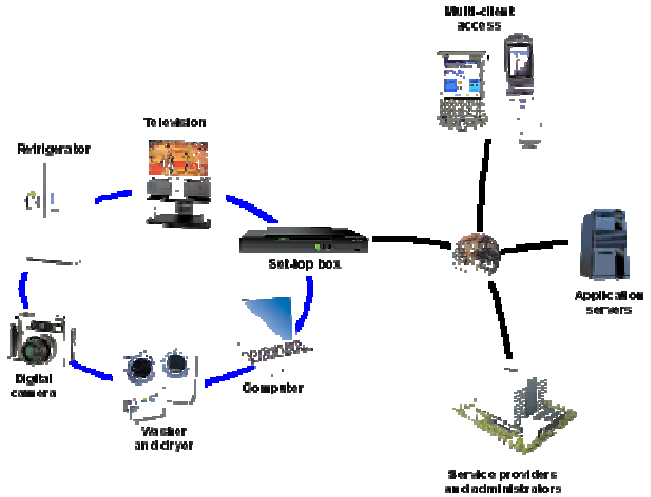
24th IEE Int. Conf. on
Advanced Information Networking and Applications (AINA 2010)
Workshop
Service Oriented Architectures in Converging Networked Environments (SOCNE 2010)

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

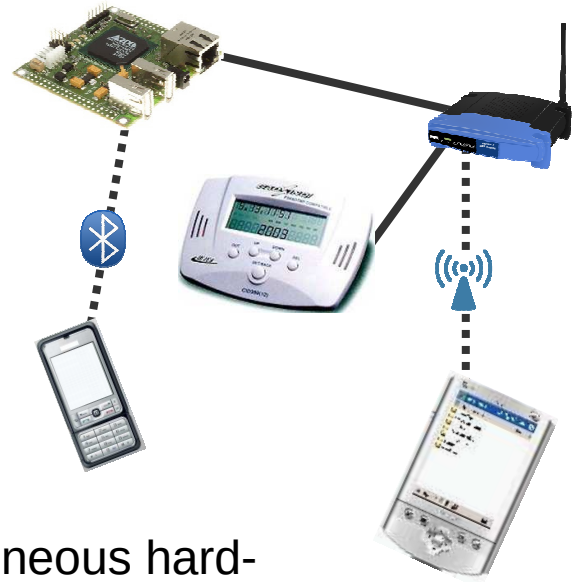
- *Motivation*
- *Technical Background*
- *System Structure*
- *Management System*
- *Example*
- *Conclusion*



TU Dortmund University
O. Dohndorf, J. Krüger, H. Krumm

MATERNA
C. Fiehe, A. Litvina, I. Lück, F.-J. Stewing

Motivation



- Due to e.g. decreased costs, the use of intelligent connected devices constantly increases
- Today, those devices become more and more loosely coupled, following the service-oriented architectural style (“device-based service systems”)
- Challenge: realization of such systems using heterogeneous hard- and software (e.g., different communications protocols or interfaces)
- Goal: flexible and adaptable systems with a predictable and reliable system behavior; e.g., regarding changing conditions or requirements
- Our approach: developing an appropriate management system to monitor a system and provide automatically reactions and adaptations while being light-weighted in order to be applicable to embedded devices with limited resources and computational power

Technical Management

- **Open System Interconnection Management**
  - Developed by the Int. Organization for Standardization (ISO)
  - FCAPS: fault, configuration, accounting, performance, security
  - Management Information Base (MIB)

- **Web-Based Enterprise Management (WBEM)**
  - Developed by the Distributed Management Task Force (DMTF)
  - Management of distributed systems
  - Common Information Model (CIM)
    - Object-oriented information model for uniform description of management information and functions
    - Representation of physical and logical devices

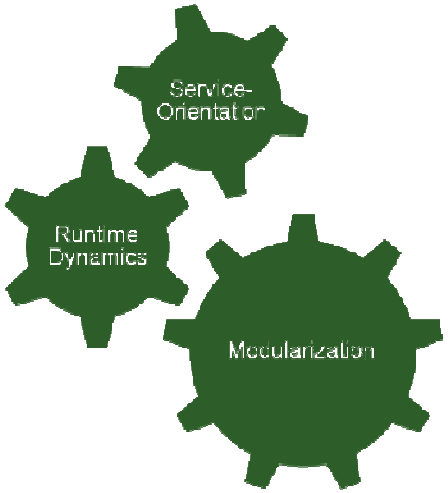
http://www.wbemsolutions.com/tutorials/CIM/cim-schema.html

