be used directly to code the application.

Another definition which focuses on the management purposes was proposed by Lupu:

> "policies are a means of specifying and influencing management behavior within a distributed system, without coding behavior into management agents" [LS99].

In his work, Wies [Wie94] points out that a large number of related terms surround the notion of "policy", such as "strategy, goal, vision, direction, mission, process, tactic, procedure, plan, scheme, course, and guideline". He places policies as a link between the corporate and technology management (Figure 5.1).

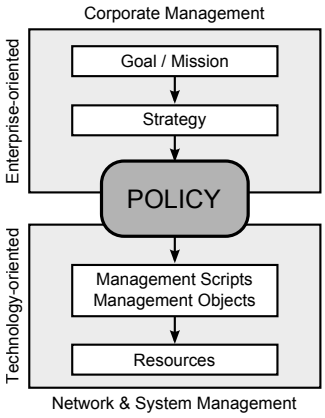

Figure 5.1: Policy Through the Loop of Enterprise Management. Adapted from [Wie94]

From the enterprise-oriented point of view, the corporate goal can be achieved through a number of long-term strategies. Policies are derived from these strategies and define the organizational measures and tactics for a particular department. From the technology-oriented point of view, policies act on their targets, i.e. management objects which are defined on the concrete resources. Thus,

> "policies are derived from the goals of management and define the desired behavior of distributed heterogeneous systems and networks" [Wie94].

In doing so, they rather "define *what* management has to accomplish", but not *how* to do it, i.e. "what technical instruments, management protocols, functional and information models are used" [Wie94].

Different as the definitions are, they all adopt the intention of policies as it stands: policies are persistent specifications of objectives and not immediate, one-time decisions to perform [MS93]. In other words, policies are supposed to be reused.

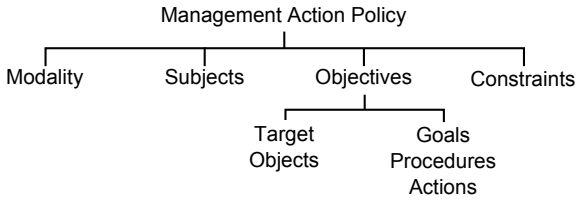

Figure 5.2: Policy Attributes. Adapted from [MS93]

A generic policy disposes of the attributes presented in Figure 5.2: modality, subjects, objectives, and constraints [MS93].

Concerning the modality, two different types of policies are to be distinguished: *imperative* and *authorization* policies. Imperative policies are used to express an imperative obliging (positive) or inhibiting (negative) a management agent to perform a certain action. By means of authorization policies management agents are permitted (positive) or prohibited (negative) to carry out an action on a corresponding target object.

Policy subjects specify actors (human or automated) to whom the policy is associated. Depending on the policy modality, policy subject is responsible for performing management actions or is provided with the legitimate power to perform an action to carry out policy objectives.

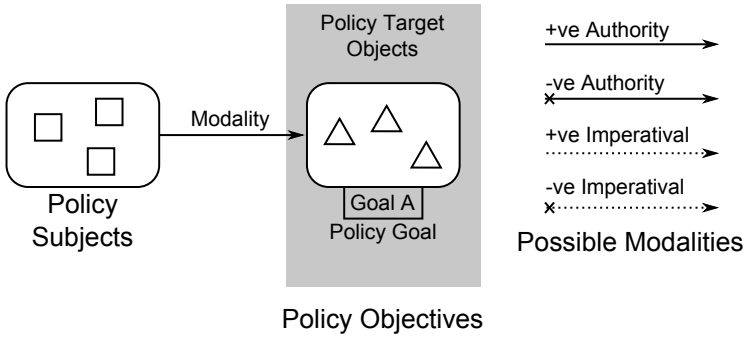

Figure 5.3: Generic Policy Notation. Adapted from [MS93]

The entities at which the policy is directed are known as the policy target objects. It is a common practice to define policy subjects and targets in terms of sets (e.g., enumeration, predicate). Thus, the concept of management domains was introduced by Sloman and Mofett working group in order to group policy target objects to which common policies apply (Figure 5.3).

In order to express aims of a policy the term policy objective is used. They are are defined as a pair of goals and target objects. Thereby, goals define either a high-level goal or a procedure (i.e. a defined sequence of actions).

Finally, constraints the express applicability of the policy. They allow to formulate the terms of usage for the policies concerning some predefined conditions, e.g. time, duration, location, cost.

In this work, we regard policies from the technical management point of view and adopt the following definition of policy term:

> Policies are reusable management instruments applied by the automated management system in order to guarantee the managed system to conduct in a flexible but constrained manner. Technically, policies are formulated on management objects and enforced by the management agents. As a whole, they

define organizational measures in order to achieve the desired management goals throughout the whole field of functions (FCAPS).

5.3 Policy Abstraction

Early studies of Maullo and Calo [MC93] have highlighted general policies in terms of several levels distinguished by the degree of precision, communication manner, and the extent of accommodation. The term of *policy hierarchy* has been coined. The authors address in their work the following distinct abstraction levels:

- Societal Policy (Principles) - class of policies applied to human interactions in general and prescribing modes of conduct.
- Directional Policies (Goals) - policies dictating the directions in which the processes are routed such as organizational or corporate goals.
- Organizational Policy (Practices) - class of policies which interpret corporate goals and directives and turn them into the form of developed plans, formulated approaches, contractual agreements met.
- Functional Policy (Targets) - functions to be accomplished to carry out specified organizational policies formulated in terms of the functional areas such as integrity requirements, workload targets, quality measures, configuration specifications.
- Process Policy (Guidelines) - class of policies presenting sets of processes to be maintained and formulated in form of pseudocode, macros, schemas, programs.
- Procedural Policy (Rules) - statements of policies exist only in form of executable encoded procedures.

Later on, this idea has been applied to the policy-based system management. Thus, RFC3198 [WSS+01] states that the policy representation can vary from high-level abstract form to low-level device-specific configuration parameters.

In order to illustrate this aspect, we regard the following abstractly formulated policy: *"save on power consumption of the system"*. Applying this form of the policy in practice demands, however, a more concrete formulation. E.g., the following concrete actions could be used in order to achieve the above mentioned goal:

- *"Use saving power plan"* The hosts used within the system are to follow a special power plan which saves energy by reducing performance where possible. This will ensure the appropriate power settings of the hosts.
- *"Reduce standby power"* System on standby should consume as little power as possible. This will provide for decreasing latent costs in matters of power consumption.

Further, *"Use saving power plan"* can be expressed more precisely by giving a concrete specification what actions under which circumstances have to be enforced. E.g., a saving power plan could make use of the following measures:

- *"Put system in a standby mode, if idle"* In case of inactivity for a certain period of time, the system should be put in a standby mode to save power. Regard the following exemplary rule: "if $T_{idle} >= 3600$ then $S_{standby} = on$".

- *"Turn off monitor after a period of inactivity"* After idling for too long, monitors should be turned off. Exemplary, this can be specified by means of the following rule: "if $T_{idle} >= 300$ then $VBETOOL_{dpms} = off$".
- *"Reduce monitor brightness"* Configure monitor brightness to the minimum user can stand, e.g. the following configuration can be used: $M_{xbacklight} = 30$.

Similarly, *"Reduce standby power"* can mean the following concrete measures:

- *"Select devices smartly"* While planning or reconfiguring the system at runtime, devices which consume in a standby mode less power are to be preferred. In practice, this can mean just checking a certain management variable: $POWER_{SB} <= 0.5$
- *"Use devices smartly"* In case, the device is not needed for a long time period, it should be unplugged. This form expresses a common rule realized normally by a human.

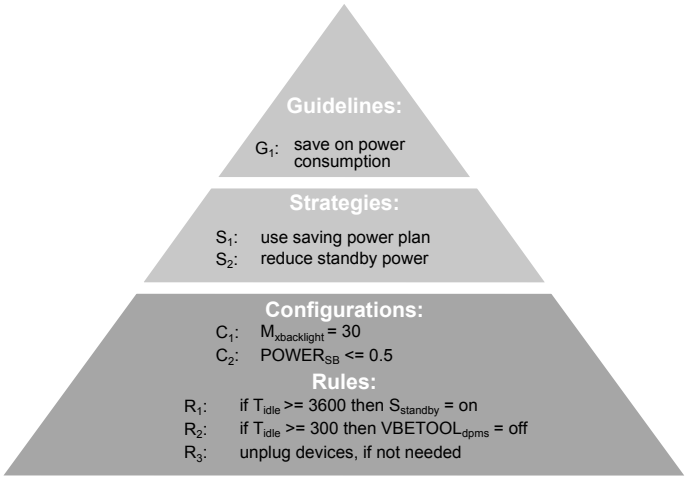

Figure 5.4: Policy Abstraction Levels

The form of expressing policies can considerable vary in its abstraction level (Figure 5.4). In the example above different levels were demonstrated: from high-level abstract requirements and targets down to low-level configurations and instructions for use.

It is common, that policies are formulated in the first place by human experts. Thus, the high-level form is mostly preferred for expressing the management targets or requirements. It deals with abstract notions, concepts, and terms. The policies contain domain specific vocabulary and are understandable, apparent, and obvious for the domain expert. On the opposite side, the automated management system itself handles normally with the low-level form of policies. It demands for concrete formulations applicable for the management environment irrespective of the managed system application domain. Thus, the low-level policies operate on configuration parameters, management variables, and operands. They express values that can be quantified by measurement or qualified by categories.

In this regard, [DJS07] and lately in the work in progress by [SHvdM17] the authors work out the term of *policy continuum*. Different constituencies of users like business users, developers and admins use different concepts and terms in their own specific language. Thereby they contribute to the policy definition at various levels of the policy continuum. A unified information model allows to build a "consensual lexicon that enables terms from one language to be mapped to terms of another language" and to support the model-to-model translation from a business level down to a technical level.

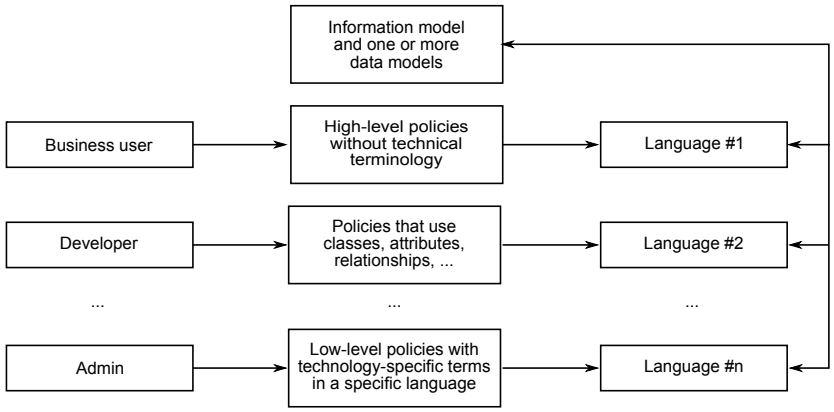

Figure 5.5: Simplified Policy Continuum. Adapted from [DJS07]

5.4 Policy Refinement

In order to bridge the gap between requirements specified and operations to be enforced a certain refinement should occur. Translation between different levels of "abstraction" from high to low is referred as *policy refinement*. Policy refinement process is a complex task, which requires information other than policy [WSS+01].

To carry on with the example above, in order to translate the high-level policy *"save on power consumption of the system"* into concrete variable assignments of single devices, e.g. *"if* $T_{idle} >= 3600$ *then* $S_{standby} = on$*"*, information is needed, what devices form up the system and how they are to be configured. Thus, expert knowledge is required as well as the managed system runtime data. It is also to mention, that this deduction demands multiple refinement steps. Thus, in the first step a concrete strategy *"Use saving power plan"* can be derived. The second step would bring parametrized rules and configurations to be used by the management systems.

Manual refinement process is error-prone and extremely tedious. Firstly, in a large distributed system a number of management policies can be very big. Secondly, the requirements are often specified by multiple stakeholders. Thus, conflicting requirements, omissions as well as errors in the requirements can result in *policy conflicts* [LS99]. They may arise primarily in case multiple policies refer to the same object. It happens oft, since the requirements to the systems belong to different management categories like fault, configuration, account, performance, and security. So, conflict detection and resolution turn out to be inevitable tasks during performing policy refinement.

Taking into consideration these issues, automated solutions for policy refinement have become a subject of research during the last decades. Different refinement approaches emerged.

5.4.1 Sloman et al.

In early 1990's, Moffett and Sloman [MS93] presented their work on *policy hierarchies* as a basic structure formed by refining general high-level policies into a number of more specific ones. The authors argue that there can be no fully automatic way of refining policies, but state an urgent need to have a "means of codifying the refinement steps and, as far as possible, generalizing them". Thus, their policy hierarchy model contributes to this.

To be specific, they identify several different relationships which may exist between policies in a hierarchy:

Partitioned Targets

While goal is the same, the target set of the lower-level policy is a subset of the target set of the higher-level one. In order to ensure that the target partitioning is complete, the whole target must "covered" by lower-level policies. E.g., all the assets of a company are to be protected, i.e the assets are the target of a company's policy (higher-level). The company has several departments to which these assets are assigned. Each department has its own department's policy (lower-level) which target is the department's assets, i.e. the subset of the company's target.

Goal Refinement

The goal of a higher-level policy is refined into one or more lower-level goals. Thereby the higher-level policy's goal and the lower-level one's refer to the same target. E.g., a higher-level policy and a lower-level policy have the same target, the files belonging to a certain department. The goal of the higher-level policy can be refined from *"Protect from loss"* into the lower-level *"Backup weekly"*. Obviously, no fully automated method of goal refinement can exist. It is up to the system planner to decide how to refine a goal and what actions are required in order to fulfill it completely. In this example, another lower-level goal is considerable: *"Backup daily"* or or a combination of *"Backup weekly"* and *"Inspect the content of the backup store monthly"*.

Arbitrary Refinement of Objectives

The arbitrary refinement of objectives means that the goal and target are quite different from the higher-level objectives. E.g., the higher-level policy *"Protect the confidentiality of files"* can be refined into the lower-level one *"The users must use a secure login procedure"*, whereby there exists no direct relationship between the subjects or targets of the two policies.

Ordered or Unordered Procedures

A higher-level policy can be refined by an unordered set of lower-level ones. E.g., the higher-level *"Protect data"* can be refined into *"Protect data from loss"* and *"Protect data confidentiality"*. A higher-level policy can also be refined by a procedure, which is an ordered sequence of actions, e.g. *"Backup to tape"* and then *"Take the tape to safe store"*.

In general, Moffett and Sloman [MS93] claim, that the policy refinement process is strongly dependent on the policy hierarchy. Thus, given a formal description of relationship between high-level policies, refined low-level policies, and the actions implementing them, the system management will benefit in the following points:

- Automated translation of high-level policies into enforceable ones can be supported by the provided policy hierarchy. Automated solutions are possible for partitioned targets where lower-level policies can be expressed as subset relationships of higher-level ones. Automation becomes, however, difficult for goal refinement and arbitrary relationships.
- Verifying whether the set of lower-level policies actually fulfills the higher-level policies. That means, all the target objects are covered, i.e. the policy goals are met. Again, the automation of the process is feasible if the relationship between the policies of different levels is of partitioned targets form and difficult for goal refinement and arbitrary refinement of objectives.

- Given the policy hierarchy, it is easier to decide which lower-level policies are to be created or changed, in case a higher-level policy is defined or changed. Thus, changes at runtime can be partially supported.

5.4.2 Bandara

Bandara et al. propose a policy refinement process, which is based on the abductive reasoning technique [BLMR04]. The approach requires that the system, i.e. its objects, their behavior and organization, is represented formally by means of the Event Calculus, a formal language for representing and reasoning about dynamic systems [BLR03].

System Definition

In order to support the user, a tool has been developed which allows to specify the *system description (SD)* using UML. Thereby the objects in the system including the types specifying the attributes and interfaces are represented as a class diagram. The actions for the types are specified in a UML state chart representation. Analog, a UML representation of the *goals (G)* is used to model their types and attributes and supports the goal refinement process. In order to use the refined goals the operations that are needed to achieve them are to be identified. These operations can be done in sequentially or in parallel. For this purpose, the concept of *strategy (S)* is introduced, i.e the mechanism by which the system can achieve a particular goal or in other words, the relationship between system description and the goal. Thus, formally it can be stated as: $SD_x, S_x \vdash G_x$, where x is the label of the abstraction level. Again, for the sake of usability, the strategies which define a method invocation trace for achieving a given goal can be represented in UML using a message sequence chart.

So, in the approach by Bandara et al., it is expected that the user provides a representation of the system, in terms of the properties and behavior of the components, and defines the goals that the system must satisfy. The definition of the behavior of the system is done by specifying the pre- and post-conditions of the operations supported by the components. These are specified by the user in a high-level notation such as state charts. Similarly, the goals to be satisfied are defined in terms of desired system states.

Goal Elaboration

The policy refinement process involves a *goal elaboration* technique which translates high-level goals, defined during the requirements gathering process, into concrete low-level policies. A goal refinement hierarchy defines the dependencies between the goals at the different levels of refinement, thereby a goal of a higher-level can be decomposed either conjunctive or disjunctive. The conjunctive decomposition means that the higher-level goal is achieved only by achieving all the sub-goals. On the contrary, the disjunctive decomposition supposes that the higher-level goal is achieved if any one of the sub-goals is achieved. Thus, the process of refinement means that a particular path down the hierarchy is followed. In doing so, the feasibility of achieving the higher-level goal in terms of the lower-level ones at each level of abstraction is verified. In case the goal cannot be achieved the information at the higher-level has to be elaborated so that suitable lower-level goals can be derived. The goal elaboration technique proposed in the KAOS approach by [DvL96] represents each goal (and also negated goals, i.e. obstacles) as a Temporal Logic rule and uses refinement patterns in order to decompose it into a set of lower-level sub-goals entailing this goal. The approach by Bandara et al. goes a step further by elaborating a mechanism to connect the lower-level goals with the behavioral description of the system.

This is done by combining the notation used by KAOS with state charts, formal system definition and abductive reasoning techniques.

Event Calculus

As the underlying formalism for the formal system definition the *Event Calculus (EC)* is used. It supports among others abduction as mode of logical reasoning as well as allows to involve events and temporal relationships while representing and reasoning about dynamic systems. Following the approach to EC presented in [RMNK02] the authors use a set of time points (mapping them to the non-negative integers), fluents, i.e a set of properties varying over the lifetime of the system, and a set of event types. Additionally, they include a number of base predicates (like *initiates*, *terminates*, *holdsAt*, *happens*) and domain independent axioms. The authors as make use of function symbols like $state(Obj, V_0, Value)$ representing the value of a variable of an object in the system, $op(Obj, Action(V_p))$ representing the operations specified in an action event, $systemEvent(Event)$ denoting events that are generated by the system at runtime, and $doAction(ObjSubj, op(ObjTarg, Action(V_p)))$ representing the event of the action specified in the operation term being performed by the subject on the target object.

A UML profile for modeling goals and goal refinement patterns introduced in [HF04] describes also a high-level notation for representing these patterns. It also presents how they can be mapped into a set of temporal logic formulas. Bandara et al. shows how to translate these temporal logic operators into EC representation. Similarly, the state charts used for the system description are transformed into EC notation in the way that each transition arrow's input becomes an action to be performed. The transition between states is mirrored in the current state values becoming false fluents and the next state values becoming true fluents. It is also how the self-transitions are to be translated and how the current state values are reflected in the preconditions. Further, a strategy is translated into EC by means of conjunction of $happens(doAction(...), T)$ predicates where the time values correspond to the order in which the actions should be performed.

5.4.3 Romeikat

A model-oriented approach to policy refinement has been introduced by Romeikat et al. in [RBS11a], [RBS11b]. The authors use models to specify ECA rules at different abstraction layers and then apply model transformations in order to refine the rules in an automated way. The approach is demonstrated for the network management domain, where the policies are used to optimize the coverage in a mobile network.

The proposed methodology rests upon the idea of policy continuum presented in [Str03]. Romeikat et al. uses different types of models at different abstraction layers. Thus, the domain model specifies domain-specific concepts in a system, whereas the policy-model policies specifies policies that manage the system. A linking model is used in order to link the both models with each other, so that the domain-specifics can be used and addressed in the policies. A flexible number of abstraction layers can be used. The authors present a relational algebra which formally defines the semantics of the domain, policy, and linking metamodels and their views at the different abstraction layers. Figure 5.6 gives an overview of the approach. A common understanding of a managed system together with its context is required in order to manage it successfully. Depending on the focus and background of the stakeholders and experts, specific terminology is used. In [RBS11b] the *domain* represents this common understanding of the system by different experts. The *domain model* covers all the relevant information and knowledge about the domain with its specific

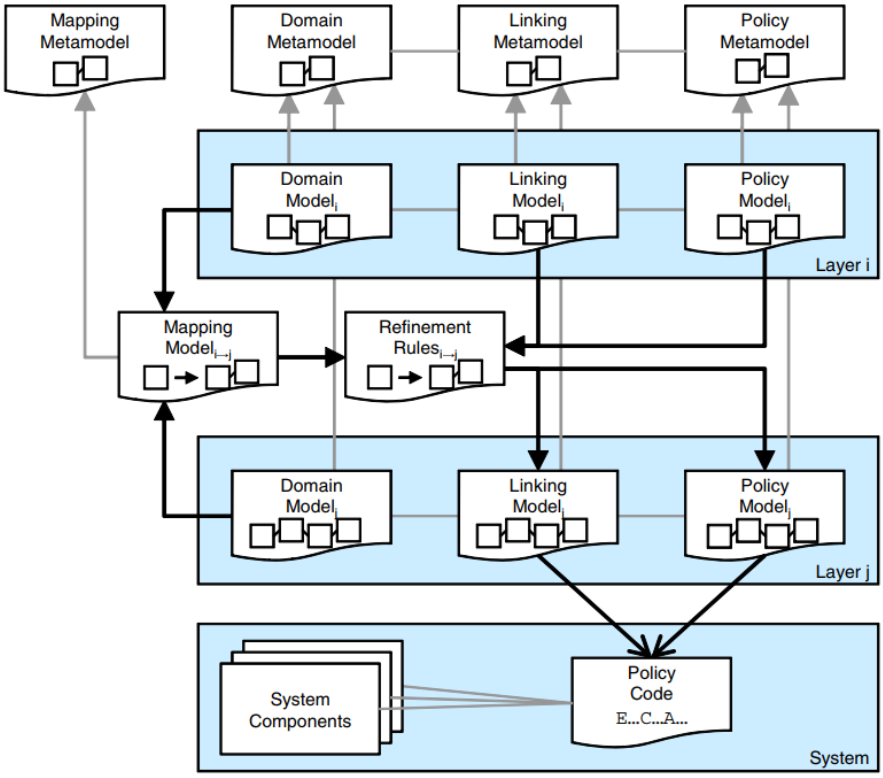

Figure 5.6: Policy Refinement. Reprinted from [RBS11a]

concepts across all abstraction layers. Similarly, the *policy model* covers any information about the policies, offering a particular view at any layer. In order to link the domain and the policy models a *linking model* is used. The linking model allows to apply the domain-specific concepts within the policies. The abstract syntax of the models is defined by means of the corresponding metamodels. The common concepts of such well-known policy languages as PonderTalk [TDLS09], KAoS [UBJ04], and Rei [KFJ03] are considered by the policy metamodel.

In order to support the refinement process a *linking model* has been introduced [RBS11a]. The linking model specifies mappings between the domain-specific entities of the higher-level to the lower-level one. The authors provide a set of *mapping patterns* which express the possible structural changes through the successive refinement process:

- Using the *identity pattern* means that an entity on the higher-level is leaved unchanged on the lower-level.
- The *replacement pattern* is used to map an entity on the higher-level to an entity on a lower-level.
- By means of the *merge pattern* multiple higher-level entities are mapped to a single entity on a lower-level.
- Representing a choice between multiple mapping options, the *split pattern* maps a higher-level entity to one out of several lower-level entities.
- The *erasure pattern* is used in case higher-level entity is not relevant at a lower level. An entity on the higher-level is mapped to no entity on a lower-level.

- The *appearance pattern* is used in case that an entity on the higher-level does not have any representation on a lower-level.

The refinement patterns are instantiated with the entities of the domain model. They refer to the instantiated refinement patterns as a *mapping model*.

After the mapping model is completed, the generation of *refinement rules* is performed. They specify a model-to-model transformation in a formal and language-independent way. A refinement rule transforms links between the domain and policy model from a higher layer into a lower layer. By applying the refinement rules to the linking and policy model, the refined policies are generated. Firstly, an intermediate model-based representation of the target language is generated. Then, a model-to-text transformation generates executable policy code in the target language. The authors have implemented the code generation for the Ponder policy language [RSB09].

5.5 Policy-Based Management Frameworks

There have been a number of efforts proposing different policy-based frameworks. In [PHS+08] a survey on exisiting policy-based frameworks for service-oriented systems is introduced. KAoS is a policy and domain services framework based on W3C's the Web Ontology Language (OWL) [MvH04] suitable for the management of distributed systems including Web Services, Grid Computing, and multi-agent system platforms [UBJ04], [UBJ+03]. Rei is a deontic logic-based policy framework [KFJ03] grounded in a semantic representation of policies in RDF-S [BG04] and policy reasoning engine based on the F-OWL [ZFC04]. The two pioneers among the policy-based management frameworks, the IETF policy framework and the Ponder policy framework are presented in the following.

5.5.1 IETF Policy Framework

The IETF working group has introduced a set of standards which define a framework for the representation, management, sharing and reusing of policies and policy information in a vendor-independent, interoperable and scalable manner.

Information Model Overview

No specific policy language has been proposed by the IETF, but an object-oriented policy information model. Thus, the managed system is represented by the Common Information Model (CIM) [DMT12b], [DMT16]. Whereas the policy-related information is represented by the Policy Core Information Model (PCIM) [MESW01], an extension of CIM. PCIM describes the basic concepts of policy groups, rules, conditions, actions, repositories and their relationships.

Figure 5.7 shows an overview of the basic classes and relationships in PCIM. The *PolicyGroup* class is a generalized aggregation container. By means of it, either *PolicyRules* or *PolicyGroups* can be aggregated in a single container providing nesting capabilities with no restriction on the depth of the nesting for administrative convenience.

The *PolicyRule* class represents the "condition-action" semantics associated with a policy. In general the condition is represented as a set of conditions (optionally negated) in DNF or CNF. If and only if the condition specified by the *PolicyRule* is true, the corresponding actions are to be performed. The *PolicyCondition* and *PolicyAction* classes are used in order to model the conditions and actions associated with a policy rule. A policy rule can be active or inactive. To indicate the schedule of activation and deactivation correspondingly, the

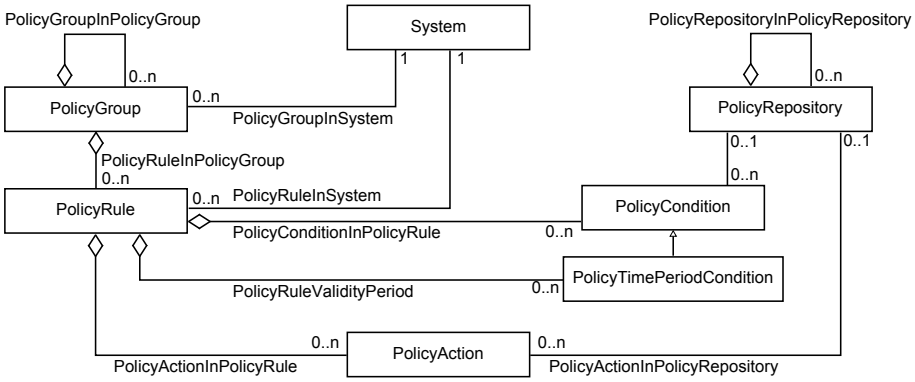

Figure 5.7: Core Policy Classes and Relationships in PCIM. Adapted from [MESW01]

PolicyRuleValidityPeriod aggregation is used. The *PolicyTimePeriodCondition* is a subclass of *PolicyCondition*. Thus, time-based criteria can be included in the condition definitions for a *PolicyRule*. In case a *PolicyRule* does not specify a *PolicyTimePeriodCondition*, it should be treated as valid always.

The *PolicyRepository* class represents an administratively defined container for reusable policy-related information. Thus, a reusable policy condition or action is always related to a single *PolicyRepository*, by means of the *PolicyConditionInPolicyRepository* or *PolicyActionInPolicyRepository* association. Rule-specific conditions and actions, however, are not related to any policy repository.

The IETF's initial focus has been on network policies to control Quality of Service (QoS) and IPSecurity. Thus, after providing a "core" model, the IETF has also published the corresponding domain-specific extensions (e.g., Policy QoS Information Model [SRS+03], IPsec Configuration Policy Model [JRV03]).

Deployment Model Overview

The IETF policy framework defines a policy deployment model, shown in Figure 5.8 based on [YPG00], [WSS+01]. According to it, four main architectural elements take part in the management process: a *policy repository*, a *policy decision point (PDP)*, a *policy enforcement point (PEP)*, and a *policy management tool*.

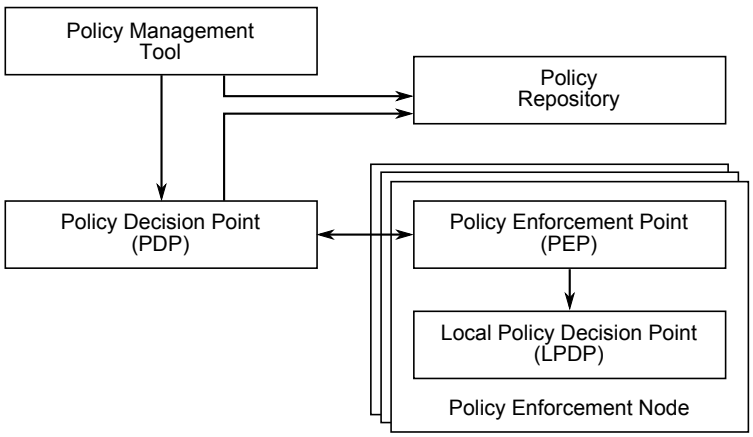

Figure 5.8: IETF Policy Deployment Model. Adapted from [YPG00], [WSS+01]

A policy management tool is an administrative tool which supports the lifecycle management of policy objects. A policy repository is an administratively defined component for storing and classifying policy elements and related data [WSS+01]. A PDP (also referred as *policy server*) is a logical entity that makes policy decisions for itself or for other managed elements that request such decisions [YPG00]. A PEP (also referred as *policy client*) is a logical entity that enforces policy decisions [YPG00].

The PEP is supposed to always run on a (distributed) policy enforcement node deployed at the managed element side. The PEP is the point where the basic interaction between the PEP and the PDP begins. The PEP detects that a policy decision is required. If needed, the PEP formulates a request for a policy decision and sends it to the PDP. In case a local decision point is available (LPDP) is available on the policy enforcement node, the PEP will first use the LPDP for a local decision. Afterward, the local decision is sent to the PDP who may override the LPDP and returns the ultimate decision to the PEP who finally enforces it. In the simplified case the PDP and the PEP may be co-located on the same policy enforcement node.

5.5.2 Ponder

The policy-based management framework Ponder has been developed over years at Imperial College [Slo94], [MS93], [Mar97], [Lup98], [Dam02]. The Ponder framework includes the specific policy specification language, a general architecture and policy deployment model including several extensions for access control and QoS management.

Information Model Overview

The partial class diagram for the information model used in Ponder's framework is presented in Figure 5.9.

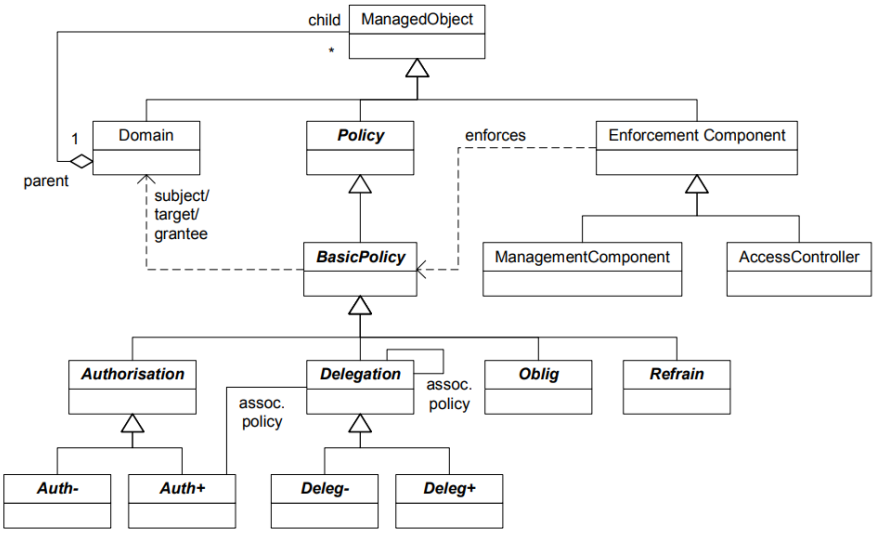

Figure 5.9: Policy Information Model in Ponder. Reprinted from [Dam02]

Within the model, all the objects (domain, policy, enforcement component) are defined as a *ManagedObject*. A domain can aggregate a number of other managed objects. By applying set operations (such as union, intersection and difference) to the objects within domains, basic policies can be defined over sets of objects. Subjects and targets of policies are defined in terms of domains. It is distinguished between ***access control*** (*Authorization* and

Delegation classes) and ***subject-based policies*** (*Obligation* and *Refrain*). The enforcement of policies in the runtime system is the task of *EnforcementComponents*. The automated enforcement of subject-based policies is the task of *ManagementComponents*, whereas *AccessControllers* enforce the access control policies.

The Ponder policy specification language is described in [DDLS00], [DDLS01]. *Authorization policies* define access rights (*positive* and *negative*), i.e. "what activities a member of the subject domain can perform on the set of objects in the target domain in terms of interface method calls". The syntax of an authorization policy is shown in Listing 5.1.

```
inst ( auth+ | auth- ) policyName {
  subject [<type>]  domain-Scope-Expression ;
  target  [<type>]  domain-Scope-Expression ;
  action            action-list ;
  [ when            constraint-Expression ; ]}
```

Listing 5.1: Authorization Policy Syntax in Ponder. Adapted from [DDLS00]

The name of a policy can be given as a path, so that the corresponding domain can be located. The specification of the domain scope for the subject and target elements includes an optional reference to the corresponding interface. In order to restrict the applicability in terms of time or attribute values, the policies can include constraint attributes. Thereby, the subject, target, action, event and time constraints are among the basic constraints applicable to all the policy types.

Delegation policies are meant to permit subjects to grant privileges, which they possess (*positive* and *negative*), to other subjects (grantees) to perform an action. Listing 5.2 demonstrates the syntax of a delegation policy.

```
inst ( deleg+ | deleg- ) ( associated-policy-name ) policyName {
  grantee   [<type>]  domain-Scope-Expression ;
  [ subject [<type>]  domain-Scope-Expression ; ]
  [ target  [<type>]  domain-Scope-Expression ; ]
  [ action            action-list ; ]
  [ when              constraint-Expression ; ]
  [ valid             constraint-Expression ; ]
  [ hops              int-value ; ] }
```

Listing 5.2: Delegation Policy Syntax in Ponder. Adapted from [DDLS00]

A delegation policy is always associated with an authorization policy, which specifies the access rights to be delegated. After a delegation is performed, the grantors still retain their access rights. Similar to the authorization policy, the specification of the domain scope for the grantee, subject and target elements includes an optional reference to the corresponding interface. In addition to the basic constraints, the positive delegation policies can optionally contain delegation constraints. In order to specify them, the *valid* and *hops* clauses are used. The *valid* attribute of the delegation policy specifies the duration or the period over which the delegation should be valid before the revocation. The *hops* attribute specifies the maximum number of cascading delegations allowed (maximum number of delegation hops).

Along with the access-control, the management of subjects in the system is to be performed. So, the subject-based policies specify the obligations which subjects must do and define the actions that subjects must not perform.

Obligation policies "specify the actions that must be performed by managers within the system when certain events occur and provide the ability to respond to changing circumstances". In Listing 5.3 the syntax of an obligation policy is shown.

```
inst oblig policyName {
  on                    event-specification ;
  subject    [<type>]   domain-Scope-Expression ;
  [ target   [<type>]   domain-Scope-Expression ; ]
  do                    obligation-action-list ;
  [ catch               exception-specification ; ]
  [ when                constraint-Expression ; ]}
```

Listing 5.3: Obligation Policy Syntax in Ponder. Adapted from [DDLS00]

The required *on* clause of an obligation policy must explicitly specify the event on which the actions defined in the *do* clause must be performed. Ponder allows to specify simple and composite events of internal (e.g. built-in timer event) and external (e.g. propagated notification of an external component) art. Composition operators can be used to specify composite events indicating their order, occurrence number, period of time, etc. The actions to be performed can be combined with concurrency operators which specify whether actions should be executed sequentially or in parallel. By means of the optional *catch* clause, an exception that is executed if the actions fail to execute for some reason is defined.

Refrain policy act as restraints on the actions that subjects perform. Listing 5.4 demonstrates the syntax of a refrain policy.

```
inst refrain policyName {
  subject [<type>]  domain-Scope-Expression ;
  target  [<type>]  domain-Scope-Expression ;
  action            action-list ;
  [ when            constraint-Expression ; ]}
```

Listing 5.4: Refrain Policy Syntax in Ponder. Adapted from [DDLS00]

The syntax of refrain is similar to negative authorization policies, but refrain policies are enforced by subjects and not by the target access controllers. Refrain policies are used in case the targets are not trusted or cannot enforce the policy because the decision depends on a state value of the subject. Further, calls on the actions may be internal to the subject (e.g. part of the interface or action script implemented by the agent).

Ponder contains also features which allow composition of the basic policies. Thus, a *group* is a syntactic scope used to group related policies together. A *role* allows a semantic grouping of policies, having the same subject and providing a means of grouping policies related to e.g. a position in an organization or a specific management agent. By means of a *relationship*, policies applying to the interactions between roles are grouped. A *management structure* allows configuration of roles and relationships into organizational units.

Deployment Model Overview

The deployment architecture supports the instantiation, distribution and life-cycle management of policies, as well as their enforcement by automated enforcement components. The overview of the architecture is introduced in Figure 5.10 from [DLSD01].

The Ponder deployment architecture includes three supporting services: a *domain service*, a *policy service* and an *event service*. The policy service serves as the interface to the policy management. The *policy administrator* interacts with the policy service in order to design and specify new policies. The policy service stores compiled policy classes, creates and distributes the corresponding policy objects. Thus, it instantiates a basic policy by creating and initializing either an *authorization policy object (APO)*, or an *obligation policy object (OPO)* or a *refrain policy object (RPO)*. The enforcement of policies is performed by the corresponding enforcement agents. Target's *access controllers (AC)* are responsible for the

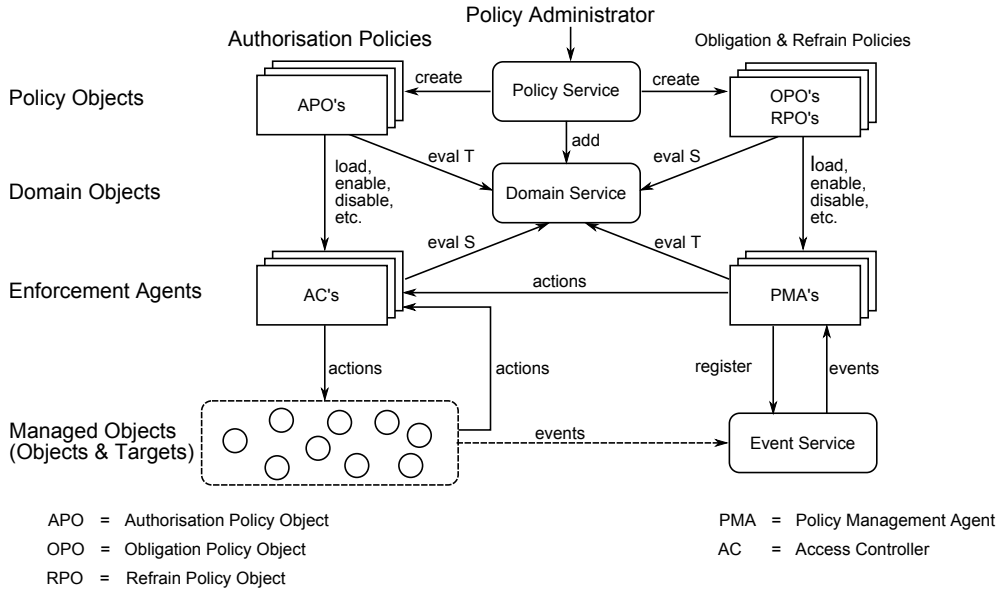

Figure 5.10: Policy Deployment Model in Ponder. Adapted from [DLSD01]

authorization policies and the subject's *policy management agents (PMA)* are responsible for refrain and obligation policies.

The main task of the domain service is to manage a distributed hierarchy of domain objects and to support the evaluation of subject and target sets at runtime. The domain objects hold references to the corresponding managed objects and policy objects that currently apply. For example, an LDAP server can be used in order to implement the domain service. In this case, the LDAP server has to generate events for changes to the membership of a directory. Another opportunity is to implement it by a database system [Dam02].

The event service provides an interface which allows to subscribe for events of certain type. The event service generates events based on the status of the underlying system and the managed objects and forwards them to the subscribed PMAs triggering obligation policy. In order to determine the target objects to which the obligation policy applies, the PMA asks the domain service and subsequently performs the corresponding policy action in case no constraint exists.

Chapter 6

Model-Based Management

Using a system model in order to support technical management has proved to be successful during the last years. Thereby the area of management has varied considerably from network through devices to services. Thus, in [VVB02] the authors present a model-based approach to network management. A reactive self-configuring model-based hybrid hard- and software system is presented in [WNN96]. Whereas [FUMK03], [GHK+01], [EKK+04] address model-based management of services-oriented systems.

6.1 Model

Model is a selective abstract formal representation of a system [Béz05]. A given system may have plenty of different models. Each of them represents a particular aspect of the system and only this aspect. Each model has a specific purpose and is described in the language of its unique *metamodel*. The metamodel defines how elements of a system are to be chosen in order to generate a given model. Thus, a metamodel, which the model is conform to, specifies what aspect of the system the model represents. Figure 6.1 provides an UML class diagram illustrating the relationship [Obj10]:

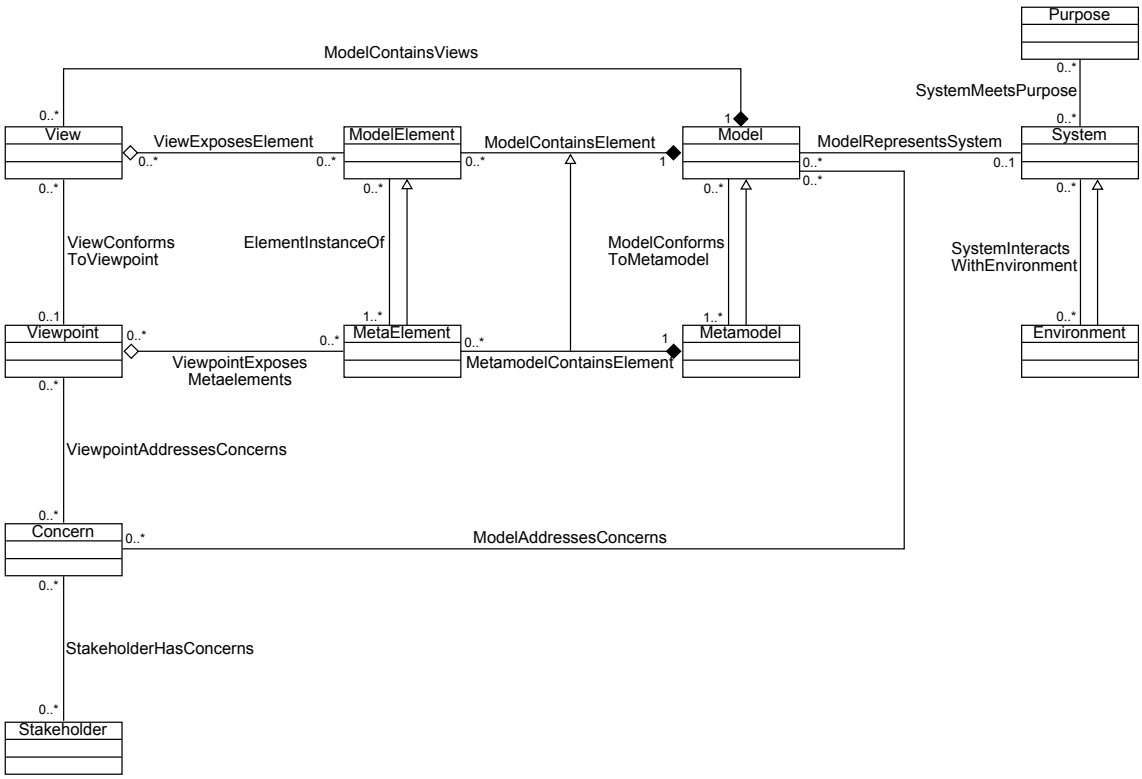

Figure 6.1: Definition of the model concept with MDA. Adapted from [Obj10]

In November 2000, the Object Management Group (OMG) introduced the Model Driven Architecture (MDA^{TM}), adhering to the global trend of and realizing Model Driven Engineering (MDE) principles [Obj14a]. Based on the established standards like MOF, XMI, OCL, UML, CWM, SPEM, the MDA separates business and application logic from the underlying platform technologies by providing platform-independent models and leverages them to "enhance the agility of planning, design, and other lifecycle processes, and improve the quality and maintainability of the resulting products".

In the field of automated technical management, the usage of models brings similar benefits and gains in importance. Several steps towards standardization have been taken lately. Thus, the DMTF's Common Information Model (CIM) provides a common definition of management information for systems, networks, applications and services, and allows vendor- and domain-specific extensions. It is an information model, a conceptual view of the managed environment, which unifies and extends the existing instrumentation and management standards using object-oriented constructs and design. The CIM standard includes the CIM Metamodel [DMT12b], the CIM Schema [DMT16] and a set of relevant specifications[1].

Relying on the OMG's UML specification [Obj11], the CIM Metamodel is the basis on which CIM schemas are defined. It defines the semantics for the construction of new conformant models and comprises common basic elements for representing models (e.g. object classes, properties, methods and associations) [DMT12b]. The actual models are described by the CIM schemas representing the resources of a managed system, including their attributes, behaviors, and relationships. The CIM Schema is structured into the distinct layers: core model (applying to all areas of management), common model (applying to the common areas like systems, applications, networks, and devices but independent of a particular technology or implementation), and extension schemas (technology-specific extensions to the common model) [DMT16]. The CIM specifications define the management infrastructure, the details for integration with other management models, the syntax, semantics, naming conventions [DMT12a] as well as the use of the Managed Object Format (MOF) language [DMT12c] for specifying CIM models. DMTF's Web-Based Enterprise Management (WBEM)[2] comprises a set of specifications that cover discovery, access, and manipulation of resources modeled using the CIM.

6.2 Model-Based Management

An innovative approach to the technical management harnessing modeling to support the process has been presented by Lück et al. in [IKP^{+}05], [IPK^{+}06], [LÖ06]. The approach includes the concept as well as the corresponding tool for automated refinement of policies in multilevel hierarchies. The tool supports the user by the interactive graphical modeling which allows easy and intuitive handling.

Figure 6.2 shows the model structure which is the basis of the approach. The model represents the managed system as well as the associated control and policy elements. The modeling process includes creating the graph model and subsequent parametrization of nodes and edges of the graph. The backend functions allow automated refinement of policies from top to bottom through the hierarchy.

The horizontal allocation of the model in abstraction levels provides a framework for a three level hierarchy. Each policy element refers to a certain element of the managed system. These relationships are represented by edges between policy and managed system's

[1] From the DMTF's site: https://www.dmtf.org/standards/cim
[2] From the DMTF's site: https://www.dmtf.org/standards/wbem

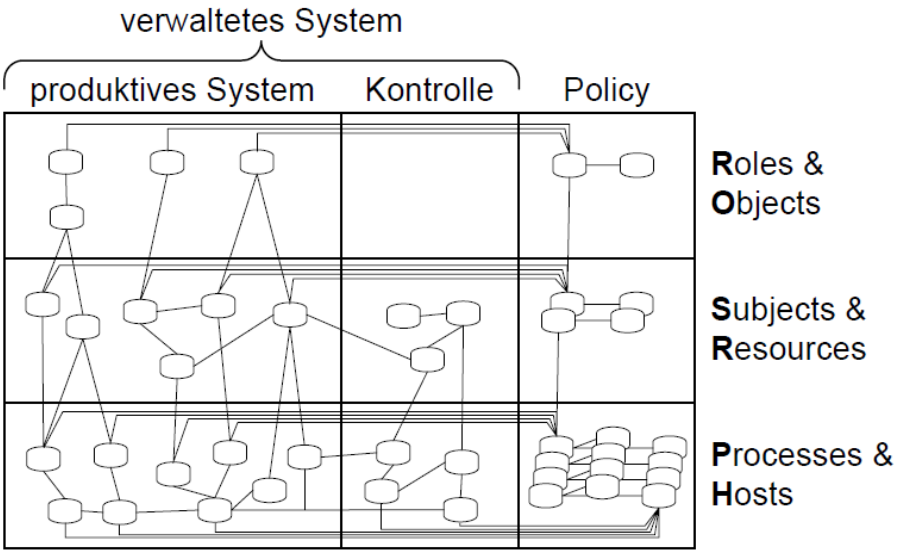

Figure 6.2: Basic Model Structure. Adapted from [LÖ06]

elements. Only intra-layer policy-managed element associations are allowed. The layers of the model have the following primary focus:

"Roles & Objects" (RO) The top layer of the model is devoted to the organizational aspects of the model. The basic elements are the *roles* and *abstract objects*. The layer can also include further elements of the similar abstraction grade. However, this is up to the actual use case and field of application. In general, the control elements are not relevant for the specification on this abstraction layer.

