In: PIK - Praxis der Informationsverarbeitung und Kommunikation, Nummer 28 / 4, Seite 211-216, 2005

Selbstmanagement-Ansätze im eBusiness-Umfeld

Markus Schmid studierte Informatik an der FH Wiesbaden und am Cork Institute of Technology (CIT) in Irland. Derzeit ist er Wissenschaftlicher Mitarbeiter im Labor für Verteilte Systeme der FH Wiesbaden, wo er sich mit Service Level Management und Selbstmanagement beschäftigt. Er ist Mitglied der GI und der IEEE Computer Society.

Reinhold Kröger studierte Informatik an der Universität Bonn, wo er auch mit einer Arbeit über verteilte Betriebssysteme promovierte. Anschließend war er ca. 10 Jahre in der GMD als Wiss. Mitarbeiter und Projektleiter tätig. Seit 1993 ist er Professor für Betriebssysteme und Verteilte Systeme an der Fachhochschule Wiesbaden – University of Applied Sciences. Dort leitet er das Labor für Verteilte Systeme. Seine Forschungsschwerpunkte sind Management Verteilter Systeme und verteilte Anwendungen in der Automatisierungstechnik. Er ist Mitglied des Leitungsgremiums der GI/ITG-Fachgruppe Betriebssysteme und Mitglied der ACM.

Zusammenfassung

Dieser Beitrag beschreibt Herausforderungen und Lösungsansätze für kooperatives Selbstmanagement von verteilten Anwendungen. Anhand eines lokalen Selbstmanagement-Beispielszenarios wird zunächst die Notwendigkeit für Kooperationen im Selbstmanagement-Kontext erläutert und auf mögliche Probleme eingegangen. Darauf aufbauend werden mögliche Kooperationsformen und -szenarien definiert und das Beispiel um Selbstmanager-Kooperation erweitert. Im Ausblick werden sich aus dieser Kooperation ergebende Herausforderungen auf dynamische SOA-Szenarien projiziert.

1 Einleitung

Service-Provider bieten eine immer größere Vielfalt unterschiedlicher Dienste an; das Spektrum reicht vom einfachen Web-Hosting bis hin zu vollständigen eBusiness-Applikationen. Komplexere verteilte Dienste sind dabei üblicherweise als Multi-Tier-Applikationen realisiert und verwenden bisher etablierte Middleware-Architekturen wie Java 2 Enterprise (J2EE), Microsoft DotNet oder auch die Common Object Request Broker Architecture (CORBA) zur Kommunikation zwischen beteiligten Komponenten (vgl. Abb. 1). Die zuvor genannten Technologien erfordern jedoch eine relativ enge Kopplung, so dass Szenarien, in denen unterschiedliche Provider flexibel interagieren, um gemeinsam einen Dienst zu erbringen (der sich wiederum aus anderen Diensten zusammensetzen kann), nur schwer realisierbar sind.

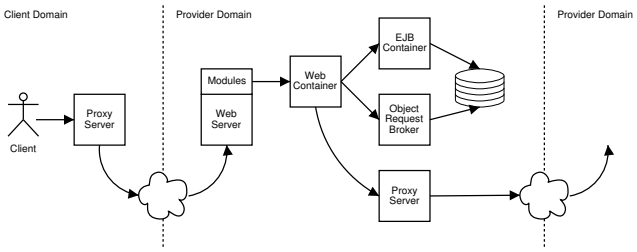

Abbildung 1: Beispiel für die technische Realisierung einer komplexen verteilten Anwendung

Für Szenarien, in denen lose gekoppelte Dienste interagieren, etabliert sich derzeit die Web Services[1] Technologie. Als Web Services bezeichnet man Dienste, die über eine in der Web Services Description Language (WSDL)[2] definierte Schnittstelle verfügen. Diese Schnittstellenbeschreibung wird in einer gemeinsamen Registry abgelegt und steht damit anderen Diensten zur Verfügung, die ggf. erst zur Laufzeit entscheiden, welche Web-Service-Instanz den gewünschten Dienst erbringen soll.

