Policy-gesteuertes Management adaptiver und gütegesicherter Dienstesysteme im Projekt OSAMI

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

Oliver Dohndorf, Jan Krüger und Heiko Krumm
Technische Universität Dortmund
(dohndorf, krueger, krumm)@ls4.cs.uni-dortmund.de

Abstract: Ziel des europäischen ITEA2-Projekts OSAMI Commons ist die Entwicklung einer Service-orientierten Komponentenplattform zur Realisierung adaptiver und gütegesicherter Dienstesysteme. Die Anwendungsdomäne des deutschen Teilprojekts ist die kardiologische Rehabilitation. Diese Domäne stellt hohe Anforderungen an Interoperabilität, Zuverlässigkeit, Datensicherheit und Adaptivität, die mittels eines Policy-gesteuerten Managements erfüllt werden. Policies lassen sich in der Planungsphase auf unterschiedlichen Abstraktionsebenen werkzeugunterstützt erstellen und werden in der Laufzeitphase durch ein leichtgewichtiges Management effizient durchgesetzt.

1 Einleitung

Das Gesundheitssystem der Zukunft wird einer immer älter werdenden Gesellschaft Rechnung tragen müssen. Es verlangt nach innovativen und intelligenten Lösungen, um der Kostensteigerung entgegenzuwirken. Die telemedizinisch gestützte Betreuung innerhalb eines überwachten häuslichen Umfelds stellt einen vielversprechenden Lösungsansatz dar und dient als Ergänzung zu einer stationären Behandlung.

Im Rahmen des europäischen ITEA2-Projekts OSAMI Commons (Open Source Ambient Intelligence Commons) soll das deutsche Teilprojekt (OSAMI-D)[1] die Voraussetzungen dafür schaffen, dass Geräte und Dienste im Gesundheitswesen schneller und zuverlässiger konfiguriert, eingesetzt und gewartet werden können. Als Anwendungsdomäne dient die kardiologische Rehabilitation. Ziele sind die Entwicklung einer grundlegenden Komponentenplattform auf Basis einer Service-orientierten Architektur und ihre Bereitstellung als Open-Source-Software. Die Grundlage bildet die von der OSGi Alliance spezifizierte Software-Plattform. Als technische Anforderungen wurden festgelegt:

Interoperabilität Durch den Einsatz heterogener Hard- und Software sind flexible verteilte Dienstesysteme nur schwer zu realisieren. Insbesondere im Gesundheitswesen wird je nach Anwendungsgebiet eine Vielzahl unterschiedlicher Standards (HL7, DICOM oder IHE Profile) verwendet. Kompatible Kommunikationsprotokolle und einheitliche Schnittstellen finden aber nur selten Anwendung. Die fortschreitende Technisierung und der steigende

[1] Das deutsche Teilprojekt wird vom Bundesministerium für Bildung und Forschung (BMBF) gefördert

Kostendruck führen jedoch häufig zu dem Problem, dass bisher isolierte Systeme zusammenarbeiten müssen. Daher rücken Standardkompatibilität und -konformität immer mehr in den Vordergrund und werden zunehmend vorausgesetzt.

Zuverlässigkeit Offene Systeme stehen in wechselseitiger Interaktion mit ihrer Umgebung und verändern sich fortlaufend, da äußere Einflüsse auf sie einwirken. Ein solches System ist konfrontiert mit veränderlichen Bedingungen und Anforderungen sowie Fehlern und Ausfällen zur Laufzeit. Auf der einen Seite benötigt man die Möglichkeit einer flexiblen und adaptiven Systemkonfiguration, auf der anderen Seite steht die Forderung nach einem vorhersehbaren und zuverlässigen Systemverhalten.

Datensicherheit Der Schutz personenbezogener Daten erfordert besondere Aufmerksamkeit. Innerhalb informationsverarbeitender Systeme sind die Schutzziele Vertraulichkeit, Integrität, Verfügbarkeit und Verbindlichkeit sicherzustellen. Die hohen Ansprüche an Datensicherheit und Datenschutz erfordern effektive und sorgfältig geplante Maßnahmen. Dabei muss gewährleistet werden, dass kein unautorisierter Zugriff auf Systemressourcen oder Daten erfolgen kann. Die Zugriffsmöglichkeiten müssen gemäß der Datenschutzbestimmungen reglementiert werden, wobei der Patient stets die vollständige Kontrolle über seine Daten ausübt. Er allein bestimmt, wer welche Informationen einsehen, nutzen und weitergeben darf.

Adaptivität Um den Patienten bestmöglich bei seiner Rehabilitation zu unterstützen, muss gewährleistet werden, dass das System Fehler und Ausfälle weitestgehend automatisiert handhabt. Die medizinischen Bedürfnisse und individuellen Präferenzen des Patienten sind bei der Systemkonfiguration zu berücksichtigen. Da sich sein Gesundheitszustand im Laufe der Zeit verändern kann, muss das System leicht anpassbar und erweiterbar sein, so dass stets eine optimale Versorgung garantiert werden kann. Im Umfeld eingebetteter Systeme mit nur beschränkten Ressourcen ist sicherzustellen, dass durch die zusätzlichen Management-Funktionen keine Beeinträchtigungen der Anwendungsfunktionen entstehen. Zur flexiblen Konfiguration des Systems ist daher ein leichtgewichtiges Laufzeit-Management erforderlich.