"Subjects & Resources" (SR) The middle layer of the model is devoted to the active elements of the model. These are *subjects*, the elements that take actions on behalf of certain users and in doing so use the *resources* of the system. The level of abstraction on this layer is dictated by the service-oriented aspect of the model. Thus, the *services* are also modeled by means of the corresponding elements on this layer.

"Processes & Hosts" (PH) The bottom layer of the model is devoted to the technical aspects of the model. The basic elements are the *processes*, *hosts* as well as *user credentials*, *system resources*, *communication protocols* and *network structure*. Since the abstraction level of the layer is very technical, it is easy to derive the concrete configurations for the control elements of the actual system from the policy elements.

The proposed approach has been applied and validated in the field of security management, in particular configuration of security services. Thus, the focus of identified abstraction levels as well as the attribution of elements to the corresponding layers lies on the aspects specific to this application field. Each policy is presented in each layer by means of a certain policy element. Each policy element controls access to certain resources.

The access control is up to the following extended logic: not allowed access has to be prohibited and allowed access must in fact be provided by the system. These authorization policies are comparable with the positive authorization policies in Ponder presented in Section 5.5.2. The explicit access interdiction is not required, since according to the specified logic the access is implicitly denied until it's explicitly allowed. Further, it is possible to define dynamic constraints for allowed access.

The process of model-based management includes multiple steps. Firstly, the ***modeling step*** is done: the managed system is modeled on the abstraction levels mentioned above. Each layer represents the system with the different level of abstraction. Thus, each abstract

model element of the higher-level has a corresponding concrete element on the lower-level. These associations are modeled also. Subsequently, the required policy is modeled by means of the policy elements on the top-level. The modeling process is performed bottom-up:

1. The components ans structures of the real system are modeled on the PH layer. The developed model presents the technical view of the managed system.

2. The services which are realized by the processes of the PH layer are modeled on the SR layer and associated with the corresponding elements of the bottom layer. The model presents the service-oriented view of the system.

3. The attendant roles and object are modeled on the RO layer and connected to the corresponding subject types and users of the SR layer. The model presents the organizational view of the system.

Secondly, the ***policy refinement step*** is performed in an automated way. Before each sub-step, the consistency check is done, in order to ensure that the model has no conflicts and the specified policy is feasible with regard to the modeled system.

On the basis of the system model and the identified abstract requirements, the policy refinement is performed in the top-down matter.

1. The authorization policies are presented in the RO layer as access permissions and associated with the corresponding roles, objects and access mode. Several arts of dynamic constraints can be defined: role cardinality constraint, exclusive role authorization, and exclusive role activation.

2. The access permissions on the SR layer are presented as service permissions. The corresponding model element allows a subject of a user by means of a service to access a certain resource. Access permissions and exclusive role activation elements of the RO layer are refined to the service permissions and exclusive subject activation elements on the SR layer.

3. The PH layer includes protocol permissions which allow a process, that can show a user credential, to communicate with another process by means of a protocol in order to access a certain system resource. A concurrent usage of certain user credentials can be prohibited by an exclusive user credential activation constraint. The service permissions and exclusive subject activation elements on the SR are refined to the protocol permission and exclusive user credential activation element of the PH layer having regard to the dependencies between the services. In order to ensure the service communication, the corresponding processes should be able to communicate with each other. Thus, the service associations of the SR layer are refined to the protocol permission elements of the PH layer.

4. For each security service the relevant protocol permission elements are estimated. For each protocol permission element of the PH layer all the communication paths are calculated. The paths which do not meet the security requirements are not considered further.

The derivation of the single configurations from the policy elements of the PH layer as well as the distribution of them to the corresponding control elements are not the subject of the policy refinement process. This is in general a syntactic adaption of generic rules on the PH layer to the specific interfaces of the control elements. At bottom of the step, the ***code generation*** and the ***distribution*** are performed.

The consistency check and the refinement algorithm guarantee that the derived elements comply with the specified policies on the whole. The proof of correctness of the refinement process is introduced in [LÖ06].

6.3 Model-Based Management Challenges

The challenges which the automated management solutions have to face have been introduced in Section 4.3. Additionally, the adoption of the introduced model-based management approach obligates to handle the following aspects.

Scalability and Modularization

Dealing with a larger systems with a great number of managed components, the model-based approach tends to lose its clarity and straightforwardness. The model becomes obscure and unclear. A common way of dealing such problems is applying the *divide and conquer* principle [dAKdG05b]. As the system is modularized into smaller segments, it becomes possible to deal with them in detail separately. The system is observed on a more abstract level considering the interaction of the separated segments.

In [dAKdG05b],[dAIKdG05] the system is partitioned into *abstract subsystems* hiding the details of the system by concentrating on the overall structure of the system and letting the particulars of each subsystem to the corresponding internal specifications. Thus, the processes of system analysis and design are separated even more strongly so that the model gets more comprehensive and scalable.

Tool Support

Providing a tool support for the users planning, designing and configuring the management system is very important. Automation of the consistency check, refinement of the provided abstract system requirements as well as generation of the configurations and management artifacts supports the users enormously. The characteristic feature of the model-based approach of modeling the system on multiple abstraction layers places additional demands from the supporting tools.

The graphical representation of models is much more clear and intuitive for the users in contrast to the textual or formal representation. At the same time, the visual models tend to get extremely large which can make them difficult to overview. Thus, the graphical modeling tool should allow filtering the model elements in order to provide the ability to group elements according to certain features and handle them in the same way. The employment of multiple views of the system facilitates the work with larger systems allowing the user to concentrate on specific points of view.

Further, the object-oriented system design of the model can facilitate the usability of the model. The ability to use multiple inheritance allows to build any hierarchical structures. The usage of graph structures for representing the elements of the model and their relationships not only provides for intuitive display of the model but also allows applying certain techniques and graph transformation algorithms for the model transformations.

General Applicability

Developing a model-based management system should face the trade-off of generality of the technique and the amount of automation. That means, creating a technique that is highly

automated is possible at the expense of trimming its usage to a very narrow application domain. The types of model elements and structures as well as the used refinement algorithms are aligned with the management scenario. Extending the management scenario involves revision and adaption of the model structures and refinement algorithms.

It demands great skill to create a metamodel which covers enough types of model elements, structures and supports the refinement process, so that the produced models cover a bright variety of management scenarios. The usage of a common metamodel allows to exchange the models on demand. Thus, exchanging configurations and management artifacts at runtime becomes possible.

For though the works of Lück [LÖ6] and de Albuquerque [dAKdG05b], [dAKdG05a], [dAIKdG05] reside in the application field of network security management, the general approach of the model-based management can be adopted in other application fields. Transferring to another application field requires indeed other (as the case may be, more general) alignment of layers in the system model, working out a set of appropriate refinement patterns and templates, but still preserve the same principle of the layered abstraction model structure, graph notion of the model, the metamodel's arrangement, etc.

Separation of Concerns

The idea of separation of concerns has been studied extensively in [JZ93] "aiming at clarifying the relationship between a formal specification and the domain of the system to be specified". The authors have shown the strong need of separation of descriptions of the domain (with its objects and operations) and the actual requirements.

The introduced model-based approach differentiates clearly between the system, control and requirements elements. That helps to keep the modification and adaptation efforts during the runtime comparatively small.

Due to the layered structure, the model-based approach supports the strong separation of abstraction layers so that the system and requirements definition are tailored to the certain layer allowing the different user groups to operate with the system elements on the desired abstraction level.

Formal Representation and Proof

A formal representation of the model provides not only a formalism to describe the system and its requirements, but also to support their analysis and verification. In general, choosing the formal technique demands proving its *correctness*, *completeness*, *consistency*, and *minimality* [SSS+16].

The model-based approach allows in the most cases a more comfortable, illustrative and thus quality-assured definition of the system and requirements. Thus, the graph notion is natural and intuitive for the representation of objects and their relations and the object-oriented methodology supports the principles of abstraction and generalization. However, a complex and laborious work has to be done in order to transform the model representation into the formal representation and to prove that the transformation conforms to the above referred features.

The detailed proof of the correctness of the refinement rules and the model structure is elaborated by Lück in [LÖ6] and is extended with the notion of abstract subsystems by de Albuquerque in [dAKdG05a] .

Chapter 7

Runtime Management System

In order to carry out the management process at runtime, a management system is needed. This chapter introduces the management system which we use to put into execution the developed approach. The access to the management data is performed by means of the management tree data structure, firstly introduced in [BFL+13] and with full details elaborated in [BKF15]. The policy-based approach to the management presented in [DKK+10b], [DKK+11b] is applied. Thus, the policies represent the logic which determines the behavior of the managed system at runtime.

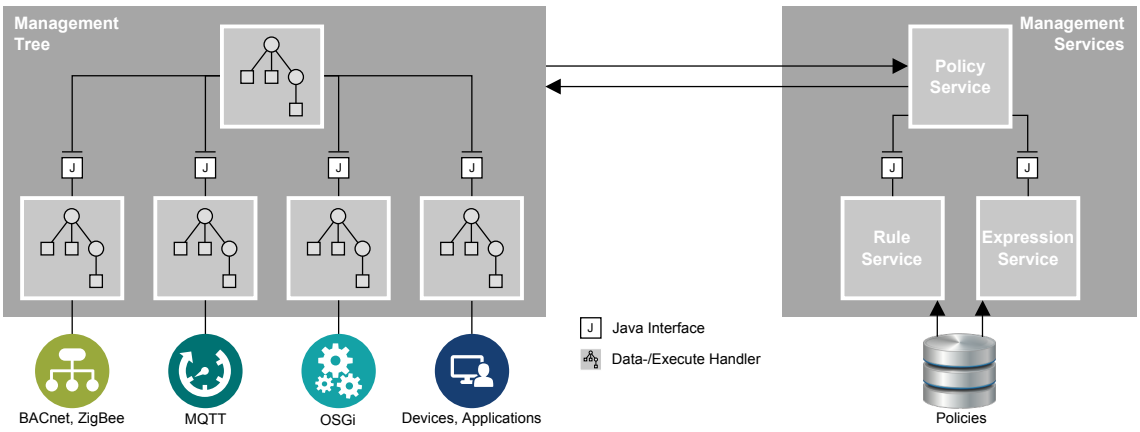

Figure 7.1: Overview of the Runtime Management System. Adapted from ([BKF15]

Figure 7.1 provides an overview of the runtime management system ([BKF15]). The core of the system is a distributed ***management tree*** which follows the approach of the OMA Device Management Alliance described in the Device Management Tree and Description Specification [Ope07]. The data and execute handlers encapsulate the protocol- as well as use case-specific management data acquisition and propagation.

Policies, i.e. policy rules and expressions, are the actual guidelines governing the choices of the management system operations. They are stored in the policy storage which is addressed directly by the corresponding management services. Policies are defined on the management data which is organized in the management tree.

The operations on the management tree are performed by the ***management services***. The primary management service is the policy service which encapsulates the access to the rule and expression services. The rule service enforces the execution of policy rules whereas the expression service is responsible for the evaluation of the policy expressions on demand. The main data flow occurs between the management tree and the policy service and comprises basically reading and setting the management data.

The following sections introduce the components, their composition and structure as well as the communication between them in detail.

7.1 Management Tree

The management tree is a virtual data access structure, it is not used to store data, but rather provides a uniform hierarchical view of the data offered by the management agents. The handler implementations map the management data to the corresponding hierarchically organized object model. Therefore, they manage to abstract the management data access from the actual access operations and event notifications. Owing to this access facade a homogenous view on the resource landscape is offered. Thereby, the resource landscape spans all three layers of the system model presented in Section 8.1: from low-level hard- and software components through the services and applications up to the high-level use cases with their functions and assets.

7.1.1 Management Data

The management data is provided in form of *management variables*. We divide logically the management variables into *status* and *configuration variables* [FLL+09], [DKK+10a]. The configuration variables are set by the management service. Whereas the status variables are used by the resources to represent their state. Thus, the control loop is performed: the changes of the managed object's state trigger the management service to set the corresponding configuration variables what in turn causes the change in the internal state of the managed objects. The transition from one system state to another takes place.

It is obvious, that due to the existing event queue there occurs a certain temporal offset between the requesting of the status change and the actual execution of the corresponding action. Fails the transition from one state to another or it is not executed within the predefined time slot, an inconsistency between actual and desired state occurs. The variable separation, however, allows the system to grip such cases and undertake the corresponding measures.

7.1.2 Tree Nodes

As the name suggests, the management tree is a virtual hierarchical tree-like data structure consisting of nodes starting at a root node. Each node is uniquely identified by an absolute URI starting from the root. Tree nodes can be divided into interior and leaf nodes. The interior nodes can have children nodes. The leaf nodes, on the contrary, have no children nodes. They represent an abstraction of the actual management variable. These are primitive values of the following basic types: Boolean, Short, Integer, Long, Float, Double, String, Binary and Object.

In order to control the access to the nodes, an Access Control List (ACL) is used. ACL determines which access operations (Get, Add, Replace, Delete and Execute) on the node are allowed for which principal.

Moreover, nodes can be associated with the corresponding meta data describing them and their sibling nodes. The meta data may specify default values for the leaf nodes as well as allowed access operations for the nodes.

The following list summarizes the settable (and in some cases only readable) properties which can be defined for a tree node:

Property	Description
Name	The name of the node must be unique among the node's siblings. It is up to the underlying handler implementation, whether the node can be renamed at runtime.

to be continued.....

...to be continued...

Property	Description
Title	The title of the node must be human readable in comparison to the node's name. Depending on the handler implementation, this property is optional.
ACL	The Access Control List defines the allowed access operations, which are allowed to be performed on the node and its descendants.
Version	The version number which starts at 0 and is incremented on every node modification. Modifications include changes of the node value and any of its properties (also including ACLs) as well as addition and deletion of nodes. The value is read-only. Depending on the handler implementation, this property is optional.
Timestamp	The timestamp of the last version change. The value is read-only. Depending on the node's implementation, this property is optional.
Data Type	The data type of the node's value. This property is supported by the leaf nodes only.
Mime Type	The mime type of the node's value. This property is supported by the leaf nodes only.
Schema	The name of the schema which defines the structure of the subtree starting at the node.
Value	The value contained in the leaf node. This property is supported by the leaf nodes only.

Table 7.1: Overview of the Node Properties. Adapted from [BKF15]

To sum up, to perform the management process means just to perform operations on the management tree: node creation, node removal as well as node property assignment. A management node, whether interior or leaf, represents an abstraction of the actual management information. Thus, the management tree offers a uniform view of the management data.

7.1.3 Data and Execution Handlers

The management tree hides the protocol- and application-specific parameters for data acquisition by means of data and execute handler implementations. The handlers are realized in the form of self-contained software components. In order to improve the system performance, the handlers are provided dynamically at runtime. This means that they are released when the corresponding management data does not need to be accessible any more.

Every tree node is provided by the corresponding data handler implementation. This implementation is responsible for offering a hierarchical view of management data to the consumer. Thereby it adapts the data, takes care of propagation and collection of the management data in time. A data handler can be of two types: a base data handler and a protocol-specific one. The base data handler restricts the tree structure and is schema-controlled. It reflects the resource landscape (e.g. devices, software components) of the managed system in accordance with the use case. The other handler type is a protocol-specific one. This handler implementation encapsulates the communication and platform technology details (e.g. BACNet, ZigBee, MQTT, OSGi).

7.1.4 Tree Structure

The structure of the management tree is controlled by the schemas. A schema can be applied to and removed from a node at runtime. The schemas are declarative and specify

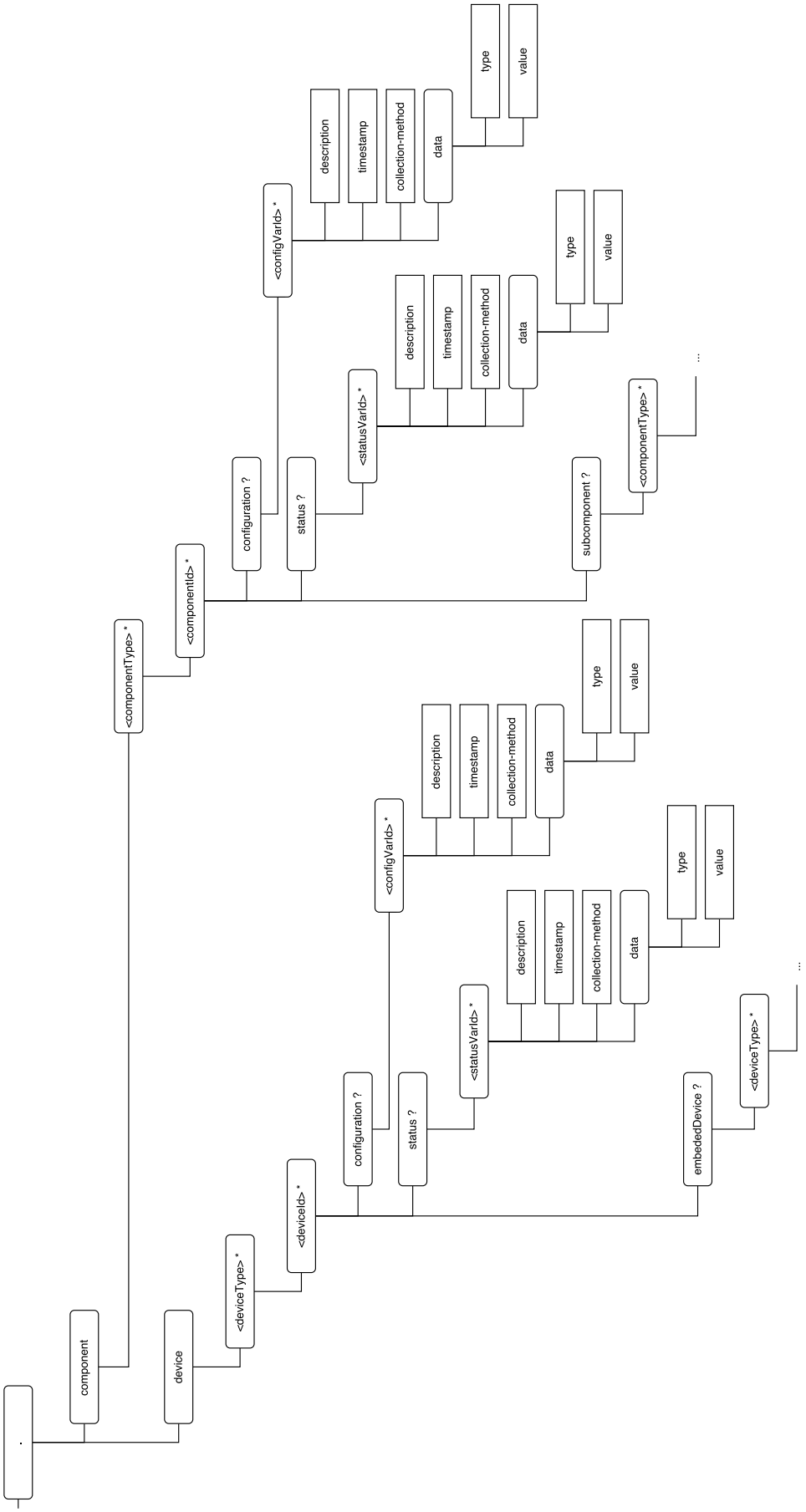

Figure 7.2: Management Tree DDF

the structure of the underlying subtree. In case a node is typed with a schema during its creation, it represents the root node of the subtree which must conform to the schema. In case the node stays untyped, it becomes a part of the declared schema valid within this current subtree. All the modification actions (adding, deleting and changing of properties) on the tree node must be conform with the valid schema, otherwise the action must fail.

Figure 7.2 demonstrates an example schema specifying the allowed tree structure in the simplified graphical notation used by the OMA DM Device Description Framework, DDF [Ope07], [Ope16]. The notation in the shape of a block diagram allows to distinguish between interior (rounded block) and leaf (unrounded block) nodes. Similar to the syntax of DTDs for XML, a special character is used to denote the number of occurrences of the node:

"?" "zero or one"
"*" "zero or many"
"+" "one or many"

In case a character is omitted, the node is present exactly once per default. Each block in the graphical notation corresponds to a described node, the title of the block stands for the name of the node. The unnamed blocks act as placeholders and are instantiated at runtime when the node is created, they are marked with angle brackets ("<>"). In order to construct the URI for each node in the management object, the names of the ancestral nodes are used.

Following the approach of the management variables, the management data is organized in the following way: the root node has two similarly organized subtrees, one for the devices, one for the software components. Each of them has multiple subtrees for different device (or component) types. Each device (or component) identified by its id is described in a separated subtree. It contains three optional collections of configuration variables, status variables and embedded devices (or subcomponents). The subtrees for the status and configuration variables are similarly arranged. They have three leaf nodes for holding the description, timestamp and collection method of the management variable as well as the data node which in turn is organized into the type and value leaf nodes holding the data type of the management variable and its value itself. The embedded devices and subcomponents branches are organized in the same way as the device and component subtrees.

7.1.5 Management Tree Access

In order to ensure atomic or transactional data access sessions are used, i.e. a valid session is required to interact with the management tree. The session type is to be specified at the beginning of the interaction with the management tree by the client. The following session types are supported:

Shared Session Shared sessions are used for the read-only access to the management data. An unlimited number of concurrent shared sessions is allowed. The creation of an exclusive session on the overlapping subtrees must be blocked, however.

Exclusive Session Exclusive sessions are used for the write access to the management data. Concurrent access to the overlapping subtrees is not allowed for multiple exclusive sessions. Thus, a write lock on the subtree must be acquired by the exclusive session blocking the creation of other sessions for operating on the same parts of the tree.

Transactional Session Similar to exclusive sessions, transactional sessions are used for the write access to the management data. They similarly block the corresponding subtree from any other sessions. Additionally, transactional sessions can be rolled back any time as long as they are still opened. Thereby, the initial status of the subtree is saved. In

case of a rollback, the modifications done on the blocked parts of the tree are undone and the initial status is restored.

7.2 Policies

The logic which determines the behavior of the managed system at runtime is realized by means of policies. The management system performs the management actions by applying the policies at runtime. As it was defined in Section 5.2, technically, policies are formulated on the management objects and enforced by the management agents. Speaking in terms of the proposed model-based management approach, we reside in the following on the bottom layer of the system model.

Resorting to the management tree as the virtual data access structure, we define the policies exclusively on the management variables [DKK+10b], [DKK+11b] which values reside in the management tree on the leaf nodes (Figure 7.2). The implemented low-level policies are of two main types, introduced in the following: policy rules and policy expressions.

7.2.1 Policy Rule

The policy rules we use, follow the definition of positive imperative policies obliging the management agent to perform a certain action as introduced by the Sloman and Moffett working group [MS93]. The policy rules follow the *event-condition-action* paradigm as presented in Section 8.2.3. Thus, the *event* part of the rule can be realized as a certain tree event, e.g. a node is added, deleted or changed, the *condition* part is realized by checking the tree structure, node properties and values, in order to trigger the appropriate action. The *action* part corresponds to executing the equivalent tree operation, e.g. node addition, deletion or change.

Since the management data is presented in form of status and configuration variables, the condition part is reduced to evaluating the nodes containing the status variables and the action part is reduced to updating the nodes containing the configuration variables.

The interface of a policy rule is shown in Listing 7.1. The interface is built in a flexible manner in order to allow covering the greatest variety of constellations. Thus, *event-condition-action* parts of the rule can be realized in different ways by spreading them through multiple methods. In the following we look closely at the methods of the rule interface.

```
1  package com.materna.nodes.tree.rule;

3  public interface Rule {

5    public String getCorrelator();

7    public void activate(RuleContext context) throws Exception;

9    public void deactivate(RuleContext context) throws Exception;

11   public boolean evaluate(ExecutionContext context) throws Exception;

13   public void execute(ExecutionContext context) throws Exception;

15 }
```

Listing 7.1: Policy Rule Java Interface

The rule can be activated and deactivated within its rule context by means of the corresponding methods: *activate(RuleContext context)* and *deactivate(RuleContext context)*. A registered policy rule is activated automatically after its registration. After the deactivation, the policy rule does not fire until it is activated again.

The *evaluate(ExecutionContext context)* method allows to execute the *event-condition* parts of the policy rule. Through the execution context, the access to the management tree event is done. It is up to the underlying system model, what tree event is used as a trigger for the rule firing. For example, a node is deleted (e.g. a device has fallen out) or the value of a leaf node exceeds the predefined threshold (i.e. a status variable is out of the allowed range). In case the event applies, the evaluation of the condition should occur. This can be in order to check the current tree structure (e.g. whether a replacement device is available) or the node value (e.g. what is the value of a status variable). For this purpose, the execution context is used to access the management tree. A shared session is opened in order to perform the required read operations. The *evaluate(ExecutionContext context)* method returns *true* if the *condition* part of the rule is satisfied and otherwise - *false*. In the first case the execution of the *action* part is triggered.

The *execute(ExecutionContext context)* method realizes the *action* part of the policy rule. Again, the access to the management tree is done by means of the execution context. The corresponding subtree can be locked exclusively for a write session. During the session an appropriate operation on the management tree is executed. That means nodes can be created (e.g., a replacement device is initialized), deleted (e.g., a flawed sensor is inactivated) or changed (e.g., a configuration variable is set with a new value).

The use of execution context allows additional flexibility in realization of the classes implementing the rule. In case the evaluation of the *condition* part is expensive (e.g. time-consuming or requires locking of large subtrees), the intermediate evaluation results can be saved in properties of the execution context in order to be accessible within the *execute(ExecutionContext context)* method. The interface of the execution context is shown in Figure 7.2.

```
1  package com.materna.nodes.tree.rule;
2
3  import java.util.Map;
4  import java.util.regex.MatchResult;
5
6  import com.materna.nodes.tree.eventing.TreeEvent;
7  import com.materna.nodes.tree.path.pattern.PathPattern;
8
9  public interface ExecutionContext {
10
11   public TreeEvent getEvent();
12
13   public RuleContext getRuleContext();
14
15   public Map<PathPattern, MatchResult> getMatchResults();
16
17   public Map<String, Object> getProperties();
18
19 }
```

Listing 7.2: Policy Rule Execution Context Java Interface

The concrete implementation of the rule interface, i.e. how the values of the configuration variables are computed, what is the triggering event and what status variable values are to be checked, is up to the underlying system model.

7.2.2 Policy Expression

Another form of policies we advocate is a policy expression [DKK+10b]. This form has not been addressed by any work known to us so far.

In order to provide a flexible approach to the policy-based management, it is sometimes appropriate to allow the managed system to address the management system itself. Thus, a policy-aware developed component can request from the management system the evaluation of a policy expression at runtime. Depending on the result, the component can react by adapting its program flow. E.g., a managed component requests the management system before entering a critical operation to check if the current system state (i.e. resource availability and performance) is favorable enough. Due to this approach, a special form of the policy-based resource control level occurs.

```
1  package com.materna.nodes.tree.expression;
2
3  public interface Expression<T> {
4
5    public void activate(ExpressionContext context) throws Exception;
6
7    public T evaluate(EvaluationContext context) throws Exception;
8
9    public void deactivate(ExpressionContext context) throws Exception;
10
11 }
```

Listing 7.3: Policy Expression Java Interface

The interface of a policy expression is presented in Listing 7.3. It is a generic type interface that takes an output type parameter. The policy expression interface's central method is *evaluate(EvaluationContext context)* which is responsible for the evaluation of the expression. The classes implementing this interface request a shared session on the management tree. During the session the values of the status variables are read. Similarly to the policy rule, the evaluation of the expression, is done depending on the underlying system model.

```
1  package com.materna.nodes.tree.expression;
2
3  import java.util.Map;
4
5  public interface EvaluationContext {
6
7    public ExpressionContext getExpressionContext();
8
9    public Map<String, Object> getParameters();
10
11 }
```

Listing 7.4: Policy Expression Evaluation Context Java Interface

The interface of the evaluation context used as an input parameter of the method *evaluate(EvaluationContext context)* is shown in Listing 7.5. The evaluation context within which the evaluation of a policy expression occurs, allows the access to the corresponding expression context by means of the method *getExpressionContext()* and to its parameters in form of a map by means of the method *getParameters()*.

The policy expression can be activated and deactivated within its expression context by means of the corresponding methods: *activate(ExpressionContext context)* and *deactivate(ExpressionContext context)*. A registered policy expression is activated automatically after the registration. After the deactivation, the evaluation of the policy expression is not possible until it is activated again.

```
1  package com.materna.nodes.tree.expression;
2
3  import java.util.Map;
4
5  import com.materna.nodes.service.registry.ServiceQuery;
6
7  public interface ExpressionContext {
8
9    public ExpressionReference getExpressionReference();
10
11   public Map<String, Object> getProperties();
12
13   public ServiceQuery services();
14
15 }
```

Listing 7.5: Policy Expression Context Java Interface

The expression context allows to access the expression reference within its runtime environment. The access is performed by means of the *getExpressionReference()* method. Further, the method *getProperties()* returns the corresponding properties in form of a map. The method *services()* returns a service reference to the responsible expression manager.

Policy expressions allow taking the evaluation logic out of the application, but still guarantee that the evaluation is initiated on demand by the application. The management system is indeed responsible for the evaluation but still the interpretation is up to the requester. Thus, the policies applied by the automated management system at runtime provide a reusable powerful instrument which forces the managed system to conduct in a flexible but predefined manner.

7.3 Management Services

The management system is realized in the form of the multiple management services: policy, rule and expression services. The following sections introduce them in detail.

7.3.1 Policy Service

The policy service is the central part of the management system. It is a switching point for the policy management and execution at runtime. The policy service encapsulates the access to the underlying rule and expression managers which act as registries for the rules and expressions, manage their lifecycle and enforce them.

```
1  package com.materna.nodes.tree.policy;
2
3  import java.io.InputStream;
4  import java.util.Map;
5
6  import com.materna.nodes.service.ServiceReference;
7  import com.materna.nodes.tree.expression.ExpressionService;
8  import com.materna.nodes.tree.rule.RuleService;
9
10 public interface PolicyService extends ExpressionService, RuleService {
11
12   public interface Builder {
13
14     public PolicyService.Builder expressionService(ExpressionService
       expressionService);
```

```
15
    public PolicyService.Builder ruleService(RuleService ruleService);
17
    public PolicyService.Properties getProperties();
19
    public ServiceReference<PolicyService> build();
21
  }
23
  public interface BuilderFactory {
25
    public PolicyService.Builder newBuilder();
27
    public PolicyService.Builder newBuilder(InputStream in);
29
    public PolicyService.Builder newBuilder(Map<String, Object> properties);
31
  }
33
  ...
35
}
```

Listing 7.6: Policy Service Java Interface

Listing 7.6 demonstrates the interface of the policy service. The *BuilderFactory* and the *Builder* implemented as inner static classes follow the factory and builder pattern approaches and offer a more maintainable, less error-prone and robust service construction available to the client. Thus, the builder factory offers several options for the policy service builder construction. The latter one allows to build the policy service (i.e. the method *build()*) and hook the corresponding rule and expression services up (i.e. the methods *ruleService(RuleService ruleService)* and *expressionService(ExpressionService expressionService)* correspondingly).

7.3.2 Rule Service

The rule service manages the policy rules and is responsible for their firing at runtime.

```
1  package com.materna.nodes.tree.rule;

3  import java.io.InputStream;
   import java.util.Collection;
5  import java.util.Map;

7  import com.materna.nodes.service.ServiceReference;
   import com.materna.nodes.service.registry.ServiceRegistry;
9
   public interface RuleService extends RuleEventService {
11
     public RuleRegistration registerRule(Rule rule, Map<String, Object>
       properties) throws RuleException;
13
     public Collection<RuleReference> getRuleReferences();
15   ...
   }
```

Listing 7.7: Policy Rule Service Java Interface

The interface of the rule service is presented in Listing 7.7. The central method is *registerRule(Rule rule, Map<String, Object> properties)*, which is intended for registration

of rules with their properties in form of a map. The method *registerPolicy (Policy policy, Map<String, Object> properties)* registers the specified policy object with the specified properties with the management system. Among the obligatory properties which are to be specified during the rule registration are the name, run level, and path patterns of the rule. By means of the path patterns those subtrees of the management tree are specified to which the rule applies. Thus, a more fine-grained rule usage can be achieved.

The collection of the currently registered rule references can be acquired by means of the *getRuleReferences()* method.

Moreover, the implementation of the rule service is configurable with eventing, execution and listener default parameters. The eventing specific parameters allow to specifying the event delivery timeout, logging level, and the threadpool size. The execution parameters include the threadpool size and the execution timeout. The listener parameters include the delivery mode.

7.3.3 Expression Service

Listing 7.8 shows the interface of the expression service. The expression services manages the policy expressions and is responsible for their evaluation.

```
package com.materna.nodes.tree.expression;

import java.io.InputStream;
import java.util.Collection;
import java.util.Map;

import com.materna.nodes.service.ServiceReference;
import com.materna.nodes.service.ServiceRegistration;
import com.materna.nodes.service.registry.ServiceRegistry;

public interface ExpressionService {

  public ServiceRegistration<Expression<?>> registerExpression(Expression<?>
     expression, Map<String, Object> properties) throws ExpressionException;

  public Collection<ExpressionReference> getExpressionReferences();

  public ExpressionReference getExpressionReference(String expressionId);

  public EvaluationRequest evaluateExpression(String expressionId) throws
    ExpressionException;

  ...

}
```

Listing 7.8: Policy Expression Service Java Interface

The method *registerExpression(Expression<?> expression, Map<String, Object> properties)* registers the generic type expression with its properties in form of a map. The return type of the method is a generic service registration *ServiceRegistration<Expression<?>>* typed with the corresponding expression. Thus, the service-oriented approach is followed. The collection of service references registered by the expression manager is accessible by means of the *getExpressionReferences()* methods whereas the *getExpressionReference(String expressionId)* method returns the particular expression of interest.

The central method of the interface is *evaluateExpression(String expressionId)*. The method allows to trigger the evaluation of a policy expression identified by its id. The

return type of the method is an evaluation request, the interface of which is shown in Listing 7.9.

```
1  package com.materna.nodes.tree.expression;
2
3  import java.time.Duration;
4  import java.util.Map;
5  import java.util.concurrent.CompletableFuture;
6
7  public interface EvaluationRequest {
8
9    public EvaluationRequest parameters(Map<String, Object> parameters);
10
11   public EvaluationRequest parameter(String name, Object value);
12
13   public EvaluationRequest within(Duration timeout);
14
15   public <T> T await() throws ExpressionException;
16
17   public <T> CompletableFuture<T> submit() throws ExpressionException;
18
19 }
```

Listing 7.9: Policy Expression Service Java Interface

The evaluation request allows the access to the parameters of the policy expression (*parameters(Map<String, Object> parameters)* and *parameter(String name, Object value)* respectively).

The method *within(Duration timeout)* is of particular importance, since it defines the maximal duration for the policy expression evaluation. Thus, the consideration of time supports the dependability aspect of the management. It is configurable, how long the response can last. In case the response takes longer as allowed, the self-contained managed system must continue its workflow autonomously. Thereby, the two methods for the evaluation initiation are distinguished: *await()* and *submit()*. The first one returns when it completes its calculation waiting if necessary, but not later than the predefined timeout. The last one returns immediately without waiting.

Owing to the currently available implementation of the management system in Java 8 which supports the Concurrency API improvement, the runtime management system attains a more advanced level of flexibility. The aspect of asynchronous computation, where the callback actions are scattered across the code, nested inside each other and have to face likely errors, becomes more and more complex. Further, the additional completion and computation logic (e.g. premature cancellation of the execution, time-controlled flow) can be needed. Thus, the use of *CompletableFuture* class allows to handle these aspects in the more advanced way.

7.4 Management System Characteristics

To sum up this section, the interfaces provided by the runtime management system as well as the implementation of the management system support the advanced handling of the following aspects.

Time Restriction The user can control the maximal duration of the management operations. Both the rule service and the expression service support the parametrization with default execution timeout parameters. The expression service allows also to define actively the desired strategy for the evaluation request logic.

Selective Intervention The user is free to choose the scope of the management intervention. He defines and specifies the management objects and the supported management operations. Moreover, he decides what policy expressions can be requested actively from the management system. The handling and interpretation of the management response is up to the managed system itself.

Error Handling Another important aspect addressed by the management system is the error handling. The management system operates on the data which is distributed, transient and dynamic. Thus, the management system separates the exception-handling code, groups and differentiates error types and propagates the exceptions up the call stack.

Lightweight Infrastructure In order to be manageable, the managed system basically has just to expose its status and react to reconfiguration. For this purpose, management variables are used. At bottom, the managed system sets the status variables and takes care of the corresponding reactions in case the configuration variables are set by the management system.

Administrative Isolation The concept of management domains presented in Section 4.3 unfolds in the presented solution. The hierarchical structure of the underlying management tree allows logical partitioning of management objects according to their type, location, or organizational affiliation. Thus, the management system can not only address the whole groups of management objects at once but can also benefit from the dedicated restriction of the application scope.

Thus, the used policy-based runtime management system supports a dependable, autonomous, and flexible behavior of the managed system. The managed systems stay self-contained and gain on additional characteristics at the same time.

Chapter 8

Model-Based Management of Medical Systems

In this chapter the model-based management approach described in Chapter 6 is enhanced and generalized in order to be applicable to multiple application domains. Subsequently, the approach is refined in order to be applied to the medical application domain.

In particular, the general metamodel structure is presented in Section 8.1. Firstly, the main building blocks of the metamodel are presented. From layer to layer, the intra-layer structure is described. Secondly, in order to assemble the whole metamodel, we resort to the inter-layer elements which underlay the policy refinement process.

The application domain specific knowledge gathered in Chapter 3 has flown into the concretized metamodel for the management of medical devices and systems. Section 8.2 shows how the general metamodel has been transformed into this domain metamodel.

8.1 General Metamodel Structure

8.1.1 Metamodel Layers

The three-layered system model reflects the system from three different points of view: use cases, service infrastructure and implementing soft- and hardware components. In the following, the main building blocks of each layer are introduced. In doing so, we make a distinction between the *system elements* and *policy elements* as referred in Section 6.3.

8.1.1.1 "Use Cases" Layer

The top layer of the model provides the most abstract view of the system. It is described at the highest level of abstraction representing the mission of the system and expressing the main stakeholder goals.

System Elements

Being a widely used and long-standing term, a *use case*, has been defined in literature in very different ways. In [CL01], the authors discuss the diverse focuses of well-established definitions. For example, the OMG [Obj11] defines a use case in the UML specification as:

> "... a set of actions performed by a system, which yields an observable result that is, typically, of value for one or more actors or other stakeholders of the system."

According to [Coc00],

> "A use case captures a contract between the stakeholders of a system about its behavior. The use case describes the system's behavior under various conditions

> as it responds to a request from one of the stakeholders, called the primary actor. The primary actor initiates an interaction with the system to accomplish some goal."

These definitions vary greatly from the original definition introduced in 1992 by Jackobson [JCJO92]:

> "A use case is a specific way of using the system by using some part of the functionality. Each use case constitutes a complete course of interaction that takes place between an actor and the system."

Since we focus on user requirements and management aspects in this work, we are going to shift the emphasis on the purpose-centered system-behavior viewpoint and define a use case, similar to [CL01] and [Wie03], as:

> a self-contained, concise, and well-defined task comprising the user intentions and the system responsibilities expressed using the language of the application domain, free of implementation, technology details and particulars.

Thus, the use cases modeled on this layer involve certain *functions* which have to be performed in order to achieve the intended behavior of the system. They define the scope of the system and structure the main functional activities of the system. *Actors* represent the types of users interacting with the system and typically initiating the use case. They stand representative for primary participators in the described scenario. Further, relevant *assets* used within the use case are modeled on this layer. They include data, material or financial resources contributing to the use case delivery. Along with assets, the provision of use cases takes account of certain *aspects*, also presented in the model. Figure 8.1 demonstrates the main model building blocks which form the "Use Cases" layer.

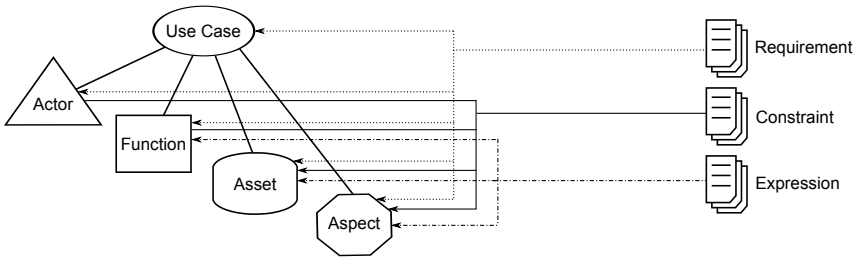

Figure 8.1: Use Case Layer Building Blocks

Policy Elements

The desired system behavior is expressed by means of concise *requirements* which can address various facets (e.g., performance, security, billing, application domain). The requirements are formulated at a high level and determine the target system state. They can also express the necessity or prerequisite of the system state. The requirements are assigned to the use cases, functions, or directly to the assets modeled on his layer.

Along with requirements, it is sometimes desirable to define some *constraints* which concern the function, asset, actors and aspects of the use case. They express a condition that must be satisfied, a certain restriction of the system state. The constraints are assigned to the corresponding model elements.

In order to evaluate, if the specified constraints are satisfied, the requirements are met or just to judge on the current system state, it is sometimes advisable to request the

evaluation of a predefined *expression*. These are modeled on the top layer of the model and refer to the corresponding functions, assets and aspects.

Alike other model elements of this layer, the requirements, constraints and expressions are to be formulated in the language of the application domain and must be comprehensible to the domain specialist.

Summary

The identified model elements of the "Use Cases" layer and corresponding intra-layer associations standing for the relations between the model elements are summarized in Table 8.1.

	Use Case	**Actor**	**Function**	**Asset**	**Aspect**	**Requirement Constraint Expression**
Use Case			performs▶	uses▶ produces▶	takesAccountOf▶	requests▶
Actor	initiates▶ participatesIn▶					
Function						
Asset						
Aspect						
Requirement Constraint Expression	refers▶	refers▶	refers▶	refers▶	refers▶	

Table 8.1: Model elements of the "Use Cases" layer

8.1.1.2 "Services" Layer

The middle layer provides the service-oriented view of the system. Following the main service-orientation principles, we focus on the following design aspects stated by [Erl07]:

Standardized Service Contract Expressing service functionality, purpose, and capability should be done via a standardized and appropriately granular service contract.

Loose Coupling The level of dependency between the service contract, its implementation, and its service consumers should be as low as possible.

Abstraction Hiding as much of service details and logic as possible provides for composability. Services can be regarded as "black boxes" where the only information available is the one published in the service contract.

Reusability Services should be designed with intention to promote reuse. Thus, they are to be positioned as enterprise resources with agnostic functional contexts.

Autonomy Increasing service's reliability and behavioral predictability is supported by means of exercising a high level of control over the underlying runtime execution environment by services.

Statelessness Delegation and deferral of state management benefits service scalability and resource consumption.

Discoverability Supplemented meta data should allow to discover and interpret the services.

Composability Services should be capable of participating as effective composition participants in a composition of any size and complexity.

System Elements

Resorting to the existing standards and taking into consideration the above listed aspects, we refer to the term of *service* in the way OASIS does it within the scope of the Reference Model for Service Oriented Architecture 1.0 [OAS06]:

> "A service is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description."

The similar intentions are targeted by *applications*. Services, however, address in most cases smaller and more isolated problems than applications. Applications often perform a wide range of operations and in doing so expose and use other services. Thus, the model layer contains services and applications providing the tightly defined set of functions specified on the use cases layer.

The applications are used by the *users* representing the actors modeled on the top layer. Figure 8.2 demonstrates the main model building blocks of the "Services" layer.

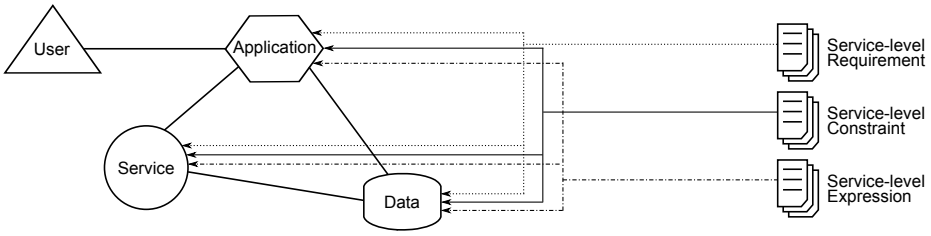

Figure 8.2: Service Layer Building Blocks

The OASIS SOA Reference Model emphasizes the importance of precise definition of information and data exchanged with a service. For this purpose, it introduces the services's information model which "includes the format of information that is exchanged, the structural relationships within the exchanged information and also the definition of terms used" [OAS06]. Especially in case of data exchange across an ownership boundary, these issues require close attention. The data interpretation on a semantic as well as syntactic level is highly application dependent. Taking into consideration these problems, we also introduce *data* model elements connected to the services and applications encountering them.

Policy Elements

The OASIS SOA Reference Model endorses the association of services with policies which provide "necessary information for prospective consumers to evaluate if a service will act in a manner consistent with the consumer's constraints" [OAS06].

Thus, corresponding service-level policy elements are required in order to express specific requirements on this abstraction level. For this purpose, policy constructs are introduced which define the usage, provision and deployment of services, applications and related data. Similarly to the policy elements of the "Use Cases" layer, these policy constructs have a form of *service-level requirements*, *constraints*, and *expressions*. In contrast to them, however, these elements can be specified not only during the design time, but rather generated and added to the model automatically during the refinement process.

The service-level policy elements refine the policy elements of the "Use Cases" layer and address those service-level specifics which are to be considered in order to satisfy the defined requirements and constraints. They also expand the defined requirements and

constraints from the service-oriented point of view. Moreover, the policy elements are formulated using the service-level specific terminology.

In the model, the policy constructs are connected to the associated application, service and data elements.

Summary

Table 8.2 brings together the identified model elements and the intra-layer associations.

	Application	**Service**	**Data**	**User**	**Service-level Requirement Constraint Expression**
Application		uses▶ exposes▶	encounters▶		requests▶
Service			encounters▶		
Data					
User	uses▶ interactsWith▶				
Service-level Requirement Constraint Expression	refers▶	refers▶	refers▶	refers▶	

Table 8.2: Model elements of the "Services" layer

8.1.1.3 "Components" Layer

The bottom layer of the system model represents the most concrete technical view of the system. Elements depicted here compose the runtime system and deliver to the user the tangible services and applications defined on the middle layer.

System Elements

The layer comprises *devices* which are available at runtime. To be specific, we model medical domain specific devices introduced in Section 3.3, instruments and appliances as well as common IT devices like computers, peripherals, and telecommunication equipment. Devices can host required *software platforms*, acquire or allocate relevant *data sources*.

As a special case we also model *sensors*, devices that detect or measure some physical quantities (e.g. physiological or ambient parameter) and respond with a transmitted signal. They can be self-contained or hosted by the corresponding device.

Software platforms hosted by the devices, in their turn, hold *software components* and provide an operating environment under which they run.

The term "software component" provides a considerable room for diversity of definitions. In [Ols06] a comprehensive research on the term's definition and historical evolution has been carried out. In this work we rely on the definition worked out by Szyperski [SGM02]:

> "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."

The details of this definition are discussed in [Vö03]. Thus, "unit of composition" implies that the components are meant to be used and reused in composition with other components. Together, they can assemble a component-based application. Assembling the applications from components must not necessarily be accomplished by the same developers.

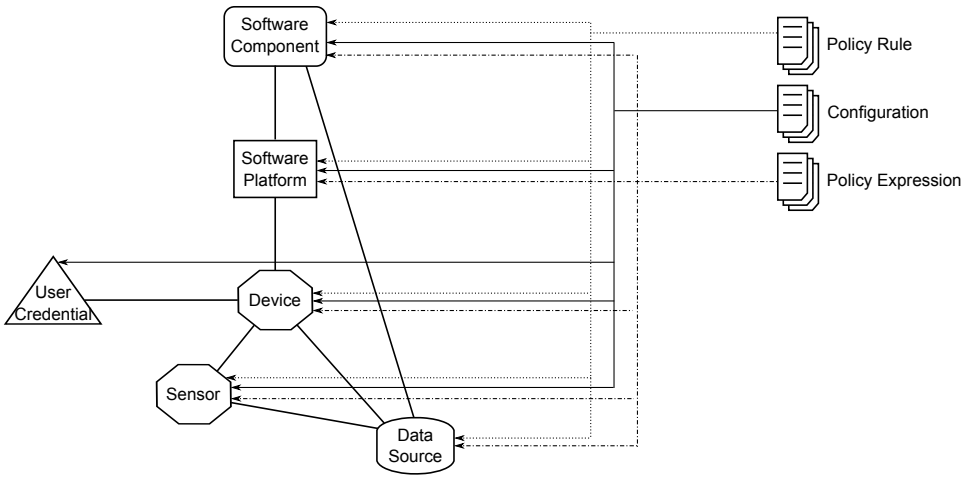

Figure 8.3: Components Layer Building Blocks

For instance, suppose the implementation technology is OSGi[1]. In this case, a device hosts an OSGi Framework (software platform), where a set of OSGi bundles (software components) is running. The bundles can expose and use OSGi services (services).