Erst mit der Web-Services-Technologie scheint es zu gelingen, verteilte Anwendungen in größerem Maße unternehmensübergreifend zu realisieren. Aktuell geht man dazu über, im Rahmen der sog. Service Oriented Architecture (SOA) [1] unternehmensübergreifende Geschäftsprozesse zu modellieren, die sich ihrerseits aus unterschiedlichen IT Services zusammensetzen. Der Geschäftsprozess selbst sowie Abhängigkeiten zu IT Services werden in einer abstrakten Beschreibungssprache (z.B. Business Process Execution Language (BPEL)) notiert, die zur Laufzeit von einer zentralen Komponente ausgewertet wird, um den Ablauf des Geschäftsprozesses voranzutreiben ("Orchestration").

Die vertragliche Basis der Auslagerung von Diensten an i.d.R. externe Dienstleister bildet ein sog. Service Level Agreement (SLA) [2, 3]. Dieses definiert u.a. Umfang und Güte (z.B. in Bezug auf Antwortzeit oder Verfügbarkeit) des zu erbringenden Dienstes. Absprachen über die Messung der vereinbarten

[1] Vgl. http://www.w3.org/2002/ws/
[2] Vgl. http://www.w3.org/TR/wsdl.html

Dienstgüte (QoS SLA-Parameter) sind ebenfalls Bestandteil eines SLAs. Service Level Objectives (SLOs) definieren Referenzwerte für SLA-Parameter. Um Verletzungen der Dienstgüte erkennen und darauf basierend entsprechende Reaktionen einleiten zu können, müssen die SLOs zur Laufzeit von einem Managementsystem ausgewertet werden. Dieser Vorgang wird als Service Level Management (SLM) bezeichnet.

Parallel zur Verbreitung von auf eBusiness-Komponenten abgebildeten Geschäftsprozessen ist somit die Bedeutung des SLM gewachsen. Traditionelle Managementsysteme werden hierbei sowohl durch die starke Zunahme von Komponenten innerhalb von verteilten Anwendungen und die damit verbundene Komplexität des zu verwaltenden Umfeldes mit einer Vielzahl sich wandelnder SLAs als auch durch häufige Änderungen der verwendeten Middleware-Technologien vor enorme Herausforderungen gestellt.

Aufgrund der gestiegenen Komplexität verteilter Anwendungen wird unter dem Oberbegriff "Selbstmanagement" (auch Autonomic Computing [4] / Adaptive Enterprise [5]) in letzter Zeit verstärkt nach Möglichkeiten zur Automatisierung von Managementaufgaben bzw. zur Reduktion der nach außen hin sichtbar werdenden Komplexität von Komponenten gesucht (vgl. [6, 7]). Einen guten Überblick aktueller Ansätze im Bereich des Selbstmanagements gibt [8].

Aktuelle Ansätze konzentrieren sich bisher eher auf den Bereich einzelner zu überwachender Komponenten und haben weniger das automatisierte Zusammenspiel unterschiedlicher Selbstmanager zur Verfolgung übergeordneter Ziele im Auge. Hierzu ist die Formalisierung von Managementwissen und -zielen auf unterschiedlichen Abstraktionsstufen sowie das Erkennen und Auflösen von Zielkonflikten bei der Kooperation unterschiedlicher Selbstmanager notwendig.

Dieser Beitrag beschreibt Lösungsansätze für diese Herausforderungen beispielhaft für das Konfigurationsmanagement als Teildisziplin des Selbstmanagements von Anwendungen im verteilten Kontext. Im Folgenden werden zunächst der an der FH Wiesbaden entwickelte Selbstmanager und ein vorhandenes Testbed beschrieben. Danach wird anhand eines lokalen Selbstmanagement-Beispielszenarios die Notwendigkeit für Kooperationen im Selbstmanagement-Kontext erläutert und auf mögliche Probleme eingegangen. Der Beitrag beschreibt dann Prinzipien der Kooperation für den verteilten Fall. Im Ausblick werden sich aus dieser Kooperation ergebende Herausforderungen auf SOA-Szenarien projiziert.

2 Selbstmanagement Testbed

Im Labor für Verteilte Systeme der FH Wiesbaden existiert ein modular aufgebauter Selbstmanager [9] sowie eine Performance-instrumentierte, heterogene eBusiness-Testumgebung [9], die als Anwendungsumfeld für den Selbstmanager dient.

Zunächst werden die wesentlichen Eigenschaften der modularen Selbstmanagementinfrastruktur erläutert, danach wird auf die verwendete eBusiness-Testumgebung eingegangen.