- **Web Services Distributed Management (WSDM)**
  - Developed by the Organization for Advancement of Structured Information Standards (OASIS)
  - Based on Web Services
  - Primary domains:
    - Management *using* Web Services (MUWS)
    - Management *of* Web Services (MOWS)

OSGi

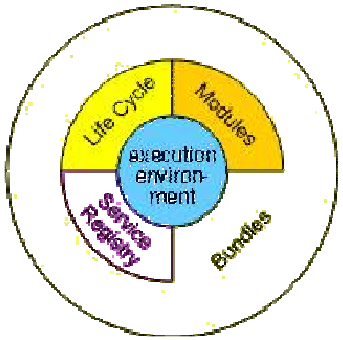
- Dynamic module system for Java

- Allows integration and management of modules („bundles“) and services



- Bundles
  - Each bundle is a JAR
  - Each bundle uses its own classloader
  - Can hide or share their packages

- Services
  - Each bundle can provide services to be used by other bundles
  - Each service is a Java object, implementing a specified interface
  - Service Registry allows locating services



- Bundles (and thus services) can be installed, started, updated, stopped, and uninstalled during runtime

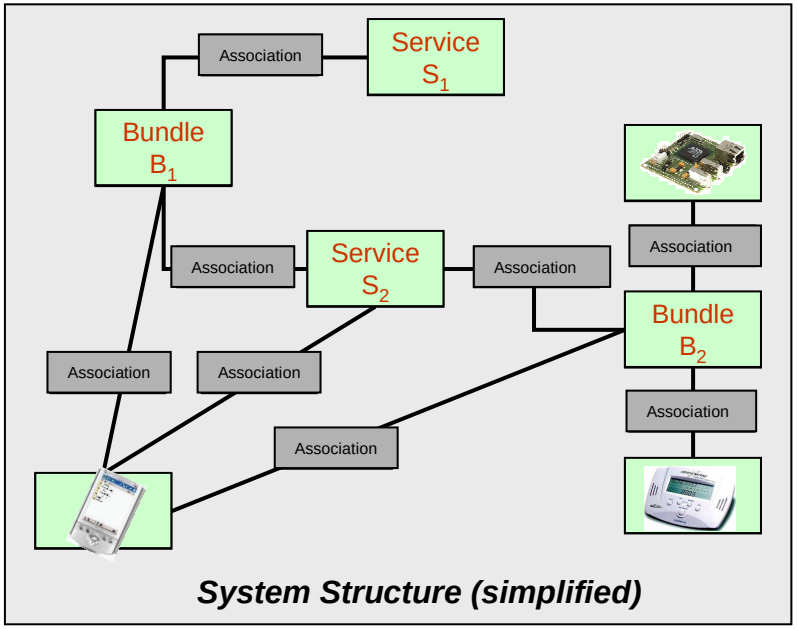
Managed System Structure

- A manageable system consists of components
  - Hardware as well as software components
  - Software components are of particular interest

- Between components, different associations may exist
  - “located on”
  - “consists of”
  - “depends on”
  - “belongs to”
  - etc.