In order to prove user identity and control access to data and other resources, *user credentials* are used. The model element standing exemplarily for a credential of any form (user account-password combination, public key certificate, biometrics, etc.) is modeled on the "Components" layer. In case credentials are to be stored physically on a computer, a corresponding association is modeled between the user credential and device element.

Policy Elements

The policy elements on the "Components" layer are the concrete technical representations of the previously defined abstract requirements, constraints and expressions. They are the management artifacts used by the management system in order to perform its functions. We distinguish three types of the policy elements: *policy rules*, *policy expressions* and *configurations*.

A policy rule represents an *event-condition-action* rule following the paradigm of positive imperative policies presented by the Sloman and Moffett working group [MS93]. That means, on a certain event concerning some components, in case a condition applies to a set of components, a corresponding action on a set of components is performed. In order to model this, we associate the policy rule element with the corresponding model elements. The *"event"* part of the rule refers to a component which triggers the policy rule, whereas the *"condition"* part of the rule supposes checking the status of the corresponding components. If the case may be, the *"action"* is undertaken on the associated model elements.

A policy expression represents a technical representation of the approach presented in [DKK+10b]. A managed component can request from the management system to evaluate a policy expression at runtime. Depending on the result, the component can react by adapting

[1] http://www.osgi.org/

its program flow. Thus, the requesting component (software component) is associated with the corresponding policy expression. The components (software components, platforms, devices, sensors, data sources) which status is considered for the evaluation of the policy expression are associated with the policy element.

The "Components" layer also includes the concrete technical configurations of the components needed in order to fulfill the defined abstract requirements and constraints. The configuration elements are associated directly with the corresponding elements (user credentials, software components, platforms, devices and sensors).

To sum up, by means of the configurations the initial state of the managed system can be set. The policy rules are used to govern the management process by specifying how the state of the managed system changes reactively (and/or even proactively). The policy expressions allow the managed system to inquire its state at runtime. In contrast to the more abstract layers in the model, the policy elements become more technical and specific. In order to keep the policy definition technology independent, it is appropriate to operate with *management variables* of the managed elements. Thus, we resort to a *status variable* in order to express the state of the component and to a *configuration variable* in order to configure the component.

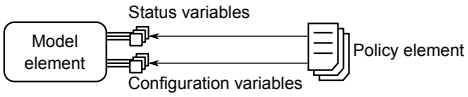

Figure 8.4: Used Notation: Management Variables

In the following, we extend the model with the management variable elements and use the notation shown in Figure 8.4. The management variables are assigned to the corresponding model elements, whereas the policy and control elements are associated with those variables, on which they are defined on.

Summary

The model elements and associations of the "Components" layer are listed in Table 8.3.

	Device	**Sensor**	**Data Source**	**Software Platform**	**User Credential**	**Software Component**	**Policy Rule Policy Expression Configuration**
Device		hosts▶	acquires▶ allocates▶		stores▶		
Sensor			acquires▶ allocates▶				
Data Source							
Software Platform	runsOn▶						
Software Component		acquires▶ allocates▶		runsOn▶			requests▶
User Credential							

to be continued...

...to be continued...

	Device	Sensor	Data Source	Software Platform	User Credential	Software Component	Policy Rule Policy Expression Configuration
Policy Rule **Policy Expression** **Configuration**	refers▶	refers▶	refers▶	refers▶	refers▶	refers▶	

Table 8.3: Model elements of the "Components" layer

8.1.2 Building Together the Metamodel

In order to assemble the whole system model and be able to derive the management relevant information, we need a kind of "glueing" elements to put the single model building blocks together. Such elements support the policy refinement process by indicating how the more abstract elements and their attributes are to be translated into the more technical ones. In other words, these elements give information, how to traverse the system model from top to bottom and express how to realize the abstract requirements on the more technical level.

Firstly, *refinement relations* are used for this purpose. They comprise primarily top-down inter-layer associations. Secondly, *refinement patterns* can imply the strategy for retrieving the model elements and their attributes on the next lower level. A part of the elements is added to the model during the modeling process by the domain specialist. This corresponds to the modeling step of the model-based management presented in Section 6.2. However, some of the model elements are added to the model in an automated manner. These are mainly the elements derived within the policy refinement step. In the following, the main types of refinement relations are introduced.

Figure 8.5 sums up the main building blocks described above and extends them with inter-layer associations. These associations represent refinement relations between model elements of adjacent layers. Thus, model elements of an upper layer are connected to those model elements of the next lower layer which provide for their realization, implementation, or provision.

Moreover, we also extend the model with the management variables introduced in Section 8.1.1.3.

In the following, the single inter-layer relationships are described. They build the basis for the refinement process by reducing the abstraction grade from top to bottom and providing more and more technical and implementation details to the next lower layers.

8.1.2.1 From "Use Cases" to "Services"

Table 8.4 illustrates the relationship between the "Use Cases" and "Services" layers.

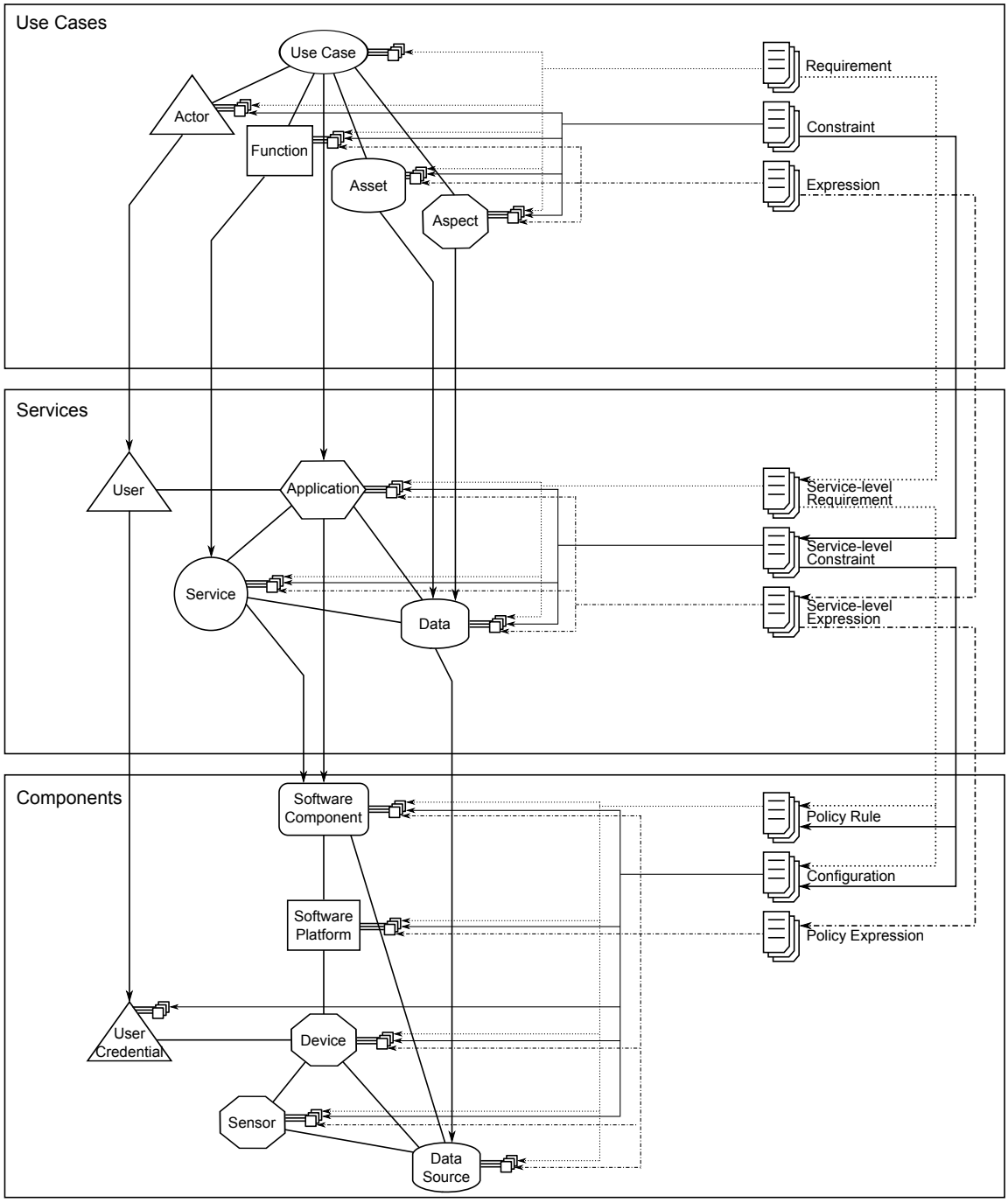

Figure 8.5: Synopsis of Main Building Blocks

	Application	Service	User	Data	Service-level Requirement	Service-level Constraint	Service-level Expression
Use Case	realizedBy▶						
Actor			refinesTo▶				
Function		providedBy▶					
Asset				refinesTo▶			
Aspect				refinesTo▶			
Requirement					refinesTo▶		
Constraint						refinesTo▶	
Expression							refinesTo▶

Table 8.4: Inter-layer Associations "Use Cases" - "Services"

The use cases modeled on the "Use Cases" layer are realized by means of a set of applications. Thus, a vertical association *realizedBy* ▶ is used to express this relationship and connect the use case element with the corresponding application elements. The set of functions performed within the use case is provided by the corresponding services modeled by the service elements of the "Services" layer. This constellation is modeled by vertical associations *providedBy* ▶ between the use case element and a set of corresponding service elements.

Assets which are used or produced within the use cases are refined to the data elements involved by the service provisioning on the "Services" layer. The corresponding elements are connected in the model by *refinesTo* ▶ relationship. Similarly, the aspects of which the use cases take account are associated with the corresponding data elements.

Actors involved on the "Use Case" layer are reflected in user model elements on the "Services" layer. Accordingly, *refinesTo* ▶ relationships are used to model this circumstance.

An inter-layer association of special interest is a *refinesTo* ▶ association between the policy elements of the "Use Cases" and "Services" layers. It is a matter of derivation of high-level policies into the more concrete and technical representation from the service-oriented point of view. Policies, available on the "Use Cases" layer in form of abstract requirements, constraints and expressions applied to the model elements are refined into service-specific policy constructs. The associations of this type are added to the model within the refinement process in an automated manner.

8.1.2.2 From "Services" to "Components"

The relationships between the "Services" and "Components" layers are summarized in Table 8.5.

	Software Component	User Credential	Data Source	Policy Rule	Configuration	Policy Expression
Application	realizedBy▶					
Service	providedBy▶					
User		refinesTo▶				
Data			refinesTo▶			
Service-level Requirement				refinesTo▶	refinesTo▶	
Service-level Constraint				refinesTo▶	refinesTo▶	
Service-level Expression						refinesTo▶

Table 8.5: Inter-layer Associations "Services" - "Components"

The applications of the middle layer are realized by the software components residing on the bottom level. Corresponding associations are of type *realizedBy* ▶.

Similarly, the services of the middle layer are provided by the software components of the bottom layer. Thus, *providedBy* ▶ association connects the corresponding service and software component elements of the adjacent layers.

The data which has been used or produced by services on the "Services" layer is refined to the data sources on the "Components" layer. Thus, *refinesTo* ▶ relationship connects the data element with the appropriate data resource element.

The identity of application users are represented by the credentials which are used by the system to prove user's identity at runtime. That is modeled by *refinesTo* ▶ association which connects the users of the "Services" layer with the corresponding user credentials elements of the "Components" layer.

The above described associations are added to the model and specified during the modeling process. Additionally, refinement relationships acquired within the automated policy refinement step from the "Services" to the "Components" layer are present in the model. This refinement step provides for obtaining the highest level of detail and enriches the model with concrete implementation specifics. Thus, the service-specific requirement and constraint expression constructs of the middle layer are refined into policy rule and configuration elements defined on management variables of the "Components" layer elements. This is expressed in model by *refinesTo* ▶ associations. The expression constructs defined on the middle layer are refined into the concrete policy expressions of the bottom layer. The corresponding *refinesTo* ▶ association connects the corresponding elements of the adjacent layers.

8.2 Medical Domain Metamodel

Based on the application domain description provided in Chapter 3 we construct the metamodel in line with the worked out model building blocks as well as refinement relations presented above. The metamodel elements follow the generalization principle and are organized in a hierarchy using the inheritance or "is a" relationship. That allows to extend

the metamodel according to the current subdomain of interest. We are going to focus exemplarily on the neurological and cardiological subdomains in the following sections.

8.2.1 "Use Cases" Layer

The system elements resided on the "Use Cases" layer include *use cases*, *actors*, *functions*, and *assets*. Additionally, the layer contains *requirements* elements which refer to the system elements and express the desired system behavior.

Actors

As the medical service delivery is a central part and purpose of the most courses of actions and processes within the medical domain, it is advisable to examine the matter starting with the aspect of medical service provision. On the part of medical service consumer, the essential actor is the *Patient*.

Pursuing the idea, that the medical disease can be used as a classifying criterion, we follow the presented ICD-10 [Wor10b] classification and introduce within the metamodel a hierarchy of metamodel elements corresponding to each of the disease classification chapter, block of categories, category and subcategory. In case of multimorbidity, the primary disease is taken as the main criterion. Thus, the subtypes of the *Patient* metamodel element include the following elements:

- *Patient with certain infectious and parasitic disease*
- *Patient with neoplasm*
- *Patient with disease of the blood and blood-forming organs and certain disorders involving the immune mechanism*
- *Patient with endocrine, nutritional and metabolic disease*
- *Patient with mental, behavioral and neurodevelopmental disorder*
- *Patient with disease of the nervous system*
- *Patient with disease of the eye and adnexa*
- *Patient with disease of the ear and mastoid process*
- *Patient with disease of the circulatory system*
- *Patient with disease of the respiratory system*
- *Patient with disease of the digestive system*
- *Patient with disease of the skin and subcutaneous tissue*
- *Patient with disease of the musculoskeletal system and connective tissue*
- *Patient with disease of the genitourinary system*
- *Patient with pregnancy, childbirth and the puerperium*
- *Patient with certain condition originating in the perinatal period*
- *Patient with congenital malformation, deformation and chromosomal abnormality*

- *Patient with symptom, sign and abnormal clinical and laboratory finding, not elsewhere classified*
- *Patient with injury, poisoning and certain other consequence of external causes*
- *Patient with external cause of morbidity*
- *Patient with factor influencing health status and contact with health services*

These metamodel elements inherit directly from the *Patient* metamodel element and generalize the corresponding blocks of categories, in their turn. For example, the metamodel element *Patient with disease of the nervous system* is specialized amongst others by the following subtypes of metamodel elements:

- *Patient with inflammatory disease of the central nervous system*
- *Patient with systemic atrophy primarily affecting the central nervous system*
- *Patient with extrapyramidal and movement disorder*
- *Patient with other degenerative disease of the nervous system*
- *Patient with demyelinating disease of the central nervous system*
- *Patient with episodic and paroxysmal disorder*
- *Patient with nerve, nerve root and plexus disorder*
- *Patient with polyneuropathy and other disorder of the peripheral nervous system*
- *Patient with disease of myoneural junction and muscle*
- *Patient with cerebral palsy and other paralytic syndrome*
- *Patient with other disorder of the nervous system*

Expanding the hierarchy further, these metamodel elements generalize the corresponding categories of patients. To give an example, the metamodel element *Patient with other degenerative disease of the nervous system* is specialized by the following subtypes of metamodel elements:

- *Patient with Alzheimer's disease*
- *Patient with other degenerative disease of nervous system, not elsewhere classified*
- *Patient with other degenerative disorder of nervous system in diseases classified elsewhere*

The most fine-grained information is given by the specialization of a metamodel element of the next lower hierarchical level. E.g., the metamodel element *Patient with Alzheimer's disease* include the following subtypes of metamodel elements:

- *Patient with Alzheimer's disease with early onset*
- *Patient with Alzheimer's disease with late onset*
- *Patient with other Alzheimer's disease*

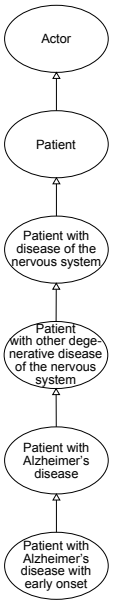

Figure 8.6: Metamodel Excerpt: Patient

- *Patient with Alzheimer's disease, unspecified*

Thus, in order to model a patient suffering from the Alzheimer's disease with early onset (ICD-10 diagnosis code G30.0) we use exemplary the following metamodel excerpt (Figure 8.6):

On the part of medical service provider, the essential actor stems from the *Health worker*. Thus, the metamodel includes the corresponding metamodel element at the top of the hierarchy. The hierarchical structure is based on the differences in skill level and skill specialization which are necessary to accomplish the tasks and duties of jobs. Resorting to the presented International Standard Classification of Occupations (ISCO, 2008) of WHO, we distinguish the following subtypes of the *Health worker* metamodel element:

- *Health professional*
- *Health associate professional*
- *Personal care worker in health services*
- *Health management and support personnel*
- *Other health service provider not elsewhere classified*

These metamodel elements generalize the subtypes of occupation categories of actors in the medical sector. Thus, e.g., the metamodel element *Health professional* is specialized by the following subtypes of metamodel elements:

- *Generalist medical practitioner*
- *Specialist medical practitioner*
- *Nursing professional*
- *Midwifery professional*
- *Traditional and complementary medicine professional*

- *Paramedical practitioner*
- *Dentist*
- *Pharmacist*
- *Environmental and occupational health and hygiene professional*
- *Physiotherapist*
- *Dietician and nutritionist*
- *Audiologist and speech therapist*
- *Optometrist and opthalmic optician*
- *Health professional not elsewhere classified*

Expanding the hierarchy further, these metamodel elements generalize the corresponding categories of health professionals. For example, the *Specialist medical practitioner* metamodel element is specialized by the following subtypes of metamodel elements:

- *Doctor in obstetric and gynaecological specialties*
- *Doctor in paediatrics*
- *Doctor in psychiatric specialties*
- *Doctor in the medical group of specialties*
- *Doctor in the surgical group of specialties*
- *Doctor in specialties not elsewhere classified*

These metamodel elements generalize specialists according the their area of practice, in their turn. Thus, e.g., the metamodel element *Doctor in the medical group of specialties* includes the following subtypes:

- *Cardiologist*
- *Dermatovenerologist*
- *Doctor in forensic medicine*
- *Gastroenterologist*
- *Haematologist*
- *Immunologist*
- *Doctor in infectious disease*
- *Doctor in internal medicine*
- *Neurologist*
- *Doctor in occupational medicine*
- *Oncologist*

- *Radiologist*
- *Doctor in rehabilitative medicine*
- *Doctor in respiratory medicine*
- *Urologist*

Similarly to the patients hierarchy, we assume that each specialist should only be counted once, according to his main area of practice or the current use case. To give an example of metamodel elements involved in modeling a doctor in neurology, who provide a service of any kind to a patient suffering from a degenerative disease of the nervous system, we demonstrate an excerpt from the metamodel (Figure 8.7):

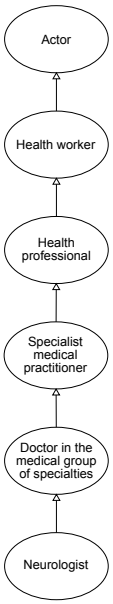

Figure 8.7: Metamodel Excerpt: Health Worker

It is to take into consideration, that a health worker can also act as a service consumer. Depending on the use case, he can use a domain specific service with an educational or informational objective. Another example is the usage of assisting applications, expertise and legacy systems by a health worker. Accordingly, the actors acting as a service provider include the following metamodel elements:

- *Health care organization*
- *Public health authority*
- *Insurance company*
- *Pharmaceutical technology manufacturer*
- *Medical technology manufacturer*
- *Medical technology vendor*
- *Medical data processing center*

These metamodel elements generalize the stakeholders according to their liability function, organizational affiliation, field of activity and sphere of interest. Thus, for example, the metamodel element *Health care organization* has got the following subtypes:

- *Hospital*
- *Clinic*
- *Prevention facility*
- *Rehabilitation facility*
- *Medical practice*
- *Ambulance*
- *Rescue service*
- *Laboratory*
- *Radiology*
- *Pharmacy*
- *Medical supply store*
- *Scientific organization*

An excerpt from the metamodel presenting a radiology as a medical domain actor is demonstrated in Figure 8.8.

Figure 8.8: Metamodel Excerpt:Radiology

Use Cases

Regarding the plenty of application domain specific use cases presented in the description of typical course of actions within the medical domain, we can distinguish between the following main use cases presented in the metamodel:

- *Treatment and prevention*
- *Information and education*
- *Research*
- *Legacy procedure*

These metamodel elements inherit directly from the *Use Case* metamodel element and generalize the typical use cases specified for the application domain of interest according to the assigned purpose and general objective. Expanding the hierarchy further, the subtypes of the metamodel elements specialize the subgroups of use cases more precisely. E.g., the *Treatment and prevention* use case metamodel element is specialized by the following subtypes of metamodel elements:

- *Therapy and diagnostics*
- *Prevention*
- *Monitoring*
- *Rehabilitation*
- *Assisted living*
- *Social services and care*

In their turn, these metamodel elements are specialized by the more specific subtypes. Thus, following the OPS-301 [DIM13], the official classification of operational procedures widely used by the German hospitals and physicians, the *Therapy and diagnostics* metamodel element can be expanded to following metamodel nodes:

- *Diagnostic procedure*
- *Radiology*
- *Operation*
- *Drugs*
- *Non-surgical therapeutic measure*
- *Additional measure*

These metamodel elements are specialized by the following subtypes of metamodel elements with regard to the involved technique. Thus, for example, the *Radiology* metamodel element generalizes the following subtypes:

- *X-ray imaging*
- *Computed tomography*
- *Ultrasonography*
- *Magnetic resonance imaging*
- *Radionuclide imaging*
- *Positron emission tomography*
- *Single photon emission computed tomography*
- *Fluoroscopy imaging*

Going down the hierarchy, the metamodel elements are specialized further by their subtypes according to the application target. E.g., the *Magnetic resonance imaging* metamodel element generalizes amongst others the following subtypes:

- *Magnetic resonance angiography*
- *Head MRI*
- *Chest MRI*
- *Abdomen and pelvis MRI*
- *Bone and joint MRI*
- *Spine MRI*

To give an example of the metamodel hierarchy of elements representing a standard use case covering a magnetic resonance imaging procedure of the head used for diagnostics of degenerative disease of the nervous system, we introduce the following excerpt of the metamodel (Figure 8.9):

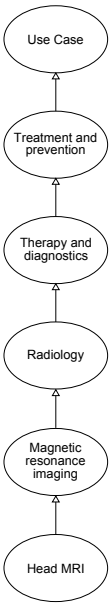

Figure 8.9: Metamodel Excerpt: MRI

Functions

In order to model the functionality carried out within the presented above use cases, a set of application specific functions is introduced. These metamodel elements inherit directly from the *Function*.

- *Medical data handling*
- *Medical device integration*
- *Patient care coordination*
- *Medical application workflow control*
- *Documentation, reporting and archiving*

These metamodel elements generalize the main function types characteristic of the medical application domain. They are specialized by the subtypes of functions introducing the functionality on the more concrete level. Thus, for example, depending on the underlying intention and task the *Medical data handling* metamodel element generalizes the following subtypes:

- *Medical data acquisition*
- *Medical data inquiry*
- *Medical data processing*
- *Medical data analysis*
- *Medical data storage*
- *Medical data representation*

Regarding the technical way of the medical data acquisition as well as the observed parameter type, the metamodel elements can be specialized by the subtypes providing the more precise level of abstraction in the description of the functions. Thus, the *Medical data analysis* can be extended by the following metamodel elements:

- *Medical narrative data analysis*
- *Medical textual data analysis*
- *Medical numeric data analysis*
- *Medical image data analysis*
- *Medical device input data*

In their turn, these metamodel elements generalize the subordinate functions described by the corresponding metamodel elements. E.g., *Medical image data analysis* metamodel element is specialized with regard to the volumetric property by the following subtypes of metamodel elements:

- *3D analysis*
- *2D analysis*

Going down the hierarchy, these metamodel elements generalize the more specific types of functions. Thus, the metamodel element *Volumetric analysis* generalizes according to lesion and organ specialty amongst others the following subtypes:

- *Volumetric abdominal analysis*
- *Volumetric chest image analysis*
- *Volumetric neuroimage analysis*
- *Volumetric bone and joint analysis*
- *Volumetric spine analysis*

The underlying image reconstruction technique can be used as a classification factor on the next lower level of hierarchy. Thus, *Volumetric neuroimage analysis* is specialized by the following subtypes of metamodel elements:

- *Voxel-based morphometric analysis*
- *Surface-based morphometric analysis*
- *Deformation-based morphometric analysis*
- *Tensor-based morphometric analysis*

To give an example of the commonly used by the diagnosis of the degenerative disease of the nervous system cortical thickness analysis, we demonstrate an excerpt from the metamodel in Figure 8.10.

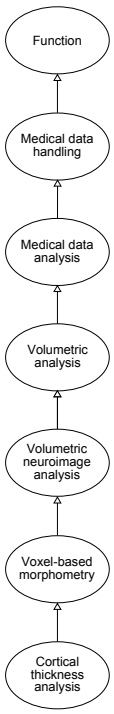

Figure 8.10: Metamodel Excerpt: Cortical Thickness Analysis

Assets

The main assets involved within the delivery of the use cases characteristic of the medical domain include a wide range of data, material and financial resources and objects. Referring to the SNOMED CT [2] which provides a most comprehensive clinical terms terminology, we define the following subtypes of assets presented as metamodel elements:

- *Clinical finding*
- *Procedure*
- *Situation with explicit context*
- *Observable entity*
- *Body structure*
- *Organism*

[2] http://www.ihtsdo.org/fileadmin/user_upload/doc/ (Aug.2013)

- *Substance*
- *Pharmaceutical biologic product*
- *Specimen*
- *Physical object*
- *Physical force*
- *Event*
- *Environment and geographic location*
- *Social context*
- *Staging and scales*
- *Qualifier value*
- *Special concept*
- *Record artifact*

These metamodel elements inherit directly from the metamodel element *Asset* and generalize the main subtypes according to their genre and purpose. Thus, according to the LOINC [3] document ontology the *Record artifact* metamodel element representing the records and documents typical of the medical domain can be specialized by the following subtypes:

- *Administrative document*
- *Consent document*
- *Clinical trial document*
- *Correspondence*
- *Public health document*
- *Legal document*
- *Clinical document*

In their turn, these metamodel elements generalize the subordinate assets providing a more precise level of description. For example, with regard to the underlying form, the *Clinical document* metamodel element is specialized amongst others by the following metamodel elements:

- *Communication note*
- *Conference evaluation note*
- *Consultation note*
- *Counseling note*
- *Diagnostic study note*

[3] http://loinc.org/

- *Digital photographic image*
- *Education note*
- *Evaluation and management note*
- *Medication management note*
- *Intervention procedure note*
- *Pathology procedure note*
- *Referral note*
- *Supervisory note*
- *Triage and care note*
- *Administrative note*

Assume a typical for diagnostics of degenerative disease of the nervous system diagnostic procedure - MRI. An asset which is produced within the procedure, a radiology diagnostic study note, can be demonstrated by the corresponding except from the metamodel (Figure 8.11).

Figure 8.11: Metamodel Excerpt: A Radiology Diagnostic Study Note

Requirements

Regarding the main domain specific issues addressed in Chapter 3, we concentrate on the common requirements ordered to the general sectors: security, accountability, billing, resources, and performance, as well as on the requirements specific to the medical application domain. According to this, we introduce the following metamodel elements representing the main requirement categories:

- *Security requirement*
- *Nonrepudiation requirement*
- *Billing requirement*

- *Resources requirement*
- *Performance requirement*
- *Medical requirement*

These metamodel elements inherit directly from the metamodel element *Requirement*. In their turn, they generalize the corresponding subcategories of the requirements. Thus, the *Security requirement* is specialized by the subordinate types:

- *Information security requirement*
- *Device security requirement*
- *Application security requirement*
- *Technological security requirement*

Expanding the hierarchy further, these metamodel elements generalize the corresponding subtypes of requirements. Thus, e.g., *Information security requirement* is specialized by the following subtypes:

- *Information confidentiality requirement*
- *Information integrity requirement*
- *Information availability requirement*

An excerpt from the metamodel hierarchy which is relevant for the description of the confidentiality requirement applied to the radiology diagnostic study note can be seen in Figure 8.12.

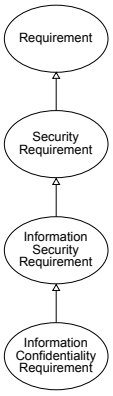

Figure 8.12: Metamodel Excerpt: Information Confidentiality Requirement

8.2.2 "Services" Layer

Similar to the "Use Cases" Layer, the "Services" Layer is resided by the system and policy elements. The system elements presented on this layer include *services (applications)*, *service bindings*, *data*, *SLAs*, *service consumer, customer* and *provider*. They are dedicated to present the system from the service-oriented point of view. The corresponding requirements expressing the agreed upon and targeted service levels are presented in the model in form of *requirements* elements.

Services (Applications)

Unlike [JAW11], who classifies the services and applications used within the medical domain according to their application-specific purpose (e.g. information, decision, education, management, and rating aids), we regard the system from the service-oriented point of view and concentrate on service-oriented issues like [Coh07] does. The services directly support the business process by being "discovered", orchestrated, and used as a system solution. They are self-contained units of work with well defined and described capabilities. A composition of services implements the actual business process by means of applications. According to the [Coh07], the "service-oriented economies thrive by promoting composition." Thus, new application-specific business logic and functions are composed together with existing business capabilities which are built in-house, bought or leased as packaged component-based solutions. Looking at the types of services, two main types are to be distinguished. The first type includes the services which provide the common infrastructure (or bus), communication, and facilities. The actual application logic is provided by the second type – application services. Following this approach, we identify the following metamodel elements:

- *Infrastructure service*
- *Application service*

These metamodel elements inherit directly from the *Service* metamodel element. Application services, in their turn, divide into four main subtypes: *entity*, *capability*, *activity*, and *process services* ([Coh07]). *Entity services* support and define access to business entities of the system. They usually implement the standard database actions: create, read, update, and delete (CRUD). *Capability services* provide the action-centric building blocks of the business process while implementing organization's business-level capabilities. Action centric business logic on the application level is exposed by *activity services*. The notion of *process services* is to implement a business process tying together their business logic with the composed functionality of the involved services in order to create a plan for the operation of the business. Thus, the corresponding metamodel elements inherit from the *Application service* element:

- *Entity service*
- *Capability service*
- *Activity service*
- *Process service*

Providing a complete hierarchy of metamodel nodes for the medical domain is a overwhelming task. However, we are going to identify the main principles of handling, in order to construct a metamodel of the *"Services" layer* for a subdomain of interest. Taking into consideration the metamodel structure identified for the *"Use Cases" layer*, the following guidelines can be applied.

Medical assets and actors of the *"Use Cases" layer* have their representatives on the *"Services" layer*. The services covering management of those components adhere generally to the *Entity services*. They can be divided into corresponding subtypes reflecting the hierarchy of defined metamodel elements for actors and assets, e.g. *Entity service* → *Asset management service* → *Record artifact management service* → *Clinical document management service* → *Diagnostic study note management service* → *Radiology diagnostic study note management service*.

Functions presented on the *"Use Cases" layer* are implemented by the corresponding services of the *"Services" layer*. The latter belong to *Capability* and *Activity services* depending mostly on their scope and granularity. Thus, services realizing functions which are action-centric on the application level are to be subordinated to *Activity services*. Their subtypes should be consequently organized in a similar way as the superior functions of the the *"Use Cases" layer*. Services providing functions which implement organization business-level capabilities belong to *Capability services*. They are to be divided into subtypes according to the hierarchical structure of the corresponding metamodel elements representing functions, e.g. *Capability service* → *Medical data handling service* → *Medical data analysis service* → *Volumetric analysis service* → *Volumetric neuroimage analysis service* → *Voxel-based morphometry service* → *Cortical thickness analysis service*.

Medical use cases of the *"Use Cases" layer* are presented on the *"Services" layer* as compositions of services and services covering more complex workflows and business processes. They are also reflected in services which have application workflow control functions or even applications themselves. Services of such nature are subordinated to the *Process services*. The hierarchy of the metamodel nodes representing the *Process services* should reflect the corresponding order of use cases and functions of the *"Use Cases" layer*, e.g. *Process service* → *Treatment and prevention service* → *Therapy and diagnostics service* → *Radiological procedure management service* → *MRI management service* → *Head MRI management service*.

To give an example of metamodel elements presenting each service type, we consider the following sample composition of services in order to implement a business process. Let us regard a standard computer-assisted MRI diagnostic procedure routine. A patient management service provides an interface to the common CRUD procedures for patient data management. Being a data-centric component, it surfaces and abstracts the business entity in the system: the patient. A patient service is an example of the *Entity service*. An example of the *Capability service* is a third-party interfacing service such as PACS integration service that can be used for communication with an external PACS system in order to upload the acquired MRI data to the central archiving directory. Another example of the *Capability service* includes an MRI acquisition service which exposes the core functionality of the composite business process while providing a short-running business activity. A more complex application specific unit of functionality such as head MRI picture processing module is exposed by the corresponding type of the *Activity service*. At last, composing the functionality offered by the above mentioned services is the task of MRI diagnostic procedure management service which accesses the patient data, chooses the scheduled MRI type, triggers the corresponding routines (a head MRI picture processing), and finally, saves the result data to the central external PACS system. This is an example of the *Process service* which implements the actual business process workflow and is used by the advanced software application running on the working station in the radiology laboratory used by the radiologist who conducts the MRI diagnostic procedure.

A metamodel excerpt used for describing this example is demonstrated in Figure 8.13.

Data

Since providing syntactic and semantic interoperability is essential for any service-oriented architecture, the metamodel includes *Data* elements. These metamodel elements are used to present any information being utilized or exchanged by the services of the *"Services" layer*, e.g. patient data, MRI data, measured blood pressure. In most cases they represent business entities of the system and are directly addressed by the corresponding *Entity services*.

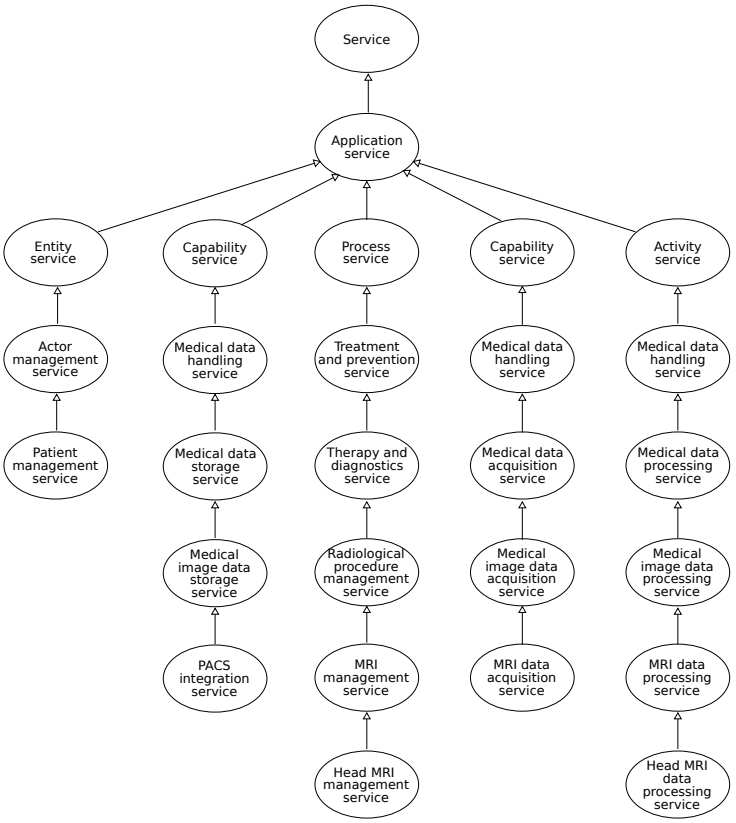

Figure 8.13: Metamodel Excerpt: MRI Diagnostic Procedure Services Composition

The assets hosted on the *"Use Cases" layer* which include information, material, or financial resources contributing to the use case delivery are refined to the appropriate data of the *"Services" layer*. For this reason, the metamodel elements representing the data reflect the corresponding hierarchy of the assets on the *"Use Cases" layer*, e.g. *Data → Asset data → Procedure data → Radiology procedure data → MRI data → Head MRI data*. The data addressed by the services on the *"Services" layer* can also be related to the associated actors on the *"Use Cases" layer*. Similar to above mentioned example, the corresponding hierarchy of the actors the *"Use Cases" layer* is to be found in the data elements of the *"Services" layer*, e.g. *Data → Actor's data → Patient's data*.

Service Consumer

Speaking of "medical services" on the abstract "business" level, the role of the medical service consumer is usually played by the patient who receives a medical treatment from the health professional (Chapter 3.1). However, regarding services from the technical point of view, the role of the service consumer is more often played by the health professional, who uses them in order to enhance his workflow and support the operation methods. Depending on the subdomain (e.g., assisted living, education, rehabilitation), the patient can act as a service consumer, also.

According to it, the metamodel elements representing the service consumer are organized in a hierarchy reflecting the actors of the *"Use Cases" layer*. Thus, for example, the radiologist who uses the *PACS integration service* in order to transfer the MRI data acquired during a standard MRI diagnostic procedure for storing it in an external PACS system, is presented in the model by the following metamodel elements *Service consumer*

→ *Health worker* → *Health professional* → *Specialist medical practitioner* → *Doctor in the medical group of specialties* → *Radiologist*. The Alzheimer's patient, who is supported at home by an ambient assisted living system and uses a special monitoring service is presented in the model by means of the following metamodel elements: *Service consumer* → *Patient* → *Patient with disease of the nervous system* → *Patient with other degenerative disease of the nervous system* → *Patient with Alzheimer's disease*.

Service Customer

A service customer is a person, organization or entity who byes a service which has been made available by the service provider to service consumers. For example, a hospital which byes for his employees external services (e.g., *Health record service*) services to be used is presented by the following hierarchy of metamodel elements: *Service customer* → *Healthcare organization* → *Hospital*.

Depending on the use case and on the subdomain, the same party can play the service customer and consumer roles both in the same scenario. Assume, an independent medical practitioner, who buys an electronic health record service from an external provider and uses it itself in his medical practice. Another example is a patient who uses a supplementary medication reminder service by choice and pays for the usage to the service provider. Taking into consideration that the medical service provision is in most countries tightly coupled with the social health insurance system, the role of the service customer can be played by the insurance company, also. In any case, the corresponding metamodel elements should reflect the hierarchical organization of the *"Use Cases" layer* actors.

Service Provider

A service provider is a person, organization, or entity responsible for making a service available to service consumers. Regardless of whether the service provision is done against payment or not, the service provider is legally bounded by the SLA agreed upon with the service customer.

To give an example, a cloud-based medical software provider acting as a service provider offers various medical IT services (practice management, appointment scheduling, patient demographics capturing, patient registration, medical fee management, etc.) to independent medical practitioners, clinics, and group practices. The corresponding hierarchy of the metamodel elements reflects the actors hierarchy of the *"Use Cases" layer*: *Service provider* → *Medical technology vendor* → *Medical software provider*.

It is worth mentioning that a lot of public health authorities provide certain IT services resided within the medical area (clinical data repositories for education and research, online nomenclatures and catalogs of diseases and medical procedures, database-supported information systems for drugs and medical devices, etc.). E.g., a notifiable disease reporting service can be provided by the official national organization presented in the metamodel by the following elements: *Service provider* → *Public health authority* → *Disease control and prevention institution*.

Requirements

The requirements defined on the *"Services" layer* apply mostly to the services, applications, and their bindings. Analog to the approach presented by [Coh07], we distinguish between the types of requirements in relation to the *infrastructure* and *application*. The first ones concentrate on resourcing and provision specific claims, the second ones - on performance and operation specific issues. Thus, the following metamodel elements can be identified:

- *Infrastructure requirement*
- *Application requirement*

Inheriting directly from the *Service requirement* metamodel element, they concentrate on their subtypes of requirements. *Infrastructure requirement* metamodel element refers to bus, communication, resourcing, and other facility requirements. They are presented in the metamodel by the corresponding subtypes:

- *Bus requirement*
- *Communication requirement*
- *Resourcing requirement*
- *Facility requirement*

Generally speaking, these categories refer to all the issues of the infrastructural nature. Thus, bus requirements cover the data transfer, communication requirements - communication, resourcing requirements - hardware and software set up, facility requirements - installation, contrivance, and other things facilitating the service provisioning. For example, a requirement concerning the geographical location of the resources involved in the service hosting can be expressed by the following excerpt from the metamodel hierarchy (Figure 8.14):

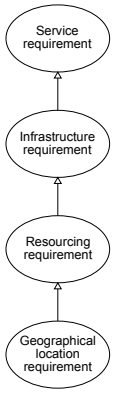

Figure 8.14: Metamodel Excerpt: Geographical Location Requirement

Application requirements, in their turn, divide into the following subtypes presented in the metamodel hierarchy:

- *Performance requirement*
- *Availability requirement*
- *Billing requirement*
- *Data treatment requirement*
- *Service level requirement*

These subtypes of metamodel elements inherit from the *Service requirement* and mostly refer the issues of the application specific nature. To be specific, requirements with respect to the performance issues are covered by the *Performance requirement* metamodel element.

Time constraints in terms of service availability are covered by the *Availability requirement* metamodel element. Monetary issues are addressed by the *Billing requirement* metamodel element subtype. *Data treatment requirement* metamodel element generalizes requirements with respect to data handling (e.g., data security issues, storage, archiving, format). Service level aspects in terms of contractual issues are covered by the *Service level requirement* metamodel element.

Thus, Figure 8.15 gives an example of the metamodel hierarchy excerpt used for description of the maximal service response time requirement.

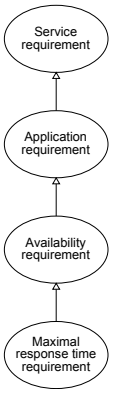

Figure 8.15: Metamodel Excerpt: Maximal Response Time Requirement

8.2.3 "Components" Layer

The "Components" layer of the system model is resided by the *devices*, *service platforms*, *data resources*, *user credentials*, *software components*, *service implementations*, and *service interfaces* system elements. The runtime policies which refer to them are presented on the "Components" layer by the *ECA rules* and *configuration* elements. Thus, the bottom layer presents the actual managed system with the corresponding management infrastructure.

Device

The delivery of medical services involves using a variety of common IT and dedicated medical devices (Chapter 3.3). The first ones include all the computers, peripherals, and telecommunication appliances, the second ones - the instrumentals and devices that are exclusively domain specific. The metamodel, therefore, contains the two corresponding metamodel elements which inherit from the *Device* metamodel element:

- *Common IT device*
- *Medical device*

The metamodel elements are specialized in their turn by the more specific subtypes. Thus, following the Medical Device Directives of the European Union Legal Framework ([Eur93], [Eur98], [Eur90]), the *Medical device* metamodel element type can be divided into subtypes corresponding to the specified medical devices types:

- *Class Is medical device*
- *Class Im medical device*
- *Class IIa medical device*

- *Class IIb medical device*
- *Class III medical device*

Further, hierarchical classification of the subtypes can be done subject to device invasiveness, activity, and intended purpose. For example, *Class IIa Medical device* metamodel element generalizes the corresponding class of medical devices and can be specialized according to measure of invasiveness on the patient's body by the following subtypes:

- *Non-invasive medical device*
- *Invasive medical device*

In their turn, these metamodel elements generalize the subordinate medical devices and provide a more specific level of description. Thus, with regard to the fact, if the non-invasive medical device is active or not, we identify further metamodel elements for device description:

- *Non-active medical device*
- *Active medical device*

The intended purpose can be used as a classification factor on the next lower level of hierarchy. Thus, *Active medical device* is specialized by the following subtypes of metamodel elements:

- *Diagnostic medical device*
- *Therapeutic medical device*
- *Auxiliary medical device*

To give an example of metamodel elements involved in modeling an active medical devices used in diagnostics - magnetic resonance imaging equipment, the following excerpt from the metamodel is demonstrated in Figure 8.16.

Sensor

As introduced in Section 8.1.1.3 sensors detect and measure information on an attribute and transform it into an analytically useful signal. Classification of the sensors can be done according to the basic type of sensing principle, recognition process, sensing element (transducer), etc.

Sensors are widely used in a lot of application fields. Although the application field can be multidisciplinary and interdisciplinary, we put forward the application domain and classify the sensors accordingly. Thus, e.g. biomedical sensors are distinguished. The biomedical sensors gain the information on human's body and pathology. The metamodel elements *Biomedical sensor* is a subtype of the *Sensor* metamodel element and can be specialized according to the type of input signal by the following subtypes:

- *Physical sensor*
- *Chemical sensor*
- *Biosensor*

Figure 8.16: Metamodel Excerpt: MRI Equipment

In their turn, these metamodel elements generalize the subordinate biomedical sensors and provide a more specific level of description. Thus, with regard to the measurand the *physical sensor* we refer to the primary physical quantity analyzed and identify further metamodel elements for device description:

- *Electrical signal sensor*
- *Blood pressure sensor*
- *Body temperature sensor*
- *Blood flux sensor*

To give an example of the metamodel elements used in order to model a sensor measuring the electrical signal produced by heart, an ECG electrode, the following excerpt form the metamodel is shown in Figure 8.17.

Data Source

The data source encompasses all its representation of each and every single data sources available during the provision of a use case. The data source can be a physical phenomena which quantified and qualified properties are detected and measured by sensors. The data source can be also provided by a data medium which can be used by devices, software component and users directly. The metamodel, therefore, contains the two corresponding metamodel elements which inherit from the *Data source* metamodel element:

- *Physical phenomenon*
- *Data medium (Carrier)*

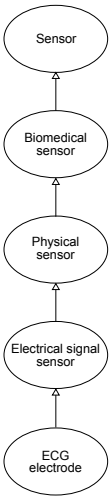

Figure 8.17: Metamodel Excerpt: ECG Sensor

The physical phenomena are classified according to the type of measured properties. E.g., in the above introduced context the biomedical sensors can detect physical quantities, chemical substances and biological materials. Thus, we identify three metamodel elements which specialize the *Physical phenomenon* metamodel element type:

- *Physical quantity*
- *Chemical substance*
- *Biological material*

The physical quantities in their turn can be structured according to the underlying energy domain. Thus the following metamodel elements specialize the *Physical phenomenon* metamodel element:

- *Electrical quantity*
- *Mechanical quantity*
- *Thermal quantity*
- *Hydraulic quantity*
- *Geometrical quantity*
- *Magnetic quantity*
- *Optical quantity*
- *Radiation quantity*

Figure 8.18 demonstrates an excerpt from the metamodel showing a data source for temperature measurements.

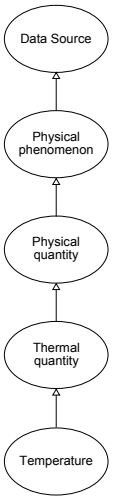

Figure 8.18: Metamodel Excerpt: Temperature

User Credential

The users of the "Services" layer are represented on the technical "Components" layer by means of their user credentials used in order to perform the process of user authentication. Based on the modality, it is common to distinguish five types of authentication factors : knowledge, ownership, inheritance, user location and current time [Tod07], [DRN17]. Thus, the metamodel contains the following subtypes which specialize the *User Credential* metamodel element according to the underlying authentication factor:

- *Ownership-based credential*
- *Knowledge-based credential*
- *Inheritance-based credential*
- *Location-based credential*
- *Time-based credential*

Going down the metamodel hierarchy, the ownership-based credentials can be divided into multiple subtypes based on the underlying form. The *Ownership-based credential* metamodel element is extended by the following subtypes:

- *Key*
- *Certificate*
- *ID card*
- *Security token*
- *Software token*
- *Hardware token*

Figure 8.19 shows an excerpt from the metamodel demonstrating the elements required to model a private SSH key.

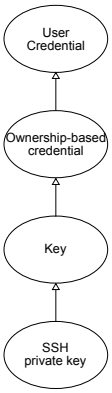

Figure 8.19: Metamodel Excerpt: Private SSH Key

Software Platform

OMG defines *platform* as a "set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns, which any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented" [Obj14b]. OMG also distinguishes between a *hardware* and *software platform*. Thus, we introduce the metamodel elements of the same name.