2.1 Der Selbstmanager

Der hier vorgestellte Selbstmanager basiert auf einer von der FH Wiesbaden gemeinsam mit Lufthansa Systems entwickelten, in ihrer Ursprungsform zum Management CORBA-basierter Anwendungen verwendeten Managementlösung. Verglichen mit der in [10] vorgestellten Lösung wurde die dort als Management-Kern bezeichnete Komponente noch um wesentliche Teile erweitert, so dass sich der Anwendungsfokus vom Management CORBA-basierter Anwendungen hin zum Selbstmanagement von Applikationen und Infrastrukturen auf Basis unterschiedlicher Middleware-Technologien verschieben konnte. Abbildung 2 zeigt die Kommunikation des Selbstmanagers mit einer unter Management stehenden Komponente sowie Möglichkeiten zur Kommunikation mit der Außenwelt. Der Selbstmanager zusammen mit "seiner" Anwendungskomponente wird als eine sich selbst managende Komponente aufgefasst.

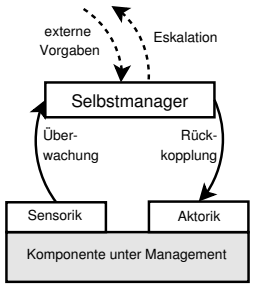

Abbildung 2: Management einer Komponente durch den Selbstmanager

Der Selbstmanager besteht intern aus vier wesentlichen Komponenten (vgl. Abb. 3): Zentrales Element des Selbstmanagers ist der `Module Manager`. Von diesem bereitgestellte `Adapter` dienen der Anbindung von Erweiterungsmodulen, die Sensorik, Aktorik und Logik für den Selbstmanager bereitstellen. Ein (in der Abbildung nicht separat dargestelltes) Messaging-Subsystem innerhalb des `Module Managers` ist für die interne Weiterleitung von Nachrichten zuständig.

Der Selbstmanager unterstützt `Event Modules`, die mittels eigener Threads selbsttätig Ereignisse aufgrund von Veränderungen der Außenwelt registrieren und daraus interne Nachrichten generieren. Ferner erlaubt er die Integration von `Action Modules`, die durch interne Nachrichten gesteuert über applikationsspezifische Schnittstellen Werte ermitteln oder Aktionen in der Außenwelt durchführen können.

Sensorik entsprechend Abbildung 2 lässt sich durch `Event Modules` (Push-Modell) oder `Action Modules` (Pull-Modell) realisieren. Zur Realisierung

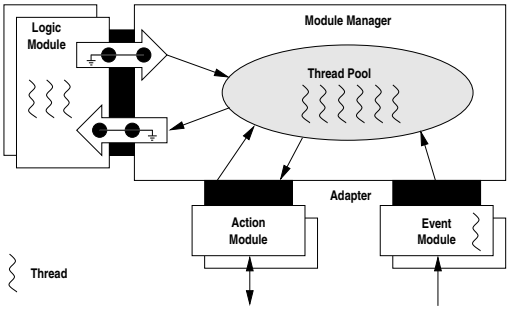

Abbildung 3: Modulare Struktur des Selbstmanagers

von Aktorik werden `Action Modules` eingesetzt. In einem einfachen Szenario können Aktoren z.B. Betriebssystemaufrufe zum Terminieren bzw. Erzeugen von Prozessen implementieren. Auch können sie eine generische Schnittstelle zur Kommunikation mit einer Anwendungsklasse nutzen (z.B. für Web-based Enterprise Management (WBEM) oder Simple Network Management Protocol (SNMP) instrumentierte Anwendungen).

`Logic Modules` stellen in Verbindung mit formalisiertem Managementwissen das "Gehirn" des Selbstmanagers dar. Zur Repräsentierung von Managementwissen werden derzeit endliche Automaten und eine XML-basierte Policy-Sprache unterstützt, eine Erweiterung auf Neuronale Netze oder andere Regelungsmechanismen ist aufgrund der modularen Struktur möglich. `Logic Modules` werden regelmäßig oder aufgrund eingehender Nachrichten aktiv. Sie treffen Management-Entscheidungen und erzeugen als Ergebnis wiederum interne Nachrichten, die an die anderen Module des Selbstmanagers weitergeleitet werden.

Der Selbstmanager wurde für die Betriebssysteme AIX und Linux in C++ implementiert. Das Design zielt hauptsächlich auf das Erreichen von hoher Modularität und Hochverfügbarkeit ab.

