Policy-Based Management for Resource-Constrained Devices and Systems

Oliver Dohndorf, Jan Krüger
and Heiko Krumm
TU Dortmund University
44221 Dortmund, Germany
(dohndorf, krueger)@ls4.cs.uni-dortmund.de

Christoph Fiehe, Anna Litvina, Ingo Lück
and Franz-Josef Stewing
MATERNA Information & Communications
44141 Dortmund, Germany
(christoph.fiehe, anna.litvina)@materna.de

***Abstract*—The presented policy-based management system supports autonomous control and adaptation of a distributed system according to changing conditions and requirements by means of event-condition-action (ECA) rules. Furthermore, it supports policy-aware application programming. Application components can request evaluations of policy expressions and decisions in order to govern their behavior depending on global system state and environment conditions. That rich functionality has to be provided very efficiently since the distributed system consists of resource-constrained devices. The model-based management (MBM) approach is applied separating comfortable tool-assisted policy definition and refinement at design time from lightweight runtime policy enforcement. New enhancements of MBM extend the policy refinement to the derivation of ECA rules. The new backend functions generate executable Java bytecode for policy expressions and decisions as well as for policy rules. The code is appropriately partitioned and allocated to the devices. A simplified healthcare scenario demonstrates the approach and its application.**

***Keywords*-policy, model-based management, device**

I. Introduction

In the current project OSAmI, a healthcare application scenario is defined for which a comprehensive automated management is needed. Exceptions and changing conditions are common in that scenario and must not impair the proper operation of the system. We rely on policy-based management which has been acknowledged as a promising way for management automation [1]. The additional policy hierarchy approach supports the representation of abstract business goals and objectives as well as of detailed low-level policies describing device configurations, parameter settings, and corrective actions. The actions particularly are defined by event-condition-action (ECA) rules. The policy design can concentrate on the definition of abstract and easy-to-understand high-level policies since the detailed low-level policies are obtained later on by transformation which is referred to as *policy refinement* [2]. According to [2], fully automated refinement is not possible. In comparison with the abstract policies, the low-level ones contain additional details and domain-specific information which has to be provided by wide expert knowledge.

We enhance and apply the special approach of model-based management (MBM) [3]. MBM separates policy definition and refinement from policy enforcement. Definition and refinement are performed at design time using the modeling and refinement tool MoBaSeC (Model Based Service Configuration). The frontend of MoBaSeC is a graphical model editor recording the system model and the abstract policies. The refinement functions compute the refined policies. At the end of the design phase, MoBaSeC's backend functions generate configuration data in accordance with the derived low-level policies. MBM in fact supports automated policy refinement since it utilizes a hierarchically structured system model. The model is defined by the user. It represents the managed system on three interrelated levels of abstraction. The high-level policies are directly linked with the system objects of the highest model layer. System refinement relations connect adjacent model layers linking objects with their refined implementations of the next lower layer. The automated policy refinement follows these system refinement relations in order to obtain the additional information for generating the low-level policies. The model architecture is inspired by the DMTF Common Information Model (CIM) [4] which is an object-oriented information model for the uniform description of system structure, functions, and information. It allows to represent physical and logical objects as well as their mutual relations.

Our enhancements of MBM extend the policy refinement to the derivation of executable ECA rules. Rule conditions and actions are defined over configuration and status variables forming the Management Information Base (MIB) [5] of the system. Furthermore, the notions of policy expressions and decisions refer to the configuration and status variables. Beside of configuration data, MoBaSeC's new backend functions generate executable Java bytecode for policy expressions and decisions as well as for rule conditions and actions. Moreover, MoBaSeC computes the appropriate partitioning and allocation of the generated Java bytecode. Code and data deployment complete the design phase. At runtime, the code serves as implementation of a management system which is lightweight and exactly tailored to the efficient enforcement of the defined policies.

In the sequel, Section II introduces related work. Section III presents the system structure which forms the basis of our approach. Sections IV and V depict the design phase, respec-

tively, the runtime phase of our policy-based management. It is exemplified by an application example from the medical home care domain. Section VI concludes the paper.

II. Related Work

The work particularly is related to approaches for policy-based management of healthcare systems covering resource-constrained devices. AMUSE [6] introduces the architectural pattern of the *Self-Managed Cell (SMC)*. It provides local feed-back control based on ECA rule policies which allow to express the adaptation strategy required in response to context changes. Policy specification and enforcement is supported by the *Ponder2* tool [7]. MATCH [8] proposes the policy language APPEL [9] and its adaption to the healthcare domain by provision of typical triggers, conditions, and actions. Stylized natural language policies can be edited remotely using a web-based policy wizard. In the case of complex application systems, however, the task of policy specification can become very tedious. In contrast, our approach represents abstract policies graphically not using a special policy language. At runtime the low-level policies are represented as executable Java bytecode, thus avoiding the interpretation of policy language statements.