The software platforms include a diversity of arts and types. According to the underpinning designation, the *Software platform* element is specialized by the following subtypes:

- *Application platform*
- *Operating system*
- *Virtual machine*
- *Database*
- *Runtime environment*
- *Middleware*

In [KB15] the authors present a helpful categorization of middleware which forms the basis for the adopted hierarchical metamodel structure. Thus, the *Middleware* metamodel element is specialized by the following subtypes of metamodel elements:

- *Integration-oriented middleware*
- *Specialized middleware*

In its turn, the *Integration-oriented middleware* is divided into the following subtypes presented in the metamodel hierarchy:

- *Application-oriented middleware*
- *Communication-oriented middleware*

The *Application-oriented middleware* metamodel element stands for middleware which provides support for decomposition or other generic programming abstractions, assisting

application development in multiple aspects as well as providing a runtime environment to control the life-cycle execution of application components [Emm00]. According to the decomposition art there are two main subtypes to distinguish:

- *Component-oriented middleware*
- *Agent-oriented middleware*

Component-oriented middleware realizes the idea of interchangeable and reusable software components [CL02]. They implement a component model, which defines syntax and semantics of component definitions and their relations [CSVC11]. OSGi Service Platform is an example of a component-oriented middleware platform. Figure 8.20 shows a metamodel excerpt used to model a concrete OSGi Framework implementation, an Eclipse Equinox OSGi Service Platform.

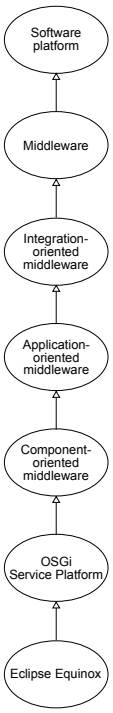

Figure 8.20: Metamodel Excerpt: Eclipse Equinox

Software Component

The components of the bottom layer stand for the concrete technical objects, artifacts and items which are present in the managed system at runtime. Among them the software components realize the applications and provide the services of the "Services" layer. They are the actual physical representation of the at runtime available executable code. The software components encapsulate a set of related functions provided by the corresponding services and applications. In terms of the architectural embedding, the software components are subjects of composition with contractually specified interfaces and explicit context dependencies [SGM02].

Generally, a software component is a unit of deployment [Szy03]. In order to enable dynamic scenarios, it also has to be a unit of versioning and replacement. So, in most cases, a software component includes a collection of modules and resources. The first ones contain code, e.g. a set of classes, the second ones contain immutable data, e.g. serialized

objects. The component can be accompanied with a metadata describing the code, data and deployment descriptors.

According to he underpinning technology, we specialize the *Software component* metamodel element by the following subtypes:

- *Software package*
- *Software module*
- *Web service*
- *Web resource*
- *Object & class library*
- *Standalone software system*

Figure 8.21 demonstrates the metamodel elements used to model a software component used within the OSGi Services Platform, an OSGi Bundle.

Figure 8.21: Metamodel Excerpt: OSGi Bundle

Policy Elements

In order to fulfill the specified abstract requirements, they are to be translated into concrete technical representations formulated for the model elements of the bottom layer: devices, sensors, software components and platforms, data sources, etc. Thus, we are looking for the initial configurations of the components and rules specifying the reconfigurations to be undertaken subject to the status changes of the components. Moreover, the evaluation rules of the specified abstract expressions are to be refined into the evaluation rules on the most technical level: formulated on the status variables of the components.

Therefore, the metamodel contains an element *Policy* which is extended by the subtypes *Configuration*, *Policy rule*, and *Policy expression*. Following the approach used for the specification of the service-level requirements, we distinguish between the infrastructure- and application-specific policy elements. The metamodel is organized similarly to the solution presented in Section 8.2.2. The infrastructure-specific policy elements address resourcing and provision requirements whereas the application-specific policy elements address performance and operation-specific requirements.

Suppose, the battery level of a device must be monitored in order to fulfill the abstract requirement of the system dependability. Figure 8.22 shows an excerpt of the metamodel used for modeling the corresponding policy rule.

Another policy type is a policy expression which is requested from the management system by the application. Suppose, that performing certain functions of an application is advisable only under certain ambient condition. In this case, before entering the

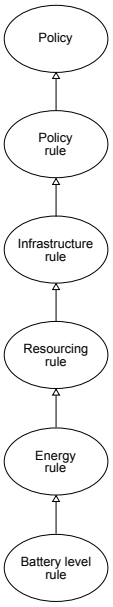

Figure 8.22: Metamodel Excerpt: Battery Level Rule

corresponding blocks in the application code, the application requests the management system to evaluate the current ambient environment. For example, if the current ambient environment is favorable, the application enters the critical block in the application code. Figure 8.23 demonstrates an excerpt of the metamodel used for modeling the corresponding policy expression.

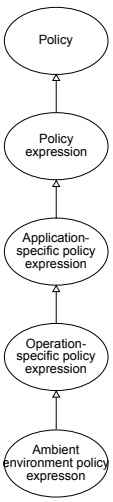

Figure 8.23: Metamodel Excerpt: Ambient Environment Policy Expression

The presented metamodel follows the generalization principle and uses the inheritance relationships in order to form the elements hierarchy. Up to the domain of interest, the metamodel can be extended appropriately. A system model for the concrete medical case is constructed as an instance of the metamodel. The following chapter is devoted to the policy refinement process which applies policy derivation patterns in order to produce the runtime configurations and policies from the system model.

Chapter 9

Policy Derivation Patterns

The policies and configurations used by the management system at runtime are derived within the policy refinement process. In order to support the refinement process, we propose a set of *policy derivation patterns*. Policy derivation patterns are model patterns defined on certain types of model elements. In fact, they imply a subgraph with nodes and edges of special types which builds an excerpt from a system model. Policy derivation patterns can be parametrized by the system developer, who has an opportunity to adjust the refinement process to his needs as for the current use case and specific requirements and conditions.

According to the purpose and architectural structure we distinguish three basic types of policy derivation patterns: *evaluation*, *control*, and *refinement patterns*. The following sections describe them closely and provide some demonstrative examples.

9.1 Evaluation Patterns

The purpose of *evaluation patterns* is to support the definition of abstract status variables with their range of values within a model layer. Such a construct can combine inputs of multiple status variables and give an opportunity to evaluate them with appropriate techniques. In doing so, an evaluation pattern comprises a function which relates a set of inputs with a set of permitted outputs. Figure 9.1 outlines schematically the common structure of evaluation patterns.

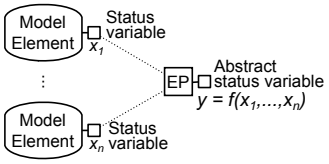

Figure 9.1: Evaluation Pattern Structure

An evaluation pattern model element is connected to those status variables which contribute to the value of the dedicated abstract variable. The status variables providing the input for the pattern can originate from several model elements. The input can come from an abstract variable, also. Thus, composition of multiple evaluation patterns is allowed. In order to specify the target system state, the modeler is able to limit the allowed range of values of the output abstract variable.

Based on the underlying function we distinguish the following types of evaluation patterns: *aggregation*, *attribution*, and *fuzzy relation patterns*.

9.1.1 Aggregation Pattern

The notion of an aggregation pattern is to aggregate multiple status variables into an abstract status variable by means of an arithmetic expression. The evaluation of an

arithmetic expression is dependent on the definition of the mathematical operators and the used system of values. The value may be undefined depending on the underlying function.

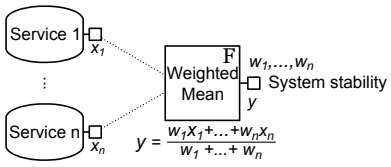

Figure 9.2: Aggregation Pattern for System Stability

A simplified example in Figure 9.2 demonstrates the usage of aggregation patterns. Suppose, the abstract system stability is calculated from the stability values of the single services composing the system: *Service 1*, ..., *Service n*. The model elements are provided with appropriate status variables: x_1, ..., x_n. For the sake of simplicity the variable values are measured in percent. It is also possible to give some weights w_1, .., w_n to the services in order to express their ranking. We calculate the system stability value as a weighted mean:

$$y = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$$

and model it by means of the *weighted mean* aggregation pattern, which connects the corresponding status variables with the derived abstract variable *system stability* and can be parametrized with appropriate weights.

9.1.2 Attribution Pattern

The idea of attribution patterns is to specify functional attribution of status variables to multiple value ranges. It allows to assign the single values to (ordered) groups and in doing so to define an abstract measurement scale for the values of the variable. The underlying function which maps a set of inputs to exactly one distinct output value can be expressed as ordered pairs, set membership, graph, or relation.

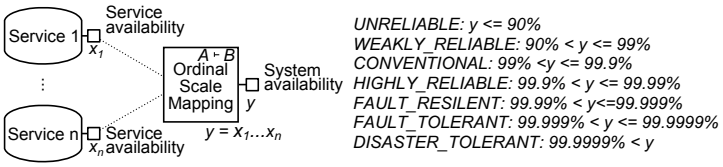

Figure 9.3: Attribution Pattern for Availability

Figure 9.3 outlines an example of attribution pattern usage. Suppose a user makes the most high demands on the system availability. This can be expressed by specifying a corresponding requirement model element with value "DISASTER_TOLERANT". As discussed before, this requirement is formulated quite abstractly. The measurement and interpretation of this term is rather use case specific and can vary considerably. In other words, one needs a concrete definition of the term "DISASTER_TOLERANT" concerning the availability for this particular use case. Let us assume, the system availability y is measured in percent and means the percentage of time when system is operational. It can be calculated from the availability of independent services composing the system x_1, ..., x_n. Thus, we are going to use an aggregation pattern with an underlying product function:

$$y = \prod_{i=1}^{n} x_i$$

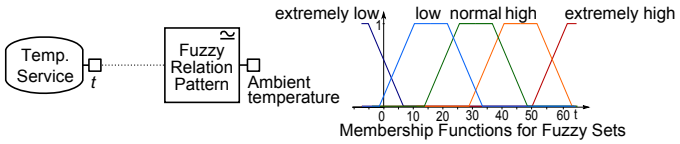

Figure 9.4: Fuzzy Relation Pattern for Temperature

In order to translate the numeric availability value in abstract notation expressing its level, we enhance the aggregation pattern with a functional attribution function. It maps the numeric value given in percent to an ordinal scale of measure. The *ordinal scale mapping* attribution pattern connects the corresponding status variables and can be parametrized as required. E.g., the system availability value of greater than 99,9999% is mapped to the "DISASTER_TOLERANT" level. It is to point out, that an evaluation pattern expresses rather "what" than "how" strategy, it does not specify in this case how to achieve the targeted level of availability but what does it mean technically.

9.1.3 Fuzzy Relation Pattern

Based on the fuzzy logic principles proposed by Zadeh [Zad65], we introduce a fuzzy relation pattern which allows to express a degree of vagueness while modeling abstract status variables. Let us assume, the inputs of the fuzzy relation pattern are universal sets $X_1, \ldots, X_n$. Fuzzy relation R on $X_1 \times X_2 \times \ldots \times X_n$ is a fuzzy subset of the Cartesian space $X_1 \times X_2 \times \ldots \times X_n$ mapping each n-tuple $(x_1, x_2, \ldots, x_n)$ to the interval [0,1], expressing the strength of the relation. In case the sets are discrete, fuzzy relation may be given as matrices. If the sets are continuous, relations are given mostly analytically. Further, the members of the input sets can be members of fuzzy sets. In this case the relation between the members is often given as a function of their grades of membership in fuzzy sets [SB05].

Consider the following example, high and so much the worse extremely high ambient temperature can cause system's malfunction. In order to define these imprecise concepts, we can use fuzzy sets. E.g., the following membership functions are used to graphically represent the fuzzy sets: *extremely low*, *low*, *normal*, *high*, and *extremely high*. To model this situation we use a fuzzy relation pattern with one input illustrated in Figure 9.4. The pattern input is a real number representing the temperature measurement values acquired by a service. The pattern is parametrized with above mentioned fuzzy set membership functions. E.g., the measured value is $t_0 = 55°$. The evaluation of the pattern results in values:

$$\mu_{extremely\_low}(t_0) = 0,00$$
$$\mu_{low}(t_0) = 0,00$$
$$\mu_{normal}(t_0) = 0,00$$
$$\mu_{high}(t_0) = 0,45$$
$$\mu_{extremely\_high}(t_0) = 0,75$$

and means, that 55° has a grade of relationship to *high* of 0,45 and a grade of relationship to *extremely high* of 0,75, a grade of relationship to any other defined set is 0,00. Thus, an abstract variable *ambient temperature* is specified by means of discrete categories and allows modeling a desired grade of fuzziness depending on current condition.

To sum up, evaluation patterns support the definition of application domain-specific terms. That can considerably facilitate the formulation of requirements and constraints on the abstract level benefiting the acceptance and understanding of the presented policy-based management approach.

9.2 Control Patterns

The notion of *control patterns* is to specify the target management control loop of the system. They define the dynamic behavior of control elements by mapping abstract declarative objectives on a higher layer to imperative enforcement mechanisms on the next lower layer. Thus, in contrast to evaluation patterns, control pattern express rather "how" strategy for implementing the management solution. The structure of control patterns is covered in Figure 9.5.

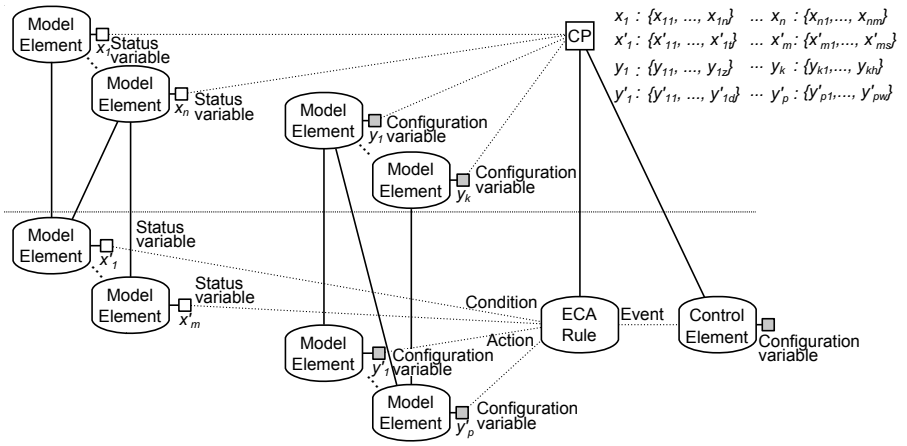

Figure 9.5: Control Pattern Structure

A control pattern takes values of status variables as an input. Based on the desired management strategy, the input values are monitored and some managements actions are undertaken, if necessary. The management actions have a form of setting certain configuration variables. Thus, the runtime management utilizes a set of ECA rules derived from the specified control patterns. The condition and action parts of the rules are defined on the status and configuration variables, the triggering event can come from an additional control element.

In the following some common models for control patterns are introduced: *watchdog timer*, *heartbeat*, *fuzzy logic control*, *on-off controller*, and *multiplexer*.

9.2.1 Watchdog Timer Pattern

The watchdog timer model is inspired by the common watchdog timer feature used in embedded systems [MB01], [Lam12]. It implies an external or integrated microprocess supervisory circuit which monitors the software abnormalities and takes appropriate corrective actions (e.g., a reset) if an infinite execution loop occurs.

The similar idea is used in the watchdog timer control pattern, which provides a time-controlled mechanism for monitoring the system state and taking corrective actions in case of detection of errors, anomalies, or undesirable system state. Suppose the following simplified watchdog model (Figure 9.6).

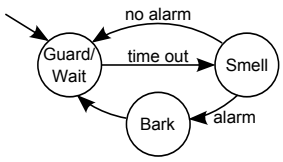

Figure 9.6: Watchdog Timer Model

An internal timer (or counter) sets the pace for the enforcement of controlling and corrective actions respectively. Thus, the watchdog is initially in the state "Guard/Wait", upon timeout a condition check takes place ("Smell"). The condition check can be performed for example by reading a status variable or making sure that the corresponding component is available. In case the condition check fails ("alarm"), the watchdog timer component undertakes a corrective action ("Bark"). E.g., the action can be in form of setting a corresponding configuration variable or triggering a service restart or substitution. In case the condition check succeeds ("no alarm"), the watchdog timer is reset.

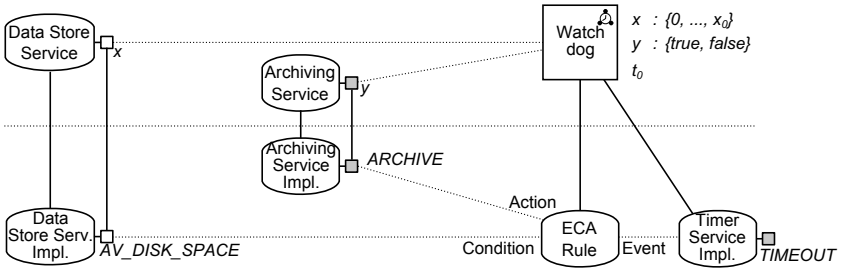

Figure 9.7: Watchdog for Available Disk Space

Figure 9.7 gives an example of a simplified watchdog timer control pattern which models a control mechanism for available data storage. Assume, a data storage service requires a certain amount of available disk space. The corresponding status variable x of the service reflects the current value. In case the value gets out of the predefined range $\{0, ..., x_0\}$, an archiving service should be activated to archive the data and free some disk space. Activation of the archiving service is done via setting the appropriate Boolean configuration variable y. Thus, a watchdog timer control element is connected to the status and configuration variables. The element is also provided with the required variable value ranges. On the next lower layer the modeled elements are refined to the more concrete implementations with dedicated variables. The watchdog timer's logic is covered by an ECA rule defined on these variables. An event which triggers the rule is generated by a timer service implementation which is preconfigured to the specified timeout t_0. The derived ECA rule has the following form:

event: *Timer's timeout event*
condition: $AV\_DISK\_SPACE \leq x_0$
action: $ARCHIVE = true$

9.2.2 Heartbeat Pattern

The heartbeat pattern extends the ability of the watchdog timer pattern. Assume, the monitored component, does not provide an opportunity to be periodically monitored with an appropriate periodical time interval. In this case, an additional heartbeat component can join up in circuit between the monitored component and the watchdog timer mechanism. The latter will function in a common way as described above by monitoring the heartbeat component and enforcing adequate corrective actions. The heartbeat component is to be implemented as an adapter of the monitored component and is responsible for periodical advertisement ("heartbeat") of the monitored component's state. As the case may be, the heartbeat component can also combine heartbeats of several components simultaneously.

Figure 9.8 demonstrates an example usage of a heartbeat control pattern. Let us suppose, we want to monitor whether the application is running. If the application is not running,

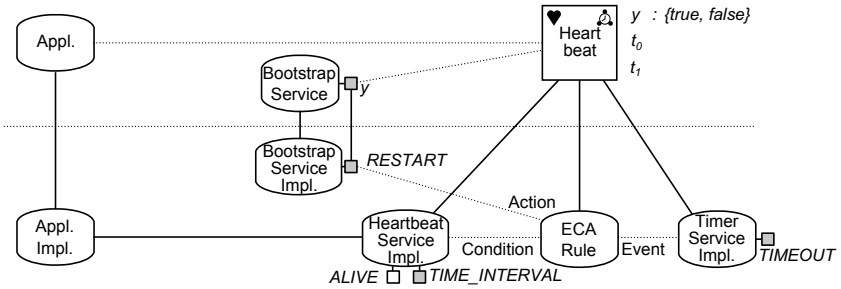

Figure 9.8: Application Heartbeat

we want to enforce the bootstrap service in order to restart it. Granted that there is no supported mechanism within the application to advertise its state by means of a concise status variable. The solution is to introduce a heartbeat control pattern which would encapsulate the corresponding control logic and publish the current application's status in form of ALIVE status variable. The time interval for status update as well as the watchdog timer's timeout are configured through the corresponding configuration variables. The heartbeat pattern is refined to an ECA rule with corresponding control components implementations which are preconfigured to the concrete management variables. The ECA rule is triggered by events fired by the timer service implementation and has the following form:

event:	*Timer's timeout event*
condition:	$ALIVE = false$
action:	$RESTART = true$

9.2.3 Fuzzy Logic Control Pattern

The fuzzy logic control pattern allows to provide a desired degree of vagueness while defining a management control loop. It is adapted from a common fuzzy logic control system used broadly in machine control nowadays. The fuzzy logic control strategies are based on heuristics methods and therefore have the advantage of being easily understood by application domain experts. That means that the management tasks successfully performed by human operators can be automatized in a simple way [Ibr03], [BH02].

The fuzzy logic controlling includes the following components. Firstly, the fuzzification block, where the crisp input of the control system is fuzzificated. That means the inputs are converted into fuzzy values for each input fuzzy set. Secondly, the knowledge base (i.e., a collection of *if-then-rules*) which comprises a set of linguistic control rules provided by the application domain experts and expressing the desired control policy and the domain knowledge. Thirdly, the decision making logic which actually embodies the fuzzy inference mechanism (e.g., *max-min inference*, *max-prod inference*) and is responsible for determining how to draw conclusions from the set of firing rules (composition of rules). At last, the output of the decision making logic performed on the knowledge base is defuzzificated (e.g., center of gravity, mean of maxima methods). That means the results of the fuzzy inference process are converted to the crisp values.

Applying this idea, we introduce a fuzzy logic control pattern. The pattern takes as input values of management variables and translates them to ECA rules and control elements. The translation is done according to the preconfigured parameters: linguistic variables specified for each input, inference mechanism, fuzzy knowledge base and defuzzification

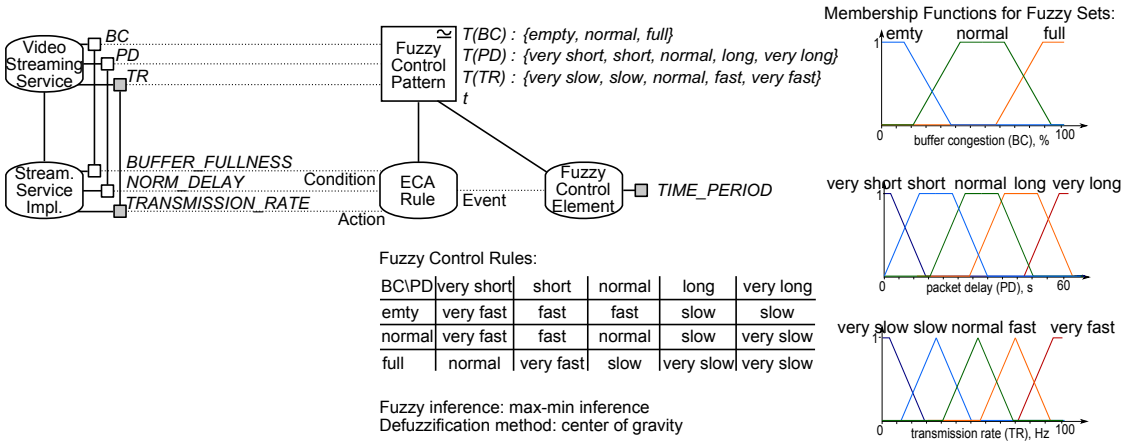

Figure 9.9: Fuzzy Control Pattern for Video Streaming Service

rule. As usual, the derived ECA rules are defined in the model on status and configuration variables of the next lower layer.

Assume the following example, the transmission rate of a video streaming service has to be controlled in the above described manner regarding such factors like congestion of the buffer and current normalized packet delay [FJRG10] (Figure 9.9). On the basis of the predefined fuzzy control rules specified by the domain expert (e.g., "if the buffer congestion (BC) is low and the current packet delay (PD) is very short then set the transmission rate (TR) to very fast"), it is possible to derive the concrete ECA rules defined on crisp values. The control element is preconfigured to fire the corresponding event periodically or on certain circumstances (e.g., status variable value exceeds predefined range). To be more specific, we have a fuzzy system with two inputs (BC, PD), one output (TR) and a set of control rules of the form:

$$R_i\text{: "}if\ BC\ is\ BC_i\ and\ PD\ is\ PD_i\ then\ TR\ is\ TR_i\text{"}$$

Several control rules can apply at a time. In order to generate a corresponding ECA rule defined on crisp values, we need to find out which of the fuzzy rules are applicable and to compose them. Firstly, a premise membership function is calculated for each fuzzy rule on the basis of the current input. If it is greater than 0, the corresponding fuzzy rule is on:

$$\mu_{premise}(BUFFER\_FULL, NORM\_DELAY) > 0$$

In the related ECA rule the firing fuzzy rules are reflected in the condition term which is formulated in accordance with the provided linguistic variables and membership functions. Since the linguistic variables characterize subranges of continuous variables, we need to determine how to translate the premise terms into crisp values without going through all possible values. A feasible solution is to define the condition terms of ECA rules on those subranges where the value of the membership function is positive. Secondly, an inference step occurs which combines the recommendations of the firing rules. E.g., we use a simple *max-min* inference method.

$$TR = max \circ min(\mu_i(BUFFER\_FULL), \mu_i(NORM\_DELAY)).$$

The conclusion of the ECA rule specifies the crisp value of the transmission rate variable. Depending on the preconfigured defuzzification method (e.g., center-of-gravity) we can calculate its value, which is a function of the composite membership function $TR_{crisp} = COG(\mu(TR))$. A derived ECA rule can have the following form:

event: *Fuzzy control's timeout*
condition: $0 \leq BUFFER\_FULL \leq 15$ and $0 \leq NORM\_DELAY \leq 20$
or
$0 \leq BUFFER\_FULL \leq 15$ and $15 \leq NORM\_DELAY \leq 35$
action: $TRANSMISSION\_RATE = COG(\mu(TR))$

9.2.4 On-off Controller Pattern

The on-off controller pattern is inspired by the common on-off (or bang-bang) controller used broadly for years in control engineering [Art80]. Figure 9.10 demonstrates the main idea of the controller. The on-off controller is a discrete feed back controller switching between two control limits: "on" and "off". The main idea of the control is to bring the actual value of the controlled variable to the desired specified value, i.e. set point. In doing so, the bang-bang controller compares the control variable with the set point and sets the controller output variable to y_{min} ("off") if the control variable x exceeds the set point x_0 and to y_{max} ("on") if it goes below the set point. In order to avoid flattering the on-off controller is often preconfigured to have a certain "deadband", a range of control variable $x_{min} \leq x \leq x_{max}$ where no action occurs.

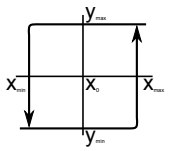

Figure 9.10: On-Off Controller Model

A simple application of the on-controller pattern is presented in the following example (Figure 9.11). Assume a thermostat service of a gas-fired heating system which turns the gas burner off or on. A temperature service provides the current measurement of temperature which is a control variable in this case. The on-off controller pattern is connected to the corresponding variables of the both services. It is parametrized with the allowed range of the control variable: $[x_{min}, x_{max}]$.

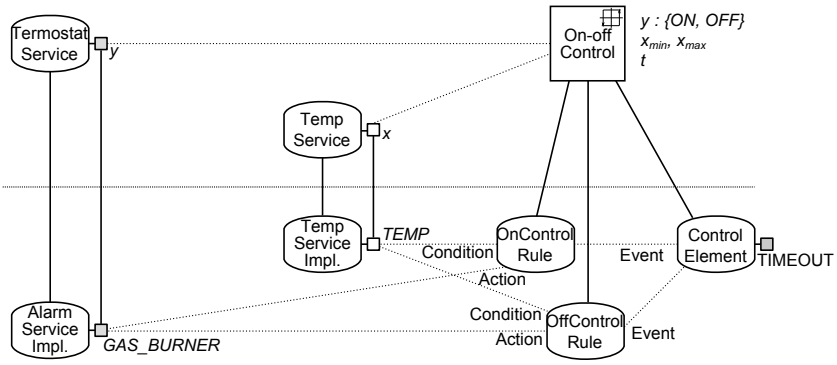

Figure 9.11: On-off Controlled Heating System

The pattern refines into two ECA rules which have the following form:

event: *Timer's timeout event*
condition: $TEMP < x_{min}$
action: $GAS\_BURNER = ON$

event: *Timer's timeout event*
condition: $TEMP > x_{max}$
action: $GAS\_BURNER = OFF$

9.2.5 Multiplexer Pattern

The multiplexer pattern is inspired by the idea of multiplexer in electronics, a combinational logic switching device which allows multiple signals to share a single common output by acting as a multiple position rotary switch. The multiple input lines of multiplexers are switched one at a time to an output [Mai07]. Similarly, the notion of the multiplexer pattern is meant to allow switching dynamically between the corresponding model elements depending on their availability or environmental condition.

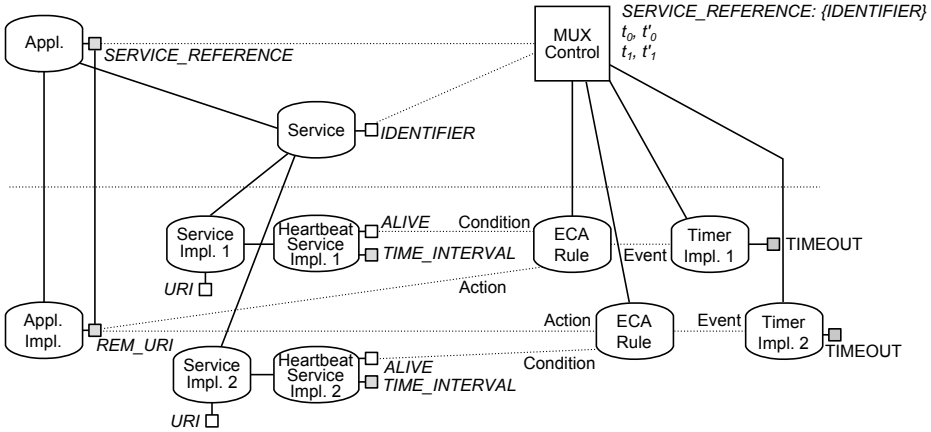

Figure 9.12: Multiplexer Pattern for Fault Tolerant Behavior

Figure 9.12 outlines an example usage of a multiplexer pattern. Suppose, the fault tolerant behavior is one of the requirements made on a certain service used by an application. Therefore, the target service is introduced redundantly. The application must be reconfigured dynamically by multiplexing the corresponding configuration variable, in case of switching between the service implementations. Let us assume, there exist two service implementations of the target service type. Each of them can be identified by the corresponding management variable URI expressing its unified resource identifier. The configuration variable REM_URI of the application implementation will be switched in its value between the values of the URI variables of the corresponding service implementations. The availability of each of the service implementations is monitored by the heartbeat component which is preconfigured to check if the monitored component is present in periodic time slots. In case the component is not available, the status variable $ALIVE$ is set to $false$. The two derived ECA rules (one for each monitored component) are triggered periodically to check, if the reconfiguration should take place. E.g., the ECA rules can have the following form:

event: *Timer's timeout event*
condition: $REM\_URI = S2.URI$
and
$HB\_S2.ALIVE = FALSE$
action: $REM\_URI = S1.URI$

event: *Timer's timeout event*
condition: $REM\_URI = S1.URI$
and
$HB\_S1.ALIVE = FALSE$
action: $REM\_URI = S2.URI$

9.3 Refinement Patterns

The purpose of *refinement patterns* is to support the policy derivation process by specifying how the values of abstract elements are to be propagated downwards to the more detailed values. Together with the refinement relations presented in Section 8.1.2 they direct the refinement from top to bottom. In contrast to refinement relations, they operate on management variables directly. Thus, they map values of management variables of adjacent layers and provide calculation rules used within the policy refinement process. The common structure of refinement patterns is outlined in Figure 9.13.

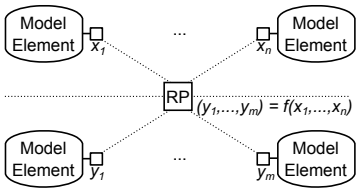

Figure 9.13: Refinement Pattern Structure

Within the system model refinement pattern model elements are situated between the adjacent layers. Management variables of the model elements on the upper layer form the input of the refinement pattern. The output of the refinement pattern flows into the management variables of the model elements on the lower layer. Thus, refinement patterns can be used to extend any inter-layer relation and consequently enrich any policy derivation pattern defined on management variables of adjacent layer elements.

Based on the underlying refinement function, we distinguish the following refinement pattern types: *repeater pattern*, *translator pattern*, and *data selector pattern*.

9.3.1 Repeater Pattern

The notion of the *repeater pattern* is to provide a mechanism to "repeat" the values from an upper layer to the next lower one. The pattern has only one input - a management variable of the upper layer. Its value is exactly echoed to the variables on the lower layer which form the output of the repeater pattern.

An example usage of repeater pattern is outlined in Figure 9.14. Assume a service having two implementations which must be modeled separately within the system model. A service has a configuration variable $SECURITY\_LEVEL$, expressing the security relevant system settings. Suppose, the use case demands the definition of requirements concerning this value, e.g., $SECURITY\_LEVEL = high$. By using the repeater pattern the system modeler can specify that within the policy refinement process the value of the

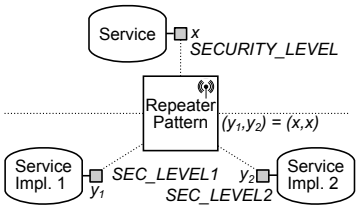

Figure 9.14: Repeater Pattern for Security Level

variable should be propagated one-to-one to the corresponding variables of the service implementations. Thus, the configuration variables of the lower layer become the same values:

$$SEC\_LEVEL1 = high$$
$$SEC\_LEVEL2 = high.$$

9.3.2 Translator Pattern

Similar to the repeater pattern, the *translator pattern* echoes the value of a management variable of the upper layer to variables of the next lower layer. However, the pattern allows to specify special rules for the policy refinement process which govern the "translation" of specified values. The rules manifest how the input value of the management variable of the upper layer is to be transferred to the values of the output variables.

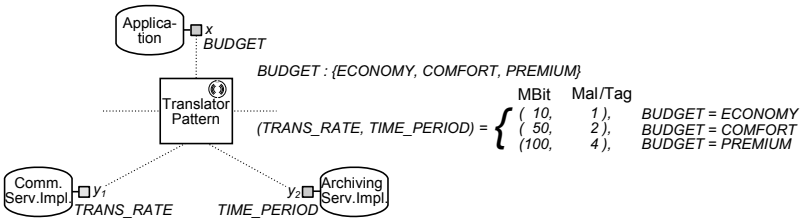

Figure 9.15: Translator Pattern for Budget

A simple example demonstrates the usage of the translator pattern (Figure 9.15). An application has a configuration variable expressing the desired budget level which is supposed to indicate how sustainable the application's performance should be. E.g., how high the transmission rate of the corresponding communication service must be set and how often the responsible archiving service should undertake backup actions. This can be achieved by setting the relevant configuration variables: $TRANS\_RATE$ and $TIME\_PERIOD$, respectively. Thus, a translator pattern with two outputs is chosen and configured in a desired way. Assuming, the modeler has defined that the target budget level should be "economy", the pattern caters for setting the configuration variable $TRANS\_RATE$ of the communication service and the configuration variable $TIME\_PERIOD$ of the archiving service to the following values:

$$TRANS\_RATE = 10$$
$$TIME\_PERIOD = 1.$$

9.3.3 Data Selector Pattern

The *data selector pattern* is the advanced version of the translator pattern, which uses for the rule definition an additional input from a status variable of the upper level. This

status variable indicates the supplementary information needed for the underlying pattern rule in order to govern the translation of specified values depending on some condition. Thus, subject to the value of the status variable, the pattern can switch between input management variables of the upper layer and "select" the corresponding one to transfer its value downwards in the predefined way.

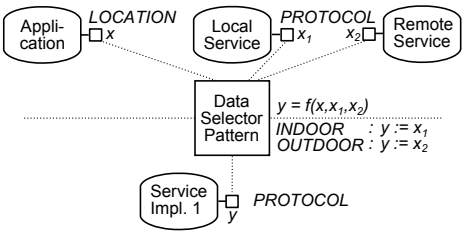

Figure 9.16: Data Selector Pattern

Figure 9.16 demonstrates the usage of the data selector pattern for configuring the used communication service. Assume, the service must be configured to use either HTTP or HTTPS application layer protocol depending on the target service. In case of the local service, HTTP protocol is used, in case of the remote service, - HTTPS. The corresponding configuration variable $PROTOCOL$ is set on the basis of the pattern output. Suppose, the application has a status variable $LOCATION$ which indicates the current location $\{outdoor, indoor\}$. Within the policy refinement process, the configuration variable $PROTOCOL$ of the communication service is set to the corresponding value of the $PROTOCOL$ variable of the service on the upper layer. The decision is done on the basis of the underlying rule. E.g., the $LOCATION$ variable value has a value "*outdoor*", the value of the $PROTOCOL$ value will be set to the value of the $PROTOCOL$ variable of the remote service:

$$PROTOCOL = https.$$

The application of the introduced policy derivation patterns will be demonstrated in the following chapters. With the superior objective of providing an automated management approach for dynamic, adaptive, and flexible medical systems, we conduct a case study in order to validate the proposed method. We apply the identified policy derivation patterns within the application domain model in order to derive the concrete runtime policies supporting the management process.

Chapter 10

Case Study: MEDOLUTION

Within the Medolution[1] project, a systematic and efficient development of Big Dependable Systems (BDS) as integration of networked reliable systems is to be researched. The combination of versatile sensors, mobile services, common IT, medical devices as well as cloud services providing Big Data analysis functions supports application systems, which are functionally rich, dynamically adaptable but at the same time restricted in reliability. Thus, technical management solutions are in demand which can support such systems in the typical life-cycle stages they go through during their lifetime.

The conceptual solution of Medolution is to be validated by means of a medical demonstrator supporting after-care and rehabilitation of patients with an implanted Left Ventricular Assist Device (LVAD) after their release from hospital. Treatment of LVAD-supported patients requires constant supervision by their doctor in order to guarantee patient's in time monitoring as well as optimal parameter tuning of medical devices and applications. Moreover, permanent telemedical monitoring by the doctor gives patients a feeling of safety that improves their living standard significantly. Clinical findings collected during telemedical monitoring of many LVAD-supported patients (e.g. patient data, vital signs, device recordings, context data, etc.) can be evaluated, interpreted and analyzed. This allows employing multiple knowledge discovery techniques. Thus, performing data-mining algorithms on Big (Medical) Data facilitates disease research as well as acquisition of new perceptions and empirical findings about treatment of patients with LVAD support. Another aspect is early detection of critical situations such as device malfunction (e.g. low battery performance) or patient's medical condition (e.g. thrombotic risk, neurological complications, fluid imbalance) with the help of classification algorithms applied to data gathered and stored in the cloud. Timely intervention or even prevention of such issues is possible due to the employed Big Medical Data approach.

The BDS-supported enhanced monitoring and controlling functions of Medolution can be categorized according their time-slotted operational range into three main categories: short-, middle- and long-term (Table 10.1).

	Operation site	**Description**
Short-term	Certified medical sensors, devices, applications operating most often within the patient's environment	Monitoring and controlling with short-term operational consequences based on internal built-in functions. These functions are assumed to be perfect and extremely reliable, e.g. the LVAD controller regulating the blood flow of the artificial heart.
Middle-term	Common IT devices and systems of patient surrounding and clinical area	Monitoring and controlling with middle-term operational consequences. The functions are supported by a distributed management system which provides for the timely and accurate intervention.

to be continued...

[1] http://medolution.org/ - Medical Care Evolution - ITEA3 research project

...to be continued

	Operation site	Description
Long-term	Cloud infrastructure as well as common IT devices and systems of patient surrounding and clinical area	Monitoring and controlling with long-term operational consequences. According to the knowledge discovered within the sophisticated data analysis in the cloud, required actions are enforced in a dependable manner.

Table 10.1: Monitoring Control-Loops

The focus of this work is the middle-term management provided by a distributed management system located in the patient surrounding and clinical areas. In order to demonstrate the meaningful usage of management, the central question is to be answered: "How can management enrich the application?" Table 10.2 arranges the management tasks according to the main functional areas of management.

Management Task	Examples
Fault Management	
Monitoring	controlling of operational capacity, monitoring of resources availability and sufficiency (hardware, sensors, communication and data networks)
Self-testing	keeping track of self-tests, observing and adapting execution frequency
Fault handling	repair mechanisms, fallback solutions, recovery actions, notifications, alarms, turning on/off the emergency mode
Reporting	logging and recording of fault-relevant events and actions, forwarding relevant data to the long-term management
Configuration Management	
Initial configuration	software distribution, network topology setup, security keys distribution, boot up
Maintenance	backup execution, version control, upgrade execution
Inventory	keeping track of used hardware and software components, taking stock of sensors, user specific allocation of components
Recording	keeping records and documentation of all configurations taken place
Reconfiguration	network topology reconfiguration, substitution of failed components, selection of appropriate services due to the agreed SLAs
Accounting Management	
Resource usage	administration, monitoring and surveillance of resource usage data (data transfer, data storage, power)
Performance Management	
Monitoring	performance monitoring, analysis, evaluation, reporting functions (data transfer, reconfigurations)
Security Management	
Security services	configuration and reconfiguration of security services: access control, authentication, authorization, encryption, malware detection, logging, audit

Table 10.2: Management Tasks According to the Functional Areas

10.1 Demonstration Scenario

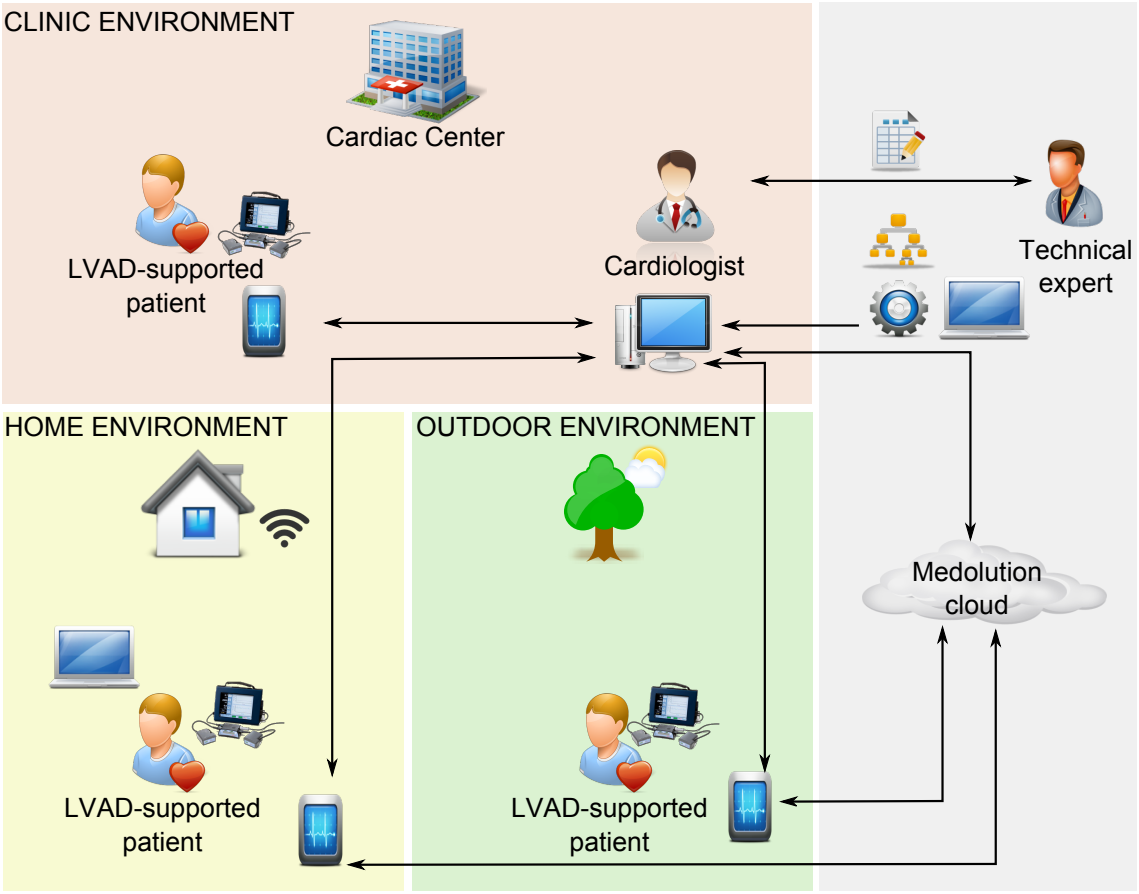

Figure 10.1: Medolution Scenario: Overview with Data Flows

The demonstration scenario is depicted in Figure 10.1. It spans three main environments (clinic, home and outdoor), the LVAD-supported patient finds himself during and directly after his in-patient treatment. The scenario concentrates on three main tasks providing a dependable assistance of LVAD-supported patients: monitoring, alarming, and adaptation.

Monitoring

Monitoring is an automated or manual regular observation, recording and supervising of processes, assets as well as their parameters and activities. It can be differentiated between the medical and the technical monitoring.

Medical Monitoring

- Vital parameters are constantly monitored by the application. The parameters are supposed to stay within a predefined range which is specified individually for each patient by his treating cardiologist.
- Monitoring of medication is essential since the LVAD-supported patients are dependent on permanent medicine taking.

Technical Monitoring

- The complex devices landscape as well as precarious patient condition demand for specific ambient parameters (temperature, humidity, air pressure, magnetic field, etc.). These are to be monitored by the management application.

- The status and status changes of the common and medical devices are to be permanently monitored, since they provide for vital functions supporting patient's life.

Thus, the monitoring process supports the requisition of the data and its transmission to the storage, analysis (cloud) and inspection location (cardiologist, patient). Figure 10.2 illustrates the main data flows within the monitoring process. The data is gathered by the sensors and devices (LVAD controller, CardioMEMS, smartwatch, smartphone, INR tester) and is transferred by the monitoring application hosted on the smartphone and home PC (via home gateway) to the corresponding destination (patient, cardiologist, cloud).

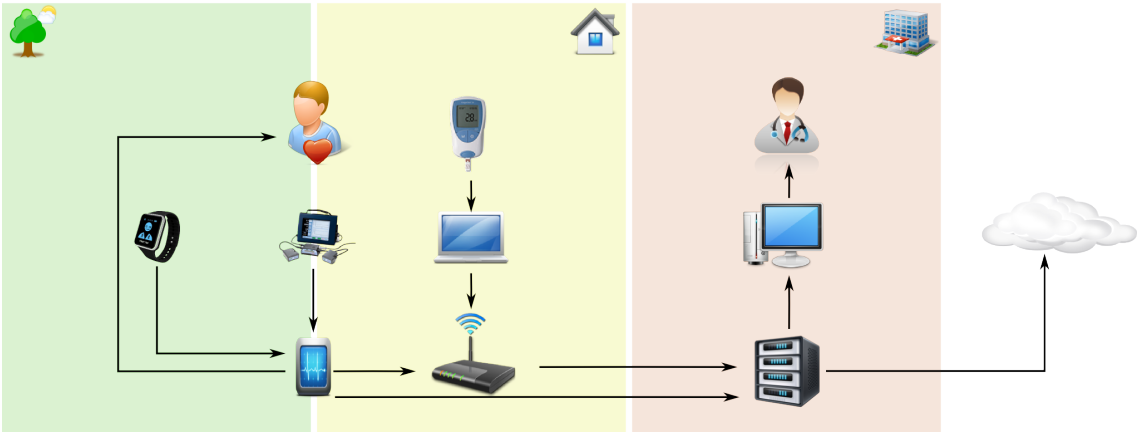

Figure 10.2: Monitoring Use Case: Data Flow

Alarming

Alarming is an automated detection of existing or approaching dangerous situations and the corresponding warning. It can be differentiated between the medical and the technical alarming.

Medical Alarming

- In case the patient's vital parameters exceed the predefined range, an adequate alarming should take place. The monitoring application takes care of prompt and proper alarms.

- If a complex medical condition is detected, the monitoring application should alarm the patient in a suitable way.

- If the monitoring application detects any deviations from the predefined medication plan, an appropriate alarm is to be produced.

Technical Alarming

- In case the relevant ambient parameters exceed the predefined ranges, the management application is supposed to produce a corresponding alarm.
- Detected abnormalities in device statuses are to be alarmed by the monitoring and the management applications.

The alarming supposes that the analysis algorithms (cloud), tele-monitoring application hosted in the clinic or the inspecting authority (cardiologist) recognize a dangerous situation and propagate a corresponding alarm or notification to the patient or the cardiologist. Figure 10.3 illustrates the main data flows of the alarming use case. The transfer of the alarms and notifications is done via the clinic server and the home gateway if applicable. The monitoring application (hosted on the smartphone and home PC) and the clinic application (hosted on the clinic PC) are responsible for the presentation and display of the alarms and notifications to the patient and the cardiologist.