Die Funktionalität des Kerns (Event Modules, Action Modules und Logic Modules) ist in Form von Bibliotheken implementiert, die zur Laufzeit vom Module Manager nachgeladen werden können. Um das Management unterschiedlichster Anwendungen zu ermöglichen, wurde eine Reihe verschiedener Event Modules und Action Modules realisiert:

- Mithilfe der **Log-Analyse** können System-Logs überwacht werden.
- **Shell Scripting** dient dem Aufruf von Unix Shell Scripts zur Erledigung einfacher Aufgaben. Hierüber kann beispielsweise ein Systemprozess neu gestartet werden.
- Die **CORBA**-Unterstützung bietet eine generische CORBA-Management-Schnittstelle, über die CORBA-Objekte aufgerufen werden können.
- Die **SNMP**-Unterstützung erlaubt das Ansprechen von beliebigen SNMP-Agenten über `GetRequest` und `SetRequest`. Hierbei agiert der Selbstmanager als SNMP-Manager. Zusätzlich werden SNMP-Traps unterstützt: Diese laufen als asynchrone Ereignisse in einem `Event Module` auf, wo sie durch andere Module aufgegriffen und weiterverarbeitet werden können.
- Über die **WBEM/CIM**-Unterstützung kann der Selbstmanager als CIM Client agieren und beliebige Anfragen an einen CIMOM senden.

Hochverfügbarkeit wird durch die Realisierung des Kerns als ein sich gegenseitig überwachendes Prozesspaar erreicht; der Kern überwacht zudem permanent die Funktionstüchtigkeit einer Zielkonfiguration von geladenen Modulen und startet diese nach einem Ausfall neu.

Bei der Formalisierung des Managementwissens durch einen Experten werden implizit für das Management relevante Wirkungszusammenhänge zwischen administrativen Eingriffen und durch die Sensorik gelieferten Kenngrößen beschrieben. Auf Basis dieser Zusammenhänge beeinflusst der Selbstmanager dann zur Laufzeit das Verhalten der Komponente.

2.2 Die eBusiness-Umgebung

Voraussetzung für das Management von heterogenen eBusiness-Szenarien auf Anwendungsebene ist eine effiziente Instrumentierung der beteiligten Komponenten. Da Middleware-Implementierungen und Applikationsserver normalerweise von Haus aus nur über unzureichende und nicht einheitliche, für das Management nutzbare Schnittstellen verfügen, muss eine separate Instrumentierung unter Zuhilfenahme geeigneter Technologien durch den Entwickler erfolgen, in Abhängigkeit von der verwendeten Technologie muss der Anwendungscode um Instrumentierungsanweisungen ergänzt werden. Hierbei ist es in der Regel sinnvoll, die Instrumentierung auf Middleware-Ebene bzw. für Applikationsserver durchzuführen, da die Instrumentierung einzelner Anwendungsobjekte aufgrund ihrer großen Zahl als fehleranfällig und zeitintensiv anzusehen ist. Zudem ist davon auszugehen, dass Anwendungskomponenten eine kürzere Lebensdauer als die darunterliegenden Middleware-Komponenten haben und dass der Anwendungscode nicht in allen Szenarien für den Benutzer verfügbar bzw. abänderbar ist.

Für das Performance Monitoring von Anwendungen eignet sich insbesondere das von der CMG/OpenGroup definierte Application Response Measurement (ARM) API [11], das aktuell in Version 4.0 für C und Java spezifiziert ist. ARM bietet mit den sog. Correlators einen Mechanismus, über den ineinander verschachtelte Arbeitsabschnitte (Units of Work) in einen logischen Zusammenhang (Vater-Kind-Beziehung) gebracht werden können. Zudem lassen sich ARM-Messungen über das Metrics Schema des von der Distributed Management Task Force (DMTF) spezifizierten Common Information Model [12] (CIM)

abbilden und so in ein objektorientiertes Management-Modell der Anwendung einbeziehen.

Abbildung 4 zeigt die instrumentierten Komponenten der eBusiness-Umgebung beispielhaft im Zusammenspiel (Apache Webserver, Tomcat Web Container, JBOSS EJB Server, IONA Orbacus, MySQL-Datenbank). In der Abbildung sind Messpunkte (d.h. Start und Ende von ARM Transaktionen) mit einem schwarzen Kreis gekennzeichnet, angestoßene ARM-Transaktionen werden jeweils mit Tx bezeichnet, Korrelatoren mit Cx. Geschäftstransaktionen von Kunden können über Komponentengrenzen hinweg verfolgt werden und Antwortzeitmessungen auf einzelne beteiligte Subsysteme heruntergebrochen werden, so dass im Rahmen des SLM eine detaillierte Ursachenforschung ermöglicht wird.