Innerhalb von OSAMI-D werden übergreifende Ansätze zur Interoperabilität entwickelt, die es unterschiedlichen Nutzern und Leistungserbringern erlauben, Daten flexibel und unter Wahrung der Datensicherheit auszutauschen. Dabei wird besonderer Wert auf die Einhaltung medizinischer und technischer Standards gelegt. Um die Zuverlässigkeit des Systems zu erhöhen und Fehler oder Ausfälle bis zu einem gewissen Grad kompensieren zu können, werden Mechanismen zur Fehlertoleranz eingesetzt. Zur Gewährleistung der Datensicherheit wird auf etablierte Verfahren und Sicherungsmaßnahmen zurückgegriffen. Ein automatisiertes technisches Management auf der Grundlage einer formalen Systembeschreibung ermöglicht kontrollierte Systemanpassungen an veränderte Bedingungen und Anforderungen. Basis des Managements sind Policies, die eine zusätzliche, oberhalb des Programmcodes liegende Steuerungsebene bilden. Policies sind nicht Teil des Programmcodes und können somit flexibel zur Laufzeit ausgetauscht werden. Unterschieden wird zwischen abstrakten High-Level und techniknahen Low-Level Policies. Da eingebetteten Systemen die Ressourcen für eine komplexe Policy-Behandlung fehlen, wird der Management-Prozess in die Planungs- und die Laufzeitphase unterteilt. In der Planungsphase werden die Policies auf unterschiedlichen Abstraktionsebenen geplant und

automatisiert in techniknahe Low-Level Policies übersetzt. In der Laufzeitphase setzt dann ein leichtgewichtiges und effizientes Management die Low-Level Policies durch.

Dieses Papier ist wie folgt gegliedert: Kapitel 2 gibt eine kurze Übersicht über den Bereich des technischen Managements. Kapitel 3 beschreibt das Konzept der Service-orientierten Architektur und einige ihrer Umsetzungen. Im Kapitel 4 werden verwandte Arbeiten vorgestellt. Ausgehend von einem Beispielszenario erfolgt im Kapitel 5 die Darstellung des Systemmodells. Kapitel 6 und 7 beschreiben das Policy-gesteuerte Management in der Planungsphase und in der Laufzeitphase. Abschließend erfolgt im Kapitel 8 eine Zusammenfassung und ein Ausblick.

2 Technisches Management

Bestrebungen das technische Management zu standardisieren, führten zu einer Vielzahl unterschiedlicher Management-Konzepte:

OSI-Management Die ersten Standardisierungen fanden durch die ISO im Rahmen der Festlegung des OSI-Referenzmodells statt. Innerhalb des OSI-Managements [ISO89] werden fünf Funktionsbereiche (*FCAPS*) festgelegt: Fehlermanagement, Konfigurationsmanagement, Abrechnungsmanagement, Leistungsmanagement und Sicherheitsmanagement. *Management-Objekte* sind definiert als die OSI-Management-Sicht einer Ressource. Sie verfügen über Attribute zur Zustandsabbildung und über Management-Operationen. Alle Management-Objekte einer Ressource sind in ihrer *Management Information Base* (MIB) hinterlegt, die die Objekte strukturiert und damit eine einheitliche Sicht auf die für das Management relevanten Informationen bietet.

Web-Based Enterprise Management (WBEM) Das Web-Based Enterprise Management [Dis09] ist ein plattform- und ressourcenunabhängiger Standard der *Distributed Management Task Force* (DMTF), der ein gemeinsames Modell für das Management verteilter Systeme definiert. Kern von WBEM ist das *Common Information Model* (CIM), ein objektorientiertes Informationsmodell zur einheitlichen Beschreibung von Management-Informationen und -Funktionen. Es ermöglicht die Abbildung physikalischer und logischer Objekte. WBEM verfolgt konsequent einen objektorientierten Ansatz. Alle Ressourcen werden durch Objekte repräsentiert, die wiederum zu Klassen zusammengefasst sind.

Web Services Distributed Management (WSDM) Das Web Services Distributed Management [OAS06] wurde von der *Organization for the Advancement of Structured Information Standards* (OASIS) für das Management Service-orientierter Architekturen entwickelt. WSDM ist ein plattform- und herstellerunabhängiger Ansatz zur Definition von Management-Schnittstellen für Ressourcen. Die WSDM-Spezifikationen basieren auf Web Services und richten sich hauptsächlich an zwei Bereiche: Das *Management Using Web Services* (MUWS) behandelt das Management von Ressourcen unter Verwendung von Web Services, das *Management of Web Services* (MOWS) definiert ein Modell für das Management von Web Services.

Policy-basiertes Management Beim Policy-basierten Management werden von den Unternehmenszielen ausgehend abstrakte Policies ermittelt. Policies legen das gewünschte Systemverhalten fest und existieren auf unterschiedlichen Abstraktionsebenen [Wie94].

Um zu ausführbaren, technischen Policies zu gelangen, muss eine abstrakte Policy über mehrere Schritte zu Low-Level Policies übersetzt werden. Dieser Prozess wird als Policy-Verfeinerung bezeichnet. Es handelt sich um eine Top-Down Vorgehensweise, die ausgehend von den Unternehmenszielen konkrete, technische Regeln ableitet. Beim herkömmlichen Management versucht man hingegen, die Unternehmensziele Bottom-Up zu erreichen.

3 Service-orientierte Architekturen

Service-orientierte Architekturen (SOA) stellen das abstrakte Konzept einer Systemarchitektur dar, in der Dienste flexibel angeboten, gesucht und genutzt werden können. Die Implementierungsdetails der Dienste (verwendete Programmiersprache, zugrundeliegendes Betriebssystem, usw.) bleiben dem Client bei der Nutzung verborgen, dieser kennt lediglich die bereitgestellten Schnittstellen. Diese werden mit Hilfe einer Schnittstellenbeschreibungssprache beschrieben und in einem Dienstverzeichnis veröffentlicht. Potentielle Clients können dort nach Diensten suchen, an deren Nutzung sie interessiert sind.

Devices Profile for Web Services (DPWS) Eine weit verbreitete Umsetzung des SOA-Konzepts sind *Web Services*. Diese werden durch das World Wide Web Consortium (W3C) in Form einer Zusammenstellung plattformunabhängiger Standards spezifiziert. Aufsetzend auf einer Auswahl der *WS-**-Spezifikationen definiert das *Devices Profile for Web Services* (DPWS) [Web09] einen Standard für den Einsatz von Web Services auf ressourcenbeschränkter Hardware wie z. B. eingebetteten Systemen. DPWS definiert insbesondere *Hosting Services*, die zur Repräsentation von Geräten (*Devices*) dienen. Ferner existieren *Hosted Services*, die die eigentlichen Dienstfunktionen realisieren und von Hosting Services bereitgestellt werden. Anstelle eines „klassischen" Dienstverzeichnisses wird *WS-Discovery* verwendet. Diese Spezifikation definiert ein Protokoll zum dynamischen Anmelden, Suchen und Abmelden von Diensten in einem Netzwerk. Eine zweite in DPWS vorgenommene Erweiterung ist das Einführen ereignisbasierter Kommunikation, spezifiziert mittels *WS-Eventing*. Eine Implementierung der DPWS-Spezifikation ist der *Java Multi Edition DPWS Stack* (JMEDS) [WS4] der *Web Services for Devices* Initiative.