- Associations may be static or dynamic

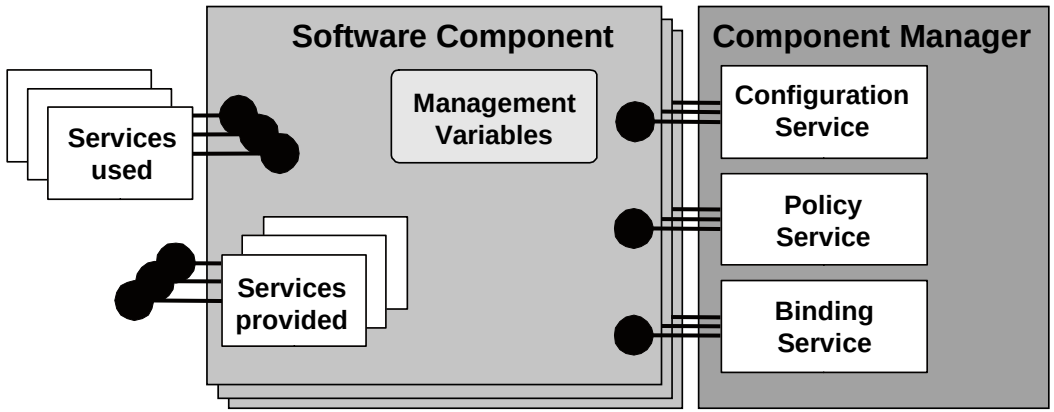
- Dynamic associations are called bindings



System Structure (simplified)

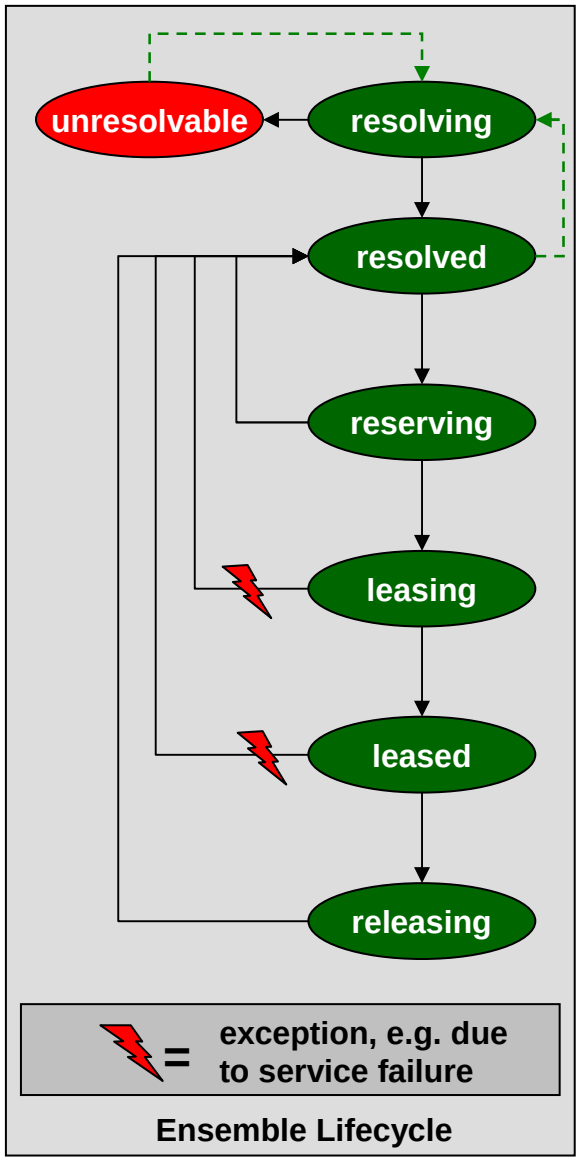
Software Components & Component Managers

- Realized by means of OSGi bundles
- Implement domain-specific application logic
- Offer their functionality by providing an unlimited number of services
- May use an unlimited number of services provided by other components
- Provided with a MIB scheme and a corresponding set of management variables
  - Status variables
  - Configuration variables
  - Accessible only using the *configuration* service of the appropriate *Component Manager* (CM)
- Furthermore, the CM provides the *policy* and *binding* service



Bindings

- Association between two components (client / server)
- Established between so-called endpoints considering the endpoints properties (e.g., security parameters)
- Dynamic, i.e. established at runtime
- Potentially transparently substitutable (depending on the bindings and / or endpoints properties)
- Bindings hold a status, e.g. requested, established, or broken
- Ensemble-/Lease-Principle for stable bindings
  - Ensemble: logical service group, allocated either as a whole or not at all
  - Lease: time-limited client/server contract
  - Only clients can release ensembles during the lease time (apart from exceptions)



Ensemble Lifecycle

Model-Based Management

- Combination of *management policies* and *policy hierarchies* with a *layered system model*

- Management Policies
  - Additional control level above the programming code
  - Used for configuration, adaptation, correction