III. System Structure

A system is represented as a set of components and associations between them. Static associations represent component relations (e.g., "runs on", "is part of", "depends on"). A dynamic association supporting the interaction between a client and a service is called a *binding*.

Software Component A software component is realized as an OSGi bundle and implements domain-specific application logic. Figure 1 depicts a component which provides *(1)* and/or uses *(2)* one or more services. The MIB scheme of the component defines the *status* and *configuration variables*. Status variables describe the management-relevant state of the component; they are set from the application logic and read by the management. Configuration variables are means of configuring the component; they are set by the management and read from the application logic.

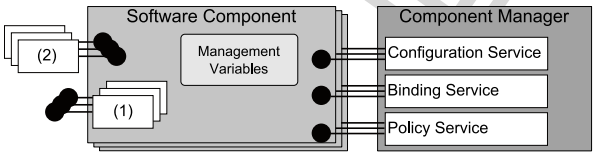

Figure 1: Component Structure

The management variables can be accessed only via a *component manager*. A component manager, implemented as a component itself, is responsible for the management of its assigned components. It offers three management services which are to be used from the application logic: the *configuration service* for accessing management variables, the *policy service* for requesting evaluations of policy expressions or decisions in the context of the current system state, and the *binding service* for the management of bindings.

Binding A binding is a client-service association between two components, one providing a service, the other using it. During its life cycle, a binding traverses a series of control states (e.g., "requested", "established", "broken"). *Binding requirements* specify the performance, reliability, substitution, and security constraints of a component.

Bindings are established at runtime by the management system. In case of failure, they are transparently substituted if that is allowed by their constraints. That supports a dynamic, flexible, and fault tolerant configuration of the system. In many use cases, a client needs stable bindings to a set of services in combination, for example, a health monitoring service has to use and correlate the measurements of several sensors. For this purpose, we resort to the notions of *ensemble* [10] and *lease* [11]. An ensemble is a set of services to be used in combination. Therefore, the services of an ensemble are allocated as a whole. The binding is performed atomically using a two-phase allocation protocol. A lease is a time-limited contract between a client and a service and can be regarded as a compromise between stability and flexibility. During lease duration, the binding is stable and can only be released in agreement with the client or due to exceptional conditions. The limited duration supports flexibility since each lease expiration offers a chance for reconfiguration.

IV. Model-Based Management

The MBM adoption used, models the system on three different layers. Each layer comprises a self-contained model of the whole system. The models of adjacent layers are connected by refinement relations. A refinement relation associates an object with those objects of the adjacent lower layer which together represent the higher-layer object. Figure 2 depicts an example. The layers are:

Use Cases & Aspects (U&A) Use cases with their aspects are modeled. The system objects are *actors*, *assets*, *functions*, and *use cases* connecting them. The policy objects represent *aspects* and their attributes. The aspects are assigned to the use cases. The attributes define properties like medical characteristics or security requirements. In Figure 2, the use case *ergometer training* connects the *cardiac patient actor* with its *training plan asset*, and the two system functions *ergometer control* and *training control*. The training control function depends on the *pulse rate analysis function*. In the policy part, for example, the *medical aspect* defines the low-risk patient profile to be applied for that use case.

Services & Domains (S&D) The system is modeled from a service-oriented point of view. The system objects are *subjects*, *data*, *applications*, and *services*. They are grouped into *domains* represented by a domain attribute. The policy objects are *objectives* defining required service levels as well as *monitoring and control functionalities*. In