OSGi Die OSGi-Spezifikation [OSG07] definiert eine Service-orientierte Technologie unter besonderer Berücksichtigung des Lifecycle-Managements von Software-Komponenten. Im Kern der Spezifikation befindet sich das *OSGi Framework*, das die Ausführungsplattform für Java-basierte Softwarekomponenten, sog. *Bundles*, ist. Dieses Framework ermöglicht insbesondere das flexible Installieren, Starten, Stoppen, Deinstallieren und Aktualisieren von Bundles zur Laufzeit, ohne das Framework selbst stoppen zu müssen. Bundles können Funktionen in Form von *Services* bereitstellen. Zum Veröffentlichen und Suchen von Services innerhalb des OSGi Frameworks wird eine *Service Registry* verwendet. Es existiert ferner ein *Service Tracker*, der registrierte Listener über Änderungen innerhalb der Service Registry informiert. OSGi ermöglicht ebenfalls die ereignisbasierte Kommunikation. Hierfür existiert der *Event Admin Service*.

Service Component Architecture (SCA) Die *Service Component Architecture* (SCA) [Ope07] spezifiziert ein Konzept zur Entwicklung von Anwendungen und Systemen nach den Prinzipien der Service-orientierten Architektur. Basis sind offene Standards wie z. B.

Web Services. Die kleinste Einheit innerhalb der SCA ist die *Component*. Eine Component ist eine Implementierung einer speziellen Funktionalität (*Business Logic*) und stellt diese über externe Schnittstellen (*Services*) zur Verfügung. Die Beschreibung der Schnittstellen ist von der Technologie abhängig, mittels derer die Services realisiert werden. Benötigt eine Component zur Erbringung ihrer Funktionalität Services, die von anderen Components bereitgestellt werden, werden diese mittels *References* definiert. Die Protokolle, die für den Datenaustausch zwischen Reference und Service notwendig sind, werden mit Hilfe von *Bindings* beschrieben. Logisch verbunden wird eine Reference mit einem Service durch ein *Wire*. Außerdem können für eine Component sog. *Properties* definiert werden, die bei ihrer Instanziierung aus einer Konfigurationsdatei gelesen und entsprechend gesetzt werden. Mehrere Components können logisch zu *Composites* zusammengefasst werden, mehrere Composites auf nächsthöherer Ebene zu *Domains*. Dieses ermöglicht die Gruppierung von Components zu einer logischen Einheit, die zur Erbringung einer bestimmten Geschäftsfunktion notwendig sind.

4 Verwandte Arbeiten

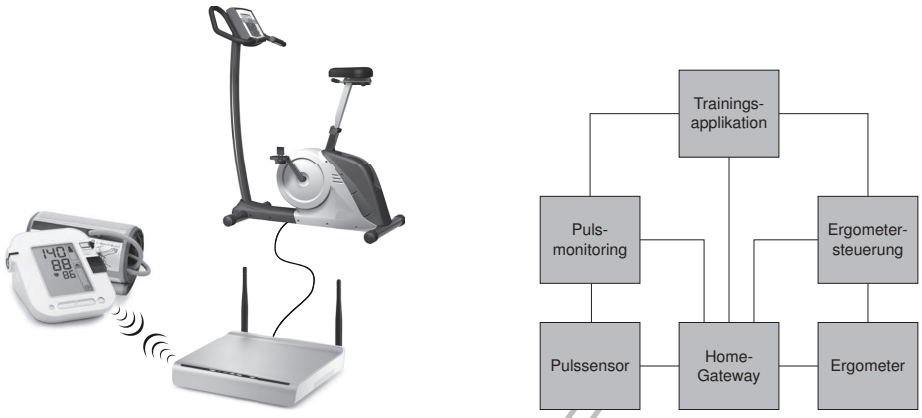
Ein Policy-gesteuertes Management von medizinischen Systemen und Anwendungen wurde bereits in einigen Projekten realisiert.

Im Projekt AMUSE [LDS+08] wurde basierend auf *Body Sensor Networks (BSNs)* das Architekturmuster der *Self-Managed Cell* (SMC) entworfen. In einer SMC wird der Management-Regelkreis anhand von Policies ausgeführt. *Obligation Policies* stellen *Event-Condition-Action* (ECA) Regeln dar und definieren Aktionen, die in Folge auf bestimmte Ereignisse ausgeführt werden müssen. *Authorization Policies* spezifizieren, welche Aktionen auf welchen Ressourcen erlaubt sind. Policies können zur Laufzeit hinzugefügt, gelöscht, aktiviert und deaktiviert werden. Das eingesetzte Werkzeug *Ponder2* [TL07] unterstützt die Spezifikation und den Einsatz von in XML dargestellten Policies [KTP+07].

Ziel des Projekts CareGrid [RDD07] war die Entwicklung eines Frameworks zur Umsetzung eines autonomen Policy-gesteuerten Managements im E-Health-Bereich. Die Formulierung der Policies erfolgte mittels Ponder2, besonderes Augenmerk wurde auf die automatische Konfliktauflösung der Authorization Policies gelegt.

Das Projekt MATCH [WT08] beschäftigt sich mit der Entwicklung von Heimpflegesystemen, die Personalisierung, Adaptierbarkeit und Zuverlässigkeit gewährleisten. Zur Definition der hier verwendeten Policies wurde eine eigens entwickelte, XML-basierte Sprache *APPEL* verwendet. Sie wurde für diese Anwendungsdomäne spezialisiert, indem für Heimpfelegesysteme typische Trigger, Bedingungen und Aktionen definiert wurden. Um mögliche Konflikte zwischen ECA-Regeln aufzulösen, wurden *Resolution Policies* [TCW07, WDT+06] eingeführt.