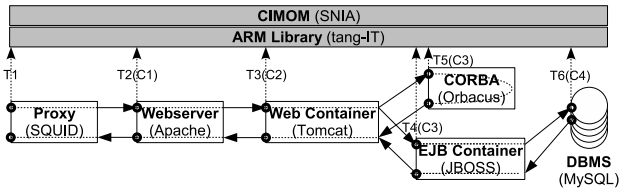

Abbildung 4: Die instrumentierte eBusiness-Umgebung des Labors für Verteilte Systeme (Auszug)

Abbildung 5 zeigt den über Korrelatoren hergestellten logischen Zusammenhang der ARM-Transaktionen des Beispiels. Die auf Basis der instrumentierten Komponenten implementierten Anwendungen müssen hierfür nicht separat instrumentiert werden, die Performance-Messungen erfolgen soweit möglich transparent für die Geschäftsanwendungen. Durch die Instrumentierung des Web Proxy Squid kann die Überwachung der Antwortzeit einer Transaktion bis auf die Kundenseite ausgedehnt werden. Die Gesamtarchitektur ist im Detail in [13] beschrieben.

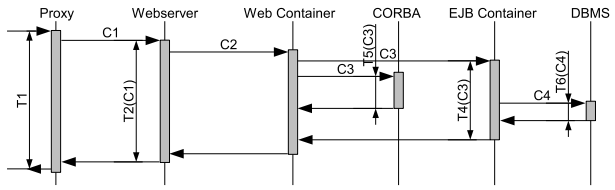

Abbildung 5: Mithilfe von ARM-Korrelatoren in einen logischen Zusammenhang gebrachte Messungen

3 Selbstmanagement und Kooperation von Managern

In einer Laborumgebung wurde der Selbstmanager bisher erfolgreich für das Selbstmanagement einzelner Komponenten der eBusiness-Infrastruktur eingesetzt, wobei bisher die Selbstkonfiguration von unabhängigen Anwendungen im Vordergrund steht. Als Anwendung dient hierbei ein Aktien-Brokerage-Dienst, der als verteilte Anwendung unter Einbeziehung eines JBOSS J2EE-Servers sowie einer MySQL-Datenbank realisiert wurde. Zur Darstellung des Frontends werden Servlets und JSP-Seiten verwendet, die von einem Tomcat Web Container bereitgestellt werden. Ein möglicher Aufruf innerhalb der verteilten Architektur ist in Abbildung 4 dargestellt.

3.1 Lokales Selbstmanagement

In dem beschriebenen Szenario verwaltet jeweils eine Selbstmanager-Instanz alle Instanzen einer der am Ablauf beteiligten Komponenten. So existiert beispielsweise zum Management der JBOSS-Instanzen auf unterschiedlichen Rechnern nur eine einzige Selbstmanager-Instanz.

Beispielhaft wurde im Zusammenspiel mit der JBOSS-eigenen Clustering-Lösung ein Szenario realisiert, in dem der Selbstmanager bei Überschreiten einer bestimmten maximalen Antwortzeit zusätzliche Instanzen des Applikationsservers startet bzw. bei geringer Last ungenutzte Instanzen beendet. Es wird hier davon ausgegangen, dass die verschiedenen Applikationsserver-Instanzen keine Ressourcen teilen müssen, sich in ihrer individuellen Antwortzeit also nicht gegenseitig beeinflussen. Details zum hier angewandten Vorgehen zu Modellierung und Deployment des Managementwissens sowie zur Konfiguration des SLM-Systems beschreibt [14].