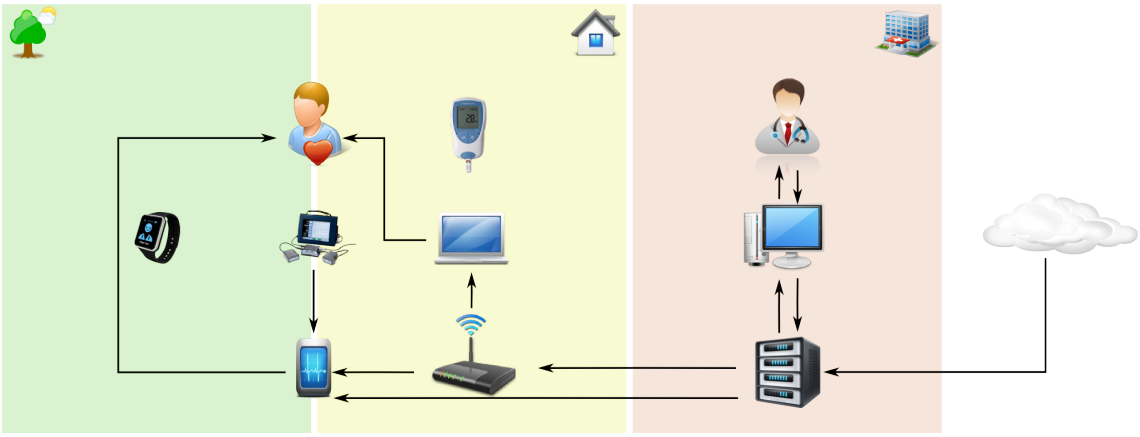

Figure 10.3: Alarming Use Case: Data Flow

Adaptation

Adaptation is an automated or manual modification or revision of processes and assets as well as their parameters and settings in order to make them applicable in situations different from originally anticipated or in order to optimize their performance in course of time. It can be differentiated between the medical and the technical adaptation.

Medical Adaptation

- The therapy of LVAD-patients requires a corresponding adjustment from time to time. That includes such values as target vital parameters ranges, medication plans, allowed ambient condition parameters. This adaptation can be initiated by the treating cardiologist directly or by the cloud-based monitoring application.

Technical Adaptation

- Adaptation of the parameters of the common and medical devices providing for life-supporting functions should be possible due to the constant changes in the patient's environment and physical condition. The changes are to be done by the monitoring and the management applications.

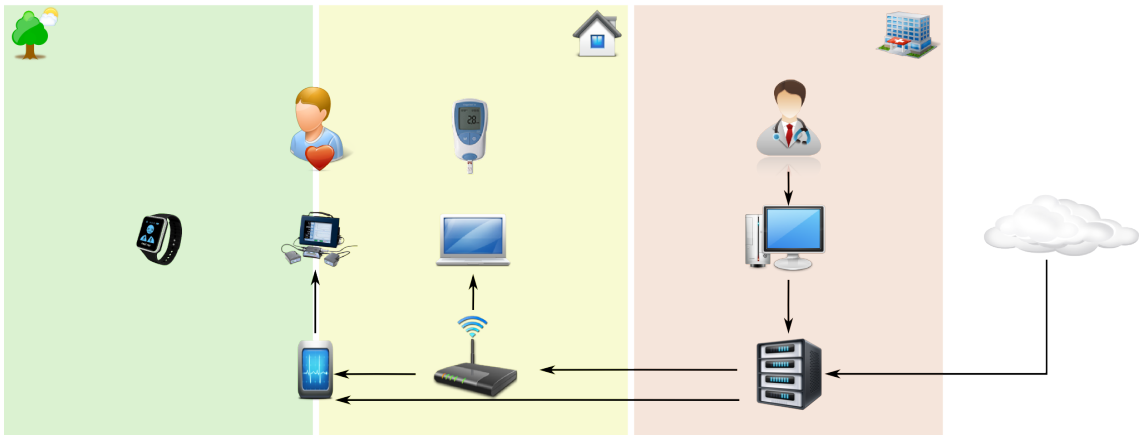

Figure 10.4: Adaptation Use Case: Data Flow

Figure 10.4 narrows the data flows within the adaptation use case down. The need of modification or revision is detected by the corresponding analysis algorithm (cloud) or by the supervising authority (cardiologist). The appropriate parameters and settings are calculated (cloud) or manually input (cardiologist). Afterward they are propagated to the responsible applications (monitoring application, INR management) and devices (smartphone, LVAD controller) via clinic server and home gateway if applicable.

Based on the descriptions above, we identify the relevant medical and technical parameters which are involved in providing the use cases of monitoring, alarming and adaptation. The following table (Table 10.3) summarizes the parameters with corresponding measuring devices which are of special interest.

Measuring Device	Parameter Art	Parameter
LVAD controller	circulatory parameter	blood flow (L/min), pulsatility index, alarms (low flow, LVAD stop, suction)
	LVAD parameter	internal clock, rotary speed (RPM), power consumption (Watts), current, voltage, pulsatility index
	battery parameter	state of charge (%), battery cycle count
INR tester	anticoagulation parameter	INR value (%)
CardioMEMS	circulatory parameter	mean arterial pressure (MAP), pulmonary artery pressure (PAP) (mmHg),
Smartwatch	circulatory parameter	oxygen saturation (%)
	geographical parameter	GPS coordinates, acceleration
Smartphone	connection parameter	GSM/WLAN/Bluetooth signal quality
	battery parameter	state of charge (%)
	ambient parameter	air pressure, humidity, light, magnetic field
Manual input	medication parameter	taken medication dose
	specific parameters	meals, personal activities

Table 10.3: Medical and Technical Parameters of Interest

10.1.1 Clinic Environment

The clinic environment offers per se a more or less stable infrastructure with a constant system landscape. The standard course of action can be assumed. The focus is on setting up the system for functioning after the hospital stay and preparing the patient for the life after the in-patient treatment supported by the system. Thus, the main three tasks are to be fulfilled: the system is to be planned, it is to be set up and configured as well as it has to be put into the first usage by the patient including necessary instructions and education and training.

Planning the system

In order to provide an individual, patient-tailored technical support and supervision of patients after their in-patient treatment, a careful and precise system planning is essential. For this purpose the system engineer and the cardiac specialist work out different patient profiles which allow a fast prototyping of system models, their requirements and proper situation specific configuration (Figure 10.1 (1)). The process is supported by the modeling tool which is used to elaborate the corresponding metamodel. When a new LVAD-patient comes up, a concrete system model is produced on the basis of the metamodel. According to the chosen patient profile, the patient specific requirements are defined by the cardiologist and are used for the automated derivation of appropriate set of configurations, settings and management rules employed by the management application at runtime (Figure 10.1 (2)).

Set up and initial configuration

After the implant surgery, the patient is equipped with a set of life-sustaining devices. Supplementary devices and systems for extended patient monitoring purposes are provided in order to support the patient in his daily activities. The initial configuration is carried out, the system is initialized, the applications and devices start working (Figure 10.1 (3)). Thus, the monitoring application monitors the vital signs, physiological parameters and medication and alarms in case of abnormalities or dangerous conditions. From now on within each environment (clinic, home, outdoor) the collected data is transmitted periodically to the cardiologist directly for the immediate evaluation (Figure 10.1 (3), (4), (6)) as well as to the cloud for the complex data analysis (Figure 10.1 (5), (7), (9)). The management application supports the whole system providing for reliable, secure, fail-safe behavior.

First usage, instructions, training

Before leaving the hospital, the patient is instructed, educated, and trained to use the devices and system. After the rehabilitation phase, the patient is released from hospital.

The course of action within the clinic environment is summarized in Table 10.4.

ID	Action	Requirements	Involved Devices
01	The technician and the cardiologist elaborate a set of LVAD-supported patients' profiles	The process has to be supported by a dedicated tool which allows the definition of corresponding characteristics and attributes	Clinic PC

to be continued...

...to be continued

ID	Action	Requirements	Involved Devices
02	The cardiologist registers a patient and provides the patient's data	A special form is required in order to enter the patient's data	Clinic PC
03	The cardiologist specifies a profile and conditions for the patient (e.g. outdoor activities allowed, long-distance walker, risk assessment based on home location or habits)	The process has to be supported by a tool which allows the cardiologist to choose between predefined profiles and settings	Clinic PC
04	The cardiologist configures manually the patient's LVAD controller with dedicated settings of the LVAD, CardioMEMS is configured		LVAD controller, CardioMEMS
05	Automated data transfer from LVAD controller into the patient data record takes place	The data transfer should be done in secure and reliable manner	LVAD controller, clinic PC
06	The surgery is performed during which the LVAD is implanted		LVAD controller, LVAD
07	Deployment and initial configuration of the system are carried out (the patient monitoring application is installed on the patient's smartphone, pairing with LVAD)	The system should be bootstrapped, the initial configuration is to be done	Patient's smartphone, LVAD controller, smartwatch, INR tester
08	The system is put into use and starts functioning	The system functions in a reliable, secure, fail-safe manner	Patient's smartphone, LVAD controller
09	The patient is instructed, educated, and trained to use and live with the implanted LVAD supported by the system		LVAD controller, patient's smartphone
10	The rehabilitation of the patient takes place in the clinic		LVAD controller, patient's smartphone, clinic PC
11	Technician equips the home environment according to the specified patient's profile	The infrastructure at the patient's home is ready for use, all the devices are initially configured	Home PC, home gateway
12	Patient goes home		

Table 10.4: Scenario for the Clinic Environment

10.1.2 Home Environment

The home environment provides a multifaceted but stable infrastructure with a constant landscape. For the patient it can be seen as an ordinary and safe environment, where he can rely on constant power supply, stable Internet connection, landlines, support of family members, available spare equipment, etc. Within the home environment, there is in fact no presence of a cardiac expert, but it still provides the patient an overall feeling of safety due to the constant monitoring by the system in a familiar surrounding area.

Data acquisition

The monitoring of the patient is continued within the home environment. Thereby the data acquisition is supported by the system which caters for the correct, timely, and secure capture of medical as well as contextual data. The captured data is to be transmitted for the further automated processing, analysis and storage to the cloud (Figure 10.1 (5)) and for the manual analysis and review to the cardiac specialist in the clinic (Figure 10.1 (4)).

Support of daily routines

In order to provide a maximal comfort and supervision of the patient in his home environment, he is supported in his daily routines with regard to his condition. Thus, the system prompts the patient on medications and measurements becoming due, advises on battery management issues, provides for timely and to the environment adjusted notifications and alarms. The context of the patient is regarded continuously in order to handle appropriately to the current situation and ambient condition.

Emergency

In case a medical emergency occurs, the patient is assisted in his critical situation according to his needs. This can include safe and timely notification of the patient, providing him with corresponding proceeding instructions (Figure 10.1 (4)).

The course of action within the home environment is summarized in Table 10.5.

ID	Action	Requirements	Involved Devices
13	The patient arrives at home	The system and devices at home are to be initialized automatically	Home gateway, home PC, smartphone, LVAD controller, LVAD
14	The monitoring of the patient continues within the home environment	The system functions in a reliable, secure, fail-safe manner	Home gateway, home PC, smartphone, LVAD controller, LVAD
15	At predefined times the patient is prompted to undertake the INR value measurement and to enter the measured value as well as the taken anticoagulation medicine dose	The prompt should take place at the appropriate time (e.g. the patient is not sleeping, currently performing his training, etc.)	smartphone, home gateway, ergometer, INR tester
16	At predefined times the patient is prompted to measure MAP and PAP values	The measurement is to be done at the appropriate situation (e.g. the patient is not training, eating, working physically, etc.)	smartphone, home gateway, ergometer, CardioMEMS
17	The captured data (INR value, LVAD records, MAP, PAP, etc.) is transmitted to the clinic and cloud in predefined time slots	The data transmission should be performed at the appropriate situation (e.g. Internet connection, mobile phone battery status)	smartphone, home gateway

to be continued...

...to be continued

ID	Action	Requirements	Involved Devices
18	The patient inspects his driveline exit site thoroughly daily. In case of driveline exit site abnormalities, the patient follows the protocol to use his smartphone to take exit site pictures which are sent to the cardiologist.	The resources on the smartphone are to be available for this operation.	smartphone, home PC, home gateway
19	The patient is regularly confronted with certain critical operations like replacing the running LVAD controller or batteries which require particular ambient conditions (e.g., a quiet well-lighted location, the patient is sitting or lying down)	It is required to check if the current situation is suitable for the critical operation, if necessary the patient should be informed	smartphone
19	The LVAD controller's battery life is constantly monitored by the application. Additionally to the state of charge, cleaning cycles and self-tests are performed by the LVAD and the power module. The patient is prompted to execute a calibration of his devices if required, as well as of the expiration date of all the batteries in use.	Support of the system is needed while estimating the actual operating life of the battery is affected not only by the rate and depth of cycles but also by other conditions such as temperature and humidity.	LVAD controller, smartphone, home gateway
20	The system regularly initiates the LVAD controller to conduct a cleaning cycle of the aortic valve	The system must recognize in a dependable manner that the patient does not perform any physical activity (and not e.g. just has forgotten his smartphone) and is awake and trigger the conduction of cleaning cycles periodically. In case the patient falls asleep or starts to perform physical exercises, he should be warned immediately.	LVAD controller, smartphone, smartwatch
21	When the patient falls asleep, the system should adjust the settings of the involved devices appropriately (The LVAD controller is connected to the power module, the volume of the speaker giving an acoustic tone in case of a detected alarm is turned on)	The system should detect if the patient has fallen asleep (vital parameters, time of the day, etc.) or can probably fall asleep. A corresponding alert must be done. After the patient wakes up, the original settings are to be used.	smartphone, home pc
22	The acquired valuable information about the patient's context (e.g. activity, environmental parameters) is captured, processed and transmitted to the cloud for further data mining purposes		smartphone, home gateway, ergometer

to be continued...

...to be continued

ID	Action	Requirements	Involved Devices
23	Any time the patient has an access to the data about the resource usage data (e.g. mobile data traffic) for the accounting purposes		smartphone, home gateway

Table 10.5: Scenario for the Home Environment

10.1.3 Outdoor Environment

The outdoor environment is characterized by the unreliable nature of wireless communication and no constant power supply. In addition the patient within the outdoor environment can feel insecure due to being on his own in the absence of his family members and supervising cardiologist.

Hazardous environment

The patient's equipment (power module, batteries, clips, charger) must be stored and transported within a certain range of ambient parameters. In order to warn the patient about unfavorable conditions, the system monitors it, resorting to the available sensors (barometer, magnetometer, gyroscope, hygrometer, GPS, accelerometer) and services (air temperature).

No wireless/mobile connection

The loss of wireless and/or mobile connection means for the patient that no data transfer to the cloud is possible. The main issue, however, is the fact that no emergency call or contact to the cardiologist can take place. The patient has to be aware of that, in order to make an appropriate decision: to betake himself to the next cell or to go on his own risk. The navigation to the next cell is supported by the system.

Emergency

In case of a medical emergency in the outdoor environment, several aspects are to be considered. Thus, if no wireless/mobile connection exists, there is no opportunity to make a call for help or contact the cardiologist. The patient feeling particularly insecure can react time-delayed, so that automation of corresponding actions can bring a considerable advantage and gain valuable time. An important facet of the outdoor environment is positioning of the patient, which can also play an important role in case of emergency if the patient needing assistance has to be located.

The course of action within the outdoor environment is summarized in Table 10.6.

ID	Action	Requirements	Involved Devices
24	The patient leaves his home for a walk	The system detects automatically that the patient has left his home environment. It should be checked, if the available resources (i.e. batteries) are going to be sufficient for a walk. In order to estimate the battery sufficiency, the system asks the patient about the intended absence length. The system also prompts the patient to take the backup LVAD controller, necessary power cables, clips and the spear batteries with him.	smartphone, LVAD controller, smartwatch
25	The system reconfigures itself for the outdoor environment	The sample rates of the pulse sensor and GPS sensor are lowed in order to save the battery of the smartwatch. The loudness of the speakers of the smartphone and the LVAD controller is increased in order to avoid missing alarms in the noisy environment	smartphone, LVAD controller, smartwatch
26	During his walk the patient can reach locations which are potentially hazardous (industry, airports, establishments with security checks, etc.) due to the risk of LVAD equipment malfunctioning (e.g. static discharge) increased considerably in certain environments (e.g. close to high-powered broadcasting and communication equipment, MRI, radio frequency energy sources, metal detector, body scanner, etc.)	The patient has to be tracked constantly, his position is to be checked given the list of registered potentially hazardous locations	smartphone, smartwatch
27	The ambient condition can influence the function of the patient's equipment (power module, batteries, clips, charger), thus it must be stored and transported within certain range of ambient parameters	It must be checked constantly and interfered, if the ambient parameters like temperature, humidity, air pressure, and magnetic field are within the predefined range	smartphone, smartwatch
28	The patient can reach a location with no cellular network coverage. He is informed about the impossibility to call the hospital in case of emergency and navigated to the next cellular cell	It should be detected when the network connection is lost and found again. The data transmission to the cloud, system updates, and downloads are to be paused or rescheduled. It also monitors if the cellular network coverage zone has been reached again.	smartphone

to be continued . . .

...to be continued

ID	Action	Requirements	Involved Devices
29	During the walk, a critical condition occurs, the LVAD controller raises an alarm. The interface of the LVAD controller shows alternating screens "Low Flow" and "Call Hospital Contact", the corresponding warning light flashes, a constant alarm tone persists.	The situation must be recognized by the system as dangerous. The system should be switched to a safeguarded mode. Corresponding settings are to be done.	smartphone, smartwatch, LVAD controller
30	The patient tries to resolve the problem acting according to the instructions by firstly checking, if the cables are attached, and then connecting to the working power source (i.e. replacing the batteries). He is assisted by providing him with corresponding proceeding instructions.	A timer must be used in order to check the time expired after the alarm has been raised.	smartphone, smartwatch, LVAD controller
31	The patient arrives at home	It must be recognized that the patient has arrived at home. The patient is to be reminded about loading the batteries and making a revision of wear and carry accessories (e.g. pack up the travel bag). The original settings for the home environment are to be done, postponed actions are to be resumed (data transmission to the cloud, system updates, and downloads)	smartphone, smartwatch, home gateway

Table 10.6: Scenario for the Outdoor Environment

10.2 Technical System

Based on the described scenario we identify the technical system which provides the above specified functionality. Therefore, we concentrate on the organization and wiring of components in the target system. The main logical unit blocks of the system of a higher abstraction are identified for the start. These are presented as components, which are self-contained modular parts of the system, encapsulating their contents. The manifestation of components is replaceable within their environment. A formal contract of the services that the component provides to the other components and those that it requires from the other components in the system is specified in terms of its provided and required interfaces (groups of operations).

Figure 10.5 presents the technical structure in form of a component diagram following the UML notation [Obj15]. The following sections describe the identified components, their dependencies and interfaces.

10.2.1 Components

The following Table 10.7 summarizes the identified components stating the inter-component dependencies and the corresponding device on which the component is hosted:

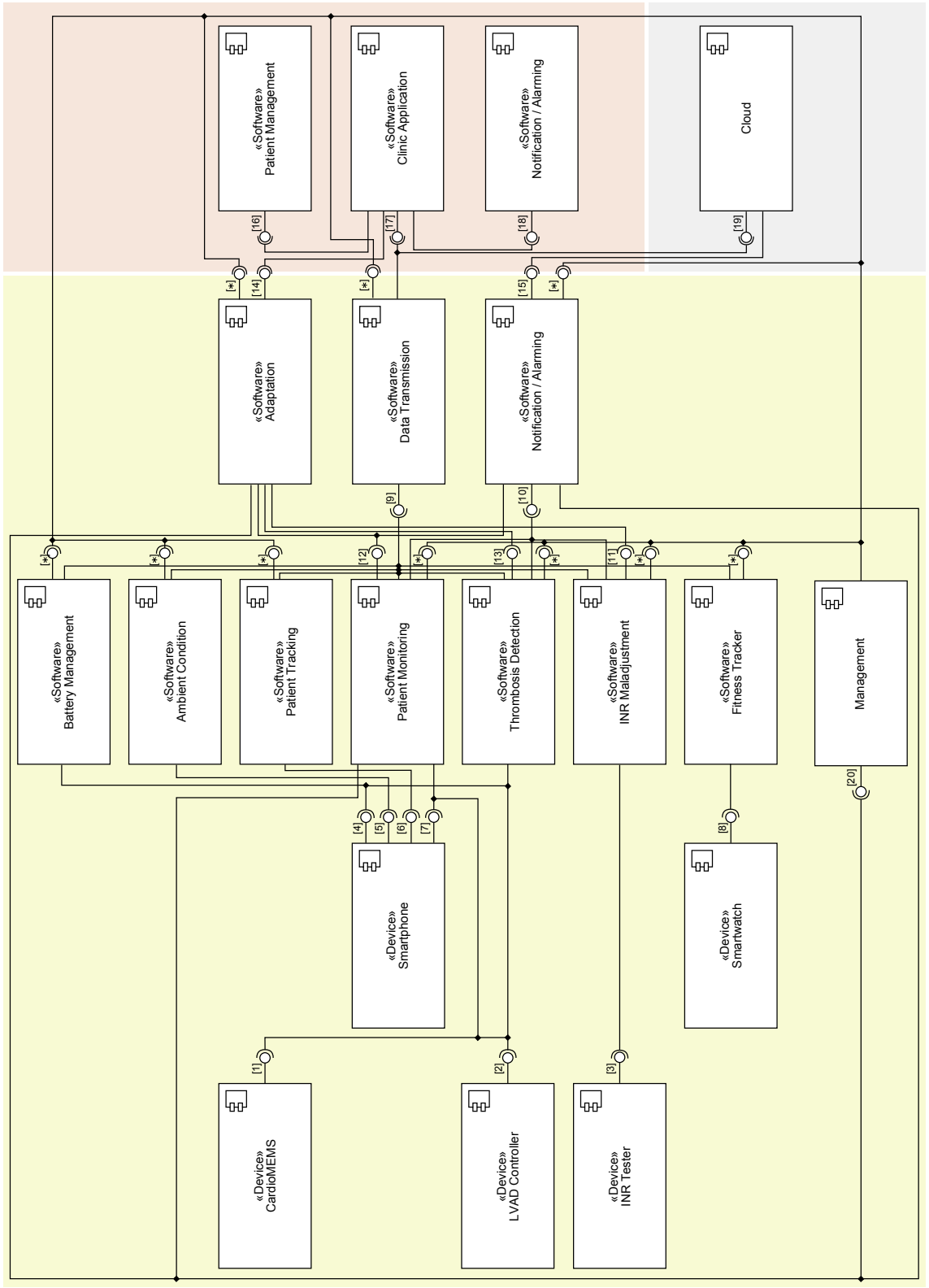

Figure 10.5: Medolution Component Diagram

Component	Description	Dependencies	Hosting Device
CardioMEMS	A component responsible for the integration of the CardioMEMS device	Patient Monitoring	Smartphone
LVAD Controller	A component responsible for the integration of the LVAD device including the corresponding controller and monitor	Patient Monitoring, Thrombosis Detection, Battery Management	LVAD controller
INR Tester	A component representing the INR tester device	INR Management	INR tester
Smartphone	A component responsible for the integration of the smartphone device with its sensors	Patient Monitoring, Battery Management, Ambient Condition, Patient Tracking	Smartphone
Smartwatch	A component responsible for the integration of the smartwatch	Fitness Tracker	Smartwatch
Patient Monitoring	A central component responsible for active gathering of monitored data (e.g. vital signs, physiological and device parameters, patient's activity) and forwarding it for the further analysis and storage	LVAD Controller, CardioMEMS, Smartphone, Data Transmission, Notification - Alarming, Adaptation, Management	Smartphone
Battery Management	A component responsible for management of batteries of involved devices (e.g. LVAD controller, smartphone, smartwatch) including such functions like estimation of remaining lifetime considering the current ambient condition and settings, reminder of a required charging, count of rate and depth of cycles	Smartphone, Smartwatch, LVAD Controller, Ambient Condition, Data Transmission, Management	Smartphone
Ambient Condition	A component responsible for monitoring of the ambient condition (e.g. temperature, humidity, air pressure, magnetic field) which is to stay within a predefined range in order not to influence the function of patient's equipment (e.g. batteries) and patient's condition (blood circulation) as well as to provide for information which can turn out to be significant during the analysis of the medical data (e.g. correlation of air pressure and blood circulation parameters)	Data Transmission, Smartphone, Management	Smartphone

to be continued . . .

...to be continued

Component	Description	Dependencies	Hosting Device
Patient Tracking	A component for tracking the patient's position (GPS, potentially hazardous locations, distance to the next cell) and activity status	Data Transmission, Smartphone, Management	Smartphone
Thrombosis Detection	A component supporting the thrombosis detection based on the measured LVAD power consumption	Data Transmission, LVAD Control, Notification - Alarming, Management	Smartphone
INR Maladjustment	A component for requisition of INR relevant values, like INR measured values, nutrition diary, medication dose taken, edema, etc.	INR Tester, Data Transmission, Notification - Alarming, Adaptation, Management	Smartphone
Fitness Tracker	A component which monitors the activity of the patient, e.g. physical active, doing sport, sleeping, at rest	Smartwatch, Data Transmission, Management	Smartphone
Adaptation	A central component responsible for executing the modifications and adjustments of settings (e.g. target vital parameters ranges, medication plans, allowed ambient condition parameters) which have been supplied by the medical personnel based on the treating doctor's decision and (or) the analysis of data in the cloud	Clinic Application, Patient Monitoring, Ambient Condition, Thrombosis Detection, INR Maladjustment, Management	Smartphone
Data Transmission	A component for transmission of the gathered data for the further analysis and storage to the clinic application and cloud	Patient Monitoring, Ambient Condition, Patient Tracking, Battery Management, Thrombosis Detection, INR Maladjustment, Fitness tracker, Management, Clinic Application, Cloud	Smartphone
Notification - Alarming	A central component responsible for notification and alarming of the patient (or the cardiologist) about existing or approaching dangerous conditions and situations	Patient Monitoring, Ambient Condition, Patient Tracking, Battery Management, Thrombosis Detection, INR Maladjustment, Fitness Tracker, Management, Clinic Application, Cloud	Smartphone

to be continued ...

...to be continued

Component	Description	Dependencies	Hosting Device
Management	The central management component	Patient Monitoring, Ambient Condition, Patient Tracking, Battery Management, Thrombosis Detection, INR Maladjustment, Fitness Tracker	Smartphone
Clinic Application	A central component providing for the monitoring of the patients on the clinic side including the main functions of telemonitoring, notification and alarming of the cardiologist as well as pseudonymization of the patient data and transmission of it to the cloud for the storage and further data processing and analysis	Data Transmission, Patient Management, Cloud, Notification - Alarming, Adaptation	Clinic PC
Patient Management	A component for the management of the patient data	Clinic Application	Clinic PC
Cloud	A component for the storage and analysis (e.g. thrombosis, derailed BP detection, INR maladjustment, etc.) of the gathered data	Data Transmission, Clinic Application	Cloud

Table 10.7: Summary of Components

10.2.2 Interfaces

A provided interface of a component represents the services and obligations that the component offers to its clients. A required interface on the contrary specifies services that the component "needs in order to perform its function and fulfill its own obligations to its clients" [Obj15]. Table10.8 summarizes the identified interfaces, stating the corresponding components and the data exchanged through the interfaces.

	Interface	Exchanged Data	Provided by	Required by
[1]	**MAP/PAP**	Measured blood pressure parameters like MAP, PAP (mmHg), measuring device id	CardioMEMS	Patient Monitoring

to be continued ...

...to be continued

	Interface	Exchanged Data	Provided by	Required by
[2]	**LVAD**	Measured or calculated circulatory parameters like blood flow (L/min), pulsatility index, registered LVAD alarms like low flow, LVAD stop, suction, measured LVAD parameter like internal clock, rotary speed (RPM), power consumption (Watts), current, voltage, pulsatility index, battery status	LVAD Controller	Patient Monitoring, Thrombosis Detection, Battery Management
[3]	**INR**	Measured anticoagulation parameter like INR value (%), measuring device id, manually input data concerning the nutrition, taken medication dose and edema, device id	INR Tester	INR Maladjustment
[4]	**Battery**	Measured state of charge (%), battery cycle count, registered maximal number of load cycles, expiration date, last calibration date for the main and the backup battery, battery id	LVAD Controller	Battery Management
[5]	**Ambient Parameter**	Measured ambient parameters like air pressure (mmHg), humidity (%), light (lx), magnetic field (μT) and the corresponding measuring sensor ids	Smartphone	Ambient Condition
[6]	**GPS Accelerometer**	Measured GPS coordinates (latitude and longitude), proper acceleration (m/s^2), measuring device id	Smartphone	Patient Tracking
[7]	**Notification Alarm**	Notifications and alarms to be displayed to the patient	Smartphone	Patient Monitoring
[8]	**Activity**	Patient's current estimated activity status (physical active, doing sport, sleeping, at rest), patient's id	Smartwatch	Fitness Tracker
[9]	**Data**	Measured data to be transmitted for the further analysis and storage to the clinic application and the cloud	Data Transmission	Patient Monitoring, Battery Management, Ambient Condition, Patient Tracking, Thrombosis Detection INR Maladjustment, Fitness Tracker

to be continued ...

...to be continued

	Interface	Exchanged Data	Provided by	Required by
[10]	**Alarm**	Dangerous condition detected by the local components	Notification - Alarming	Patient Monitoring, Thrombosis Detection, INR Maladjustment
[11]-[13]	**Parameter**	Adaptation to be applied (target vital parameters ranges, medication plans, allowed ambient condition parameters), patient id	Patient Monitoring, Thrombosis Detection, INR Maladjustment	Adaptation
[14]	**Adaptation**	Modifications and adjustments of settings supplied by the cardiologist based on the treating doctor's decision and (or) the analysis of data in the cloud (target vital parameters ranges, medication plans, allowed ambient condition parameters), patient id	Adaptation	Clinic Application
[15]	**Notification/ Alarm**	Notifications and alarms to be displayed to the patient generated in the cloud by the corresponding applications	Smartphone	Patient Monitoring
[16]	**Patient**	Patient personal data, EHR, id	Patient Management	Clinic Application
[17]	**Data**	Measured data which is transmitted for the further analysis and visualization to the clinic application	Clinic Application	Data Transmission
[18]	**Notification/ Alarm**	Dangerous condition detected by the Clinic Application to be displayed to the cardiologist	Notification - Alarm	Clinic Application
[19]	**Data**	Measured data which is transmitted for the further analysis and visualization to the cloud	Clinic Application	Data Transmission
[20]	**Management Data**	Request for a policy expression evaluation	Management	Patient Monitoring, Battery Management, Ambient Condition, Patient Tracking, Thrombosis Detection INR Maladjustment, Fitness Tracker
[∗]	**Management Data**	Management data exchanged between the management and the managed components: configuration and status variables	Management	Patient Monitoring, Adaptation, Notification - Alarming

Table 10.8: Summary of Interfaces

10.3 MEDOLUTION System Model

Based on the presented demonstration scenarios, we model the system according to the approach introduced in Section 8.1. Thus, the identified application scenarios and their main medical functions are reflected on the "Use Cases" layer, the services provided and required by the system are presented on the "Services" layer and the involved hosting devices, measuring sensors as well as software components are allocated to the "Components" layer.

For the sake of brevity, Figure 10.6 shows just a small excerpt of the system concentrating on the monitoring of a LVAD-supported patient within his home and outdoor environment.

10.3.1 "Use Cases" Layer

Two main actors can be identified in the presented application scenario: the LVAD-supported patient after his in-patient treatment and the cardiologist supervising the patient after his stay in the hospital. Thus, the model elements *LVAD Patient* and *Cardiologist* are used in the model to represent the corresponding actors.

The demonstration scenario includes three main use cases: monitoring of the LVAD-supported patients, alarming the patient and the supervising cardiologist about existing and approaching dangerous situations as well as adaptation of the patient's treatment. Hence, we distinguish three corresponding model elements standing for this use cases: *Patient Monitoring*, *Alarming* and *Adaptation*.

The functions provided within the scope of the *Patient Monitoring* use case include measurement of the relevant parameters, processing of the acquired data, and transmission of the data to the clinic. Thus, the following functions can be identified: *Data Collection*, *Data Processing*, and *Data Transmission*.

For the *Alarming* use case the main identified supported functions are maintenance of the catalog of the target system state, detection of abnormality, alarm and notification. That means the model elements *Target State Catalog*, *Detection of Abnormality*, *Alarm*, *Notification* are introduced.

The functions provided within the scope of the *Adaptation* use case include detecting the need for adaptation, calculation of the adjustment, as well as the actual control of the system. Thus, the model elements can be identified: *Detecting Need for Adaptation*, *Calculation of Adjustment* and *Control of System*.

Diverse aspects are of special interest within the demonstration scenario. For the sake of brevity we bring out only some of them, e.g. such aspects as the ambient condition, the patient's activity and the patient's environment. Corresponding model elements *Ambient Temperature*, *Ambient Environment*, *Patient's Activity*, *Transmission Connection* are introduced within the model.

The requirements formulated for the aspects, functions and use cases of the demonstration scenario are also great in numbers. E.g., the requirements formulated for the above mentioned aspects include constraints on the ambient temperature, patient's activity and the transmission connection. The model element *Ambient Temperature Constraint* represents the restrictions on the ambient temperature which can be configured with the minimal and maximal allowed temperatures to allow a normal operation of LVAD accumulators. The *AV Cleaning Cycle Resting Constraint*, *AV Cleaning Cycle Awake Constraint* model elements represent the constraints on the estimated likelihood that the LVAD patient performs no physical activity and is awake during the conduction of the aortic valve cleaning cycle. The *Data Transmission Constraint* model element is used

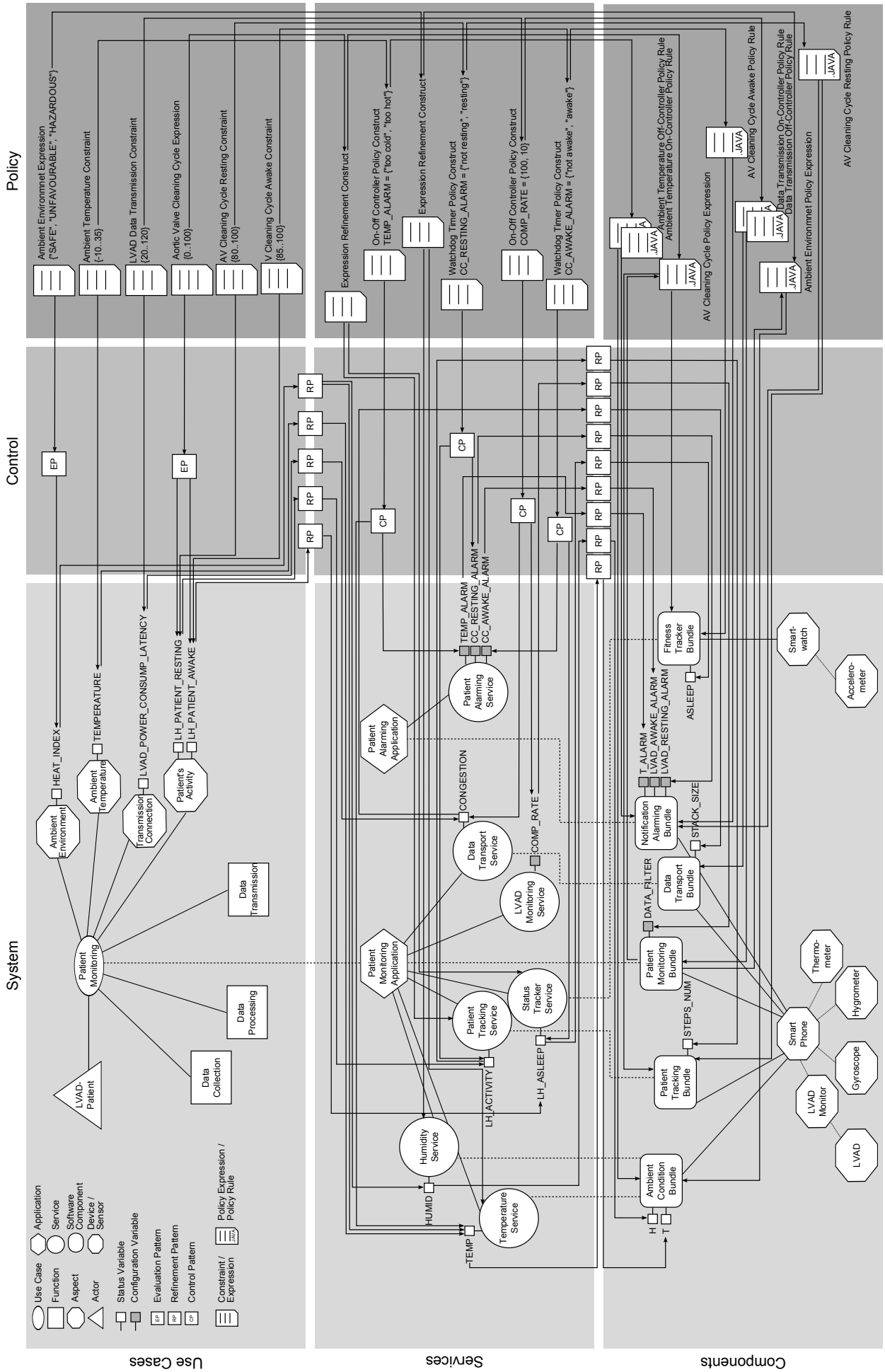

Figure 10.6: Excerpt of the Medolution System Model: "Patient Monitoring" Use Case

to define restrictions on the latency of the LVAD power consumption data during its transmission to the clinic.

In addition to the use case requirements, use case specific policy expressions can be used to formulate expressions to be evaluated during the runtime. Thus, e.g. *Ambient Environment Expression* represents a construct formulated on the ambient parameters requiring a complex computation which is beyond the competence of the managed system. Another example is the *AV Cleaning Cycle Expression* model element which is used to evaluate the aggregated likelihood that the LVAD patient performs no physical activity and is awake. The interpretation of the evaluation output (i.e. whether it is safe to conduct the cleaning cycle or not) is, however, up to the managed system.

10.3.2 "Services" Layer

The "Services" layer accommodates the model elements representing applications, services and related requirements.

The central application which provides the identified use cases is the *Patient Monitoring Application* presented in the model as one of the main model elements within the home and outdoor environments. In the clinic environment there is a *Clinic Telemonitoring Application* which is modeled by means of the corresponding model element.

The applications use multiple services required to enable access to one or more capabilities described in the sections above. The services are presented in the model by the corresponding model elements. To give an example, the model elements representing the services providing the ambient parameters are *Humidity Service*, *Temperature Service*, *Ambient Light Service*, and *Air Pressure Service*. The services providing the measured physiological parameters have the corresponding model elements introduced: *MAP/PAP Measurement Service*, *LVAD Monitoring Service*, *Patient Tracking Service*, *Status Tracker Service*. The model elements *Patient Alarming Service* and *Clinic Alarming Service*, stand for the services responsible for the patient's and cardiologist's notification and alarm. The services for providing access to devices are reproduced by the corresponding model elements: *LVAD Flow Control Service*. Another example of external services used by the applications in the clinic is the service responsible for managing the patient information; it is represented by the *Patient Management Service* model element.

The service and application specific requirements are presented in the model by means of the corresponding model elements. In most cases these are rather auxiliary constructs used during the policy refinement process. As may be the case dedicated constraints can be introduced also. They express restrictions, requirements, constraints, and conditions on the use, provision and deployment of the services and applications. For instance, the model elements *Rule Construct* and *Expression Construct* are introduced on this layer and are connected to the corresponding model elements.

10.3.3 "Components" Layer

The presented application scenario involves a set of medical and common devices used in different environments: home, clinic and outdoor. The home and in parts the outdoor environments include e.g. a LVAD with its controller and monitor, INR tester, smartphone, smartwatch as well as Home PC. The clinic environment includes just a clinic PC. Thus, the model elements *LVAD*, *LVAD Controller*, *LVAD Monitor*, *INR Tester*, *Smartphone*, *Smartwatch*, *Home PC* and *Clinic PC* are introduced within the model.

A set of embedded and autonomous sensors measuring physiological as well as ambient parameters is used in the context of the scenario, e.g. a CardioMEMS sensor, GPS

sensor, accelerometer, gyroscope, magnetic field sensor, temperature sensor, barometer, hygrometer and camera. The corresponding model elements are introduced in the model: *CardioMEMS*, *GPS sensor*, *Accelerometer*, *Gyroscope*, *Magnetic Field Sensor*, *Temperature Sensor*, *Barometer*, *Hygrometer*, and *Camera*.

Further, a set of software components used in the home and outdoor environments can be identified, the model elements which stand for them are: *Patient Monitoring Bundle*, *Notification Alarming Bundle*, *Patient Tracking Bundle*, *Fitness Tracker Bundle*, *Ambient Condition Bundle*, *Data Transport Bundle*, *LVAD Flow Control Bundle*, *Thrombosis Detection*, *INR Maladjustment*, and *Battery Management Bundle*. Within the clinic environment the software components are presented by the following model elements: *Telemonitoring Application Bundle*, *Clinic Notification Alarming Bundle* and *Patient Management Bundle*.

10.4 Policy Derivation

In order to perform the refinement from the model elements of the "Use Cases" to "Services" layer and afterward from the "Services" to "Components" layer, refinement patterns are to be specified and properly configured. They state how the variables of the model elements of the upper layer are to be transformed into the variables of the adjacent lower layer. We exemplify the policy derivation process on the basis of the introduced model. Thereby, we look closely on the model elements which represent the patterns, how they are configured and modeled in the MoBaSeC tool and how does the outcome of the policy derivation process look like. The summarizing overview of the presented model elements is depicted in Figure 10.6.

10.4.1 Derivation of Ambient Temperature Policy Rules

Let us consider an example of the ambient temperature. For the proper function of the LVAD controller device, it is important to ensure that the ambient temperature stays within the predefined range, since the function of the batteries can be considerably impaired. In case the allowed range is exceeded, a corresponding action should take place. As soon as the temperature normalizes, the undertaken measures become unnecessary. The model element *Ambient Temperature* represents the ambient temperature measured in degree Celcius (Figure 10.7). A corresponding status variable $TEMPERATURE$ is provided within the related aspect. To model this we add a constraint model element related to the $TEMPERATURE$ status variable. The allowed temperature range is mirrored in the configuration of the constraint model element, stating the maximal and minimal permitted values:

$$double\ TEMPERATURE_{min} = -10;$$
$$double\ TEMPERATURE_{max} = 35;$$

The service responsible for the measurement of the temperature which is modeled as the *Temperature Service* performs operation in degree Fahrenheit. Thus, the status variable $TEMP$ is defined within this model element. A translator pattern is needed (see Section 9.3.2) which states how the computation from Fahrenheit into Celsius is done:

$$double\ TEMPERATURE = (TEMP - 32) * 5/9;$$

So, the model element for the translator pattern added to the model and is placed between the "Use Cases" to "Services" layer.

The *Temperature Service* measures the ambient temperature and exposes the measured value by means of the status variable $TEMP$. Suppose, the value is out of the allowed range.

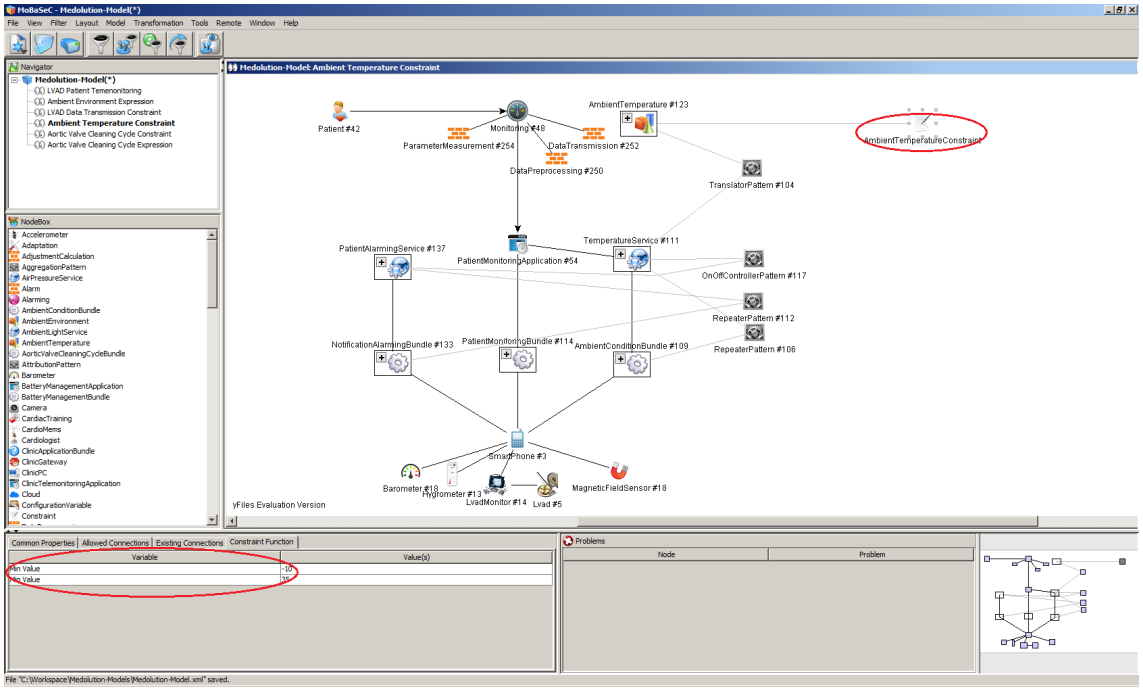

Figure 10.7: Example: Ambient Temperature Constraint[2]

The *Patient Alarming Service* can be used to provide the patient with the corresponding alarm of the exceeded temperature range. For this purpose, the $TEMP\_ALARM$ configuration variable can be used. Let us consider two cases: the registered temperature is higher than the allowed maximal value and the registered temperature is lower than the minimal value. In the first case, the $TEMP\_ALARM$ configuration variable is set to the value "too cold", in the second "too cold". To model this situation we reside to a control pattern, namely an on-off controller pattern (see Section 9.2.4) which controls the output variable, i.e. $TEMP\_ALARM$ of the *Patient Alarming Service*, subject to the control variable, i.e. $TEMP$ of the *Temperature Service*. The on-off controller pattern is configured accordingly by stating the control variable and setting the output parameters. The control pattern means that two management rules are required. The first one should fire on exceeding the maximal allowed threshold and the second on exceeding the minimal one. Going down in the hierarchy from the "Services" to "Components" layer, we need an instruction, how to refine the status variable $TEMP$ into the concrete technical status variable provided by the software component *Ambient Condition Bundle*. Suppose, the component provides a status variable T measured in degree Fahrenheit. As a refinement pattern, a repeater pattern can be used (see Section 9.3.1), which simply repeats the value of the $TEMP$ variable:

$$double\ TEMP = T;$$

Similarly, the T_ALARM configuration variable is provided by the software component *Notification Alarming Bundle*. It is to be set with a string value which is to repeat the value of the $TEMP\_ALARM$ configuration variable. Thus, a repeater pattern is used:

$$string\ TEMP\_ALARM = T\_ALARM;$$

Figure 10.7 demonstrates an extract of the model in the MoBaSeC tool exemplifying the definition of the ambient temperature constraint.

[2]To be viewed with a PDF Viewer

The model after the policy derivation process is shown in Figure 10.8. Two model elements for the policy rules are generated and added to the model.

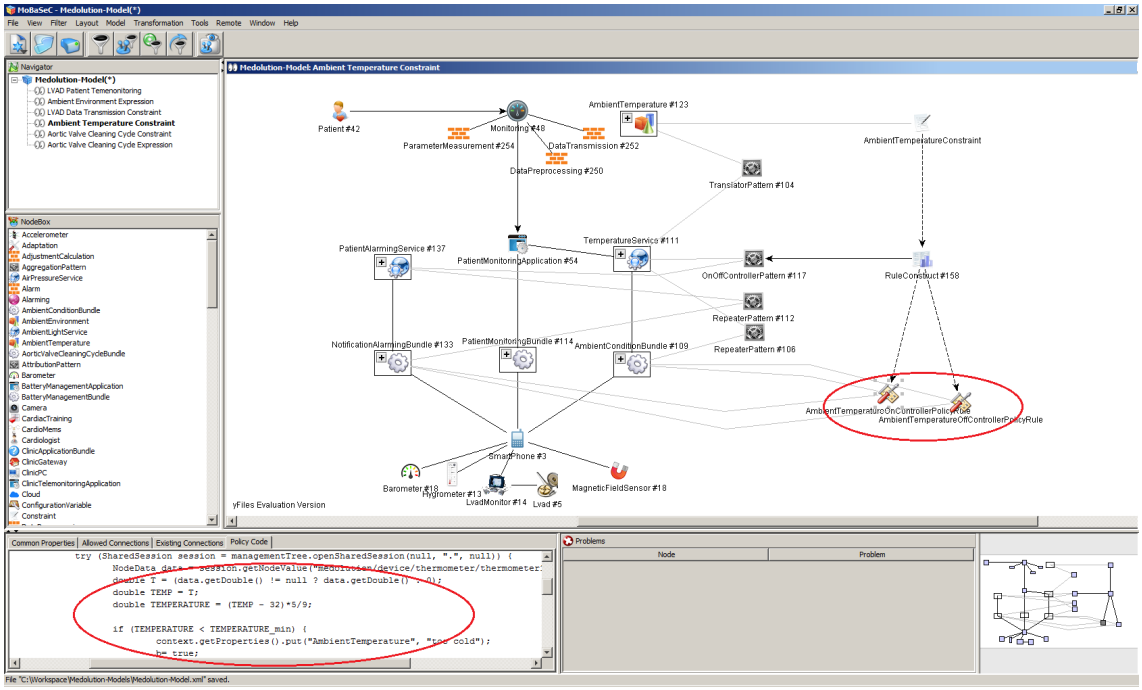

Figure 10.8: Example: Ambient Temperature Constraint - Derived Model Elements

The source code for the two policy rules to be executed during the runtime is generated by the MoBaSeC's backend functions. The policy rules represent the management rules resulting from the used control pattern.