Im Vergleich zu diesen Ansätzen verwenden wir Policies auf unterschiedlichen Abstraktionsebenen, die in einer vorbereitenden Planungsphase automatisiert verfeinert werden. Zur Laufzeit werden keine Skriptsprachen schwergewichtig interpretiert, sondern übersetzte und für die einzelnen Manager-Komponenten individuell zusammengestellte Policy-Module ausgeführt. Weiterhin grenzen wir Anwendungsprogrammierung und Policies stärker von-



(a) Physikalischer Aufbau

(b) Komponenten und Assoziationen

Abbildung 1: Beispielszenario

einander ab. Wir verwenden ECA-Regeln nur als abgeleitete Low-Level Policies und stellen sie nicht zur Definition der abstrakten Policies zur Verfügung. Die abstrakten Policies definieren keine Abläufe, sondern Spielräume, Präferenzen und Gewichtungen. Dies wirkt der Vermischung von Anwendungslogik und Policy-Steuerung entgegen.

5 Systemmodell

Zur Umsetzung adaptiver, zuverlässiger und flexibler Systeme setzt das technische Management ein geeignetes Systemmodell voraus. Dieses kann als eine Gruppe von Objekten aufgefasst werden, die untereinander in Beziehung stehen. Ausgehend von einem Beispielszenario aus der kardiologischen Rehabilitation wird im Folgenden ein Systemmodell entwickelt, das ein in Anlehnung an klassische Management-Konzepte standardkonformes und effizientes Management ermöglicht.

Beispielszenario Ein Patient soll nach einem stationären Klinkaufenthalt in seiner häuslichen Umgebung ein überwachtes Cardiotraining durchführen. Ein individueller Trainingsplan gibt die Trainingsart, die Trainingsdauer und die Grenzwerte einzelner Vitalparameter (z. B. Puls oder Blutdruck) vor. Die für das Training erforderliche Hard- und Software wird ihm von der Klinik bereitgestellt. Dies umfasst die medizinischen Sensoren, ein Ergometer sowie ein Home-Gateway, das als Ausführungsplattform für die Software dient und den Verbindungsaufbau zur Klinik ermöglicht. Die Abbildung 1a zeigt einen Ausschnitt aus der medizinischen Anwendungsdomäne. Aus Vereinfachungsgründen wird angenommen, dass für die Überwachung der Vitalparameter lediglich ein Pulssensor verwendet wird.

In unserem Systemmodell stellen die unterschiedlichen Elemente des Szenarios, wie z. B. Pulssensor, Home-Gateway, Ergometer, Pulsmonitoring, Ergometersteuerung und Trainingsapplikation, *Komponenten* dar. Logische und physikalische Beziehungen zwischen Komponenten werden durch *Assoziationen* modelliert. Das Systemmodell ist in der Abbildung 1b dargestellt.

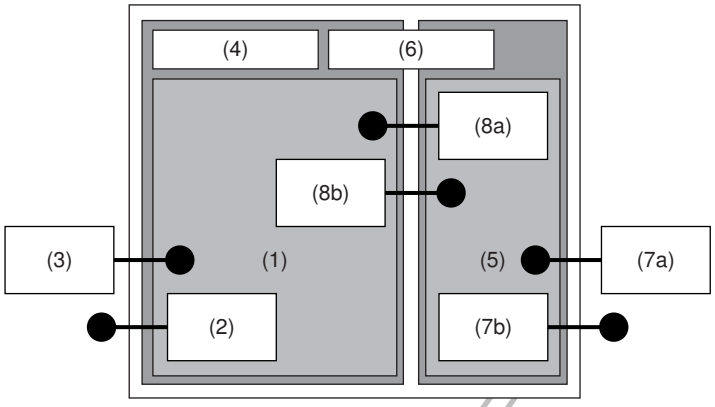


Abbildung 2: Aufbau einer Komponente

5.1 Komponenten

Unter einer Komponente wird ein Systemelement mit zugehörigen Metainformationen verstanden. Eine besondere Stellung haben Software-Komponenten, die als OSGi Bundles implementiert werden können. Die Komponenten führen Anwendungsfunktionen aus und stellen Services in Form von OSGi- oder DPWS-Services zur Verfügung. Ein Service implementiert eine Schnittstelle, über die Operationen und Ereignisse zugreifbar sind. Eine Software-Komponente lässt sich unterteilen in einen *Anwendungs-Teil* zur Erbringung ihrer Nutzfunktion und einen *Management-Teil* zur Verwaltung, Überwachung und Steuerung. Diese Komponentenstruktur ist in der Abbildung 2 dargestellt. Innerhalb des Anwendungs-Teils *(1)* kann sich der Komponentenenwickler auf die Implementierung der Nutzfunktion konzentrieren und beliebig viele Services anbieten *(2)* oder nutzen *(3)*. Des Weiteren wird er bei der Definition von *Anwendungs-Variablen (4)* nicht eingeschränkt. Der Management-Teil *(5)* dient als Proxy zum Komponenten-Manager *(7a, 7b)*, der für die Überwachung und Konfiguration der Komponente zuständig ist. Zu diesem Zweck muss der Komponentenentwickler den aus Managementsicht sichtbaren Zustandsraum spezifizieren. Er definiert ein MIB-ähnliches Schema und legt eine Menge von *Management-Variablen (6)* fest, die vom Komponenten-Manager gelesen werden können. Die Management-Variablen zerfallen in zwei Gruppen: die *Status-Variablen* und die *Konfigurations-Variablen*. Status-Variablen werden ausschließlich aus dem Anwendungs-Teil heraus gesetzt und können vom Komponenten-Manager lediglich gelesen, nicht aber geschrieben werden. Sie repräsentieren den Management-relevanten Zustand der Anwendung. Konfigurations-Variablen werden ausschließlich vom Komponenten-Manager gesetzt. Im Sinne des MUWS-Ansatzes stellt der Management-Teil dafür eine Menge von Services bereit, die den Zugriff auf die Management-Variablen ermöglichen. Die Verbindung zwischen Anwendungs- und Management-Teil wird durch eine schmale Schnittstelle realisiert. Sie ermöglicht das atomare Setzen von Status-Variablen *(8a)* und die Notifikation des Anwendungs-Teils bei Änderungen von Konfigurations-Variablen *(8b)*.