- Management phases: *design phase* and *runtime phase*

- Design phase
  - System is modeled on different layers of abstraction
  - Abstract high-level policies are defined manually
  - Policy tool for support of the modeling
  - Automated refinement and deployment of low-level, technical policies

- Runtime phase
  - Refined low-level policies are efficiently enforced

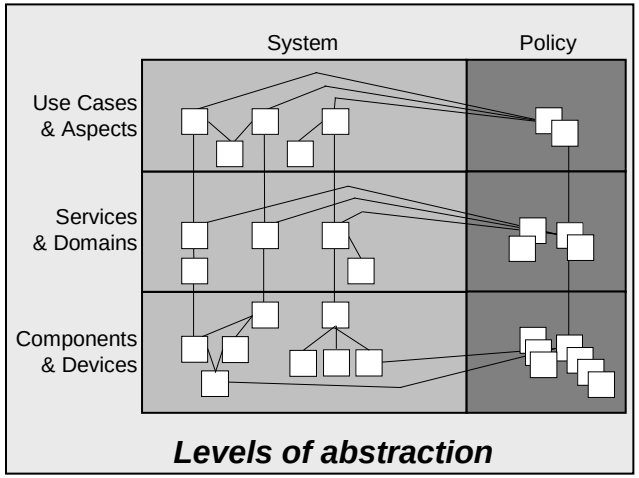
What are policies?

„Policies are rules governing the choices in behaviour of a system.“

M. Sloman,
Policy Driven Management
for Distributed Systems,
Journal of Network and
Systems Management, Vol. 2, 1994

Model & Abstraction Layers

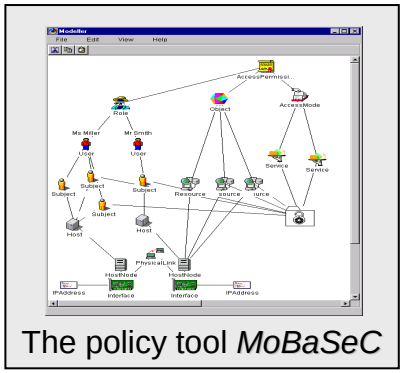
- System is modeled on 3 layers with different degree of abstraction
- Each layer model is self-contained + independent ➔ 3 models of the same system, differing in the degree of abstraction
- Refinement relations to associate elements of adjacent models
- *Use Cases & Aspects layer*: barely technical details, instead aspects like non-functional requirements and quality criteria
- *Services & Domains layer*: service-oriented point of view; clients, services, dependency relations are modeled and assigned to domains
- *Components & Devices layer*: elements existing at runtime; i.e., software components (bundles) and devices