Abbildung 6 zeigt in einer vereinfachten Form das für die Kontrolle der Anzahl verfügbarer Applikationsserver-Instanzen in einem Automaten formalisierte Managementwissen. Hierbei operiert der Selbstmanager auf der durch die ARM-Instrumentierung bereitgestellten durchschnittlichen Antwortzeit aller verfügbaren JBOSS-Instanzen RT. Zu Beginn befindet sich der Automat im Zustand `Operating`. Übersteigt RT die Schranke $\$enter\_critical$, wird eine weitere Instanz erzeugt und der Automat wechselt in den Zustand `Heavy Load`. Hier werden, falls RT nicht unter die Schranke $\$enter\_critical$ sinkt, weitere Instanzen erzeugt. Ist die maximal zulässige Anzahl von Instanzen $\$max\_instances$ erreicht, wechselt der Automat in den Zustand `Overload` und generiert ein Event vom Typ $\$event\_overload$ für die übergeordnete Management-Instanz. Beim Unterschreiten der Schranke $\$leave\_critical$ wird ein Event vom Typ $\$event\_no\_overload$ generiert und der Automat wechselt zurück in den Zustand `Heavy Load`. Bei Unterschreiten der Schranke $\$low\_load$ wechselt er von hier nach `Operating`. In diesem Zustand werden dann ungenutzte Instanzen entfernt, solange die Antwortzeitschranke $\$low\_load$ unterschritten bleibt und noch mehr als eine Instanz aktiv ist.

Bei der Entwicklung des Management-Automaten wurde existierendes Expertenwissen in die in Abbildung 6 dargestellte formale Automaten-Darstellung überführt, um es für das Selbstmanagement nutzbar zu machen. Hierbei sind insbesondere zwei Aspekte von Bedeutung: Zum einen wird ein abstraktes Managementziel definiert (hier: Kontrolle der durchschnittlichen Antwortzeit RT),

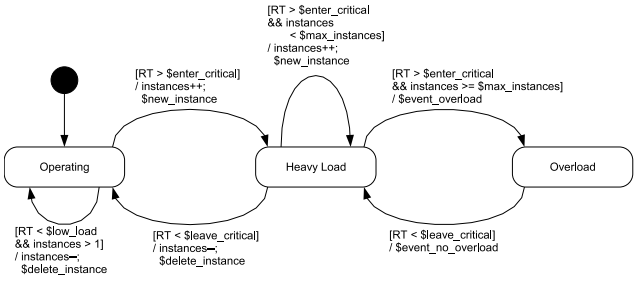

Abbildung 6: Automatenbasierte Kontrolle der Anzahl von Applikationsserver-Instanzen

das der Selbstmanager verfolgen soll, zum anderen wird über einen Regelungsalgorithmus eine Beziehung zwischen Sensorik und Aktorik hergestellt, die es dem Selbstmanager ermöglicht, das abstrakt formulierte Ziel zu erreichen (im Beispiel: durchschnittliche Antwortzeit *RT*, Anzahl der Instanzen).

In der hier untersuchten verteilten Umgebung hängt die am JBOSS-Server gemessene Antwortzeit des System allerdings auch von der Antwortzeit der im Request-Flow dahinterliegenden Datenbank ab. Unter Umständen bringt das Erzeugen weiterer JBOSS-Instanzen hier also keine Verbesserung, sondern resultiert lediglich in der Inanspruchnahme zusätzlicher Ressourcen. Aus seiner lokalen Sicht (die durch seine Sensorik definiert wird) kann der für die JBOSS-Instanzen verantwortliche Selbstmanager dies allerdings nicht erkennen, so dass ein Informationsaustausch bzw. eine Kooperation mit dem Selbstmanager der Datenbank notwendig wird.

Im Folgenden werden zunächst allgemein mögliche Formen der Kooperation von Managern diskutiert, bevor das hier gezeigte Beispiel um Kooperationsaspekte erweitert wird.

3.2 Prinzipien der Kooperation

Generell können bei der Betrachtung von Kommunikationsverhalten und Ressourcenabhängigkeiten komplexer Anwendungssysteme unterschiedlichste Konstellationen auftreten. Betrachtet man das Selbstmanagement als integralen Bestandteil einer jeden Komponente, erscheint Kommunikation auf Managementebene immer dort sinnvoll, wo es auf Applikationsebene Berührungspunkte mit anderen Komponenten gibt. Hierbei sind für das Management insbesondere zwei Szenarien relevant (vgl. Abbildung 7):

a) **Anwendungskooperation** - Komponenten kommunizieren im Rahmen ihrer Geschäftsaufgaben miteinander, bilden z.B. aufeinanderfolgende Bestandteile eines Workflows.

b) **Konkurrenz** - Komponenten konkurrieren um Ressourcen, z.B. um die Rechenleistung eines Prozessors. Zur Beschreibung konkurrierender Komponenten können Abhängigkeitsgraphen [15] eingesetzt werden. Dabei betrachtet man gemeinsam benutzte Komponenten auch als Ressourcen.