5.2 Assoziationen

Eine Assoziation beschreibt die Relation zwischen zwei Komponenten. Dabei lassen sich verschiedene Assoziationstypen unterscheiden, wie z. B. Service-Nutzung und Kommunikation sowie örtliche und administrative Zuordnungen. Des Weiteren besitzt eine Assoziation einen Status. Sie kann sich beispielsweise im Aufbau oder Abbau befinden. Eine Assoziation verläuft zwischen so genannten *Endpunkten*. Sie dienen der Adressierung und komponentenseitigen Verwaltung. Für die Überwachung und Unterstützung der Assoziationen steht ein einheitliches Assoziationsmanagement zur Verfügung. Es unterscheidet zwischen statischen und dynamischen Assoziationen. Im Gegensatz zu statischen Assoziationen werden dynamische Assoziationen zur Laufzeit aufgebaut und erlauben je nach Typ eine transparente Ersetzung. Innerhalb einer dynamischen SOA-Umgebung, in der Komponenten gestartet und wieder beendet werden, ermöglichen dynamische Assoziationen die lose Kopplung von Komponenten. Demgegenüber steht jedoch die Forderung nach Einhaltung von Gütekriterien, die planbare Strukturen zur Vereinbarung und Überprüfung voraussetzt. Zu diesem Zweck findet ein erweitertes Lease-Konzept Anwendung.

Leases Ein *Lease* ist eine zeitlich befristete Vereinbarung zwischen zwei Parteien [GC89]. Dem Lease-Inhaber wird vom Lease-Anbieter für einen gewissen Zeitraum eine Menge von Eigenschaften zugesichert. Das Lease kann lediglich vom Lease-Inhaber vorzeitig freigegeben werden. Lease-basierte Interaktionen durchlaufen drei Phasen: die *Kopplungs-*, *Interaktions-* und *Entkopplungsphase*. In der Kopplungsphase wird zunächst zwischen einem Client und einer Komponente eine Vereinbarung bzgl. geforderter Eigenschaften und Lease-Dauer getroffen. In der anschließenden Interaktionsphase kann der Client die Komponente nutzen. Leases können durch den Client verlängert werden, allerdings kann die Komponente eine solche Verlängerung ablehnen. Der Prozess geht in die Entkopplungsphase über, wenn das Lease ausläuft oder der Client das Lease freigibt. Die Anzahl der Leases, die gleichzeitig vereinbart werden können, hängt von der Komponente selbst und ihrem aktuellen Zustand ab: Es ist möglich, dass eine Komponente nur eine exklusive Nutzung gestattet und somit maximal ein Lease eingeht. Andere Komponenten erlauben eventuell eine Mehrfachnutzung, wobei die Anzahl der Leases, die gleichzeitig vereinbart werden können, beschränkt oder unbeschränkt sein kann. Leases bewahren Komponenten nicht vor einem Ausfall, führen aber zu einem vorhersagbaren und planbaren Systemverhalten.

Ensembles Häufig benötigt ein Client mehrere Leases im engen Zusammenhang. Wir sprechen dann von einem *Ensemble* [PKH+08]. Der Client ist darauf angewiesen, dass ihm alle Elemente des Ensembles zur Verfügung stehen. Die Leases eines Ensembles werden atomar in einem Zwei-Phasen-Commit-Verfahren vereinbart. In der ersten Phase fordert der Client Lease-Angebote aller Ensemble-Mitglieder an. Jeder Anbieter ist für eine kurze Reservierungszeitspanne gebunden und muss bereit sein, einen Lease-Abruf des Clients zu akzeptieren. Erhält ein Client für ein gewünschtes Ensemble nicht alle erforderlichen Lease-Angebote, lässt er die Reservierungen verfallen und weicht z. B. auf ein alternatives Ensemble aus.

6 Modellbasiertes Management

Das in [IKL+06] entwickelte Konzept des modellbasierten Managements wird an dieser Stelle aufgegriffen und verallgemeinert. Das modellbasierte Management ermöglicht die automatisierte, schrittweise Verfeinerung von Policies entlang eines Modells, das aus drei hierarchisch angeordneten Abstraktionsebenen besteht. Die Policies einer Ebene beziehen sich jeweils auf die modellierten Systemelemente derselben Ebene. Der Prozess des modellbasierten Managements gliedert sich in zwei Phasen: In der ersten Phase, der *Planungsphase*, wird das reale System in allen drei Abstraktionsebenen modelliert. Das entstehende dreischichtige Modell bildet die Grundlage für die Definition abstrakter High-Level Policies und ihre automatisierte Ableitung zu techniknahen Low-Level Policies. Mittels dieser Policies wird in der anschließenden zweiten Phase, der *Laufzeitphase*, das Management durchgeführt. Die Modellierung des realen Systems und die Ableitung der Low-Level Policies wird durch ein Werkzeug unterstützt.