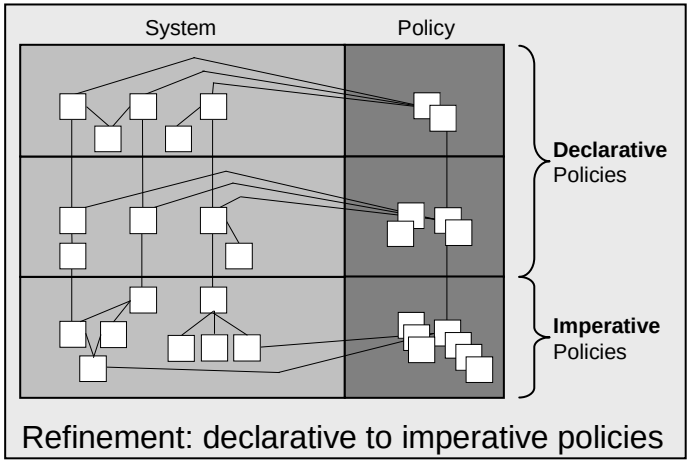
Levels of abstraction

Policy Refinement

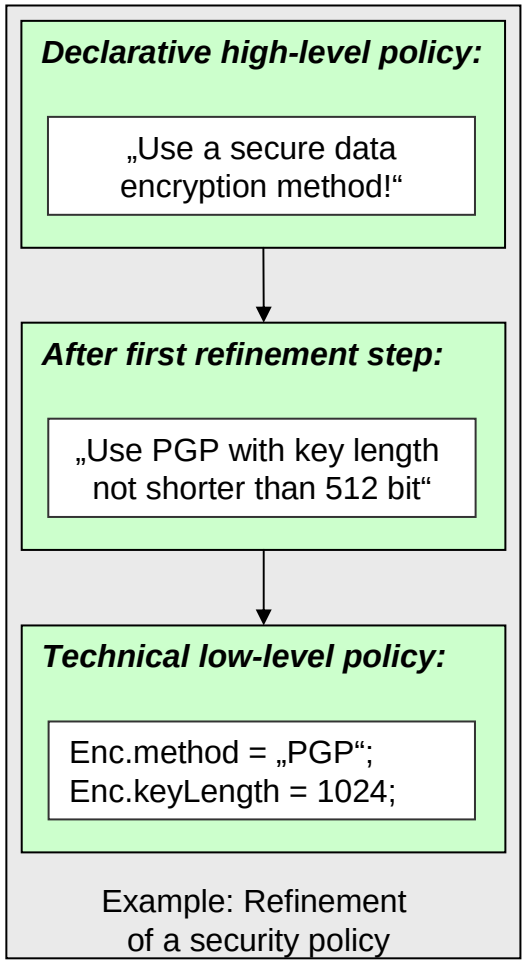
- Step-by-step derivation of imperative, technical low-level policies for use at runtime from declarative high-level policies
- Derivation according to refinement relations associating corresponding elements of adjacent model layers
- Low-level policies refer only to management variables
- Modeling & refinement is supported by the policy tool "***MoBaSeC***"



The policy tool *MoBaSeC*



Refinement: declarative to imperative policies



Example: Refinement of a security policy

Policy-Based Runtime Management

- Carried out by the Component Managers

- Enforcement of previously derived low-level policies (existing as efficiently executable Java byte code)

Trigger: ”running state“ of a service changed

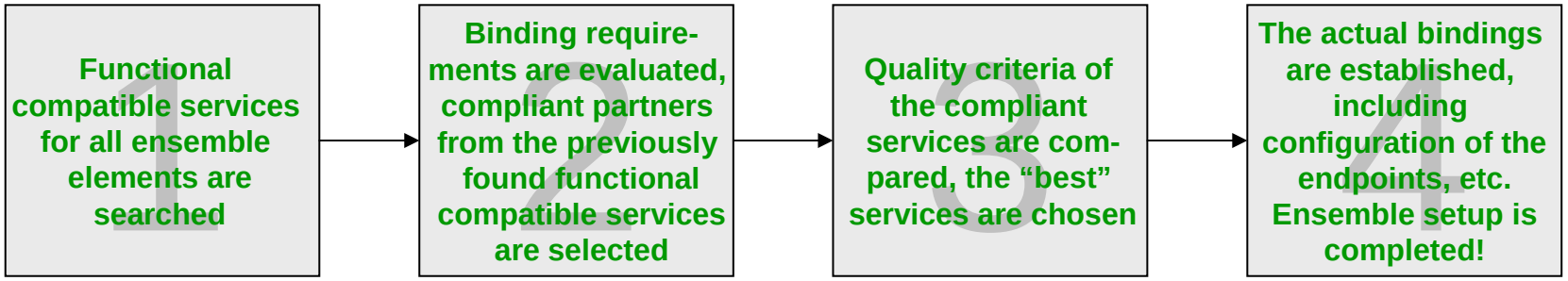
```
if ("running state" == "unstable")
mgmtData.setConfig("Client Bundle",
   "service error", "true");
```

Exemplary Policy Rule (simplified)

- Four different types of low-level policies:
  - ***Policy Expression***: expression over management variables, constants, and operations. Evaluated on demand of a component; may return any Java element as result of the evaluation
  - ***Policy Condition***: special case of the Policy Expression; return type is restricted to boolean
  - ***Policy Rule***: event-condition-action (ECA-) rule specifying the action to be performed in case of a certain event; actions are performed by assigning values to configuration variables
  - ***Binding Requirement***: specifies non-functional requirements for establishing bindings; defined as boolean expression over management variables