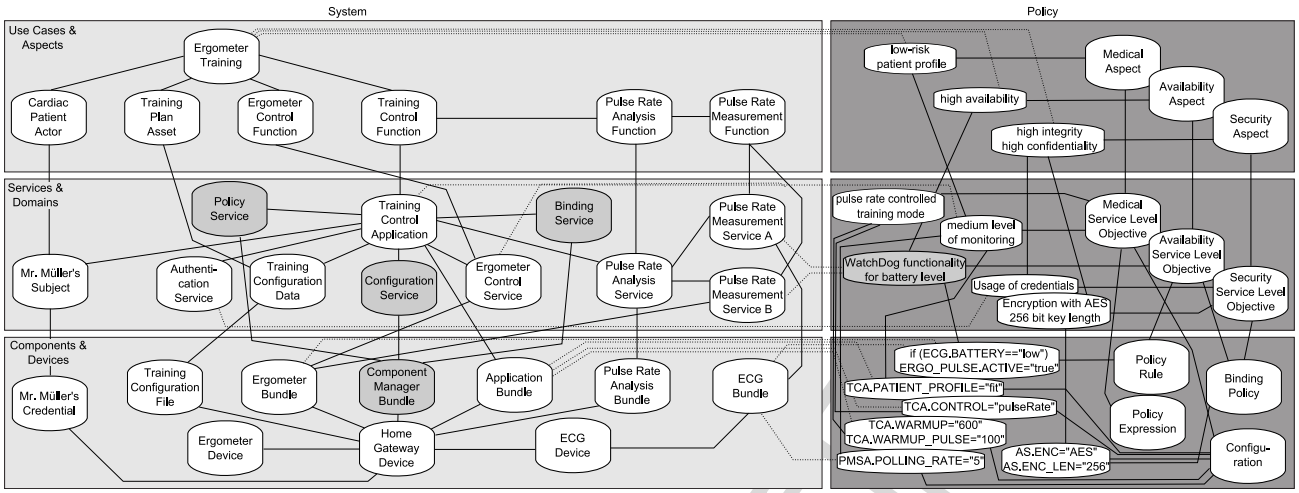

Figure 2: Application Example

Figure 2, the patient's subject is user of the *training control application* which orchestrates a series of services (e.g., *ergometer control*) and is supported by the management services for policy evaluation, configuration, and binding. In the policy part, for example, the *medical service level objective* combines the two objectives *pulse rate control* and *medium level of monitoring*. Also, refinement relation links exist (e.g., between the cardiac patient actor and the patient's subject). The function *pulse rate measurement* is linked with the *pulse rate measurement service A* and *pulse rate measurement service B* which redundantly provide the abstract function.

Components & Devices (C&D) The actual software components, resources, and devices of the system are modeled. The system objects are *credentials*, *files*, *bundles*, and *devices*. The policy objects are *ECA rules*, *binding requirements*, *configuration settings*, and *policy expressions*. In Figure 2, the patient's credential is present at the *home gateway device* which hosts the *ergometer bundle*, the *application bundle*, and the *pulse rate analysis bundle*. It is connected with the *ergometer device* and the *ECG device*. In the policy part, for example, one ECA rule is triggered by a low battery state event of the ECG device. In effect, the alternative pulse rate measurement service offered by the ergometer bundle is activated. The patient's subject and his credential are connected with a refinement relation link. Pulse rate measurement service A is refined to the implementing ECG bundle, pulse rate measurement service B is refined to the implementing ergometer bundle which provides an alternative service for pulse rate measurement.

The policy refinement is a step-by-step transformation from abstract high-level policies to technical low-level policies, carried out according to the refinement relations. Formally, it is a function from the domain of U&A to the range of S&D and from S&D to C&D. It fulfills homomorphic properties since it preserves the assignment of policy objects to system objects. For example, an abstract security requirement for high confidentiality is mapped to the protection requirements of the corresponding services. This results in specific binding requirements for cryptographic methods and minimal key length in the low-level policies. Thus, the policies are refined from *declarative* to *imperative* ones.

V. Policy-Based Runtime Management

The runtime management system includes component managers enforcing the derived low-level policies. They monitor, control, and configure the corresponding components. The assignment of components to their managers and the allocation of low-level policies are planned by MoBaSeC during the design phase. Low-level policies are defined on management variables, event parameters, and constants; they have the form of conditions, expressions, and variable assignments. Four policy types can be distinguished:

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

Policy Condition A policy condition is a special case of a policy expression returning a Boolean value (e.g., an application requests the decision, whether to stop the operation or not depending on the current system state).

Policy Rule A policy rule embodies an ECA rule specifying the actions (variable assignments) to be taken on a certain event (e.g., the application's operating mode can be adjusted in response to an increased energy consumption).

Binding Requirements A binding requirement specifies non-functional requirements for a client-service association. To establish the association, the client's and service's binding

requirements are intersected. The configuration space of the association is specified by the result of the operation. For example, the client has a binding requirement that communication channels must be encrypted with a key length of 128 or 256 bit. The service's binding requirements prescribe the encryption with a key length of 256 bit. The intersection result defines that the encryption key length of the communication channel will be 256 bit.