6.1 Modell und Abstraktionsebenen

Das reale System wird auf drei Abstraktionsebenen modelliert und pro Ebene durch ein in sich geschlossenes Modell repräsentiert. Da jede Ebene dasselbe System abbildet, jedoch mit unterschiedlichem Abstraktionsgrad, existiert zu einem abstrakten Element einer Ebene in der jeweils darunter liegenden Ebene eine entsprechende Menge konkreter Elemente. Es lassen sich die folgenden Ebenen unterschieden:

- *Use Cases & Aspects*: In der obersten, abstraktesten Ebene stehen technische Gesichtspunkte im Hintergrund. Hier werden die vorhandenen Anwendungsfälle mit ihren *Aspekten* modelliert. Ein möglicher Anwendungsfall ist z. B. „Cardiotraining“ mit den Aspekten „Datensicherheit“ und „Zuverlässigkeit“.
- *Services & Domains*: Hier wird das System Service-orientiert als Struktur aus Clients, Services und Nutzungsbeziehungen dargestellt. Sie sind *Domänen* zugeordnet. Im Beispiel tritt die Trainingsapplikation als Client der Dienste „Ergometerdienst“ und „Pulssensordienst“ auf. Sie befinden sich in der Domäne „Patientenwohnung“.
- *Components & Devices*: Auf der untersten Ebene werden die konkreten Komponenten und Geräte des Systems mit ihren technischen Details dargestellt. Die Modellierung orientiert sich an der Konfigurierung und stellt insbesondere alle Konfigurationsparameter dar. Das im Kapitel 5 vorgestellte Szenario und seine Abbildung durch Komponenten und Geräte ist ein entsprechendes Beispiel.

6.2 Policy-Verfeinerung

Die Policy-Verfeinerung ist die schrittweise Übersetzung von abstrakten Policies in techniknahe Low-Level Policies. Die Übersetzung wird durch *Verfeinerungsbeziehungen* gesteuert. Eine Verfeinerungsbeziehung verbindet ein Modellelement mit denjenigen Elementen des Modells der nächsttieferen Abstraktionsebene, welche dieses Element in der tieferen Ebene modellieren. Eine auf der „Use Cases & Aspects“-Ebene modellierte abstrakte Operation steht z. B. in Verfeinerungsbeziehung mit denjenigen Diensten der „Services & Domains“-Ebene, welche zur Durchführung dieser abstrakten Operation erforderlich sind. Der De-

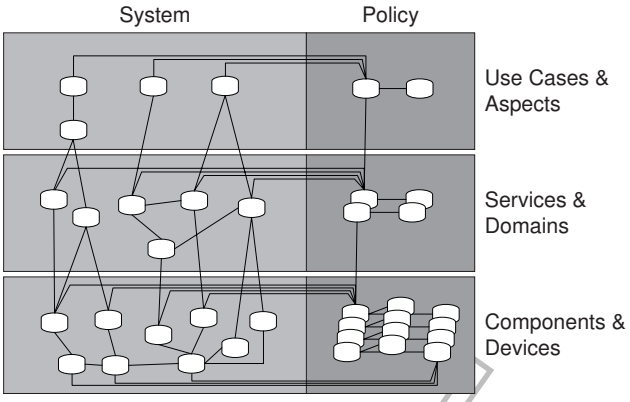


Abbildung 3: Abstraktionsebenen und Policy-Verfeinerung

tailierungsgrad der Policies nimmt mit jedem Verfeinerungsschritt zu, so dass Low-Level Policies sich schließlich auf die Management-Variablen der Komponenten beziehen. Im Beispiel besteht die abstrakte Sicherheitsanforderung nach hoher Vertraulichkeit. Sie wird auf entsprechende Schutzanforderungen der betroffenen Service-Nutzungen abgebildet. In den Low-Level Policies führt dies zu Assoziationsanforderungen nach Kryptoverfahren und Mindestschlüssellängen. Der Zusammenhang zwischen Modellebenen und den verfeinerten Policies ist in der Abbildung 3 dargestellt.

6.3 Werkzeug-Unterstützung

Die Policy-Verfeinerung wird automatisiert mit dem in Dortmund entwickelten Java-basierten Werkzeug *MoBaSeC* (*Model Based Service Configuration*) durchgeführt. Es unterstützt den Anwender bei der Modellierung der Abstraktionsebenen, der Definition der abstrakten Policies und ihrer automatisierten Ableitung zu konkreten Policies. In MoBaSeC wird dazu ein Modell aus Knoten und Kanten entwickelt, in dem die Knoten die Elemente des Systems bilden, und die Kanten Beziehungen zwischen diesen Elementen darstellen. Sowohl Knoten als auch Kanten sind typisiert; der Typ eines Knoten definiert, um welche Art von Element es sich handelt (z. B. einen Use-Case auf abstrakter Ebene oder eine Dienstimplementierung auf konkreter Ebene), der Typ einer Kante definiert, welche Beziehung zwischen den verbundenen Elementen vorliegt (z. B. eine Konkretisierungsbeziehung zwischen Elementen verschiedener Modellebenen). Die möglichen Typen der Knoten und Kanten, die bei der Modellierung benutzt werden können, werden in MoBaSeC über ein *Metamodell* definiert. Grundlage für dieses Metamodell ist ein an CIM-orientiertes Informationsmodell, das für das jeweilige Management-Szenario zu entwickeln ist. Die Knoten- und Kantentypen sind Java-Klassen, deren Instanzen die Knoten und Kanten des Modells bilden. Ferner kann in einem Metamodell eine Prüffunktion implementiert werden, die es ermöglicht, Modelle auf Konsistenz hinsichtlich aufgestellter Bedingungen zu überprüfen. Durch die Austauschbarkeit des Metamodells kann MoBaSeC für unterschiedliche Management-Aufgaben eingesetzt werden.

7 Policy-gesteuertes Laufzeit-Management

Das Laufzeit-Management wird von einer *Manager-Hierarchie* ausgeführt. Es wird zwischen den folgenden Manager-Typen unterschieden: Komponenten-, Service- und Use-Case-Manager. Lediglich Komponenten-Manager interagieren direkt mit den Komponenten. Komponenten-Manager sind Service-Managern und Service-Manager wiederum Use-Case-Managern unterstellt. Manager verwalten und kontrollieren die ihnen direkt unterstellten Manager. Um dies zu ermöglichen, sind Manager nach dem Komponentenprinzip realisiert. Komponenten-Manager können beliebig viele Komponenten managen, jede Komponente ist aber exklusiv einem Komponenten-Manager zugeordnet. Die Zuordnung der Manager untereinander ist hingegen nicht exklusiv, da ein Service beispielsweise von unterschiedlichen Komponenten bereitgestellt werden kann. Die Manager-Hierarchie und die Zuordnung der Komponenten zu ihrem Komponenten-Manager werden mittels MoBaSeC geplant, das ebenfalls die abgeleiteten Low-Level Policies an die jeweiligen Manager verteilt.