Binding Management

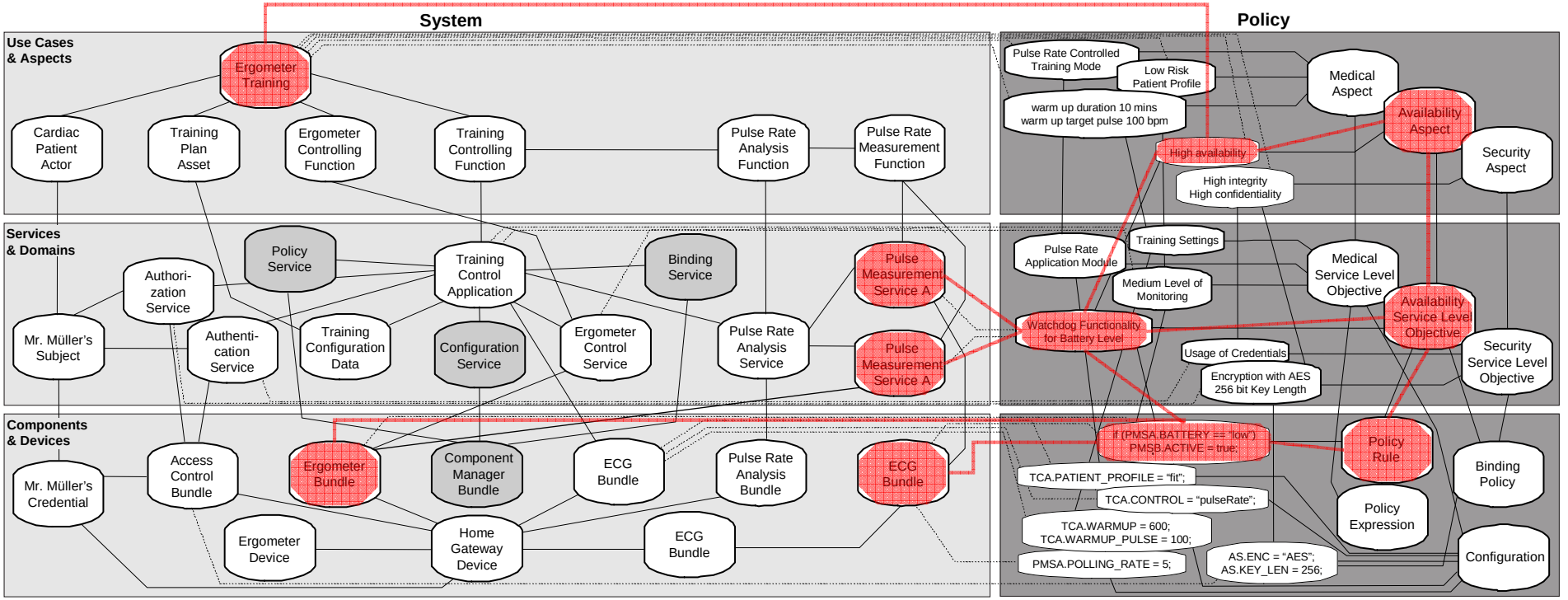
- Responsible for establishing, monitoring, and releasing ensembles
- On an ensemble request, four phases are traversed:



- Established ensembles are monitored; in case of failures, the binding management tries to reconfigure the ensemble (e.g., by transparently substituting the communication partner)
  - If reconfiguring fails, this results in an exception, leading to premature release of the ensemble
- On a client releasing an ensemble, the binding management notifies all used services, which in turn can release allocated resources

Example: System Modeling & Refinement

- Example from the E-Health domain (OSAmI project)
- Training system for a cardiac patient is planned
- Main use case: “Ergometer Training”



Conclusion

- Realizing reliable and predictable device-based service systems using a light-weight, policy-based management
- Management is divided into a design and a runtime phase
- The design phase utilizes a model to represent the system on three different layers of abstraction, hierarchically ordered
- Policies are specified as abstract high-level policies in the design phase, and automatically refined to technical low-level policies
- At runtime, low-level policies exist in form of efficiently executable Java byte code, enabling the efficient management even for a device-based service system
- Low-level policies can be divided into Policy Expressions, Policy Conditions, Policy Rules, and Binding Requirements

Thanks for the attention!

Questions?