The outcome of the policy derivation for the defined ambient temperature constraint is shown in Listings 10.1, 10.2.

```
1  package com.materna.medolution.management.showcase.policy;
   ...
3  public class AmbientTemperatureOnControllerPolicyRule extends AbstractRule {
     public boolean evaluate(ExecutionContext context) throws Exception {
5      Tree managementTree = context.getRuleContext().services().contract(Tree.
       class).execute().getService();

7      double TEMPERATURE_min = -10;
       double TEMPERATURE_max = 35;
9      boolean b = false;
       TreeEvent event = context.getEvent();
11     Path path = event.getPath();

13     switch (event.getType()) {
         case NODE_CHANGED:
15         try (SharedSession session = managementTree.openSharedSession(null,
       ".", null)) {
           NodeData data = session.getNodeValue("medolution/device/
       thermometer/thermometer1XXX/status/measurement/data/value");
17         double T = (data.getDouble() != null ? data.getDouble() : 0);
           double TEMP = T;
19         double TEMPERATURE = (TEMP - 32)*5/9;

21         if (TEMPERATURE < TEMPERATURE_min) {
             context.getProperties().put("AmbientTemperature", "too cold");
```

```
23            b= true;
            } else if (TEMPERATURE > TEMPERATURE_max) {
25            context.getProperties().put("AmbientTemperature", "too hot");
              b= true;
27          } else {
              context.getProperties().remove("AmbientTemperature");
29          }
          } catch (TreeException e) {
31          e.printStackTrace();
            break;
33        }
          break;
35      default:
          break;
37    }
      return b;
39  }

41  public void execute(ExecutionContext context) throws Exception {
      String value = "Temperature is out of the allowed range: " + context.
    getProperties().get("AmbientTemperature");
43    Tree managementTree = context.getRuleContext().services().contract(Tree.
    class).execute().getService();
      try (ExclusiveSession session = managementTree.openExclusiveSession(null
    , ".", null)) {
45      session.setNodeValue("medolution/component/notification/
    notification1XXX/configuration/temp-alarm/data/value", NodeData.of(value)
    );
      } catch (TreeException e) {
47      e.printStackTrace();
      }
49  }
}
```

Listing 10.1: Ambient Temperature On-Controller Policy Rule Java Class

```
package com.materna.medolution.management.showcase.policy;
2 ...
public class AmbientTemperatureOffControllerPolicyRule extends AbstractRule
    {
4   public boolean evaluate(ExecutionContext context) throws Exception {
      Tree managementTree = context.getRuleContext().services().contract(Tree.
    class).execute().getService();
6
      double TEMPERATURE_min = -10;
8     double TEMPERATURE_max = 35;
      boolean b = false;
10    TreeEvent event = context.getEvent();
      Path path = event.getPath();
12
      switch (event.getType()) {
14      case NODE_CHANGED:
          try (SharedSession session = managementTree.openSharedSession(null,
     ".", null)) {
16          NodeData data = session.getNodeValue("medolution/device/
    thermometer/thermometer1XXX/status/measurement/data/value");
            double T = (data.getDouble() != null ? data.getDouble() : 0);
18          double TEMP = T;
            double TEMPERATURE = (TEMP - 32)*5/9;
20
```

```
            b = (TEMPERATURE <= TEMPERATURE_max) && (TEMPERATURE >
    TEMPERATURE_min) ? true : false;
22          } catch (TreeException e) {
              e.printStackTrace();
24            break;
            }
26          break;
          default:
28          break;
      }
30    return b;
    }
32
    public void execute(ExecutionContext context) throws Exception {
34    String value = "Temperature is within the allowed range";
      Tree managementTree = context.getRuleContext().services().contract(Tree.
      class).execute().getService();
36    try (ExclusiveSession session = managementTree.openExclusiveSession(null
      , ".", null)) {
        session.setNodeValue("medolution/component/notification/
      notification1XXX/configuration/temp-alarm/data/value", NodeData.of(value)
      );
38    } catch (TreeException e) {
        e.printStackTrace();
40    }
    }
42 }
```

Listing 10.2: Ambient Temperature Off-Controller Policy Rule Java Class

10.4.2 Derivation of Ambient Environment Policy Expression

Another example of usage of the refinement patterns addresses the ambient condition, also. The evaluation of the current ambient condition is important for carrying out the use case (Figure 10.9). Let us assume, the ambient condition is represented by a parameter stating the heat index of the environment. An attribution pattern (see Section 9.1.2) is needed in order to define how the evaluation is to be done, e.g.:

$$
\begin{aligned}
&"SAFE" : HEAT\_INDEX < 80\\
&"UNFAVOURABLE" : 80 <= HEAT\_INDEX < 90\\
&"HAZARDOUS" : HEAT\_INDEX >= 90
\end{aligned}
$$

So, the corresponding status variable $HEAT\_INDEX$ is included into the model element *Ambient Environment*. The calculation instruction forms the basis of the policy expression to be evaluated at runtime by the management. The heat index is not measured directly, but rather calculated from the ambient air temperature and relative humidity. For this purpose, the *Temperature Service* and *Humidity Service* model elements with their status variables $TEMP$ measured in Fahrenheit and $HUMID$ measured in % are used. The calculation of the $HEAT\_INDEX$ from the lower layer status variables requires also a translator pattern (see Section 9.3.2) with the following calculation instruction:

$$
\begin{aligned}
&double\ HEAT\_INDEX = -42.379 + 2.04901523 * TEMP + 10.14333127 *\\
&HUMID - 0.22475541 * TEMP * HUMID - 0.00683783 * TEMP * TEMP -\\
&0.05481717 * HUMID * HUMID + 0.00122874 * TEMP * TEMP * HUMID +\\
&0.00085282 * TEMP * HUMID * HUMID - 0.00000199 * TEMP * TEMP *\\
&HUMID * HUMID;
\end{aligned}
$$

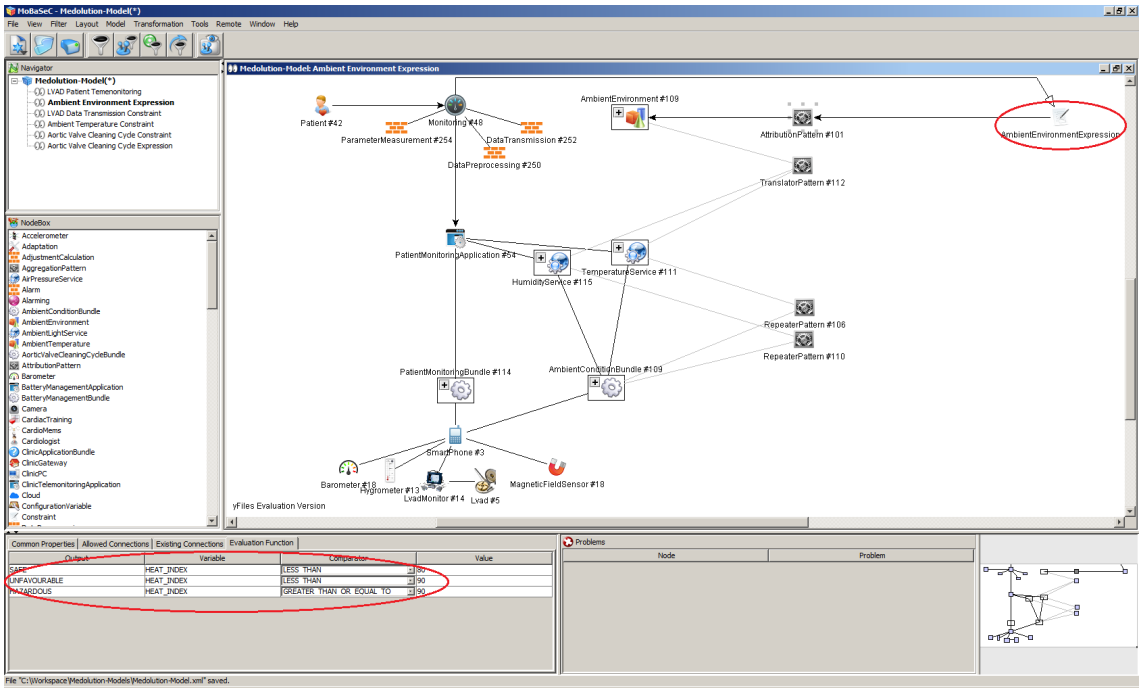

Figure 10.9: Example: Ambient Environment Policy Expression

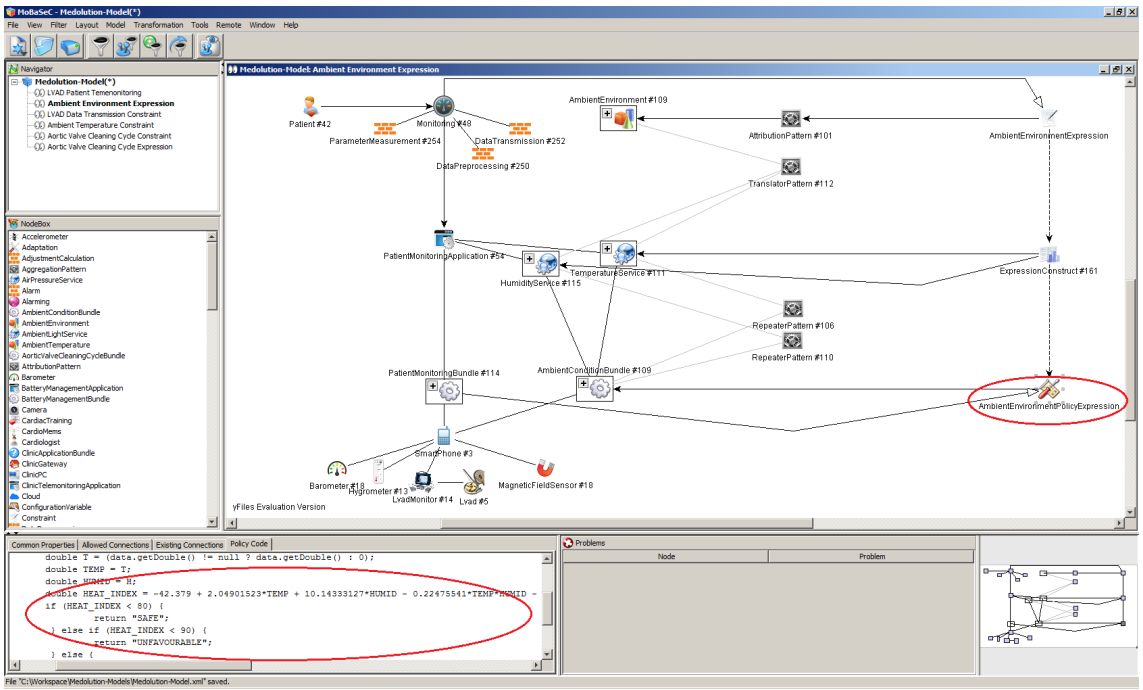

Figure 10.10: Example: Ambient Environment Policy Expression - Derived Model Elements

Descending in the hierarchy from the "Services" to "Components" layer, we refine the status variables $TEMP$ and $HUMID$ into the variables of the lower level. Suppose, the software component *Ambient Condition Bundle* provides a status variable T measured in degree Fahrenheit and H measured in %. Again, a repeater pattern can be used (see Section 9.3.1) which repeats the values of the corresponding variables:

$$double\ TEMP = T;$$
$$double\ HUMID = H;$$

Figure 10.9 demonstrates an extract of the model in the MoBaSeC tool exemplifying the definition of the ambient condition policy expression.

During the policy derivation process, it is iterated in the model from top to bottom. Based on the configured pattern elements, the tool generates the necessary model elements automatically.

Figure 10.10 shows the model after the policy derivation process. The backend functions generate a corresponding source code of the policy expression to be executed during the runtime. The outcome of the policy derivation for the defined ambient condition expression is shown in Listing 10.3.

```
package com.materna.medolution.management.showcase.policy;
...
public class AmbientEnvironmentPolicyExpression extends AbstractExpression<
    String> {
  public String evaluate(EvaluationContext context) throws Exception {
    Tree managementTree = context.getExpressionContext().services().contract
    (Tree.class).execute().getService();

    String ret = "";
    NodeData data = null;

    try (SharedSession session = managementTree.openSharedSession(null, ".",
     null)) {
      data = session.getNodeValue("medolution/device/hygrometer/
    hygrometer11XXX/status/measurement/data/value");
      double H = (data.getDouble() != null ? data.getDouble() : 0);
      data = session.getNodeValue("medolution/device/thermometer/
    thermometer1XXX/status/measurement/data/value");
      double T = (data.getDouble() != null ? data.getDouble() : 0);
      double TEMP = T;
      double HUMID = H;
      double HEAT_INDEX = -42.379 + 2.04901523*TEMP + 10.14333127*HUMID -
    0.22475541*TEMP*HUMID - 0.00683783*TEMP*TEMP - 0.05481717*HUMID*HUMID +
    0.00122874*TEMP*TEMP*HUMID + 0.00085282*TEMP*HUMID*HUMID -0.00000199*TEMP
    *TEMP*HUMID*HUMID;
      if (HEAT_INDEX < 80) {
         return "SAFE";
       } else if (HEAT_INDEX < 90) {
         return "UNFAVOURABLE";
       } else {
         return "HAZARDOUS";
      }
    } catch (TreeException e) {
      e.printStackTrace();
    }
    return ret;
   }
}
```

Listing 10.3: Ambient Environment Policy Expression Java Class

10.4.3 Derivation of Data Transmission Policy Rule

In order to provide a reliable LVAD patient monitoring, the regular data transmission to the clinic is done. Besides the measured physiological parameters (e.g. circulatory, anticoagulation, medication, exercise parameters), the transmitted data also includes technical (e.g. LVAD-specific, battery, connection parameters) as well as context parameters (e.g. geographical, ambient, personal activity parameters). The data is of different nature and is subject to different restrictions and requirements concerning confidentiality, integrity, or availability. These requirements apply necessarily to the data transfer. Thus, a control of the data transfer should take place.

Let us consider a simplified example of controlling the throughput of the data transmission. While transmitting the measured LVAD parameters, the monitoring application can transfer the measured data at several predefined rates. This means technically that different filters can be are applied to the transmitted data. A filter defines the compression rate used in order to process the data before forwarding. E.g., the LVAD power consumption data can be provided by the monitoring application with frequency of 10 or 100 Hz. The data transmission component supports the required data compression, blocking and buffering in order to optimize the transfer on the transport protocol level when moving large amount of data. At the same time there exist certain medical as well as technical requirements for the latency of the transmitted data. E.g., the latency of the transmitted LVAD power consumption data must be between 10 and 120 seconds. In order to measure the latency, the data transmission component exposes the current congestion of the connection. Thus, an appropriate feedback to the monitoring application can be done, which filter to apply. The connection congestion is manifested by means of a number of elements in the storage. So the dynamic adjustment of the compression rate can be done according to the connection status.

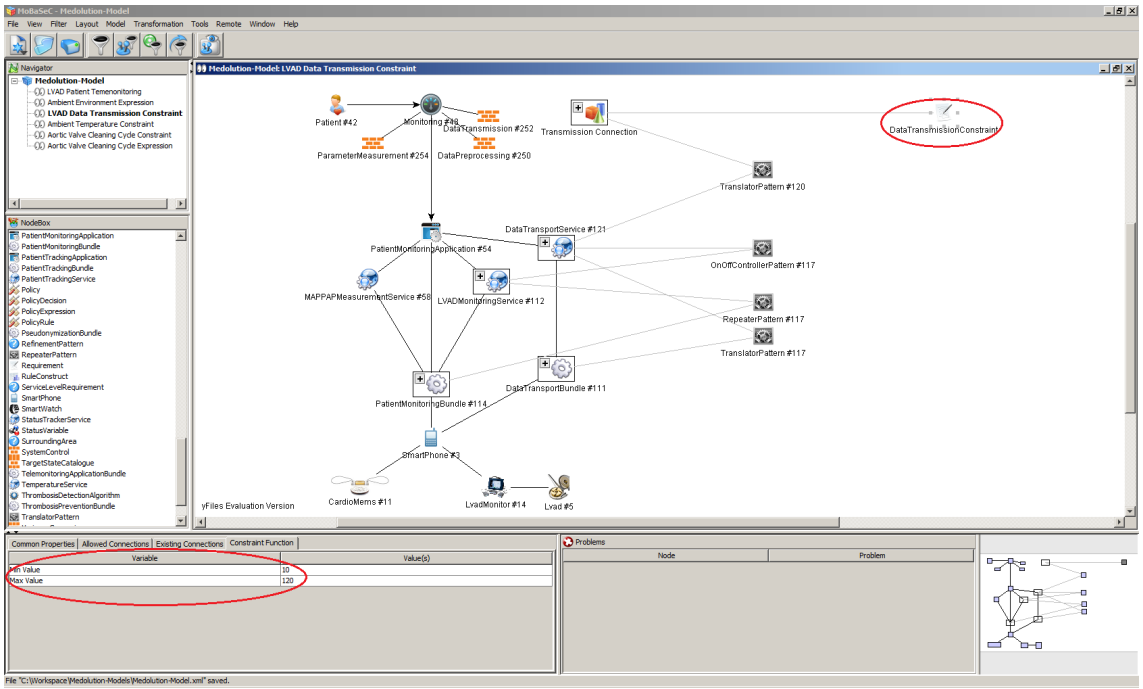

Figure 10.11: Example: Data Transmission Constraint - Derived Model Elements

The model element *Data Transmission* represents the correspondent function provided within the *Monitoring* use case (Figure 10.11). By means of the model element *Trans-*

mission Connection we introduce the monitored asset. The corresponding status variable $LVAD\_POWER\_CONSUMP\_LATENCY$ measured in seconds is provided within the related aspect. To model the requirement, we add a constraint model element related to this status variable. The allowed target range of the latency is mirrored in the configuration of the constraint model element, stating the maximal and minimal permitted values:

$$double\ LVAD\_POWER\_CONSUMP\_LATENCY_{min} = 10;$$
$$double\ LVAD\_POWER\_CONSUMP\_LATENCY_{max} = 120;$$

The service responsible for the data transmission is modeled as the *Data Transport Service*. It exposes its state by means of a status variable $CONGESTION$ measured in % is defined within this model element. A translator pattern is needed (see Section 9.3.2), which states how the computation from the congestion status variable into the expected latency of the LVAD power consumption data is done, e.g.:
$int\ LVAD\_POWER\_CONSUMP\_LATENCY = 300 * CONGESTION/100 + 30;$
The model element for the translator pattern added to the model and is placed between the "Use Cases" to "Services" layer.

We go down the hierarchy in the model to the "Services" layer. The services *LVAD Monitoring Service* and *Data Transport Service* responsible for the measuring the LVAD power consumption data and its transport to the clinic are modeled on this layer. Suppose, the measured latency of the LVAD power consumption data is greater than the predefined maximal value. In this case the corresponding configuration variable $COMP\_RATE$ of the *LVAD Monitoring Service* presenting the configurable compression rate should be set to 10 Hz. On the contrary, if the latency of the LVAD power consumption data is lower than the predefined minimal value, the compression rate of the *LVAD Monitoring Service* should be set to 100 Hz. To model this, we introduce a control pattern, namely an on-off controller pattern (see Section 9.2.4) which controls the output variable, i.e. $COMP\_RATE$, subject to the control variable, i.e. $CONGESTION$ of the *Data Transport Service*. The on-off controller pattern is configured accordingly by stating the control variable and setting the output parameters. The control pattern means that two management rules are required. The first one should fire on exceeding the allowed maximal or minimal threshold and the second one on reaching the allowed target range.

Going down in the hierarchy from the "Services" to "Components" layer, we need an instruction, how to refine the status variable $CONGESTION$ into the concrete technical status variable provided by the software component *Data Transport Bundle*. Suppose the component provides a status variable $STACK\_SIZE$ measured in integer. As a refinement pattern, a translator pattern is used (see Section 9.3.2), which states how the value of the $CONGESTION$ variable is computed from the $STACK\_SIZE$ variable:

$$int\ CONGESTION = STACK\_SIZE * 100/10;$$

Similarly, we need to refine the configuration variable $COMP\_RATES$ into the technical configuration variable $DATA\_FILTER$ provided by the software component *Patient Monitoring Bundle*. For this purpose, a refinement pattern, namely a repeater pattern (see Section 9.3.1), is used, which just repeats the variable values:

$$int\ COMP\_RATE = DATA\_FILTER;$$

Figure 10.12 demonstrates the derived policy elements. Two model elements for the policy rules are generated and added to the model.

The backend functions of the MoBaSeC generate a corresponding source code of the policy rules to be executed during the runtime. The outcome of the policy derivation for the defined data transmission constraint are shown in Listings 10.4, 10.5.

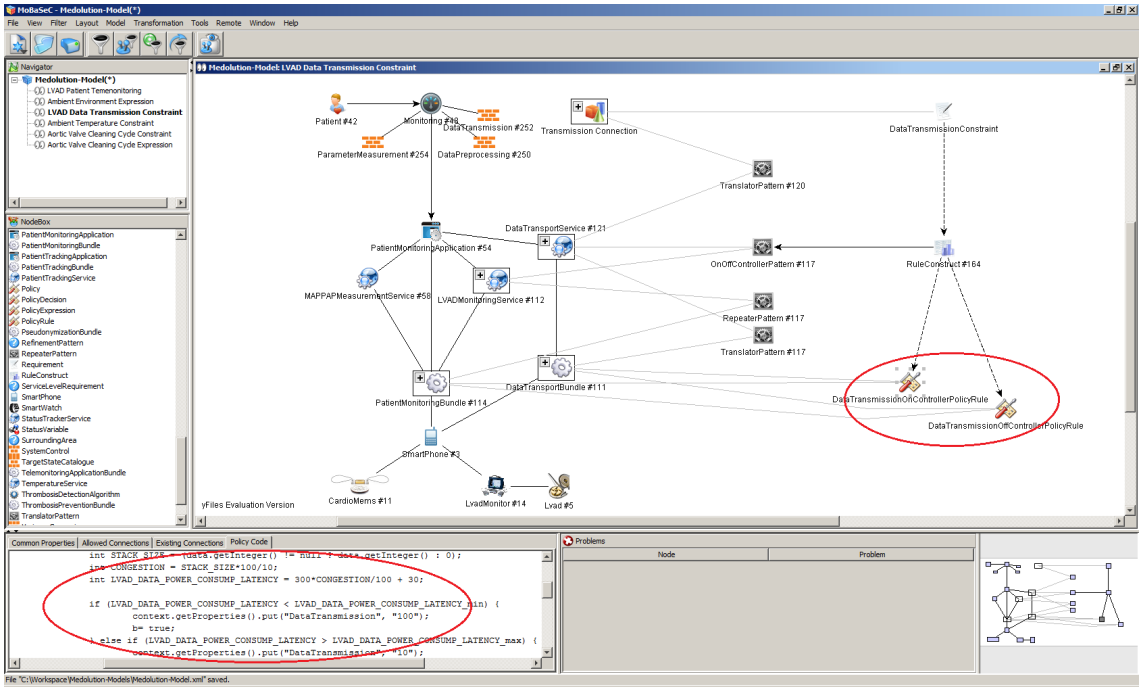

Figure 10.12: Example: Data Transmission Constraint

```
package com.materna.medolution.management.showcase.policy;
...
public class DataTransmissionOnControllerPolicyRule extends AbstractRule {
  public boolean evaluate(ExecutionContext context) throws Exception {
    Tree managementTree = context.getRuleContext().services().contract(Tree.
    class).execute().getService();

    double LVAD_DATA_POWER_CONSUMP_LATENCY_min = 10;
    double LVAD_DATA_POWER_CONSUMP_LATENCY_max = 120;
    boolean b = false;
    TreeEvent event = context.getEvent();
    Path path = event.getPath();

    switch (event.getType()) {
      case NODE_CHANGED:
        try (SharedSession session = managementTree.openSharedSession(null,
     ".", null)) {
          NodeData data = session.getNodeValue("medolution/device/mobile-
    phone/mobile-phone1XXX/status/stacksize/data/value");
          int STACK_SIZE = (data.getInteger() != null ? data.getInteger() :
    0);
          int CONGESTION = STACK_SIZE*100/10;
          int LVAD_DATA_POWER_CONSUMP_LATENCY = 300*CONGESTION/100 + 30;

          if (LVAD_DATA_POWER_CONSUMP_LATENCY <
    LVAD_DATA_POWER_CONSUMP_LATENCY_min) {
            context.getProperties().put("DataTransmission", "100");
            b= true;
          } else if (LVAD_DATA_POWER_CONSUMP_LATENCY >
    LVAD_DATA_POWER_CONSUMP_LATENCY_max) {
            context.getProperties().put("DataTransmission", "10");
            b= true;
          } else {
```

```
            context.getProperties().remove("DataTransmission");
          }
        } catch (TreeException e) {
          e.printStackTrace();
          break;
        }
        break;
      default:
        break;
    }
    return b;
  }

  public void execute(ExecutionContext context) throws Exception {
    int value = (int) context.getProperties().get("DataTransmission");
    Tree managementTree = context.getRuleContext().services().contract(Tree.
    class).execute().getService();
    try (ExclusiveSession session = managementTree.openExclusiveSession(null
    , ".", null)) {
      session.setNodeValue("medolution/device/lvad/lvad1XXX/configuration/
    data-filter/data/value", NodeData.of(value));
    } catch (TreeException e) {
      e.printStackTrace();
    }
  }
}
```

Listing 10.4: Data Transmission On-Controller Policy Rule Java Class

```
package com.materna.medolution.management.showcase.policy;
...
public class DataTransmissionOffControllerPolicyRule extends AbstractRule {
  public boolean evaluate(ExecutionContext context) throws Exception {
    Tree managementTree = context.getRuleContext().services().contract(Tree.
    class).execute().getService();

    double LVAD_DATA_POWER_CONSUMP_LATENCY_min = 10;
    double LVAD_DATA_POWER_CONSUMP_LATENCY_max = 120;
    boolean b = false;
    TreeEvent event = context.getEvent();
    Path path = event.getPath();

    switch (event.getType()) {
      case NODE_CHANGED:
        try (SharedSession session = managementTree.openSharedSession(null,
     ".", null)) {
          NodeData data = session.getNodeValue("medolution/device/mobile-
    phone/mobile-phone1XXX/status/stacksize/data/value");
          int STACK_SIZE = (data.getInteger() != null ? data.getInteger() :
    0);
          int CONGESTION = STACK_SIZE*100/10;
          int LVAD_DATA_POWER_CONSUMP_LATENCY = 300*CONGESTION/100 + 30;

          if ((LVAD_DATA_POWER_CONSUMP_LATENCY <=
    LVAD_DATA_POWER_CONSUMP_LATENCY_max) && (LVAD_DATA_POWER_CONSUMP_LATENCY
    > LVAD_DATA_POWER_CONSUMP_LATENCY_min)) {
            b = true;
            context.getProperties().put("DataTransmission", session.
    getNodeValue("medolution/device/lvad/lvad1XXX/configuration/data-filter/
    data/value").getInteger());
          }
```

```
        } catch (TreeException e) {
          e.printStackTrace();
          break;
        }
        break;
      default:
        break;
    }
    return b;
  }

  public void execute(ExecutionContext context) throws Exception {
    int value = (int) context.getProperties().get("DataTransmission");
    Tree managementTree = context.getRuleContext().services().contract(Tree.
    class).execute().getService();
    try (ExclusiveSession session = managementTree.openExclusiveSession(null
    , ".", null)) {
      session.setNodeValue("medolution/device/lvad/lvad1XXX/configuration/
    data-filter/data/value", NodeData.of(value));
    } catch (TreeException e) {
      e.printStackTrace();
    }
  }
}
```

Listing 10.5: Data Transmission Off-Controller Policy Rule Java Class

10.4.4 Derivation of Aortic Valve Cleaning Cycle Policy Expression

In order to prevent the aortic valve from adhering or developing a thrombosis, an aortic valve cleaning cycle should be conducted periodically. A cleaning cycle means technically that the LVAD pump is switched off for a short period of time in order to allow the aortic valve to open due to the natural blood pressure. In order to allow a LVAD safe cleaning cycle activation, the patient should be awake and not practicing physical activity. Thus, the system must recognize in a dependable manner, these requirements are met. A policy expression can be introduced in order to assist the system in its decision whether it is safe to initiate the aortic valve cleaning cycle or not. Determining whether the patient is awake or not as well as if he is physically active at the moment is not a trivial task. Multiple devices and sensors can be used to recognize the patient's status and activity but the measured results can often be unreliable. Thus, an evaluation of the likelihood of the correct measurement is of interest.

Figure 10.13 demonstrates an extract of the model in the MoBaSeC tool exemplifying the definition of the aortic valve cleaning cycle policy expression. Let us assume, the aortic valve cleaning cycle policy expression evaluates the probability that the required condition is met on the basis of the available measurements and the reliability of the measuring device. The corresponding model element *Aortic Valve Cleaning Cycle Expression* is presented in the model and is connected to the use case model element *Monitoring*. Suppose the patient's activity is represented by two parameters stating the expected likelihood that the patient is awake and resting. So, the corresponding status variables $LH\_PATIENT\_AWAKE$ and $LH\_PATIENT\_RESTING$ are included into the model element *Patient Activity*. Suppose the policy expression is calculated as a weighted arithmetic mean of the both parameters. An aggregation pattern 9.1.1 is used to define the evaluation of the policy expression. With the help of the corresponding model element the status variables are weighted as in Figure 10.13.

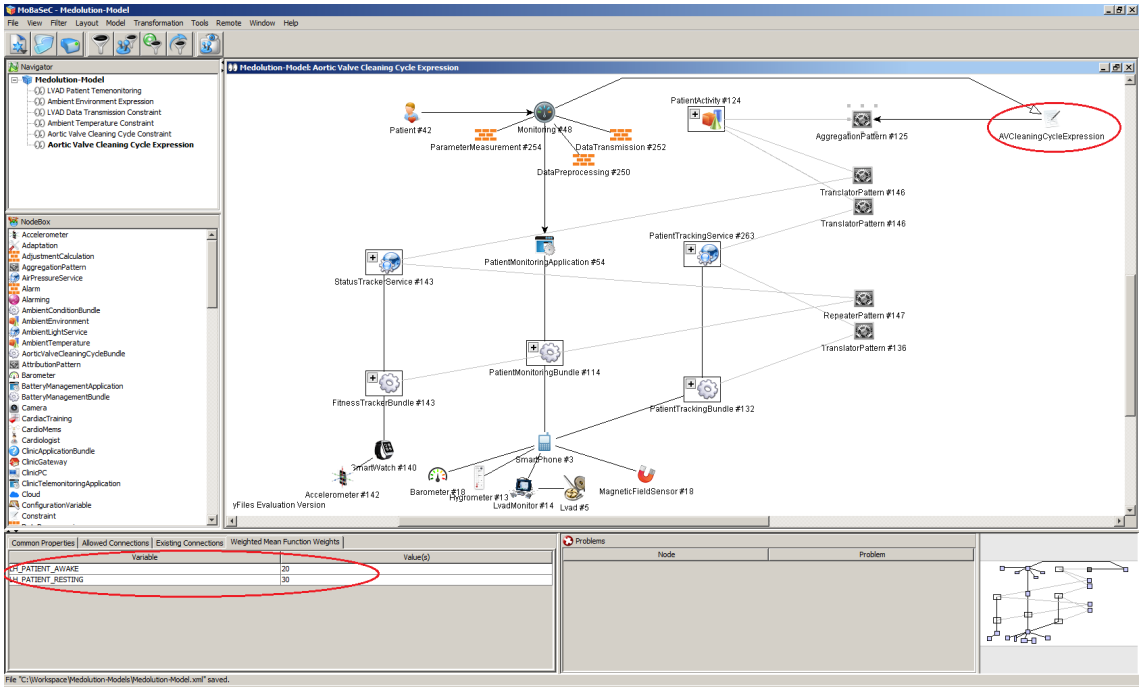

Figure 10.13: Example: Aortic Valve Cleaning Cycle Policy Expression

The probability that the patient is awake at the moment is calculated from the probability that the patient is asleep estimated by the corresponding *StatusTrackerService* presented in the model as a model element on the "Service" layer. The model element includes a status variable LH_ASLEEP. A refinement pattern, in this case a translator pattern (see Section 9.3.2), can be used in order to define how to calculate the status variable of the "Use cases" layer from the "Service" layer variable:

$$int\ LH\_PATIENT\_AWAKE = 1 - LH\_ASLEEP;$$

Similarly, the status variable $LH\_PATIENT\_RESTING$ is calculated from the status variable $LH\_ACTIVITY$ of the *Patient Tracking Service* model element. The translator pattern is parametrized with the code (see Section 9.3.2):

$$int\ LH\_PATIENT\_RESTING = 1 - LH\_ACTIVITY;$$

Descending in the hierarchy from the "Services" to "Components" layer, we refine the status variable LH_ASLEEP into the variables of the lower level. Suppose, the software component *Fitness Tracker Bundle* provides a status variable $ASLEEP$ measured in integer stating the estimated probability that the patient is asleep at the moment. The software component is hosted on the smartwatch device which is presented in the model on the "Components" layer. A repeater pattern can be used (see Section 9.3.1) which repeats the values of the corresponding variables:

$$double\ LH\_ASLEEP = ASLEEP;$$

The status variable $LH\_ACTIVITY$ of the *Patient Tracking Service* is also refined to the status variables of the lowest layer. Thus, the software component *Patient Tracking Bundle* exposes its status as a $STEPS\_NUM$ variable stating the registered number of steps done during the last minute. A translator pattern (see Section 9.3.2) can be used in order to refine the $LH\_ACTIVITY$ status variable:

double LH_ACTIVITY = STEPS_NUM/60;

During the policy derivation process, it is iterated in the model from the "Use Case" to the "Component" layer. Based on the configured pattern elements, the tool generates the necessary model elements automatically.

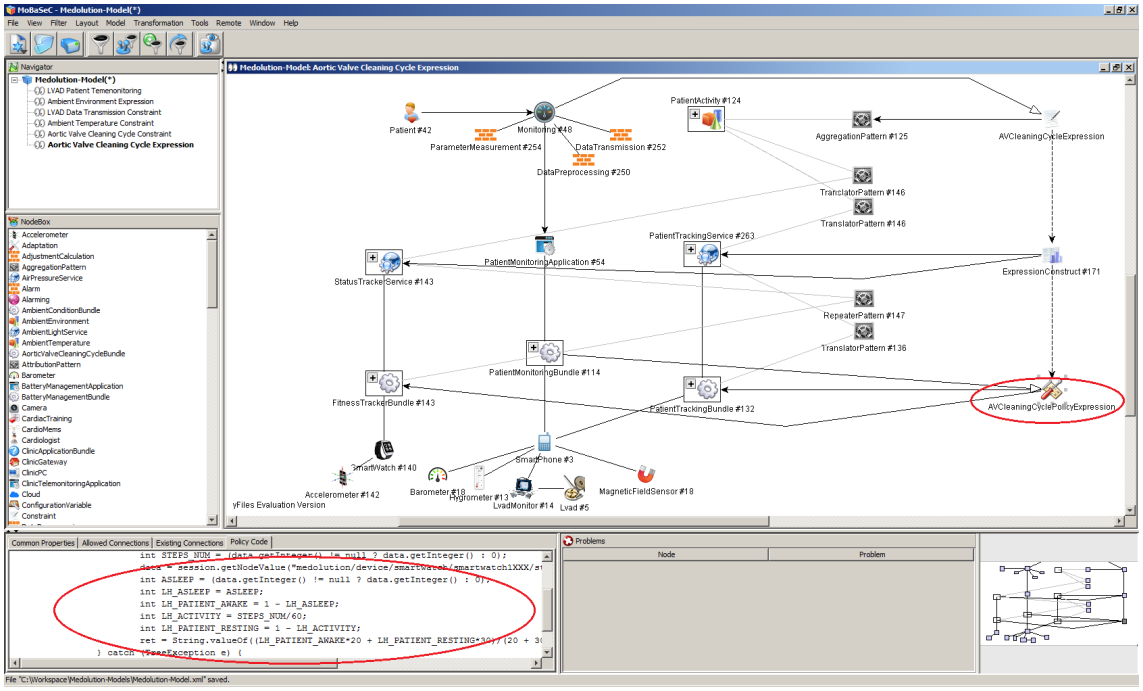

Figure 10.14: Example: Aortic Valve Cleaning Cycle Policy Expression - Derived Model Elements

Figure 10.14 shows the model after the policy derivation process. The backend functions generate a corresponding source code of the policy expression to be executed during the runtime. The outcome of the policy derivation for the defined ambient condition expression is shown in Listing 10.6.

```
1  package com.materna.medolution.management.showcase.policy;
   ...
3  public class AVCleaningCyclePolicyExpression extends AbstractExpression<
       String> {
     public String evaluate(EvaluationContext context) throws Exception {
5      Tree managementTree = context.getExpressionContext().services().contract
       (Tree.class).execute().getService();

7      String ret = "";
       NodeData data = null;
9
       try (SharedSession session = managementTree.openSharedSession(null, ".",
        null)) {
11       data = session.getNodeValue("medolution/device/mobile-phone/mobile-
       phone1XXX/gyroscope/gyroscope1XXX/status/steps-number/data/value");
         int STEPS_NUM = (data.getInteger() != null ? data.getInteger() : 0);
13       data = session.getNodeValue("medolution/device/smartwatch/
       smartwatch1XXX/status/asleep/data/value");
         int ASLEEP = (data.getInteger() != null ? data.getInteger() : 0);
15       int LH_ASLEEP = ASLEEP;
         int LH_PATIENT_AWAKE = 1 - LH_ASLEEP;
17       int LH_ACTIVITY = STEPS_NUM/60;
```

```
        int LH_PATIENT_RESTING = 1 - LH_ACTIVITY;
19      ret = String.valueOf((LH_PATIENT_AWAKE*20 + LH_PATIENT_RESTING*30)/(20
      + 30));
      } catch (TreeException e) {
21      e.printStackTrace();
      }
23    return ret;
    }
25 }
```

Listing 10.6: Aortic Valve Cleaning Cycle Policy Expression Java Class

10.4.5 Derivation of Aortic Valve Cleaning Cycle Policy Rules

As explained above, the initiation of the aortic valve cleaning cycle can be only conducted under certain constraints: the patient is awake and not practicing any physical activity. If the system recognizes that the constraints are violated during the cleaning cycle, the patient should be notified immediately. Figure 10.15 demonstrates an extract of the model in the MoBaSeC tool exemplifying the definition of the aortic valve cleaning cycle policy rules.

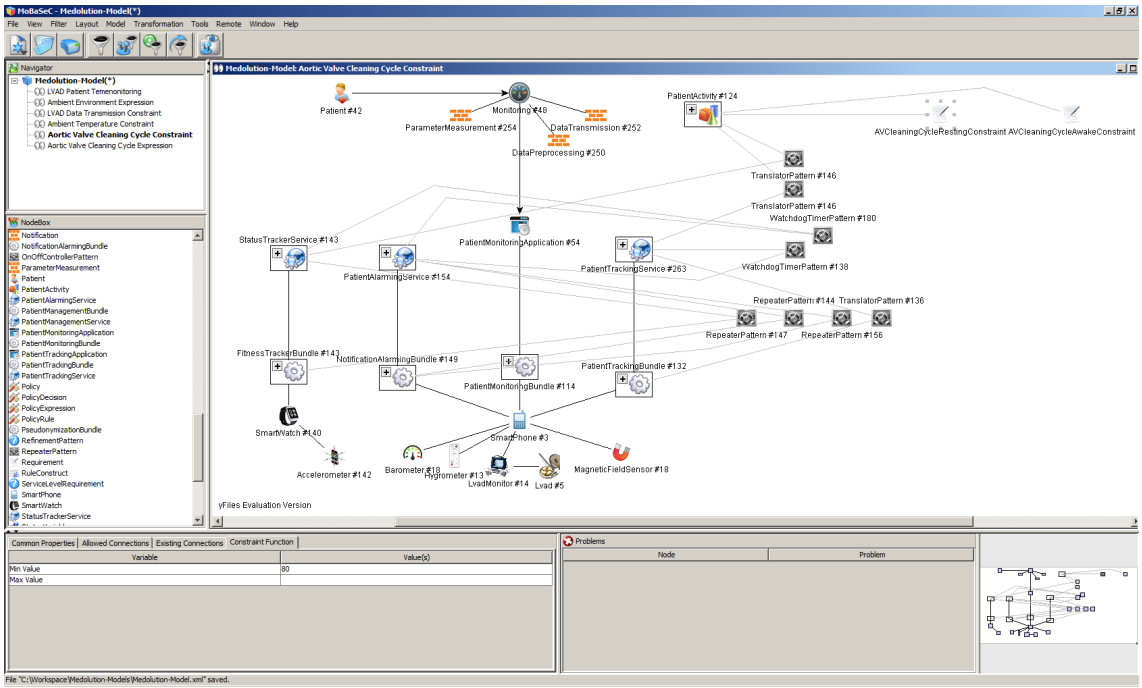

Figure 10.15: Example: Aortic Valve Cleaning Cycle Constraints - Derived Model Elements

Suppose, the probability that the patient is still awake during the aortic valve cleaning cycle should be at least 85%. In order to model this, we introduce a constraint presented as a model element which is configured with a parameter correspondingly:

$$double\ LH\_PATIENT\_AWAKE_{min} = 85;$$

Similarly, the probability that the patient is still not conducting a physical activity should be at least 80%. The introduced model element of the constraint is parametrized in the following way:

$$double\ LH\_PATIENT\_RESTING_{min} = 80;$$

The both status variables are refined to the status variables of the "Services" layer in the same way as in the example above. The "Services" layer is just to extend with the control patterns which define the desired system behavior on the constraint violation. For this purpose, the *Watchdog Timer Patterns* are used (see Section 9.2.1). They are parametrized with the status variable to be monitored: the LH_ASLEEP of the of the *Status Tracker Service* and the $LH\_ACTIVITY$ of the *Patient Tracking Service* model elements. The *Watchdog Timer Patterns* are also parametrized with the configuration variables to be set if the constraint is violated. The configuration variable $CC\_RESTING\_ALARM$ of the *Patient Alarming Service* can be set to "not resting" or "resting" otherwise, the configuration variable CC_AWAKE_ALARM can be set to "not awake" or "awake" correspondingly.

Going down in the hierarchy from the "Services" to "Components" layer, an instruction is needed, how to refine the management variables of the "Services" layer into the concrete technical status and configuration variables provided by the software components of the "Components" layer. In addition to the refinement patterns explained in the section above, we use repeater patterns (see Section 9.3.2) in order to refine the configuration variables of the *Patient Alarming Service* to the configuration variables of its providing software component *Notification Alarming Bundle*. The following refinement functions can be used:

$$String\ CC\_RESTING\_ALARM = LVAD\_RESTING\_ALARM;$$
$$String\ CC\_AWAKE\_ALARM = LVAD\_AWAKE\_ALARM;$$

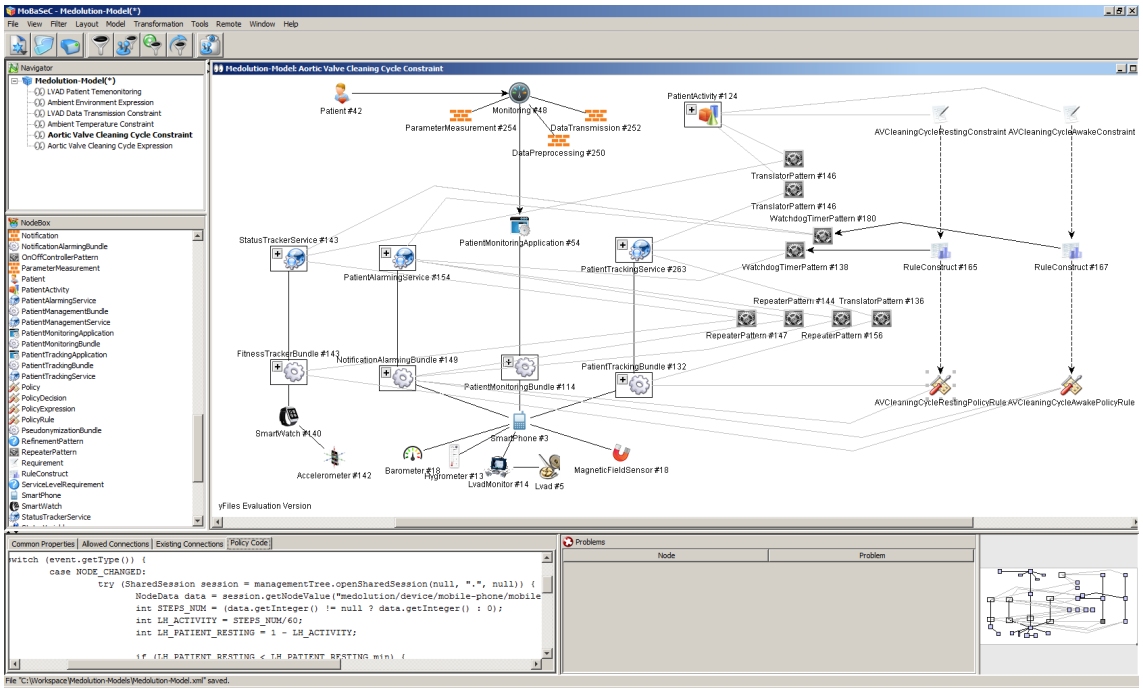

Figure 10.16: Example: Aortic Valve Cleaning Cycle Constraints - Derived Model Elements

Similarly, the policy derivation process includes iterating the model from top to bottom. Based on the configured pattern elements, the tool generates the necessary model elements automatically. Figure 10.16 demonstrates the derived policy elements.

The backend functions generate a corresponding source code of the policy rules to be executed during the runtime.

The result of the policy derivation for the aortic valve cleaning cycle constraints are shown in Listings 10.7, 10.8.

```
package com.materna.medolution.management.showcase.policy;
...
public class AVCleaningCycleRestingPolicyRule extends AbstractRule {
  public boolean evaluate(ExecutionContext context) throws Exception {
    Tree managementTree = context.getRuleContext().services().contract(Tree.
    class).execute().getService();

    int LH_PATIENT_RESTING_min = 80;

    boolean b = false;
    TreeEvent event = context.getEvent();
    Path path = event.getPath();

    switch (event.getType()) {
      case NODE_CHANGED:
        try (SharedSession session = managementTree.openSharedSession(null,
     ".", null)) {
          NodeData data = session.getNodeValue("medolution/device/mobile-
    phone/mobile-phone1XXX/gyroscope/gyroscope1XXX/status/steps-number/data/
    value");
          int STEPS_NUM = (data.getInteger() != null ? data.getInteger() :
    0);
          int LH_ACTIVITY = STEPS_NUM/60;
          int LH_PATIENT_RESTING = 1 - LH_ACTIVITY;

          if (LH_PATIENT_RESTING < LH_PATIENT_RESTING_min) {
            context.getProperties().put("AVCleaningCycleResting", "not
    resting");
            b= true;
          } else {
            context.getProperties().remove("AVCleaningCycleResting");
          }
        } catch (TreeException e) {
          e.printStackTrace();
          break;
        }
        break;
      default:
        break;
    }
    return b;
  }

  public void execute(ExecutionContext context) throws Exception {
    String value = "AV cleaning cycle is running under critical conditions:
    " + context.getProperties().get("AVCleaningCycleResting");
    Tree managementTree = context.getRuleContext().services().contract(Tree.
    class).execute().getService();
    try (ExclusiveSession session = managementTree.openExclusiveSession(null
    , ".", null)) {
      session.setNodeValue("medolution/component/notification/
    notification1XXX/configuration/lvad-cc-resting-alarm/data/value",
    NodeData.of(value));
    } catch (TreeException e) {
      e.printStackTrace();
    }
  }
}
```

Listing 10.7: Aortic Valve Cleaning Cycle Resting Policy Rule Java Class

```
package com.materna.medolution.management.showcase.policy;
...
public class AVCleaningCycleAwakePolicyRule extends AbstractRule {
  public boolean evaluate(ExecutionContext context) throws Exception {
    Tree managementTree = context.getRuleContext().services().contract(Tree.
    class).execute().getService();

    int LH_PATIENT_AWAKE_min = 85;

    boolean b = false;
    TreeEvent event = context.getEvent();
    Path path = event.getPath();

    switch (event.getType()) {
      case NODE_CHANGED:
        try (SharedSession session = managementTree.openSharedSession(null,
     ".", null)) {
          NodeData data = session.getNodeValue("medolution/device/smartwatch
    /smartwatch1XXX/status/asleep/data/value");
          int ASLEEP = (data.getInteger() != null ? data.getInteger() : 0);
          int LH_ASLEEP = ASLEEP;
          int LH_PATIENT_AWAKE = 1 - LH_ASLEEP;

          if (LH_PATIENT_AWAKE < LH_PATIENT_AWAKE_min) {
            context.getProperties().put("AVCleaningCycleAwake", "not awake")
    ;
            b= true;
          } else {
            context.getProperties().remove("AVCleaningCycleAwake");
          }
        } catch (TreeException e) {
          e.printStackTrace();
          break;
        }
        break;
      default:
        break;
    }
    return b;
  }

  public void execute(ExecutionContext context) throws Exception {
    String value = "AV cleaning cycle is running under critical conditions:
    " + context.getProperties().get("AVCleaningCycleAwake");
    Tree managementTree = context.getRuleContext().services().contract(Tree.
    class).execute().getService();
    try (ExclusiveSession session = managementTree.openExclusiveSession(null
    , ".", null)) {
      session.setNodeValue("medolution/component/notification/
    notification1XXX/configuration/lvad-cc-awake-alarm/data/value", NodeData.
    of(value));
    } catch (TreeException e) {
      e.printStackTrace();
    }
  }
}
```

Listing 10.8: Aortic Valve Cleaning Cycle Awake Policy Rule Java Class

Chapter 11

Evaluation

This chapter depicts the evaluation of the elaborated approach on the basis of the conducted case study presented in Chapter 10:

- The development of the technical management for medical devices and systems in the proposed approach is measured and evaluated.
- It is shown that the technical management in the proposed approach supports dependable behavior of medical devices and systems.