The binding service establishes, monitors, and releases client-service associations. Binding establishment is conducted in four phases: firstly, for one association, a set of functionally suitable services is found, secondly, according to the binding requirements, a subset of suitable services is selected. From that set one service with maximum business value is chosen. The business value is computed according to a policy expression. Finally, the binding is established under negotiation of the binding requirements. The negotiated quality is monitored as follows: change events of relevant status variables trigger a set of ECA rules which check for quality of service violations and perform corrective reconfiguration actions. Only in the rare case where reconfiguration fails, the binding is released prematurely. The regular release of a binding is caused by a release request of the client or by an expiration of the lease period. On a release request, the binding service forwards the request to the service which in turn releases allocated resources.

A set of service bindings, which have to be established in combination, forms an ensemble and is allocated atomically by means of a two-phase lease-granting algorithm which is similar to the two-phase-commit protocol. During the first phase the binding service asks the chosen services for reservations. When a service grants a reservation, it is obliged to accept a lease request if it occurs within the short reservation period. In the case that all services respond positively and in time, the binding service enters the second phase and submits the lease requests. Otherwise, the reservations are canceled or expire.

VI. Conclusion

We have presented our ongoing work on the development of a policy-based management for resource-constrained devices and systems carried out within the OSAmI research project [12]. The introduced approach relies on the model-based management paradigm which combines management policies and policy hierarchies with a layered system model. In the design phase, the abstract high-level policies are transformed automatically into technical low-level policies. These are defined directly on management variables, events, and constants and are present at runtime in the form of efficiently executable Java bytecode.

By now first prototypes are operational; particularly the presented application scenario has been implemented, tested, and evaluated by end users. The runtime policy system has a size of 46 kB incl. 12 kB bytecode representing the derived low-level policies. Per event at most 5 policy ECA rules have to be checked. At runtime, the policy evaluation takes less than 1% of processor performance (100 MHz Foxboard); each ECA rule is executed in less than 1 ms. At design time, the MoBaSeC tool takes less than 1 min for the automated refinement incl. the bytecode generation (2x2.5 GHz desktop PC). Though the user of MoBaSeC is assumed to be an expert, he is supported by model examples and patterns.

This work has been funded by the German Federal Ministry of Education and Research (BMBF).

References

[1] M. Sloman, "Policy Driven Management for Distributed Systems," *Journal of Network and Systems Management*, vol. 2, pp. 333–360, 1994.

[2] J. Moffett and M. S. Sloman, "Policy Hierarchies for Distributed Systems Management," *IEEE Journal on Selected Areas in Communications*, vol. 11, pp. 1404–1414, 1993.

[3] S. Illner, H. Krumm, I. Lück *et al.*, "Model-based Management of Embedded Service Systems – An Applied Approach," in *Proc. of the 20th Int. Conf. on Advanced Information Networking and Applications (AINA '06)*. IEEE Computer Society, 2006, pp. 519–523.

[4] Distributed Management Task Force, Inc. (DMTF), *Common Information Model (CIM) Infrastructure. Specification*, 2009.

[5] ISO, "ISO/IEC 7498-4: Information processing systems – open systems interconnection – basic reference model – part 4: Management framework," 1989.

[6] E. Lupu, N. Dulay, M. Sloman *et al.*, "AMUSE: Autonomic Management of Ubiquitous e-Health Systems," *Concurrency and Computation: Practice and Experience*, vol. 20, pp. 277–295, 2008.

[7] K. Twidle and E. Lupu, "Ponder2 – Policy-Based Self Managed Cells," in *Proc. of the 1st Int. Conf. on Autonomous Infrastructure, Management and Security (AIMS '07)*. Springer-Verlag, 2007, pp. 230–230.

[8] F. Wang and K. J. Turner, "Towards Personalised Home Care Systems," in *Proc. of the 1st Int. Conf. on Pervasive Technologies Related to Assistive Environments (PETRA '08)*. ACM Press, 2008, pp. 1–7.

[9] K. J. Turner *et al.*, "APPEL: An Adaptable and Programmable Policy Environment and Language," Computing Science and Mathematics, University of Stirling, CSM-161, 2007.

[10] A. Pohl, H. Krumm, F. Holland *et al.*, "Service-orientation and Flexible Service Binding in Distributed Automation and Control Systems," in *Proc. of the 22nd Int. Conf. on Advanced Information Networking and Applications (AINA'08)*. IEEE Computer Society, 2008, pp. 1393–1398.

[11] C. G. Gray and D. R. Cheriton, "Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency," in *Proc. of the 12th ACM Symposium on Operating System Principles (SOSP '89)*. ACM Press, 1989, pp. 202–210.

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