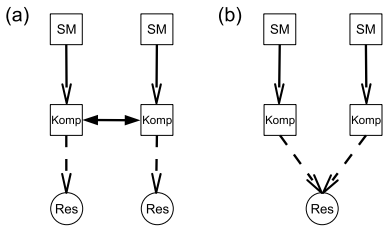

Abbildung 7: Für das Management relevante Beziehungen von Software-Komponenten

Für beide Szenarien ist eine Kommunikation auf Managementebene nicht zwingend notwendig. In diesem Fall werden die einzelnen Komponenten, wie in Abschnitt 3.1 beschrieben, unabhängig voneinander verwaltet. Managementziele unterschiedlicher Selbstmanager können hierbei allerdings im Konflikt zueinander stehen. Zum Erreichen gemeinsamer, komponentenübergreifender Managementziele ist jedoch eine Kommunikation auf Managementebene notwendig.

Bezüglich der Art der Kommunikation und Entscheidungsfindung auf Managementebene unterscheidet man, wie in Abbildung 8 dargestellt, (a) **Peer to Peer (p2p) Ansätze** und (b) **hierarchische Ansätze**, wobei auch Mischformen z.B. (c) möglich sind. Auf diese Kooperationsformen wird in den folgenden Abschnitten genauer eingegangen.

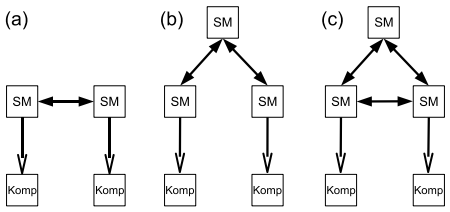

Abbildung 8: Mögliche Formen der Kooperation von Managern

Sowohl für p2p als auch für hierarchische Szenarien erfolgt die Kommunikation der Manager auf der abstrakten Ebene der Managementziele, d.h. konkretes, zum Erreichen eines Zieles notwendiges Wissen wird vom jeweiligen Selbstmanager gekapselt.

Betrachtet man mehrere der oben genannten Szenarien (Unterstützung von Geschäftsprozessen, Konkurrenz um Ressourcen) gleichzeitig, kann es auf abstrakter Ebene zu Zielkonflikten kommen, die dann im Rahmen der Manager-Kommunikation aufgelöst werden müssen.

3.3 Hierarchische Kommunikation

Wird ein hierarchisches Kommunikationsmodell für das komponentenübergreifende Management verwendet, ist letztendlich der jeweils ranghöchste Selbstmanager für das Erreichen des gemeinsamen Managementziels verantwortlich. Der ranghöchste Selbstmanager verfügt über das notwendige Managementwissen zum Erreichen des gemeinsamen Ziels, das durch Formalisierung existierenden Expertenwissens

gewonnen wird. Hierzu zählt insbesondere auch das für das Ableiten lokaler Managementziele untergeordneter Selbstmanager notwendige Wissen. Die untergeordneten Manager informieren den übergeordneten Manager über ihren Ist-Zustand, den dieser mit einem von ihm aus dem globalen Managementziel abgeleiteten Soll-Zustand vergleicht. Aus der Größe und Art der Abweichung ermittelt er dann lokale Management-Ziele für die untergeordneten Manager.

Ist als globales Managementziel z.B. das Einhalten einer in einem SLA definierten maximalen Antwortzeitschranke definiert, so ist es die Aufgabe des übergeordneten Selbstmanagers, zu ermitteln, an welcher Stelle des Systems sich der Flaschenhals in Bezug auf die Gesamtantwortzeit befindet. Der ermittelten Komponente wird als Managementziel das Verringern der Antwortzeit gesetzt.

Insbesondere hierarchische Ansätze gehen implizit davon aus, dass es möglich ist, innerhalb des unter Management stehenden Systems die formalisierten Managementziele global an einer Stelle verfügbar zu machen. An dieser Stelle ist im Rahmen des hierarchischen Ansatzes dann der Manager oder menschliche Administrator mit seinem Expertenwissen angesiedelt, der im Zweifelsfall eine endgültige Entscheidung über die Systemeingriffe trifft.