7.1 Management-Interaktionen

Das Management von Komponenten wird unmittelbar von ihren Komponenten-Managern durchgeführt. Der Management-Teil einer Komponente dient als Proxy zum zuständigen Komponenten-Manager. Somit lassen sich die Management-Interaktionen in Interaktionen zwischen Anwendungs- und Management-Teil und Interaktionen zwischen Management-Teil und Manager aufteilen. Dabei unterscheidet man die folgenden Fälle:

- Eine Komponente fordert aus ihrem Anwendungs-Teil von ihrem Management-Teil eine Policy-Auswertung bzw.-Entscheidung an, die der Manager unter Berücksichtigung der aktuellen Umgebungsbedingung durchführen bzw. treffen soll.
- Eine Komponente nutzt assoziationsspezifische Operationen des Management-Teils.
- Eine Komponente setzt atomar eine Menge von Status-Variablen und der Management-Teil erzeugt ein entsprechendes Ereignis.
- Der Management-Teil setzt atomar eine Menge von Konfigurations-Variablen und notifiziert den Anwendungs-Teil.

Die Interaktionen zwischen Management-Teil und Manager sind die folgenden:

- Der Management-Teil bietet einen Komponenten-Management-Service, der von einem Manager genutzt werden kann. Der Service ermöglicht das Lesen und Schreiben von Konfigurations-Variablen sowie das Lesen von Status-Variablen. Darüber hinaus kann sich ein Manager bei diesem Service für Ereignisse registrieren.
- Manager bieten Komponenten einen Policy-Service für Policy-Auswertungen und -Entscheidungen und einen Assoziations-Service an.

7.2 Policy-Typen

Low-Level Policies sind definiert über Management-Variablen und Ereignisparametern sowie Datentypoperationen und Konstanten. Sie haben die Form von Bedingungen, Ausdrücken und Zuweisungen. Zur Laufzeit werden sie durch effizient ausführbaren Bytecode

dargestellt. Die zugehörigen Klassen werden durch MoBaSeC mit einer Backend-Funktion generiert. Es gibt folgende Policy-Elemente:

PolicyExpression Eine PolicyExpression stellt einen Ausdruck über Management-Variablen, Konstanten und Operationen dar. Sie wird auf Anforderung ausgewertet und dient der Beurteilung des aktuellen Systemzustands. Eine Komponente kann in Abhängigkeit zum Ergebnis einer Policy-Auswertung ihren Programmablauf entsprechend bestimmen. Der Komponenten-Entwickler legt fest, welche PolicyExpressions mit welchen Rückgabetypen verwendet werden, und beschreibt die Bestimmung der Auswertung in textueller Form. Beispielsweise kann die Trainingsapplikation die Einstellungen für die nächste Trainingsphase erfragen.

PolicyCondition Die PolicyCondition ist ein Spezialfall der PolicyExpression. Der bei der PolicyExpression frei definierbare Rückgabetyp ist hier auf einen Wahrheitswert festgelegt. Policy-Entscheidungen werden anhand von PolicyConditions getroffen. Beispielsweise kann die Trainingsapplikation die Entscheidung anfordern, ob das Training abzubrechen ist.

PolicyRule PolicyRules stellen ECA-Regeln dar. Beim Eintreffen eines Ereignisses werden die Bedingungen der zugehörigen PolicyRules ausgewertet und je nach Ergebnis die Aktionen ausgeführt. Die Aktionen sind beschränkt auf das Setzen von Konfigurations-Variablen. Mittels PolicyRules wird ein reaktives Management zur Gewährleistung eines zuverlässigen und adaptiven Systemverhaltens umgesetzt. Beispielsweise kann als Reaktion auf eine Veränderung des Batteriestatus einer Sensorkomponente die Abfragerate des Sensors angepasst werden.

AssociationRequirements AssociationRequirements dienen der Formulierung von Anforderungen an Assoziationen und stellen Bedingungen über Management-Variablen dar. Beim Assoziationsaufbau werden die AssociationRequirements der beiden Endpunkte geschnitten. Diese Schnittmenge legt den Spielraum für die Konfiguration der Endpunkte fest. Beispielsweise können AssociationRequirements für einen Endpunkt A festlegen, dass die von dort ausgehenden Kommunikationskanäle mit einer Stärke von 128 oder 256 Bit verschlüsselt sein müssen. AssociationRequirements für einen Endpunkt B legen eine Verschlüsselungsstärke von 256 Bit fest. Die Schnittmenge und somit die Verschlüsselungsstärke für einen zwischen A und B aufzubauenden Kommunikationskanal ist dementsprechend 256 Bit.

7.3 Assoziationsmanagement

Die Aufgaben des Assoziationsmanagements sind Aufbau, Überwachung und Abbau von Assoziationen und Ensembles. Es basiert auf dem im Abschnitt 5.2 vorgestellten Lease-Prinzip. Grundlage dafür sind die AssociationRequirements der jeweiligen Endpunkte. Der Assoziationsaufbau geschieht in vier Phasen: In der ersten Phasen werden funktional-passende Assoziationspartner gesucht, in der zweiten Phase nicht-funktional-passende bestimmt und in der dritten Phase anhand von Qualitätskriterien bewertet. In der vierten Phase erfolgt der eigentliche Assoziationsaufbau zum ausgewählten Partner, d. h. die Konfiguration der Endpunkte. Zur Gütesicherung überwacht das Assoziationsmanagement die aufgebauten Assoziationen. Werden Gütezusicherungen verletzt, greift das Assoziations-

management steuernd ein. Ein solcher Eingriff kann beispielsweise die Ersetzung einer Assoziation sein. Soll eine Assoziation abgebaut werden, benachrichtigt das Assoziationsmanagement die beteiligten Partner, so dass die von dieser Assoziation belegten Ressourcen wieder freigegeben werden können.