11.1 Measurements

The evaluation of technologies can be performed in a *product-oriented* or *process-oriented* way [BE08]. In the first case, the focus of the evaluation is a product itself, in the last case, the focus is on the assessing the impact of a new technology as a whole on the existing practices. Thereby, multiple quality models exist. Their purpose is to define, evaluate and measure the quality by systematic specification of the relevant factors, criteria and metrics. The normative quality models are universal and are applicable to any kind of technology, whereas the concrete ones are context-specific, they are tailored to the individual requirements and environment of the stakeholders [Bäc13]. In order to construct an individual quality model, several well-known quality models have been considered: Boehm's [BBL76], McCall's [McC77], Gilb's [Gil88], Schweiggert's [Sch85] and ISO9126-1 [ISO01].

For the evaluation of the developed approach, we construct an individual quality model on the basis of the combination of Schweiggert's [Sch85] and ISO 9126-1 [ISO01] quality models. The quality model concentrates on the features which are of essential importance and most relevant for the introduced requirements and statements (Figure 11.1).

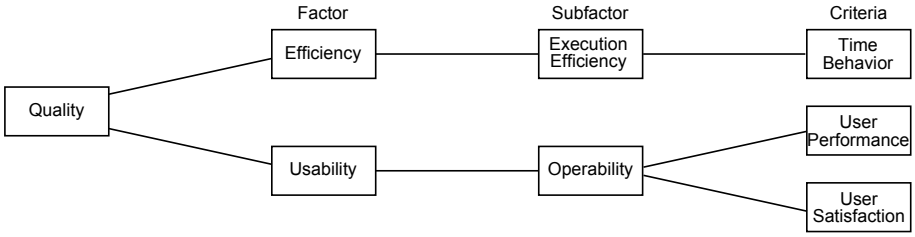

Figure 11.1: Quality Model for the Evaluation

ISO9126-1 [ISO01] quality model defines the *efficiency* as "the capability of the software product to provide desired performance, relative to the amount of resources used, under stated conditions". In particular, concerning the execution efficiency, the *time behavior* is defined as "the capability of the software product to provide appropriate response

and processing times and throughput rates when performing its function, under stated conditions."

The *usability* is defined according to ISO 9126-1 [ISO01] quality model as "the capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions". The *operability* is defined as "the capability of the software product to enable the user to operate and control it". ISO 9241-11 [ISO98] explains how (operational) usability can be evaluated in terms of *user performance* and *user satisfaction* dependent on the context of use. Thereby, user performance is measured by the extent to which the intended goals of use are achieved and the resources such as time and effort that have to be expended to achieve the intended goals. Whereas satisfaction is measured by the extent to which the user finds the use of the product acceptable.

Although the terms above apply to the software products, we resort to these definitions and transfer the concepts to the definition of the quality model for the developed approach.

The Goal-Question-Metric (GQM) approach, presented in [Bas92] and elaborated in further detail in [BE08], [Wal11], can be used in order to derive the metrics used for the measurement of the quality features.

Additionally, as stated in [HS05], the model-based design requires to consider the following aspects during the evaluation process: the size and complexity of the model as well as the size of the automatically generated code.

The following sections describe the evaluation of the identified quality factors according to the chosen quality model by using the GQM approach. Thereby we evaluate the planning and the runtime phases separately.

11.1.1 Planning Phase

During the planning phase, aside from the conceptual analysis and design, the system model is to be created. The MoBaSeC tool is used in order to assist the user in the process.

Time Behavior

Following the chosen quality model, we narrow the identified quality factor of the time behavior during the planning phase down to the capability of the developed approach

- to allow a qualified system designer to create a system model, formulate the requirements and initiate the refinement process in the modeling tool and
- to derive from the modeling tool the generated runtime policies within an appropriate processing time.

Table 11.1 demonstrates the results of the GQM analysis of the time behavior quality factor of the developed approach during the planning phase.

Goal	
Purpose	Evaluation of
Issue	the time behavior of
Object (process)	the modeling process
Viewpoint	from the system designer's point of view
Question	**Metric**

to be continued...

. . .to be continued

Question	Metric
Q1: How much time does the system designer need to produce the model?	**M1.1:** Time to instantiate the model **M1.2:** Time to instantiate the model elements **M1.3:** Time to instantiate the policy derivation patterns **M1.4:** Time to instantiate the policy elements for the expressions and constraints
Q2: How much time does the system designer need to represent the requirements in the model?	**M2.1:** Time to configure the initial model elements **M2.2:** Time to configure the policy derivation patterns **M2.3:** Time to configure the policy elements
Q3: How much time does the policy refinement process take?	**M3.1:** Time which the system designer needs to activate the refinement process **M3.2:** Time which the tool takes to automatically derive the runtime policies

Table 11.1: GQM for the Time Behavior during the Planing Phase

Experimental Setup

It is obvious that the *time and effort* spent on the modeling of the system depend considerably on the complexity and size of the system itself and the number of the defined requirements and constraints. The experiment has been conducted for the MEDOLUTION case study presented in Section 10.3.

The initial model for the presented application case encounters 32 model elements and 28 associations between them. Besides, 15 status and 8 configuration variables have been specified. The model has been extended with 20 policy derivation pattern instances defined on the management variables: 2 evaluation, 4 control and 14 refinement patterns. Further, 6 policy elements have been defined: 2 policy expressions and 4 constraints. During the policy refinement process 6 auxiliary policy constructs and 8 runtime management policies have been generated and added to the model: 2 policy expressions and 6 policy rules. Thus, the finalized model includes 72 model elements and 97 associations.

The experiment has been conducted on a PC with the following hardware configuration: Intel® Core™ i5-2500, CPU 3.30 GHz, 10 GB RAM with the 64 Bit Windows 7 Professional SP1 operating system.

Measurements

Table 11.2 demonstrates the time spent on the single tasks in the MoBaSeC tool during the planning phase of the MEDOLUTION case study.

Task	Description	Measured Time
Instantiation of the model **(M1.1)**	Per mouse click the "New Model.." menu item of the menu bar is chosen. The name, storage location as well as the corresponding metamodel are chosen.	10 sec

to be continued. . .

...to be continued

Task	Description	Measured Time
Instantiation of the model elements **(M1.2)**	The corresponding metamodel element types are put per drag-and-drop into the model. The elements are arranged as desired.	5 min
Configuration of the initial model elements **(M2.1)**	The model elements are renamed as needed. The associations between the model elements are modeled.	3 min
Instantiation of the policy derivation patterns **(M1.3)**	The corresponding metamodel element types are put per drag-and-drop into the model. The elements are arranged as desired.	3 min
Configuration of the policy derivation patterns **(M2.2)**	The policy patterns are connected with the associated management variables. The patterns are parametrized with the source code.	5 min
Instantiation of the policy elements for the expressions and constraints **(M1.4)**	The corresponding metamodel element types are put per drag-and-drop into the model. The elements are arranged as desired.	3 min
Configuration of the policy elements **(M2.3)**	The policy elements are connected with the associated management variables. The policy elements are parametrized with the corresponding values.	3 min
Policy refinement process **(M3.1, M3.2)**	Per mouse click the "Policy Refinement" menu item of the menu bar is chosen. The target location of the policy files is chosen. The refinement process is started and performs automatically.	20 sec

Table 11.2: MEDOLUTION Case Study: Measured Time Effort Per Task

The tasks are listed in the chronological order as they are performed during the planning phase. The used metrics as well as the measured time are provided.

Evaluation

The modeling of the system is a *straightforward process*. After the standard procedures of the requirements analysis, the technical structure of the system is designed: the main components are identified, the interfaces are defined and the requirements are set. On this basis, the system model including the management artifacts is created. For this purpose, the modeling tool is used. The in advance elaborated metamodel is used in order to build the system model. The model elements are initiated and configured by the user. The requirements and constraints are modeled by means of the corresponding abstract policy elements. The back-end functions of the modeling tool generate automatically the management artifacts (refined policies and configurations) ready to be used by the management system during the runtime. As the measurements have shown, the modeling process of the demonstration use case (Chapter 10) in the MoBaSeC tool including the derivation of the runtime policies and configurations has taken under 23 minutes.

We assume, that the developed approach (including the specialized medical metamodel and policy derivation patterns) can support a qualified system designer in the process of the development of the automated technical management for other comparable medical use cases. The time and effort spent during the planning phase can be evaluated as definitely appropriate, provided that the modeling process is conducted under the similar conditions (e.g., use case complexity, used hardware, management scenario).

Operability

Following the chosen quality model, we narrow the identified subfactor of *operability* down to the criteria of *user performance* and *satisfaction* while using the modeling tool during the planning phase. They are reflected in the suitability of the used approach to

- enable the user to operate the tool and
- support the user to control it.

While choosing the appropriate metrics for the evaluation, we adopt several metrics from ISO 9241-110 [ISO95]. Table 11.3 demonstrates the results of the GQM analysis for the operability of the developed approach during the planning phase.

Goal	
Purpose **Issue** **Object (process)** **Viewpoint**	Evaluation of the operability of the modeling process from the system designer's point of view
Question	**Metric**
Q1: Is the tool suitable for the modeling process?	**M1.1:** Proportion of the model elements that have been presented in the model **M1.2:** Proportion of the requirements that have been presented in the model **M1.3:** Ease of replacing a model element **M1.4:** Ease of metamodel compatibility check in the model
Q2: How self-descriptive and intuitive is the modeling process for the system designer?	**M2.1:** Ease of understanding the model elements and their relationships
Q3: How comfortable is the modeling process?	**M3.1:** Ease of handling the model elements and their relationships **M3.2:** Proportion of the elements that can be made persistent **M3.3:** Clarity of the visual representation of the model structure
Q4: How customizable are the single elements of the model?	**M4.1:** Configurability of the model elements **M4.2:** Configurability of the policy derivation patterns

Table 11.3: GQM for the Operability

Experimental Setup

The experimental setup is identical to the one presented in the section above. The MoBaSeC tool is used to model the demonstration case under the same conditions (e.g., used hardware, management scenario).

Measurements

The evaluation of the operability has been performed while conducting the modeling process in the MoBaSeC tool. All the identified model elements and the requirements have been presented in the model (**M1.1**) and (**M1.2**). The model is presented as a graph structure where the graph nodes stand for the model elements and the graph edges present

the elements' associations. This form of representation is intuitive, natural and easy to understand for the target user, i.e. system designer, (**M2.1**).

As the model elements are placed into the model or connected with each other, a direct metamodel compatibility check is performed by the MoBaSeC tool (**M1.4**) .

The ergonomics of the approach relies on the advantages of the MoBaSeC tool. Thus, the *drag-and-drop* function supports a comfortable instantiation of the model elements (**M3.1**) . The elements can be added to and also deleted from the model quickly as needed (**M1.3**) . This is beneficial in the sense of the usage comfort, since the process of modeling is a creative task and needs most likely several iterations. The drag-and-drop function allows the user to layout and organize the model elements quickly in the desired manner (**M3.3**). The layout as well as all the modeled elements can be persisted as needed any time (**M3.2**).

The size of the model itself is quite extensive. The graphical representation, however, makes the display of the model clear and intuitive in contrast to the textual or formal representation (**M2.1**). In order to avoid the potential complexity in case of a large number of the model elements, there are several MoBaSeC functions to point out.

Thus, the tool allows to employ multiple *views* (**M3.3**), so that the user can better concentrate on the specific point of view. For each view only the model elements of interest are shown, the not relevant ones are faded out. E.g., for the presented system model including 72 model elements and 97 associations 5 different views have been used, in order to reduce the number of simultaneously visible model elements.

Another ergonomic function is that the management variables are presented as separate elements assigned to the corresponding parent elements. The parent model element is implemented as a *folder* which can be expanded per mouse click if needed. This allows to model the associations between the management variables in a comfortable manner by pulling an edge between the corresponding model elements (**M3.3**). In case the management variables need not to be seen, the parent node can be just folded up.

The model elements as well as the policy derivation patterns are configured initially by the system designer. For this purpose, they are provided with *input masks*, e.g. for inputting the thresholds values (**M4.1**) or *text fields*, e.g. for providing a source code for the refinement functions (**M4.2**), so that the user can configure them in a comfortable manner. The patterns are placed per drag-and-drop into the model, connected to the associated model elements and parametrized with the corresponding values.

Evaluation

The experiments have shown that the system designer while using the MoBaSeC tool can perform the modeling process of the in Chapter 10 demonstration use case in a comfortable manner. We assume, that the developed approach (including the specialized medical metamodel and policy derivation patterns) can support a qualified system designer in the process of modeling and policy derivation for any other medical use case in the same comfortable manner.

11.1.2 Runtime Phase

During the runtime phase, the management artifacts (i.e. the derived management policies, configurations, management tree) are used by the runtime management system presented in Chapter 7 in order to support the management process.

Time Behavior

Following the chosen quality model, we narrow the identified quality factor of the time behavior during the runtime phase down to the capability of the developed approach to provide the technical management, which

- has appropriate response and processing time and
- causes acceptable overhead and load while performing its function.

Table 11.4 demonstrates the results of the GQM analysis of the time behavior quality factor of the developed approach during the runtime phase.

Goal	
Purpose **Issue** **Object (process)** **Viewpoint**	Evaluation of the time behavior of the technical management during the runtime from the system administrator's point of view
Question	**Metric**
Q1: What is the processing speed of the management operations?	**M1.1:** Execution time of a policy expression **M1.2:** Execution time of a policy rule
Q2: How much time does it take to deploy the management system and make it ready for use?	**M2.1:** Time to start and configure the runtime environment **M2.2:** Time to deploy the management system and artifacts
Q3: What is the time overhead caused by the management?	**M3.1:** System startup delay caused by the deployment of the management system **M3.2:** Delay caused by the operation of the management system at runtime

Table 11.4: GQM for the Time Behavior during the Runtime Phase

Experimental Setup

A demonstration scenario has been elaborated for the MEDOLUTION case study presented in Section 10.3. It included the deployment of the management system and the generated artifacts (i.e. policy rules and policy expressions) as well as the exemplary invocation of the policies. Each of the refined 2 policy expressions and 6 policy rules have been evaluated once during the demonstration scenario. That means 2 policy evaluation requests have taken place and 6 times a constraint violation has caused that a policy rule has been invoked.

The experiment has been conducted on a PC with the following hardware configuration: Intel® Core™ i5-2500, CPU 3.30 GHz, 10 GB RAM with the 64 Bit Windows 7 Professional SP1 operating system. As an execution platform for the demonstration scenario the Eclipse Equinox 4.6.2 OSGi platform has been chosen.

Measurements

During the experiment, the execution of the policy rules has taken about 5-13 ms each (**M1.2**). The policy expression evaluation requests have taken about 7-16 ms (**M1.1**).

Further, a rough estimate of the policy execution time in the worst case has been made. For this purpose, the structure of the policy has to be looked at. The main

time expenditure of a policy invocation can be reduced to the time expenditure of the management tree accesses. Thus, it is to estimate how many management tree accesses pro policy take place. The invocation of a policy rule comprises two relevant methods: the *evaluate(ExecutionContext context)* method which corresponds to the *event-condition* part of the rule and the *execute(ExecutionContext context)* method which realizes the *action* part of the policy rule. The number of the relevant management tree accesses is summarized for each method in the Table 11.5.

Policy Type	Method	Reading Tree Accesses	Writing Tree Accesses
Policy rule	*evaluate()*	number of status variables	0
	execute()	0	number of configuration variables
Policy expression	*evaluate()*	number of status variables	0

Table 11.5: Management Tree Accesses

Thereby, the number of the related configuration and status variables varied from 1 to 2 pro policy. It is obvious, that in general the number of the associated management variables can be greater but it is limited by the underlying model.

In order to evaluate how long does it take to perform a single management tree access, the following aspects are to be considered: the tree access session type (shared, exclusive, transactional), the access type (get, set, create, delete), the path length, and the variable type. The correct usage of the management tree supposes that the data access session is opened as low as possible in the management tree structure. Thus, in the worst case the session is opened on the root element. That means in case of an exclusive or a transactional session the access to the other management variables is not possible for the time duration of the session. The maximal time duration can be set by the developer in the configuration. In the demonstration scenario, this setting of the maximal validation time was set to 10 minutes and the setting of the maximal inactive timeout was set to 5 minutes.

Depending on the invocation strategy of the policy expression evaluation, a policy request call can block the program run or not. In the demonstration scenario, we have used the strategy, which returns the evaluation request immediately. So that the there has been no delay in the application run caused by the invocation of policies (**M3.2**). Otherwise the evaluation request can block until the calculation completes but not later than the predefined timeout. That means the developer has a control on the time overhead.

The time needed for the deployment of the management system and making it ready for the use has been measured, also.

Task	Description	Time Effort
Deployment of Runtime Environment **(M1.1)**	OSGi Framework start, initialization and start of bundles (18 framework, 17 management system, 2 demonstration scenario bundles)	15 sec
Deployment of Management System **(M1.2)**	Parsing of the DDF schema (1 root, 8 devices and 7 components schemas)	3 sec
	Management tree initialization and start	2 sec
	Setting the initial values of the management variables (15 configuration variables)	1 sec
	Registration of policy rules and expressions by the management services (6 policy rules, 2 policy expressions)	2 sec

Table 11.6: Time Expenditure for Deployment

Table 11.6 summarizes the time expenditure for the single deployment steps. The overall time expenditure for the management system deployment in the presented demonstration case using the above mentioned hardware configuration has amounted to 22 seconds (**M2.1**, **M2.2**). Since the deployment of the management system and artifacts is performed independently of the application run, the time overhead caused by it, is zero (**M3.1**).

Evaluation

The experiments have shown, that the management operations are performed within an appropriate time. We assume, that the same performance is to be expected for other comparable application use cases, provided that the runtime management is conducted under similar conditions (e.g., use case complexity, used hardware, management scenario). The overhead, which the usage of the management system causes, is minor.

Operability

Following the chosen quality model, the identified subfactor of operability comprises the administrator's performance and satisfaction while using the management system during the runtime phase. In other words, we evaluate, whether the developed approach

- allows a qualified administrator to deploy, configure and operate the management system with the included management artifacts (also while using self-contained applications) in a comfortable manner as well as
- supports him to control it (e.g. update, reconfigure) during the runtime in an appropriate way.

Goal	
Purpose **Issue** **Object (process)** **Viewpoint**	Evaluation of the operability of the management system at runtime from the administrator's point of view
Question	**Metric**
Q1: Is the approach suitable for the management of self-contained applications?	**M1.1:** Ease of making a self-contained application manageable
Q2: How comfortable is the operation of the management system at runtime?	**M2.1:** Ease of deploying the management system **M2.2:** Ease of deploying new policies
Q3: How configurable is the management system at runtime?	**M3.1:** Ease of replacing policies at runtime **M3.2:** Ease of activation and deactivation of policies at runtime

Table 11.7: GQM for the Operability

Experimental Setup

The experimental setup is identical to the one presented in the section above. The *Patient Monitoring Application* and *Ambient Condition Application* are supposed to be self-contained and are made manageable during the experiment.

Measurements

In order to be made manageable, the *Patient Monitoring Application* and *Ambient Condition Application* have been extended. Namely, the required configuration and status management variables have been labeled in the source code correspondingly. E.g. the *Ambient Condition Bundle* has been provided with the two status variables T (for temperature) and H (for humidity), the *Patient Monitoring Bundle* has been provided with the configuration variable *DATA_FILTER* (for the setting of the used data filter). The policy evaluation requests of the *Ambient Environment Policy Expression* and *Aortic Valve Cleaning Cycle Policy Expression* have been added to the source code. For a qualified developer or administrator, the effort has been marginal (**M1.1**).
The realization of the management system and the demonstration scenario on the basis of the OSGi framework has brought a lot of advantages during the runtime phase. The correct order of the deployment is ensured by means of the OSGi-specific on-board instruments. After starting the OSGi framework, the management system is initialized and started. Before the demonstration use case is started the active DDF schema is parsed so that the management tree can be initialized and filled with the initial values. Thus, the native support of the lifecycle management of components facilitates the comfortable deployment of the management system and its configurations (**M2.1**).

Because the policies are packaged as OSGi bundles, they have been registered dynamically at runtime easily (**M2.2**). As needed the policies have been activated or deactivated by the system administrator. This is possible due to the service-oriented design of the management functions (i.e. rule service and expression service) (**M3.2**).

The OSGi native management of inter-component dependencies and the modular design ensure the simplified dynamic update. The components can be started and stopped without the need for shutting down the whole system. Thus, the changes in the system model as well as the requirements can be easily transferred to the new configuration of the management system. That means that the bundle with the new management policies can just replace the old one during the runtime phase (**M3.1**).

Evaluation

If sources are available, the effort needed for making a self-contained application manageable is assumed to be acceptable, provided that the task is completed by the qualified professional.

Owing to realization in OSGi, the operation of the management system at runtime is comfortable. It concerns both the deployment of the required infrastructure and the management system itself with its configurations. Moreover, the approach allows the administrator to control the management system at runtime in a comfortable manner.

11.2 Dependable Behavior

In order to show that the technical management, in the proposed approach supports dependable behavior of medical devices and systems, we regard the effects it has had on the system in the demonstration scenario described in Chapter 10.

In particular, we evaluate, if the availability, reliability and safety of the system have been improved due to the use of the technical management. That means, we look specifically at the occurrence of predictable and/or avoidable

- system failures or breakdowns,

- emergency stops on the part of the application,
- omissions of critical blocks or functions,

caused by

- unfavorable ambient conditions,
- deficient and/or defective operating resources,
- system state

and handled by the technical management.

During the evaluation process, we resort to Event Tree Analysis (ETA) as a basic technique, performing a bottom up forward error analysis [Lev95], [Eri05], [Sut14].

In the following sections we look at the policy rules and expressions used by the management system within the presented application scenario.

11.2.1 Policy Rules

The demonstration scenario has included 4 constraints which have been refined into 6 policy rules. One of the very important constraints identified during the requirements analysis was the *ambient temperature constraint*.

Ambient Temperature Constraint

The performance of the batteries can drastically decrease or turn unpredictable at very low and high temperatures. For this reason, the management monitors the ambient temperature in order to stay within the allowed range.

If the management detects that the temperature is out of the safe zone, it configures the notification service accordingly. When the temperature reaches the safe zone again, the management reconfigures the notification service.

Goals

- Continuous as possible operation of the LVAD at temperatures that do not impair the function of the LVAD batteries
- Reduction of occurrence of terminations caused by the improper function of the LVAD batteries due to the ambient condition
- The patient stays informed about the possible risk of the battery malfunction induced by unsuitable temperature

Hazards

- The function of LVAD batteries is impaired, since they are exposed to extreme temperatures; the operation of the system terminates
- The patient is not informed that the temperature has reached the allowed range again

Before performing the event tree analysis, the exceptions which can interrupt the correct course of action are to be defined.

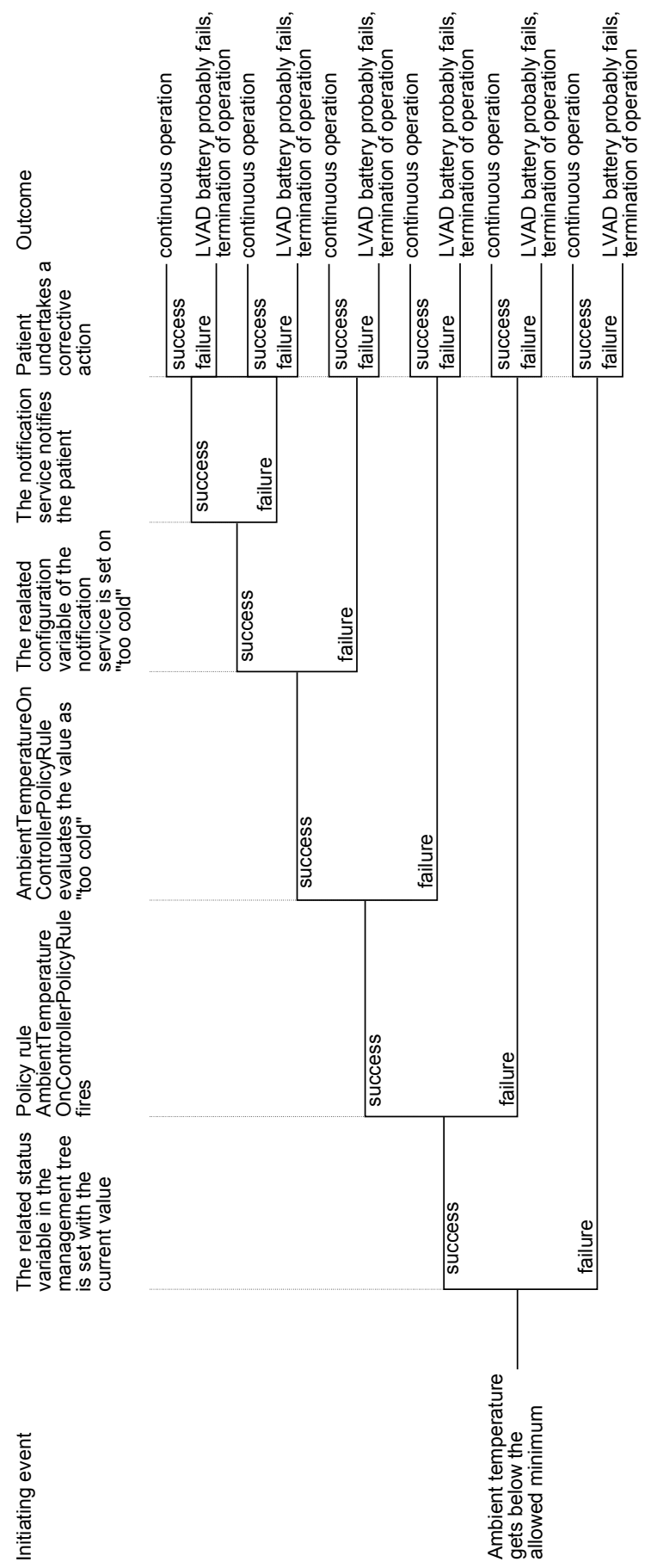

Figure 11.2: Event Tree Analysis: Ambient Temperature Constraint ("too cold")

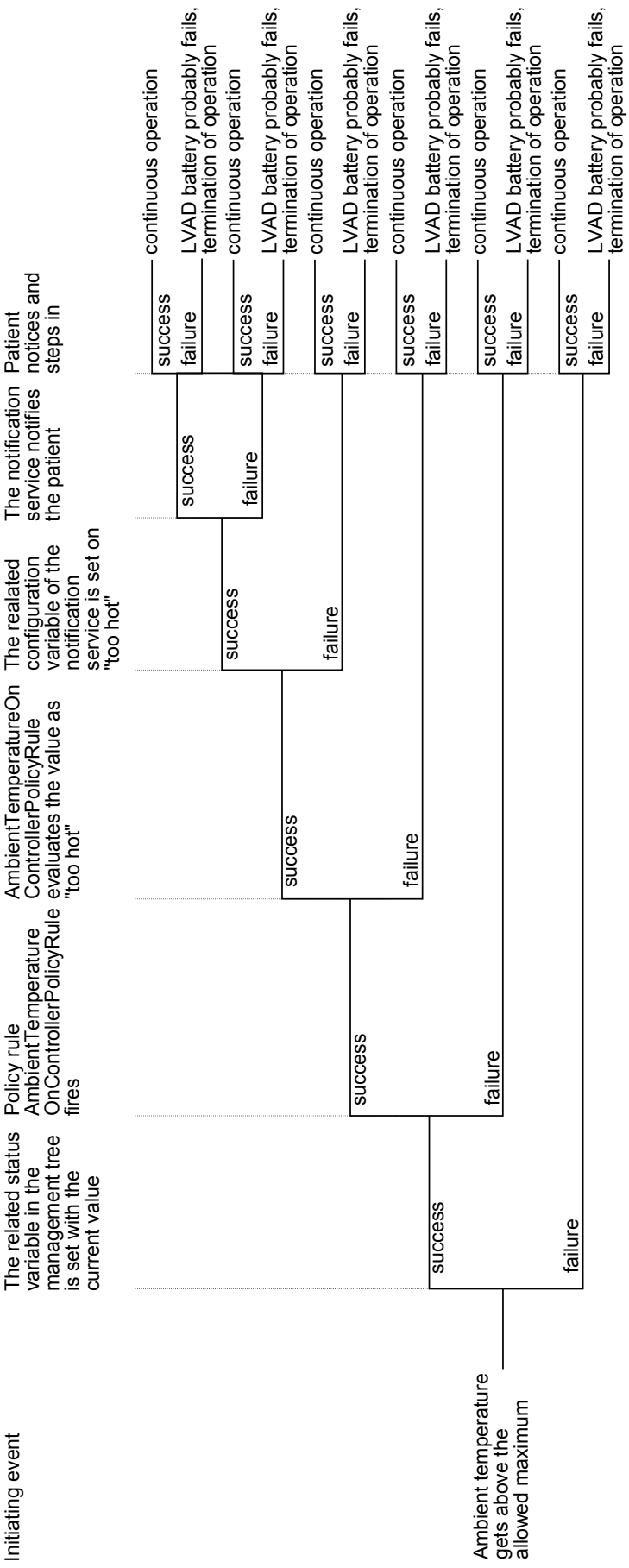

Figure 11.3: Event Tree Analysis: Ambient Temperature Constraint ("too hot")

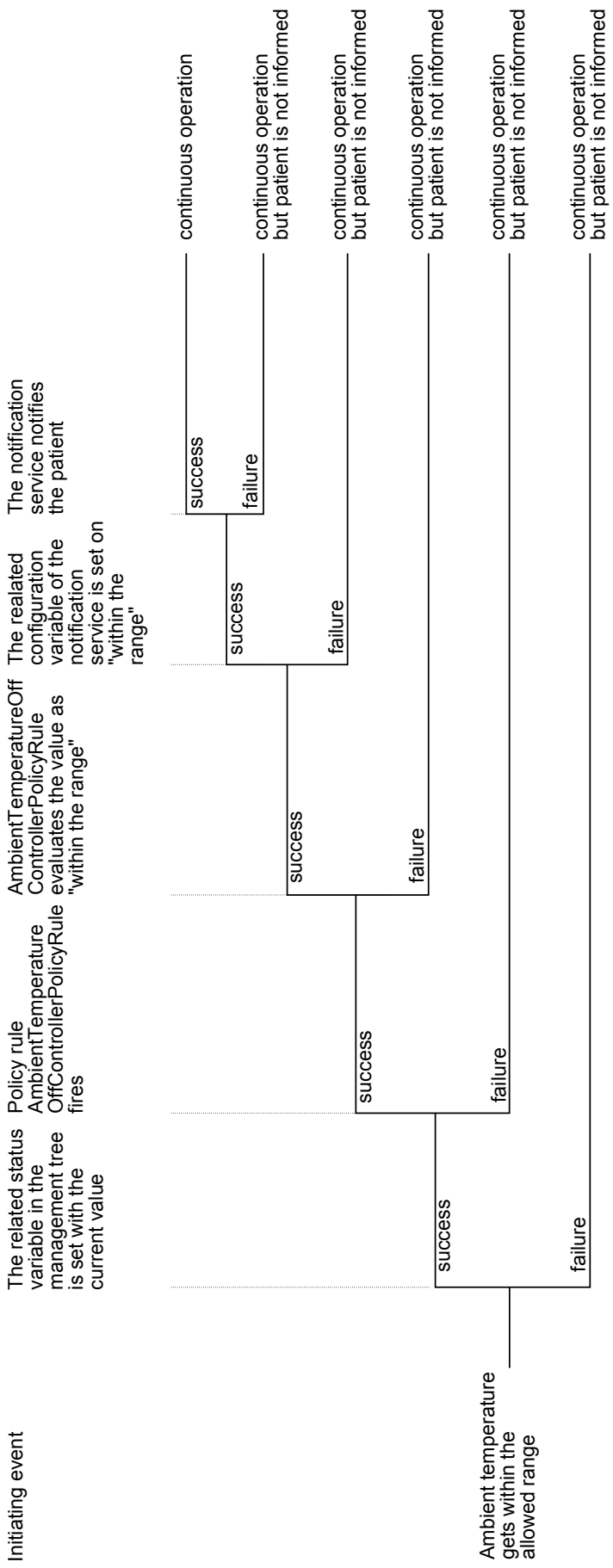

Figure 11.4: Event Tree Analysis: Ambient Temperature Constraint ("within the range")

Exceptions

- The temperature service has not exposed the status by means of status variables correctly
- The management has not recognized the dangerous temperature
- The management has not set the configuration variable of the notification service correctly
- The notification service has produced no alarm
- The patient has not noticed the alarm and/or undertaken a corrective action.

Figures 11.2-11.4 show the event tree analysis performed for the ambient temperature constraint. The violation and the satisfaction of the constraint is caused primarily as the temperature crosses the specified allowed range. Thus, three initiating events can be identified: the temperature gets below the allowed minimum, the temperature gets above the allowed maximum, and the temperature gets within the allowed range again.

In any case, the related status variable in the management tree is set with the corresponding value. If this action is to succeed, a corresponding policy rule fires: the *AmbientTemperatureOnControllerPolicyRule*, in case the temperature is out of the allowed range, or the *AmbientTemperatureOffControllerPolicyRule*, if the temperature returns back to the allowed range. The fired policy evaluates the value of the status variable and interprets the ambient temperature according to the calculation rule as "too cold", "too hot" or "within the allowed range" accordingly. Afterward, the configuration variable of the notification service in the management tree is set to the corresponding value. The notification service is reconfigured and notifies the patient about the ambient temperature status.

The patient notices the notification and can perform a corresponding action, e.g. wrap the LVAD battery or resort to a place with a more appropriate ambient temperature. In case that any of the management functions fails, there still exists a probability, that the patient acts on his own initiative without the interference of the management system.

On the contrary, in the worst case, the exposure to the extreme temperatures can cause an unreliable behavior of the LVAD batteries which can suddenly fail. If the notification of the patient about the return of the temperature to the allowed range fails, it can only cause that the patient stays uninformed. The function of the LVAD battery is not impaired or affected.

11.2.2 Policy Expressions

The demonstration scenario has included 2 policy expressions to be evaluated by the management system during the runtime. One of them was the *ambient environment expression*.

Ambient Environment Expression

It is typical for the LVAD-supported patients, that certain activities (e.g. physical activity) should be conducted within an appropriate ambient environment, otherwise a failure or termination on the part of monitoring application are very likely due to physiological strain. Thus, before entering some critical code blocks, the application can explicitly request the management to evaluate the ambient environment expression, which is calculated on the basis of the heat index in the demonstration scenario. Based upon the evaluation, the

application can decide whether to enter the critical block or not.

Goals

- Critical functions, like e.g. physical activity, are to be safely omitted, in case a failure or termination are very likely due to the physiological strain of the LVAD-supported patient caused by the ambient heat index
- Reduction in occurrence of terminations or failures during the execution of critical functions
- Unnecessary omissions of critical functions are to be avoided

Hazards

- A failure or termination occurs during performing a critical function, like e.g. physical activity

Before performing an event tree analysis, the exceptions which can interrupt the correct course of action are to be defined.

Exceptions

- The temperature service and/or the humidity service have not exposed their status by means of status variables correctly
- The management has failed to read the status variables correctly
- The management has not evaluated the heat index correctly
- The application fails to interpret the expression value correctly

Figures 11.5-11.7 show the event tree analysis performed for the evaluation of the ambient environment expression. Assume, the ambient environment can be evaluated as "safe", "unfavorable" and "hazardous". We regard three initiating events: before entering a critical block, the application requests the evaluation of the policy expression as the condition is "safe", "unfavorable" and "hazardous".

The application request initiates, that the policy expression *AmbientEnvironmentPolicy-Expression* fires. If the action succeeds, the values of the corresponding status variables of the temperature and the humidity services are read from the management tree. The fired policy evaluates the values and interprets the ambient environment according to the calculation rule of the heat index as "safe", "unfavorable" or "hazardous". The value is returned to the requesting application. The reaction of the application on the evaluated request is up to the application itself: it can enter the critical block or omit it.

In case the condition is "safe" and the application enters the critical block, the operation will continue normally, avoiding the critical block would be unnecessary. In case the condition is "unfavorable" and the application enters the critical block, the operation will possibly terminate or fail. If it does not enter the critical block, it is safely omitted. In case the condition is "hazardous", entering the critical block will most probably cause termination or failure. If not, the critical block will be safely omitted.

Thus, the decision about entering critical blocks is still up to the application. The management, however, helps to gain additional knowledge, which allows to avoid escaping critical blocks unnecessarily or entering them, if they possibly or most probably will be terminated anyway. That means the risk of a failure or termination during the runtime is reduced.

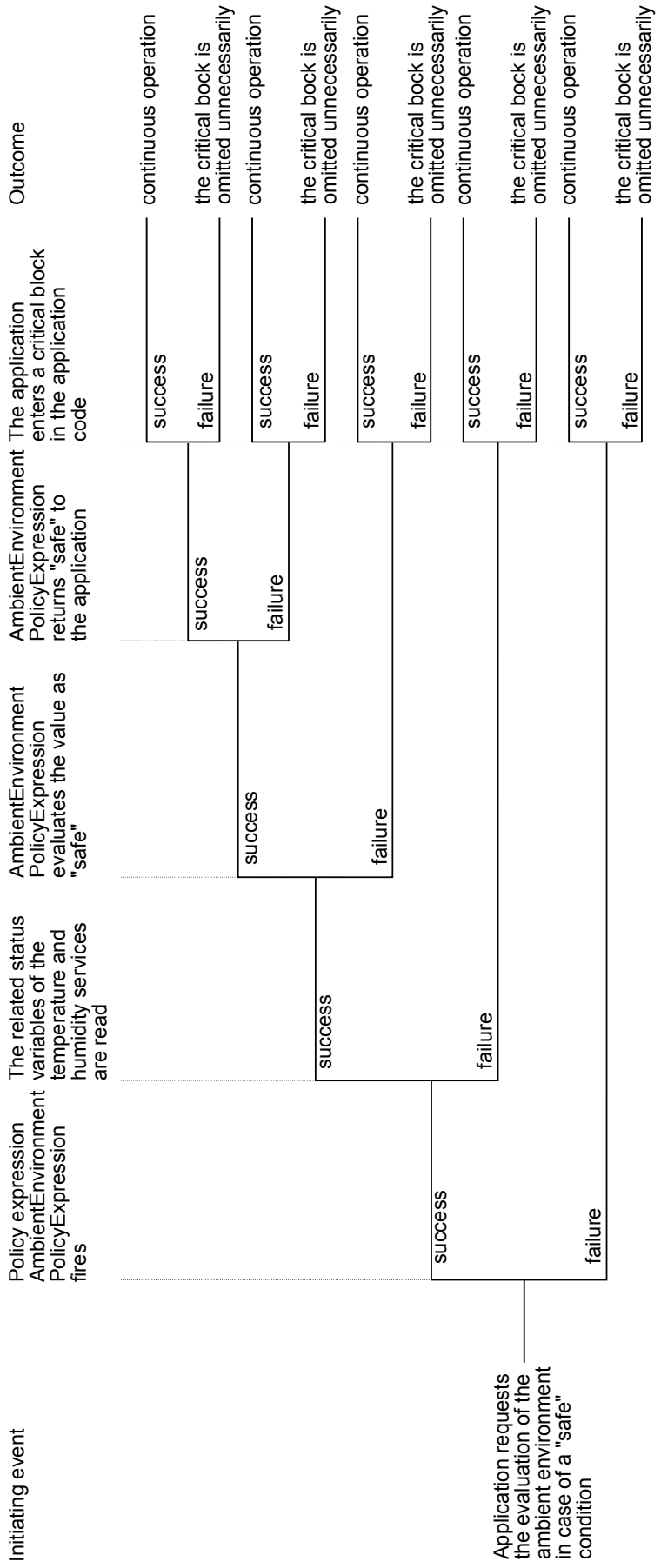

Figure 11.5: Event Tree Analysis: Ambient Environment Evaluation ("safe condition")

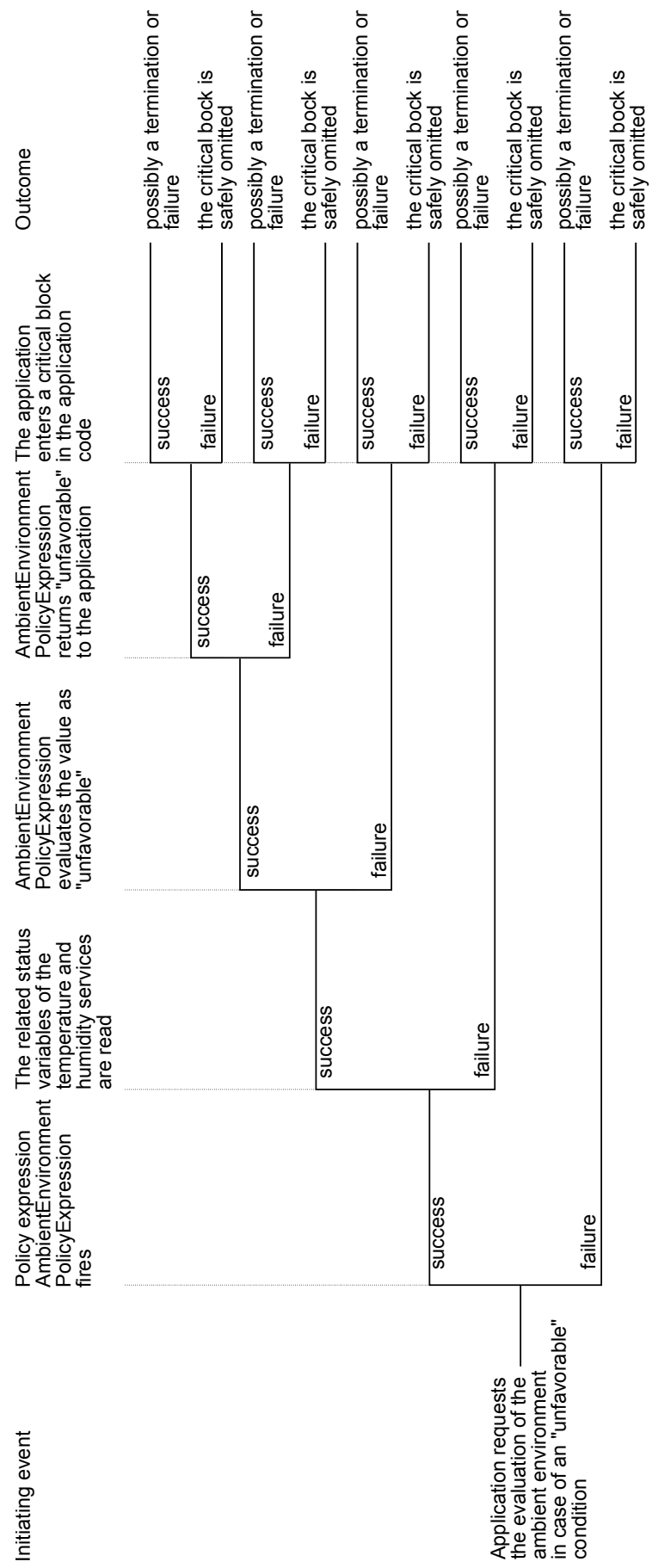

Figure 11.6: Event Tree Analysis: Ambient Environment Evaluation ("unfavorable condition")

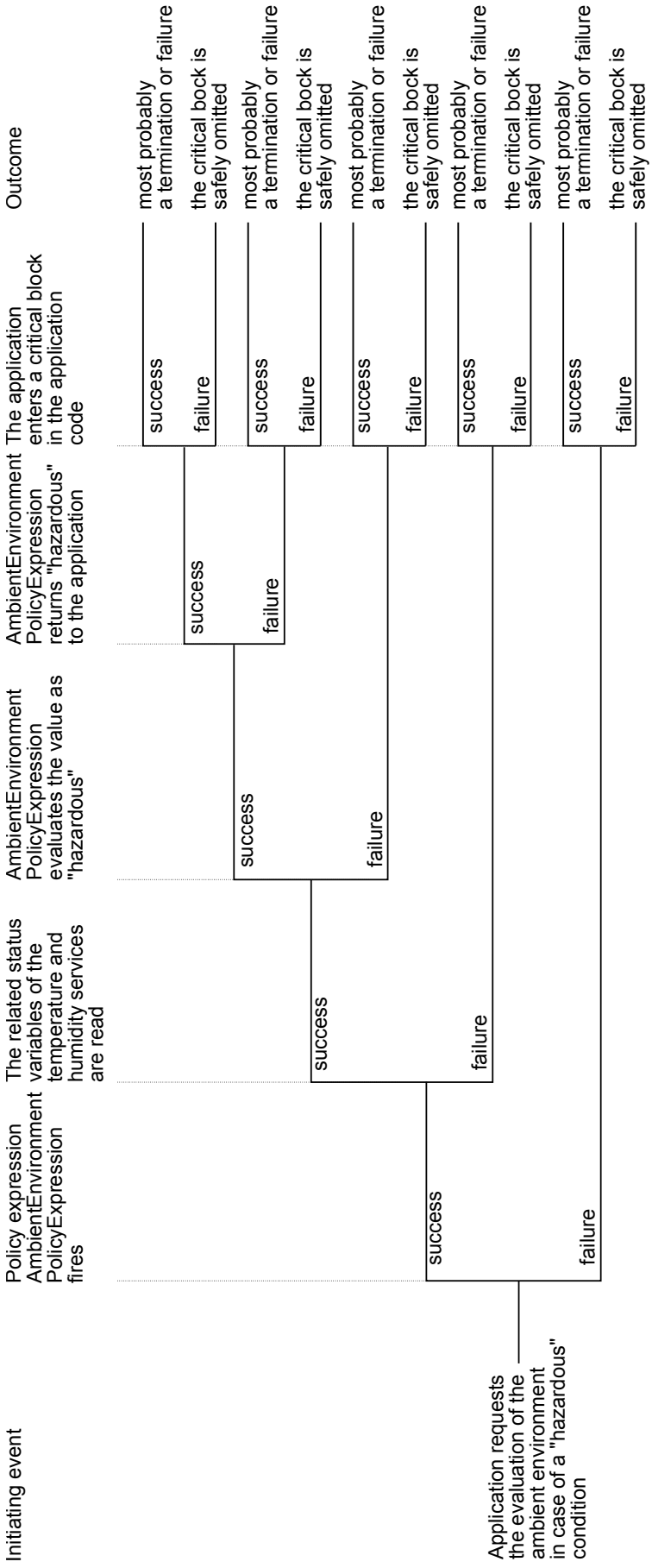

Figure 11.7: Event Tree Analysis: Ambient Environment Evaluation ("hazardous condition")

Evaluation

By means of ETA we have shown that the use of the technical management allows to reduce the occurrence of a sudden LVAD battery performance decrease due to unfavorable ambient temperature. A user's corresponding action is, however, still needed. It is to mention, that the exposure to extremely hot or cold temperatures is unlikely (but not impossible) for the LVAD batteries carried on the body, but is possible for the spare batteries which are compulsory to be carried along and must be fully operational at all times. The situation with an unexpected LVAD battery fault is critical for the patient. That is why the level of risk is to be evaluated as high.

Thus, the gain on the reliability of the system (as defined by the mean time to failure [Els12]) due to the use of management in case of operation in an unfavorable ambient environment is obvious. The avoidance of a sudden LVAD battery failure shows gain also on the availability of the system (as defined by the ratio of uptime to total time [Els12]) since the battery replacement takes time.

It has also been shown, that the ability of the application to request an evaluation of the ambient environment policy expression from the management allows to reduce the occurrence of terminations during execution of critical blocks in the application code.