Fordert eine Komponente den Aufbau eines Ensembles an, übernimmt das Assoziationsmanagement den Aufbau der Assoziationen zu den einzelnen Ensemble-Teilnehmern. Verletzt eine der Assoziationen ihre Gütezusicherungen, versucht das Assoziationsmanagement zunächst das Ensemble zu rekonfigurieren. Gelingt dies nicht, wird das Ensemble abgebaut. Beim Abbau eines Ensembles werden alle daran beteiligten Assoziationen kontrolliert abgebaut.

8 Zusammenfassung und Ausblick

In diesem Papier wurde das Konzept des Policy-gesteuerten Managements im Projekt OSAMI Commons vorgestellt. Basis für das Management ist das aus Komponenten und Assoziationen bestehende Systemmodell. Komponenten bestehen aus Anwendungs- und Management-Teilen, wobei letztere als Proxies zu Managern dienen. Assoziationen bilden Beziehungen zwischen Komponenten ab. Zur Gewährleistung eines zuverlässigen und adaptiven Systemverhaltens werden Policies eingesetzt, die in der Planungsphase als High-Level Policies erstellt und automatisiert zu Low-Level Policies verfeinert werden. Grundlage dafür ist ein Modell, das das reale System auf drei hierarchisch angeordneten Abstraktionsebenen abbildet. In der Laufzeitphase wird das Management leichtgewichtig mittels der Low-Level Policies durchgeführt, die über Management-Variablen und Ereignisparametern sowie Datentypoperationen und Konstanten definiert sind. Low-Level Policies werden unterschieden in PolicyExpressions, PolicyConditions, PolicyRules und AssociationRequirements. ECA-Regeln stehen somit ausschließlich zur Laufzeit zur Verfügung, während in der Planungsphase lediglich Spielräume, Präferenzen und Gewichtungen festgelegt werden. Dies wirkt der Vermischung von Anwendungslogik und Policy-Steuerung entgegen.

Innerhalb von OSAMI-D wird das Zusammenspiel der entwickelten Komponenten anhand eines E-Health-Demonstrators gezeigt. Die Implementierung und Erprobung des hier vorgestellten Management-Konzepts erfolgt im Rahmen dieses Demonstrators. Ferner ist beabsichtigt, das Management-Konzept auf die Anwendungsdomänen der anderen nationalen OSAMI-Teilprojekte zu übertragen.

Literatur

[Dis09] Distributed Management Task Force. *Web-Based Enterprise Management (WBEM)*, 2009. http://www.dmtf.org/standards/wbem.

[GC89] Cary G. Gray und David 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)*, Seiten 202–210. ACM Press, 1989.

[IKL+06] Stefan Illner, Heiko Krumm, Ingo 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), Seiten 519–523. IEEE Computer Society, 2006.

[ISO89] ISO. ISO/IEC 7498-4: Information Processing Systems – Open Systems Interconnection – Basic Reference Model – Part 4: Management Framework, 1989.

[KTP+07] Sye Keoh, Kevin Twidle, Nathaniel Pryce et al. Policy-based Management for Body-Sensor Networks. In *Proc. of the 4th Int. Workshop on Wearable and Implantable Body Sensor Networks (BSN '07)*, Seiten 92–98. Springer-Verlag, 2007.

[LDS+08] Emil Lupu, Naranker Dulay, Morris Sloman et al. AMUSE: Autonomic Management of Ubiquitous e-Health Systems. *Concurrency and Computation: Practice and Experience*, 20:277–295, 2008.

[OAS06] OASIS Web Services Distributed Management (WSDM) TC. *WSDM 1.1 OASIS Standard Specifications*, 2006.

[Ope07] Open SOA Collaboration (OSOA). *SCA Service Componente Architecture: Assembly Model Specification*, 2007.

[OSG07] OSGi Alliance. *OSGi Service Platform Core Specification – Release 4, Version 4.1*, 2007.

[PKH+08] Andre Pohl, Heiko Krumm, Felix 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)*, Seiten 1393–1398. IEEE Computer Society, 2008.

[RDD07] Giovanni Russello, Changyu Dong und Naranker Dulay. Authorisation and Conflict Resolution for Hierarchical Domains. In *Proc. of the 8th IEEE Int. Workshop on Policies for Distributed Systems and Networks (POLICY '07)*, Seiten 201–210. IEEE Computer Society, 2007.

[TCW07] Kenneth J. Turner, Gemma A. Campbell und Feng Wang. Policies for Sensor Networks and Home Care Networks. In Mohammed Erradi, Hrsg., *Proc. of the 7th Int. Conf. on New Technologies for Distributed Systems (NOTERE '07)*, Seiten 273–284. Hermes Science Publishing, 2007.

[TL07] Kevin Twidle und Emil Lupu. Ponder2 – Policy-Based Self Managed Cells. In *Proc. of the 1st Int. Conf. on Autonomous Infrastructure, Management and Security (AIMS '07)*, Seiten 230–230. Springer-Verlag, 2007.

[WDT+06] Feng Wang, Liam S. Docherty, Kenneth J. Turner et al. Services and Policies for Care at Home. In *Proc. of the 1st Int. Conf. on Pervasive Computing Technologies for Healthcare*, Seiten 1–10. Institution of Electrical and Electronic Engineers Press, 2006.

[Web09] Web Services Discovery and Web Services Devices Profile (WS-DD) TC. *Devices Profile for Web Services Version 1.1*. OASIS, 2009.

[Wie94] René Wies. Policies in Network and System Management – Formal Definition and Architecture. *Journal of Network and System Management*, 2(1):63–83, 1994.

[WS4] WS4D Initiative. *WS4D JMEDS-Stack*. http://www.ws4d.org/?page_id=14.

[WT08] Feng Wang und Kenneth J. Turner. Towards Personalised Home Care Systems. In *Proc. of the 1st Int. Conf. on Pervasive Technologies Related to Assistive Environments (PETRA '08)*, Seiten 1–7. ACM Press, 2008.