Provided that the application handles the result of the evaluated policy expression in an appropriate way (i.e. it omits critical blocks in the code if the ambient environment is evaluated as hazardous), a probable termination or failure during execution of a critical block is avoided. The probability of a termination or failure during the execution of the critical block stays the same if the ambient condition is evaluated as safe or unfavorable and the application enters the critical block in the application code. Thus, the overall number of terminations during execution of critical blocks in the application code is reduced.

A termination or failure during the execution of a critical block in the application code has a very high level of risk. Thus, a safe omission of the block in case of a probable failure contributes to the overall system safety.

We assume, that the technical management will contribute to the dependable behavior of the system in the same manner in the other application use cases comparable to this one.

Chapter 12

Conclusion

This work presents an advanced form of system management which combines the innovative model-based management technique with the established approach of policy-based management. The presented approach is applied within the medical application field. It has been shown that the approach supports the development and dependable behavior of medical devices and systems.

The proposed approach applies an explicit separation of the design and runtime phases of the management process. During the design phase the management and managed system are extensively planned. The requirements to the concrete managed system are defined and subsequently the required management artifacts and configurations are derived. These are used by the management system and govern the choices in its behavior during the runtime phase.

The planning of the system during the design phase means creating a comprehensive model of the management system and the managed one. The systems are modeled on three layers, each of them varying in degree of abstraction and reflecting a different view on the systems: from the abstract to the technical one. The top layer *"Use Cases"* provides the most abstract view of the system. It reflects the main application-oriented purpose of the system and expresses the main stakeholders' goals. The desired system behavior is expressed by means of concise requirements addressing various facets (e.g., performance, security, billing, application domain). The requirements are formulated at a high level and determine the target system state. They can also express necessities or prerequisites of the system state. The middle layer *"Services"* provides the service-oriented view of the system. On this layer the system is defined in terms of reusable autonomous loosely coupled services as well as applications providing and using them. The service-level requirements are introduced which define the usage, provision and deployment of services, applications and related data. In contrast to the requirements of the *"Use Cases"* layer, however, these requirements can be specified not only during the design time, but rather generated and added to the model automatically during the refinement process. The most concrete technical view of the system is represented on the bottom layer of the system model. Thus, the *"Components"* layer comprises the hardware and software components composing the runtime system and delivering to the user the tangible services and applications defined on the middle layer. The requirements of the higher level are translated to the concrete technical representations of the bottom layer. In order to assemble the whole system model, a kind of "glueing" elements is needed to indicate how the more abstract elements and their attributes are to be translated into the more technical ones. For this purpose, *refinement relations* are introduced, which comprise primarily top-down inter-layer associations between the model elements of the adjacent layers.

The proposed general three-layered metamodel structure has been specialized for the medical application domain. The presented concretized metamodel integrates the domain knowledge and allows to include the domain-specific constraints and requirements into the modeling process. The metamodel follows the generalization principle and uses the

inheritance relationships in order to form the elements hierarchy and allows extensions if necessary. A system model for the concrete application use case is constructed as an instance of the metamodel. Each of the model elements is instantiated from the corresponding metamodel element and has a predefined set of *management variables*. The *status variables* are used to express the state of the model element and the *configuration variables* are used in order to configure the model element.

At runtime the management system reads the status variables and sets the configuration of the components. To govern the choices in the behavior of the management system special management artifacts are used. We refer to them as management *policies* and recognize two types of them: *policy rules* and *policy expressions*. The policy rules follow the *event-condition-action* paradigm and are implemented technically by means of reading the status and setting the configuration variables on certain events. The policy expressions are formulated on status variables. Their evaluation is requested by a managed component itself from the management on demand. Due to policy expressions, a special form of the policy-based resource control level occurs. Policy expressions allow taking the evaluation logic out of the application, but still guarantee that the evaluation is initiated on demand by the managed system. The management system is indeed responsible for the evaluation but still the interpretation is up to the requester. Thus, the policies applied by the automated management system at runtime provide a reusable powerful instrument which forces the managed system to conduct in a flexible but predefined manner.

The management policies as well as initial assignments of the configuration variables are derived from the system model during the refinement process. In order to support this process, a set of *derivation patterns* has been proposed. The derivation patterns are model patterns defined on certain types of model elements. The patterns can be parametrized by the system developer, so that he can adjust the refinement process to the current use case and its specific requirements and conditions. According to the purpose and architectural structure three basic types of derivation patterns have been distinguished: *evaluation*, *control*, and *refinement patterns*.

Evaluation patterns are intra-layer patterns which are used to support the definition of abstract status variables within a single model layer. Such a construct can combine inputs of multiple status variables and allows to evaluate them by means of a calculation rule. In doing so, an evaluation pattern comprises a function which relates a set of inputs with a set of permitted outputs. Based on the underlying function we distinguish the following types of evaluation patterns: *aggregation*, *attribution*, and *fuzzy relation patterns*.

The notion of *control patterns* is to specify the target management control loop of the system. They define the dynamic behavior of control elements by mapping abstract declarative objectives of a higher layer to imperative enforcement mechanisms on the next lower layer. Thus, in contrast to evaluation patterns, control pattern express rather a "how" strategy for implementing the management solution. In the work, we have introduced four forms of control patterns inspired by the common techniques of control engineering: *watchdog timer*, *heartbeat*, *fuzzy logic control*, *on-off controller*, and *multiplexer patterns*.

The purpose of *refinement patterns* is to support the policy derivation process by specifying how the values of abstract elements are to be propagated downwards to the more detailed values. Together with the inter-layer refinement relations they direct the refinement from top to bottom. In contrast to refinement relations, they operate on management variables directly. Thus, they map values of management variables of adjacent layers and provide calculation rules used within the policy refinement process. Based on the underlying refinement function, we distinguish the following refinement pattern types: *repeater*, *translator*, and *data selector patterns*.

The derivation patterns and the refinement relations are the basis of the policy refinement process. It is iterated through the model elements from the top layer to the bottom layer along the refinement relations, the found patterns are applied and at the end the management policies are generated. The policies derived during the refinement process are enforced by the management system at runtime.

Within the MEDOLUTION project we have applied the proposed approach to the technical management of a medical application use case. The application use case has included monitoring, controlling and adaptation of treatment of LVAD-supported patients. The development of the technical management for the application use case as well as the operation of the management system have been measured and evaluated concerning the time behavior and operability. The measurements and evaluations have been done on the basis of the GQM approach. The time and effort spent during the planning phase and the time behavior of the management system at runtime have been evaluated as appropriate. The overhead, which the usage of the management system causes, has been evaluated as minor. The process of modeling and policy derivation and the operation of the management system (deployment, (re-)configuration, etc.) at runtime has been evaluated as comfortable for a qualified professional. We assume, that the same performance is to be expected for other comparable application use cases conducted under similar conditions.

It has been shown that the dependability (in particular, availability, reliability and safety) of the system have been improved in the application use case due to the usage of the technical management. Specifically, the occurrence of predictable and/or avoidable system failures or breakdowns, emergency stops on the part of the application and omissions of critical blocks or functions caused by unfavorable ambient conditions, deficient and/or defective operating resources, and system state have been looked at. During the evaluation process, we have resorted to ETA as a basic technique, performing a bottom up forward error analysis. We assume, that the proposed approach to the technical management will contribute to the dependable behavior of the medical devices and systems in the same manner in comparable application use cases.

The introduced approach can be also applied to the future cyber-physical systems which bring together multiple application domains. Such systems require advanced automated management solutions due to their complexity caused by involving interdisciplinary approaches and deeply intertwined applications and consumer devices.

Bibliography

[AKP+11] S. Andreassen, D. Karbing, U. Pielmeier, S. Rees, A. Zalounina, Line Sanden, M. Paul, and L. Leibovici. Model-Based Medical Decision Support – A Road to Improved Diagnosis and Treatment? In *15th Nordic-Baltic Conference on Biomedical Engineering and Medical Physics*, volume 34, pages 257–260, June 2011.

[Art80] Zvi Artstein. Disrcete and Continuous Bang-Bang and Facial Spaces or: Look for the Extreme Points. *SIAM J. REVIEW*, 22(2):172–185, 1980.

[ASB+06] I. Aib, M. Salle, C. Bartolini, A. Boulmakoul, R. Boutaba, and G. Pujolle. Business Aware Policy-Based Management. In *1st IEEE/IFIP International Workshop on Business-Driven IT Management*, pages 55–62. IEEE, April 2006.

[Asi14] Asian Harmonization Working Party, Work Group 1, Pre-Market Submission and CSDT. White Paper on Medical Device Software Regulation – Software Qualification and Classification, 2014.

[AXE11] AXELOS Limited. ITIL® Glossary and Abbreviations. Glossary of Terms English v.1.0, 2011.

[BA07] Raouf Boutaba and Issam Aib. Policy-based Management: A Historical Perspective. *Journal of Network and Systems Management*, 15(4):447–480, 2007.

[Bäc13] M. Bächle. *Qualitätsmanagement der Softwareentwicklung: Das QEG-Verfahren als Instrument des Total Quality Managements.* Deutscher Universitätsverlag, 2013.

[Bas92] Victor R. Basili. Software Modeling and Measurement: The Goal/Question/Metric Paradigm. Technical Report UMIACS TR-92-96, University of Maryland at College Park, MD, USA, 1992.

[BBL76] B. W. Boehm, J. R. Brown, and M. Lipow. Quantitative Evaluation of Software Quality. In *Proceedings of the 2nd International Conference on Software engineering, ICSE '76*, pages 592–605, 1976.

[BE08] H. Balzert and C. Ebert. *Lehrbuch der Softwaretechnik: Softwaremanagement.* Spektrum Akademischer Verlag, 2008.

[BE13] Hannes; Burwitz, Martin; Schlieter and Werner Esswein. Modeling Clinical Pathways - Design and Application of a Domain-Specific Modeling Language. In *Wirtschaftsinformatik Proceedings 2013*, 2013.

[BFL+13] A. Brinkmann, C. Fiehe, A. Litvina, I. Lück, L. Nagel, K. Narayanan, F. Ostermair, and W. Thronicke. Scalable Monitoring System for Clouds. In *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 351–356, Dec 2013.

[BG04] Dan Brickley and Ramanathan V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. *W3C Recommendation*, 10, 2004.

[BGFV11] Ayan Banerjee, Sandeep K. S. Gupta, Georgios Fainekos, and Georgios Varsamopoulos. Towards Modeling and Analysis of Cyber-physical Medical Systems. In *Proceedings of the 4th International Symposium on Applied Sciences in Biomedical and Communication Technologies*, ISABEL '11, pages 154:1–154:5, New York, NY, USA, 2011. ACM.

[BH02] Walter Banks and Gordon Hayward. Fuzzy Logic in Embedded Microcomputers and Control Systems. Technical report, BYTE CRAFT LIMITED, 2002.

[BKF15] M. Burkert, H. Krumm, and C. Fiehe. Technical Management System for Dependable Building Automation Systems. In *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8, Sept 2015.

[BLMR04] Arosha K. Bandara, Emil C. Lupu, Jonathan Moffett, and Alessandra Russo. A Goal-based Approach to Policy Refinement. In *Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*, POLICY04, Washington, DC, USA, 2004. IEEE Computer Society.

[BLR03] A. K. Bandara, E. C. Lupu, and A. Russo. Using Event Calculus to Formalise Policy Specification and Analysis. In *4th IEEE Workshop on Policies for Networks and Distributed Systems (Policy 2003)*, Lake Como, Italy, 2003.

[BSE12] Martin Burwitz, Hannes Schlieter, and Werner Esswein. Agility in Medical Treatment Processes – A Model-Based Approach. In Elmar J. Sinz and Andy Schürr, editors, *Modellierung*, volume 201 of *LNI*, pages 267–279. GI, 2012.

[Béz05] Jean Bézivin. On the Unification Power of Models. *Software and System Modeling*, 4(2):171–188, 2005.

[CCSZ03] Jiannong Cao, Alvin Chan, Yudong Sun, and Kang Zhang. Dynamic Configuration Management in a Graph-Oriented Distributed Programming Environment. *Science of Computer Programming*, 48(1):43–65, 2003.

[CL01] Larry L. Constantine and Lucy Lockwood. Object Modeling and User Interface Design. In Mark Van Harmelen, editor, *Structure and Style in Use Cases for User Interface Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, April 2001.

[CL02] I. Crnkovic and M.P.H. Larsson. *Building Reliable Component-based Software Systems*. Artech House Computing Library. Artech House, 2002.

[Cla82] D.D. Clark. Fault Isolation and Recovery. RFC 816, July 1982.

[Cla89] D.D. Clark. Policy Routing in Internet Protocols. RFC 1102, May 1989.

[CMK05] S. Chakravorty, C. Mendes, and L. Kale. Proactive Fault Tolerance in Large Systems. In *Proceedings of HPCRI Workshop*, 2005.

[Coc00] Alistair Cockburn. *Writing Effective Use Cases – Crystal Series for Software Development*. Addison-Wesley Longman, November 2000.

[Coh07] Shy Cohen. Ontology and Taxonomy of Services in a Service-Oriented Architecture. *Microsoft Architect Journal*, 2007.

[Com] Computing Science and Mathematics, University of Stirling. Adaptable and Programmable Policy Environment and Language (APPEL). https://accentsuite.sourceforge.io/. Accessed September 2018.

[CPW+01] David S. Channin, Charles Parisot, Vishal Wanchoo, Andrei Leontiev, and Eliot L. Siegel. Integrating the Healthcare Enterprise: A Primer Part 3. What Does IHE Do for ME? *RadioGraphics*, 21:1351–1358, 2001.

[CSVC11] I. Crnkovic, S. Sentilles, A. Vulgarakis, and M. R. V. Chaudron. A Classification Framework for Software Component Models. *IEEE Transactions on Software Engineering*, 37(5):593–615, 2011.

[dAIKdG05] Joao Porto de Albuquerque, Holger Isenberg, Heiko Krumm, and Paulo Licio de Geus. Improving the Configuration Management of Large Network Security Systems. In *Proceedings of the 16th IFIP/IEEE Ambient Networks International Conference on Distributed Systems: Operations and Management*, DSOM'05, pages 36–47, Berlin, Heidelberg, 2005. Springer-Verlag.

[dAKdG05a] J. P. de Albuquerque, H. Krumm, and P. L. de Geus. Policy Modeling and Refinement for Network Security Systems. In *6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'05)*, pages 24–33, June 2005.

[dAKdG05b] Joao Porto de Albuquerque, Heiko Krumm, and Paulo Licio de Geus. On Scalability and Modularisation in the Modelling of Network Security Systems. In Sabrina de Capitani di Vimercati, Paul Syverson, and Dieter Gollmann, editors, *Computer Security – ESORICS 2005. 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005. Proceedings*, pages 287–304, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[Dam02] N. Damianou. *A Policy Framework for Management of Distributed Systems.* PhD Thesis, 2002. Department of Computing, Imperial College.

[DBC+00] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry. The COPS (Common Open Policy Service) Protocol. RFC 2748 (Proposed Standard), January 2000. Updated by RFC 4261.

[DDLS00] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. Ponder: A Language for Specifying Security and Management Policies for Distributed Systems The Language Specification Version 2.3. Imperial College Research Report DoC 2000/1, Imperial College of Science, Technology and Medicine, Department of Computing, 180 Queen's Gate, London SW7 2BZ, U.K., October 2000.

[DDLS01] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In *Proceedings of the Policy Workshop 2001*, pages 29–31. HP Labs, Bristol, UK, Springer-Verlag, 2001.

[DHTM06] Keith J. Dreyer, David S. Hirschorn, James H. Thrall, and Amit Mehta, editors. *PACS. A Guide to the Digital Revolution.* Springer-Verlag New York, 2 edition, 2006.

[DIM13] DIMDI - Deutsches Institut für Medizinische Dokumentation und Information. Aktualisierungsliste zur Vorabversion OPS 2014. http://www.dimdi.de/dynamic/de/klassi/downloadcenter/ops, July 2013.

[DJS07] Steven Davy, Brendan Jennings, and John Strassner. The Policy Continuum - A Formal Model. In *Proceedings of the 2nd IEEE International Workshop on Modelling Autonomic Communications Environments, MACE*, volume 6 of *Mulicon*, pages 65–79, 2007.

[DKK+10a] O. Dohndorf, J. Krüger, H. Krumm, C. Fiehe, A. Litvina, I. Lück, and F. J. Stewing. Lightweight Policy-Based Management of Quality-Assured, Device-Based Service Systems. In *Proceedings of 24th IEEE International Conference on Advanced Information Networking and Applications Workshops*, pages 526–531, April 2010.

[DKK+10b] O. Dohndorf, J. Krüger, H. Krumm, C. Fiehe, A. Litvina, I. Lück, and F. J. Stewing. Policy-Based Management for Resource-Constrained Devices and Systems. In *Proceedings of the IEEE International Symposium on Policies for Distributed Systems and Networks*, pages 61–64, July 2010.

[DKK+11a] O. Dohndorf, J. Krüger, H. Krumm, C. Fiehe, A. Litvina, I. Lück, and F. J. Stewing. Adaptive and Reliable Binding in Ambient Service Systems. In *Proceedings of the 16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, 2011.

[DKK+11b] O. Dohndorf, J. Krüger, H. Krumm, C. Fiehe, A. Litvina, I. Lück, and F. J. Stewing. Tool-Supported Refinement of High-Level Requirements and Constraints Into Low-Level Policies. In *Proceedings of the IEEE International Symposium on Policies for Distributed Systems and Networks*, pages 97–104, June 2011.

[DLSD01] Naranker Dulay, Emil C. Lupu, Morris Sloman, and Nicodemos Damianou. A Policy Deployment Model for the Ponder Language. In *Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management (IM'2001)*, 2001.

[DMT12a] DMTF. Common Information Model (CIM) Infrastructure. Specification, DMTF Standard, Version 2.8.0. http://www.dmtf.org/standards/cim/, April 2012.

[DMT12b] DMTF. Common Information Model (CIM) Metamodel. Specification, DMTF Standard, Version: 3.0.0. http://www.dmtf.org/standards/cim/, December 2012.

[DMT12c] DMTF. Managed Object Format (MOF). Specification, DMTF Standard, Version 3.0.0. http://www.dmtf.org/standards/cim/, December 2012.

[DMT16] DMTF. Common Information Model (CIM) Schema. Specification, DMTF Standard, Version 2.45.0. http://www.dmtf.org/standards/cim/, Januar 2016.

[DRN17] D. Dasgupta, A. Roy, and A. Nag. *Advances in User Authentication*. Infosys Science Foundation Series. Springer International Publishing, 2017.

[Dru92] Colin M. Drury. *Management And Cost Accounting*. Springer US, 3 edition, 1992.

[DRW06] Leticia Duboc, David S. Rosenblum, and Tony Wicks. A Framework for Modelling and Analysis of Software Systems Scalability. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 949–952, New York, NY, USA, 2006. ACM.

[DSNH10] Simon Dobson, Roy Sterritt, Paddy Nixon, and Mike Hinchey. Fulfilling the Vision of Autonomic Computing. *Computer*, 43(1):35–41, January 2010.

[DvL96] R. Darimont and A. van Lamsweerde. Formal Refinement Patterns for Goal-Driven Requirements Elaboration. In *4th ACM Symposium on the Foundations of Software Engineering (FSE4)*, pages 179–190, 1996.

[EE96] Ezekiel J. Emanuel and Linda L. Emanuel. What Is Accountability in Health Care? *Annals of Internal Medicine*, 124(2):229–239, January 1996.

[EKK+04] Tamar Eilam, Michael Kalantar, Er Konstantinou, Giovanni Pacifici, Tamar Eilam, Michael Kalantar, Er Konstantinou, and Giovanni Pacifici. Model-Based Automation of Service Deployment in a Constrained Environment. IBM Research Report, IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704 Yorktown Heights, NY 10598, September 2004.

[Els12] E.A. Elsayed. *Reliability Engineering*. Wiley Series in Systems Engineering and Management. Wiley, 2012.

[Emm00] Wolfgang Emmerich. Software Engineering and Middleware: A Roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 117–129, New York, NY, USA, 2000. ACM.

[Eri05] C.A. Ericson. *Hazard Analysis Techniques for System Safety*. Wiley, 2005.

[Erl07] Thomas Erl. *SOA Principles of Service Design*. Prentice Hall Service-Oriented Computing Series. Prentice Hall International, 2007.

[Eur90] European Parlament; Council of European Union. Directive 90/385/EEC of the European Parlament and of the Council on the Approximation of the Laws of the Member States Relating to Active Implantable Medical Devices, June 1990.

[Eur93] European Parlament; Council of European Union. Council Directive 93/42/EEC of 14 June 1993 Concerning Medical Devices, June 1993.

[Eur98] European Parlament; Council of European Union. Directive 98/79/EC of the European Parlament and of the Council on In Vitro Diagnostic Medical Devices, October 1998.

[Eur07] European Parlament; Council of European Union. Directive 2007/47/EC of the European Parlament and of the Council, June 2007.

[Eur10] European Commission; DG Health and Consumer; Directorate B; Unit B2 Cosmetics and Medical Devices. MEDICAL DEVICES: Guidance Document MEDDEV 2. 4/1 Rev. 9. Guidelines Relating to the Application of the Council Directive 93/42/EEC on Medical Devices, June 2010.

[Eur16] European Commission; DG Internal Market, Industry, Entrepreneurship and SMEs; Directorate Consumer, Environmental and Health Technologies; Unit Health Technology and Cosmetics. MEDICAL DEVICES: Guidance Document - Qualification and Classification of Stand Alone Software (MEDDEV 2.1/6), 2016.

[FJ02] Csilla Farkas and Sushil Jajodia. The Inference Problem: A Survey. *SIGKDD Explorations Newsletter*, 4(2):6–11, December 2002.

[FJRG10] M. Fleury, E. Jamme, R. Razavi, and M. Ghanbari. Resource-Aware Fuzzy Logic Control of Video Streaming over IP and Wireless Networks. In A.E. Hassanien and J.H. Abawajy and A. Abraham and H. Hagras, editor, *Pervasive Computing: Innovations in Intelligent Multimedia and Applications*, Computer Communications and Networks, pages 47–75. Springer-Verlag London Limited, 2010.

[FK92] David Ferraiolo and Richard Kuhn. Role-Based Access Controls. In *Proceedings of the 15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[FLL+09] Christoph Fiehe, Anna Litvina, Ingo Lück, Franz-Josef Stewing, Oliver Dohndorf, Jan Krüger, and Heiko Krumm. Policy-gesteuertes Management adaptiver und gütegesicherter Dienstesysteme im Projekt OSAMI. In *Proceedings 154 - Informatik 2009 Im Focus das Leben*, pages 970–983, Lübeck, Germany, 2009. Gesellschaft für Informatik / Verlag Köellen.

[FUMK03] Howard Foster, S. Uchitel, J. Magee, and J. Kramer. Model-Based Verification of Web Service Compositions. In *Proceedings of the 18th IEEE International Conference on Automated Software Engineering*, pages 152–161, October 2003.

[GHK+01] M. Garschhammer, R. Hauck, B. Kempter, I. Radisic, H. Roelle, and H. Schmidt. The MNM Service Model - Refined Views on Generic Service Management. *Journal of Communications and Networks*, 3(4):297–306, December 2001.

[Gil88] Tom Gilb. *Principles of Software Engineering Management*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988.

[GKS+85] K. Görgen, H. Koch, G. Schulze, B. Struif, and K. Truöl. *Grundlagen der Komminikationstechnologie: ISO Architektur offener Kommunikationssysteme*. Springer Verlag, Berlin Heidelberg New York Tokyo, 1985.

[HAN98] Heinz-Gerd Hegering, Sebastian Abeck, and Bernhard Neumair. *Integrated Management of Networked Systems: Concepts, Architectures, and Their Operational Application*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[HF04] W. Heaven and A. Finkelstein. UML Profile to Support Requirements Engineering with KAOS. *IEE Proceedings - Software*, 151(1):10–27, February 2004.

[Hil90] Mark D. Hill. What is Scalability? *SIGARCH Computer Architecture News*, 18(4):18–21, December 1990.

[HM08] Markus C. Huebscher and Julie A. McCann. A Survey of Autonomic Computing: Degrees, Models, and Applications. *ACM Computing Surveys*, 40(3):7:1–7:28, August 2008.

[HR08] Hardi Hungar and Erwin Reyzl. Software-Entwicklung und Zertifizierung im Umfeld sicherheitskritischer und hochverfügbarer Systeme: Bedeutung modellbasierter und formaler Ansätze für effiziente Entwicklung und Zertifizierung. In *Software Engineering (Workshops)*, volume 122 of *LNI*, pages 299–302. GI, 2008.

[HS05] A. Hosagrahara and P. Smith. Measuring Productivity and Quality in Model-Based Design. Technical report, The MathWorks, Inc., 2005.

[HWAB04] Reinhold Haux, Alfred Winter, Elske Ammenwerth, and Birgit Brigl. *Strategic Information Management in Hospitals*. Springer-Verlag New York, 1 edition, 2004.

[IBM05] IBM. IBM Paves the Way for Mainstream Adoption of Autonomic Computing. Market Wired, April 2005.

[Ibr03] Ahmad Ibrahim. *Fuzzy Logic for Embedded Systems Applications*. Newnes, 2003.

[IEE00] IEEE Architecture Working Group. IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-intensive Systems. Technical report, IEEE, 2000.

[IKP+05] Stefan Illner, Heiko Krumm, Andre Pohl, Ingo Lück, Darius Manka, and Thomas Sparenberg. Policy Controlled Automated Management of Distributed and Embedded Service Systems. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks*, pages 710–715, Innsbruck, Austria, February 2005. ACTA Press.

[Inf04] Information Technology Industry Council. American National Standard for Information Technology – Role Based Access Control. Technical report, American National Standards Institute, Inc., February 2004. ANSI® INCITS 359-2004.

[Int92] International Telegraph and Telephone Consultative Committee (CCITT). Recommendation X.700 (09/92): Management Framework for Open Systems Interconnection (OSI) for CCITT Applications. Technical report, International Telecommunication Union, 1992.

[Int15] International Medical Device Regulators Forum (IMDRF), SaMD Working Group. Software as a Medical Device (SaMD): Application of Quality Management System (IMDRF/SaMD WG/N23), 2015.

[IPK+06] Stefan Illner, Andre Pohl, Heiko Krumm, Ingo Lück, Andreas Bobek, Hendrik Bohn, and Frank Golatowski. Model-based Management of Embedded Service Systems – An Applied Approach. In *Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA 2006)*, pages 519–523, Vienna, Austria, 2006.

[Ise04] Rolf Isermann. Model-Based Fault Detection and Diagnosis: Status and Applications. In *Proceedings of the 16th IFAC Symposium on Automatic Control in Aerospace*, pages 71–85, 2004.

[Ise06] Rolf Isermann. *Fault-Diagnosis Systems : An Introduction from Fault Detection to Fault Tolerance*. Springer, Berlin, Germany, 2006.

[ISO93] ISO. ISO/IEC 10165–1:1993 Information Technology – Open Systems Interconnection – Management Information Services – Structure of Management Information: Management Information Model. Technical report, ISO/IEC JTC 1 - Information Technology, September 1993.

[ISO95] ISO. IISO 9241-110:2006, Ergonomics of Human-System Interaction – Part 110: Dialogue Principles. Technical report, ISO/TC 159/SC 4 Ergonomics of Human-System Interaction, 1995.

[ISO98] ISO. ISO 9241-11:1998, Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs) – Part 11: Guidance on Usability. Technical report, ISO/TC 159/SC 4 Ergonomics of Human-System Interaction, 1998.

[ISO01] ISO. ISO 9126-1:2001, Software Engineering - Product Quality, Part 1: Quality Model. Technical report, International Organization for Standardization, 2001.

[ISO03] ISO. ISO 13485:2003 Medical Devices – Quality Managemen Systems – Requirements for Regulatory Purposes. Technical report, ISO/TC 210 Quality Management and Corresponding General Aspects for Medical Devices, July 2003.

[ISO07] ISO. ISO 14971:2007 Medical Devices – Application of Risk Management to Medical Devices. Technical report, ISO/TC 210 Quality Management and Corresponding General Aspects for Medical Devices, 2007.

[ISO10] ISO. ISO/IEC/IEEE 24765:2010(E) Systems and Software Engineering – Vocabulary. Technical report, International Organization for Standardization, 2010.

[ITU08] ITU-T Study Group 12. Recommendation ITU-T E.800 (09/08): Definitions of Terms Related to Quality of Service. Technical report, International Telecommunication Union, September 2008.

[JAW11] Jens H. Weber Jahnke, Anissa Agah, and James Williams. Consumer Health Informatics Services – A Taxonomy. Technical Report UVic/IPIRG-2011-TR-01, University of Victoria. Department of Computer Science, Victoria, BC V8W3P6, March 2011.

[JCJO92] Ivar Jacobson, Magnus Christerson, Patrick Jonsson, and Gunnar Övergaard. *Object-Oriented Software Engineering. A Use Case Driven Approach: A Use Case Approach.* Addison-Wesley Longman, 1992.

[JJPR09] Radha Jagadeesan, Alan Jeffrey, Corin Pitcher, and James Riely. Towards a Theory of Accountability and Audit. In *Proceedings of the 14th European Conference on Research in Computer Security*, ESORICS'09, pages 152–167, Berlin, Heidelberg, 2009. Springer-Verlag.

[Jos08] James Joshi. *Network Security: Know It All: Know It All.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[JRV03] J. Jason, L. Rafalow, and E. Vyncke. IPsec Configuration Policy Information Model. RFC 3585 (Proposed Standard), August 2003.

[JZ93] M. Jackson and P. Zave. Domain Descriptions. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 56–64, January 1993.

[KAC+05] Siheung Kim, Seongjin Ahn, Jinwok Chung, Ilsung Hwang, Sunghe Kim, Minki No, and Seungchung Sin. A Rule Based Approach to Network Fault and Security Diagnosis with Agent Collaboration. In Kim, TagGon, editor, *Artificial Intelligence and Simulation*, volume 3397 of *Lecture Notes in Computer Science*, pages 597–606. Springer Berlin Heidelberg, 2005.

[KB11] Robert M Kaplan and Yair M Babad. Balancing Influence Between Actors in Healthcare Decision Making. Technical report, BMC Health Services Research, 2011.

[KB15] Julian Kalinowski and Lars Braubach. Integrating Application-Oriented Middleware into the Android Operating System. In *UBICOMM 2015, The Eighth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, page 6. IARIA, Xpert Publishing Services, 7 2015.

[KC03] Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, January 2003.

[KDR+06] M. Kessis, P. Dechamboux, C. Roncancio, T. Coupaye, and A. Lefebvre. Towards a Flexible Middleware for Autonomous Integrated Management Applications. In *Proceedings of the International Multi-Conference on Computing in the Global Information Technology, ICCGI '06*, pages 27–27, August 2006.

[KFJ03] Lalana Kagal, Tim Finin, and Anupam Joshi. A Policy Language for a Pervasive Computing Environment. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, POLICY 2003, Washington, DC, USA, 2003. IEEE Computer Society.

[Kir05] Eric Kirchstein. Policy Management for Autonomic Computing: Solving a Business Problem Using PMAC. Technical report, IBM, DeveloperWorks, 2005.

[KM85] J. Kramer and J. Magee. Dynamic Configuration for Distributed Systems. *IEEE Transactions on Software Engineering*, 11(4):424–436, 1985.

[KP97] Stefan Kätker and Martin Paterok. Fault Isolation and Event Correlation for Integrated Fault Management. In Lazar, AurelA. and Saracco, Roberto and Stadler, Rolf, editor, *Integrated Network Management V*, IFIP – The International Federation for Information Processing, pages 583–596. Springer US, 1997.

[KTP+07] S.L. Keoh, K. Twidle, N. Pryce, A.E. Schaeffer-Filho, E. Lupu, N. Dulay, M. Sloman, S. Heeps, S. Strowes, J. Sventek, and E. Katsiri. Policy-based

Management for Body-Sensor Networks. In *Proceedings of the 4th International Workshop on Wearable and Implantable Body Sensor Networks*, pages 92–98. Springer, March 2007.

[LÖ6] Ingo Lück. *Modellbasierte Konfiguration von Sicherheitsdiensten*. PhD Thesis, 2006. Universität Dortmund, Fachbereich Informatik.

[Lam12] Jim Lamberson. Single and Multistage Watchdog Timers. White Paper. Technical report, SENSORAY. Embedded Electronic, 2012.

[LDS+08] Emil Lupu, Naranker Dulay, Morris Sloman, Joe Sventek, Steven Heeps, Stephen Strowes, Kevin Twidle, Sye Loong Keoh, and Alberto Schaeffer-Filho. AMUSE: Autonomic Management of Ubiquitous e-Health Systems. *Concurrency and Computation: Practice and Experience*, 20:277–295, March 2008.

[Lev95] Nancy G. Leveson. *Safeware: System Safety and Computers*. ACM, New York, NY, USA, 1995.

[LR07] Richard Lenz and Manfred Reichert. IT Support for Healthcare Processes - Premises, Challenges, Perspectives. *Data Knowledge Engineering*, 61(1):39–58, April 2007.

[LS99] E. Lupu and M. Sloman. Conflicts in Policy-Based Distributed Systems Management. *IEEE Transactions on Software Engineering*, 25:852–869, 1999.

[Lup98] E. C Lupu. *A Role-Based Framework for Distributed Systems Management*. PhD Thesis, 1998. Department of Computing, Imperial College.

[Mai07] Anil K. Maini. *Digital Electronics: Principles, Devices and Applications*. John Wiley & Sons, Ltd., 1 edition, 2007.

[Mar97] D. A Marriott. *Policy Service for Distributed Systems*. PhD Thesis, 1997. Department of Computing, Imperial College.

[MB01] Niall Murphy and Michael Barr. Watchdog Timers. *Embedded Systems Programming*, pages 79–80, October 2001.

[MC93] M.J. Maullo and Seraphin B. Calo. Policy Management: An Architecture and Approach. In *Proceedings of the IEEE First International Workshop on Systems Management*, pages 13–26, Hawthorne, NY, 1993. IBM Thomas J. Watson Res. Center.

[McB02] B. McBride. Jena: A Semantic Web Toolkit. *IEEE Internet Computing*, 6(6):55–59, November 2002.

[McC77] J. McCall. *Factors in Software Quality: Preliminary Handbook on Software Quality for an Acquisition Manager*, volume 1-3. General Electric, 1977.

[MESW01] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen. Policy Core Information Model – Version 1 Specification. RFC 3060 (Proposed Standard), February 2001. Updated by RFC 3460.

[MFZH99] Jean-Philippe Martin-Flatin, Simon Znaty, and Jean-Pierre Hubaux. A Survey of Distributed Enterprise Network and Systems Management Paradigms. *Journal of Network and Systems Management*, 7, 1999.

[MS93] Jonathan D. Moffett and Morris S. Sloman. Policy Hierarchies for Distributed Systems Management. *IEEE Journal on Selected Areas in Communications*, pages 1404–1414, 1993.

[MvH04] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Reference. W3C Recommendation. Technical report, W3C, February 2004. https://www.w3.org/TR/owl-ref/.

[NP10] Belverd E. Needles and Marian Powers. *Principles of Financial Accounting*. South-Western College Pub., 11 edition, January 2010.

[OAS06] OASIS. Reference Model for Service Oriented Architecture 1.0. OASIS Standard. http://docs.oasis-open.org/soa-rm/v1.0/, October 2006.

[Obj10] Object Management Group. The MDA Foundation Model. OMG Document ormsc/10-09-06l. Technical report, OMG, September 2010.

[Obj11] Object Management Group. OMG Unified Modeling Language (OMG UML), Superstructure. Version 2.4.1. Technical report, OMG, August 2011.

[Obj14a] Object Management Group. Model Driven Architecture (MDA). MDA Guide rev. 2.0. OMG Document ormsc/2014-06-01, OMG, June 2014.

[Obj14b] Object Management Group. Model Driven Architecture (MDA). MDA Guide Version 1.0.1. OMG Document omg/2003-06-01, OMG, June 2014.

[Obj15] Object Management Group. OMG Unified Modeling Language (OMG UML™). Version 2.5. Technical report, OMG, March 2015.

[Ols06] Greg Olsen. From COM to Common. *ACM Queue*, 4(5):20–26, June 2006.

[OMG02] OMG. CORBA Services Specifications: Security Service Specification, Version 1.8. Technical report, Object Management Group, Inc., March 2002. http://www.omg.org/spec/SEC/1.8/.

[Ope07] Open Mobile Alliance. OMA Device Management Tree and Description, Version 1.2. http://www.openmobilealliance.org, February 2007.

[Ope16] Open Mobile Alliance. OMA DM Device Description Framework DTD, Version 1.2. http://www.openmobilealliance.org, May 2016.

[Pat09] Patricia A. Morreale and Kornel Terplan, editor. *CRC Handbook of Modern Telecommunications*. CRC Press, 2 edition, 2009.

[Pav98] George Pavlou. OSI Systems Management, Internet SNMP and ODP/OMG CORBA as Technologies for Telecommunications Network Management. In S. Aidarous and T. Plevyak, editor, *In Telecommunications Network Management: Technologies and Implementations*, pages 63–109. IEEE Press, 1998.

[PHAB12] Klaus Pohl, Harald Hönninger, Reinhold Achatz, and Manfred Broy. *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*. Springer-Verlag Berlin Heidelberg, November 2012.

[PHS+08] T. Phan, J. Han, J. G. Schneider, T. Ebringer, and T. Rogers. A Survey of Policy-Based Management Approaches for Service-Oriented Systems. In *Proceedings of the 19th Australian Conference on Software Engineering (ASWEC 2008)*, pages 392–401, March 2008.

[PN04] Soila Pertet and Priya Narasimhan. Proactive Recovery in Distributed CORBA Applications. In *International Conference on Dependable Systems and Networks*, pages 357–366. IEEE, 2004.

[RBS11a] R. Romeikat, B. Bauer, and H. Sanneck. Automated Refinement of Policies for Network Management. In *Proceedings of the 17th Asia Pacific Conference on Communications*, pages 439–444, October 2011.

[RBS11b] Raphael Romeikat, Bernhard Bauer, and Henning Sanneck. Modeling of Domain-Specific ECA Policies. In *Proceedings of the 23d Intenational Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 52–58. Knowledhe Systems Institute, 2011.

[RC07] Anand Ranganathan and Roy Campbell. What is the Complexity of a Distributed Computing System? *Complexity*, 12:37–45, July 2007.

[RDD07a] Giovanni Russello, Changyu Dong, and Naranker Dulay. Authorization and Conflict Resolution for Hierarchical Domains. In *Policies for Distributed Systems and Networks*, June 2007.

[RDD+07b] Giovanni Russello, Changyu Dong, Naranker Dulay, Jatinder Singh, Jean Bacon, and Ken Moody. A Policy-Based Framework for e-Health Applications. In *Proceedings of the UK e-Science All Hands Meeting 2007 (AHM 07)*, Nottingham, UK, September 2007.

[RJJZ10] A. Ray, R. Jetley, P. Jones, and Y. Zhang. Model-Based Engineering for Medical-Device Software. *Biomed Instrum Technology*, 44(6):507–518, November 2010.

[RMNK02] Alessandra Russo, Rob Miller, Bashar Nuseibeh, and Jeff Kramer. An Abductive Approach for Analysing Event-Based Requirements Specifications. In *Proceedings of the 18th International Conference on Logic Programming (ICLP)*, pages 22–37, Copenhagen, Denmark, August 2002.

[RS88] D.C. Robinson and M.S. Sloman. Domains: A New Approach to Distributed System Management. In *Proceedings of Workshop on the Future Trends of Distributed Computing Systems*, pages 154–163, September 1988.

[RSB09] R. Romeikat, M. Sinsel, and B. Bauer. Transformation of Graphical ECA Policies into Executable PonderTalk Code. In *Proceedings of the 3rd International Symposium on Rule Interchange and Applications (RuleML)*, pages 193–207. Springer LNCS, 2009.

[SB05] William Siler and James J. Buckley. *Fuzzy Expert Systems and Fuzzy Reasoning*. John Wiley & Sons, Inc., Hoboken, New Jersy, 2005.

[SBM07] Jatinder Singh, Jean Bacon, and Ken Moody. Dynamic Trust Domains for Secure, Private, Technology-Assisted Living. *Second International Conference on Availability, Reliability and Security (ARES 2007)*, 2007.

[Sch85] F. Schweiggert. *Software-QualitÄt: Eine Standortbestimmung*. Wirtschaftsgut Software. R. Kölsch and W.Schmid and F. Schweiggert, 1985.

[SGM02] Clemens Szyperski, Dominik Gruntz, and Stephan Murer. *Component Software - Beyond Object-Oriented Programming*. Component Software Series. Addison-Wesley Longman, 2 edition, November 2002.

[SHvdM17] J. Strassner, J. Halpern, and S. van der Meer. Generic Policy Information Model for Simplified Use of Policy Abstractions (SUPA). Internet draft, IETF, Network Working Group, 2017.

[Sie10] Siemens AG. IHE – Integrating the Healthcare Enterprise. Our Contribution to Connectivity Across the Continuum of Care. http://www.siemens.com/IHE, September 2010.

[SJCC07] Kwang Sik Shin, Jin Ha Jung, Jin Young Cheon, and Sang Bang Choi. Real-time Network Monitoring Scheme Based on SNMP for Dynamic Information. *Journal of Network and Computer Applications*, 30(1):331–353, January 2007.

[SLL+08] X.-H. Sun, Z. Lan, Y. Li, H. Jin, and Z. Zheng. Towards a Fault-Aware Computing Environment. In *Proceedings of the High Availability and Performance Computing Workshop (HAPCW)*, 2008.

[Slo94] Morris Sloman. Policy Driven Management For Distributed Systems. *Journal of Network and Systems Management*, 2:333–360, 1994.

[SM88] Morris Sloman and Jonathan Moffett. Domain Model of Autonomy. In *Proceedings of the 3rd Workshop on ACM SIGOPS European Workshop: Autonomy or Interdependence in Distributed Systems*, pages 1–4, New York, NY, USA, 1988. ACM.

[SMM+12] Julia Schroeter, Peter Mucha, Marcel Muth, Kay Jugel, and Malte Lochau. Dynamic Configuration Management of Cloud-based Applications. In *Proceedings of the 16th International Software Product Line Conference*, volume 2, pages 171–178, New York, NY, USA, 2012. ACM.

[SRS+03] Y. Snir, Y. Ramberg, J. Strassner, R. Cohen, and B. Moore. Policy Quality of Service (QoS) Information Model. RFC 3644 (Proposed Standard), November 2003.

[SSS+16] S. N. Shirazi, S. Simpson, K. N. Syeda, A. Mauthe, and D. Hutchison. Towards Policy Refinement for Resilience Management in Cloud. In *Proceedings of the 8th International Workshop on Resilient Networks Design and Modeling (RNDM)*, pages 260–266. IEEE, 2016.

[Sta] Stanford Center for Biomedical Informatics Research (BMIR), Stanford University. Protege Ontology Editor and Knowledge Aquisition System. http://protege.stanford.edu. Accessed September 2018.

[Str03] John Strassner. *Policy-Based Network Management: Solutions for the Next Generation (The Morgan Kaufmann Series in Networking)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[Sut14] I. Sutton. *Process Risk and Reliability Management*. Elsevier Science, 2014.

[SWZ05] Gopalan Sivathanu, Charles P. Wright, and Erez Zadok. Ensuring Data Integrity in Storage: Techniques and Applications. In *Proceedings of the 2005 ACM Workshop on Storage Security and Survivability*, pages 26–36, New York, NY, USA, 2005. ACM.

[SZ08] S. M. M. Soe and M. P. Zaw. Design and Implementation of Rule-Based Expert System for Fault Management. *Journal of World Academy of Science, Engineering and Technology*, 48:34–39, 2008.

[Szy03] Clemens Szyperski. Component Technology: What, Where, and How? In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 684–693, Washington, DC, USA, 2003. IEEE Computer Society.

[TDLS09] K. Twidle, N. Dulay, E. Lupu, and M. Sloman. Ponder2: A Policy System for Autonomous Pervasive Environments. In *Proceedings of the 5th International Conference on Autonomic and Autonomous Systems (ICAS '09)*, pages 330–335, April 2009.

[Tod07] Dobromir Todorov. *Mechanics of User Identification and Authentication: Fundamentals of Identity Management*. Auerbach Publishers Inc., 2007.

[Top13] Eric Topol. *The Creative Destruction of Medicine: How the Digital Revolution Will Create Better Health Care*. Basic Books, 2013.

[TTT+13] Osamu Takaki, Izumi Takeuti, Koichi Takahashi, Noriaki Izumi, Koichiro Murata, Mitsuru Ikeda, and Koiti Hasida. Graphical Representation of Quality Indicators Based on Medical Service Ontology. *SpringerPlus*, 2:274–294, 2013.

[UBJ+03] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott. KAoS Policy and Domain Services: Toward a Description-Logic Approach to Policy Representation, Deconfliction, and Enforcement. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, Washington, DC, USA, July 2003. IEEE Computer Society.

[UBJ04] Andrzej Uszok, Jeffrey M. Bradshaw, and Renia Jeffers. KAoS: A Policy and Domain Services Framework for Grid Computing and Semantic Web Services. In *Proceedings of the 2nd International Conference on Trust Management (iTrust 2004)*, pages 16–26, 2004.

[U.S12] U.S. Food and Drug Administration. Premarket Approval of Medical Device. Code of Federal Regulations. Title 21 Volume 8, April 2012.

[VÖ3] Markus Völter. A Taxonomy for Components. *Journal of Object Technology*, 2(4):119–125, July-August 2003.

[VAH+02] R. Vilalta, C. V. Apte, J. L. Hellerstein, S. Ma, and S. M. Weiss. Predictive Algorithms in the Management of Computer Systems. *IBM Systems Journal*, 41(3):461–474, July 2002.

[VRYK03] Venkat Venkatasubramanian, Raghunathan Rengaswamy, Kewen Yin, and Surya N. Kavuri. A Review of Process Fault Detection and Diagnosis: Part

I: Quantitative Model-Based Methods. *Computers & Chemical Engineering*, 27(3):293–311, 2003.

[VVB02] Jorge E. López De Vergara, Víctor A. Villagrá, and Julio Berrocal. Semantic Management: Advantages of Using an Ontology-Based Management Information Meta-Model. In *Proceedings of the HP Openview University Association 9th Plenary Workshop (HP-OVUA'2002)*, pages 11–13, 2002.

[WA02] Elaine J. Weyuker and Alberto Avritzer. A Metric to Predict Software Scalability. In *Proceedings of the 8th International Symposium on Software Metrics*, Washington, DC, USA, 2002. IEEE Computer Society.

[Wal11] E. Wallmüller. *Software Quality Engineering: Ein Leitfaden für bessere Software-Qualität*. Hanser, 2011.

[WDT+06] Feng Wang, Liam S. Docherty, Kenneth J. Turner, Mario Kolberg, and Evan H. Magill. Services and Policies for Care at Home. *Proceedings of the 1st. International Conference on Pervasive Computing Technologies for Healthcare*, pages 7.1–7.10, 2006.

[WG06] Charles B. Weinstock and John B. Goodenough. On System Scalability. Technical Report CMU/SEI-2006-TN-012, Carnegie Mellon University, March 2006.

[Wie94] Rene Wies. Policies in Network and Systems Management - Formal Definition and Architecture. *Journal of Network and Systems Management*, 2:63–68, April 1994.

[Wie03] Karl Wiegers. *Software Requirements 2*. Microsoft Press, March 2003.

[WNN96] Brian C. Williams, P. Pandurang Nayak, and Urang Nayak. A Model-based Approach to Reactive Self-Configuring Systems. In *Proceedings of 19th National Conference on Artificial Intelligence (AAAI-96)*, pages 971–978, 1996.

[Wor10a] World Health Organization. Classification of Health Workforce Statistics. www.who.int/hrh/statistics/workforce_statistics, 2010.

[Wor10b] World Health Organization. International Statistical Classification of Diseases and Related Health Problems. 10th Revision, 2010.

[WSS+01] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, and S. Waldbusser. Terminology for Policy-Based Management. RFC 3198 (Informational), November 2001.

[WT07] Feng Wang and Kenneth J. Turner. Policy Conflicts in Home Care Systems. In *Proceedings of the 9th International Conference on Feature Interactions in Software and Communication Systems*, Amsterdam, September 2007.

[WT08] Feng Wang and Kenneth J. Turner. Towards Personalised Home Care Systems. *Proceedings of the 1st International Conference on Pervasive Technologies Related to Assistive Environments (PETRA '08)*, July 2008.

[YPG00] R. Yavatkar, D. Pendarakis, and R. Guerin. A Framework for Policy-based Admission Control. RFC 2753 (Informational), January 2000.

[Zad65] Lotfi A. Zadeh. Fuzzy Sets. *Information and Control*, 8:338–353, 1965.

[ZFC04] Youyong Zou, Timothy W. Finin, and Harry Chen. F-OWL: An Inference Engine for Semantic Web. In *Proceedings of the 3rd InternationalWorkshop on Formal Approaches to Agent-Based Systems (FAABS)*, Lecture Notes in Computer Science, pages 238–248, Greenbelt, MD, USA, April 2004. Springer-Verlag.