Für Anwendungsfälle, in denen Selbstmanagement über administrative Grenzen hinweg realisiert werden soll, wie dies z.B. in SOA-Szenarien vorgesehen ist, ist der hierarchische Ansatz somit allerdings nicht praktikabel. Hier kann zur Überbrückung administrativer Grenzen p2p-Kommunikation auf abstrakter Ebene eingesetzt werden, wobei dies unabhängig von den innerhalb der beteiligten Domänen verwendeten Paradigmen ist.

Abbildung 9 verdeutlicht die Domänen-übergreifende Kooperation der Selbstmanager beispielhaft. Innerhalb von Domäne A wird ein rein hierarchisches Modell eingesetzt, Domäne B ist intern auf p2p-Basis strukturiert. Beiden Domänen ist jedoch gemein, dass der über die Domänengrenze hinweg kommunizierende Manager ausreichenden Einfluss auf die intern zu treffenden Entscheidungen haben muss.

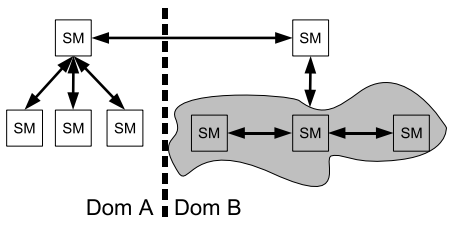

Abbildung 9: Kooperation von Managern über administrative Grenzen hinweg

3.4 p2p-Kommunikation

Im Vergleich zu hierarchischen Szenarien gestaltet sich die Koordination unterschiedlicher Selbstmanager über p2p-Verfahren ungleich komplizierter.

Lokale Ziel-Entscheidungen werden hier beispielsweise auf der Basis von Gruppenkommunikations- und Voting-Mechanismen [16] der beteiligten Selbstmanager herbeigeführt. Hierbei kennen alle beteiligten Manager das gemeinsame Managementziel, ein zentraler Überblick über den globalen Zustand des Systems ist aber nicht Voraussetzung für die Entscheidungsfindung.

Ist als globales Managementziel z.B. das Einhalten einer in einem SLA definierten maximalen Antwortzeitschranke definiert, sind alle Selbstmanager gemeinsam für das Einhalten des Ziels verantwortlich. Gemeinsam müssen die lokalen Managementziele so justiert werden, dass das gemeinsame Ziel erreicht wird.

Die (im Managementbereich notwendige) Nachvollziehbarkeit von Gruppenentscheidungen gestaltet sich im dynamischen Umfeld mit hoher Komponentenfluktuation allerdings schwierig, p2p-Kommunikation wird sich daher in erster Linie für überschaubare Szenarien eignen, die lediglich ein geringes Konfliktpotenzial bergen. Vorteile könnten sich bezüglich der Fehlertoleranz des Gesamtsystems der Selbstmanager ergeben.

3.5 Kooperation am Beispiel

Um beispielhaft komponentenübergreifendes Selbstmanagement zu realisieren, wurde das in Abschnitt 3.1 beschriebene Szenario um einen hierarchischen Ansatz zur Manager-Kooperation erweitert. Hierfür wurde zunächst ein weiteres `Logic Module` entwickelt, mit dessen Hilfe Selbstmanager miteinander kommunizieren und Managementziele austauschen können.

Abbildung 10 zeigt die etablierte Manager-Hierarchie zur Realisierung von komponentenübergreifendem Selbstmanagement im eBusiness-Testbed. Der übergeordnete Selbstmanager erhält von den untergeordneten Selbstmanagern einen Mittelwert der Antwortzeit der von ihnen verwalteten Komponenten. Basierend auf diesen Informationen entscheidet der übergeordnete Manager, welcher untergeordnete Selbstmanager seine Antwortzeit verringern soll, und gibt dieses Ziel weiter. Der in Abschnitt 3.1 vorgestellte Regelungsalgorithmus zur Kontrolle der JBOSS-Instanzen musste hierfür nur insoweit angepasst werden, dass Zustandsübergänge nur auf Veranlassung des übergeordneten Selbstmanagers erfolgen und nicht mehr wie bisher abhängig von der aktuellen Antwortzeit.

Der übergeordnete Manager kann aus den Antwortzeitmessungen der einzelnen Selbstmanager im Bedarfsfall den Engpass im System ermitteln und den jeweiligen Selbstmanager veranlassen, seine Antwortzeit zu reduzieren. Dass die Verringerung der durchschnittlichen Antwortzeit durch Bereitstellen weiterer Instanzen geschieht, bleibt dem übergeordneten Manager verborgen.

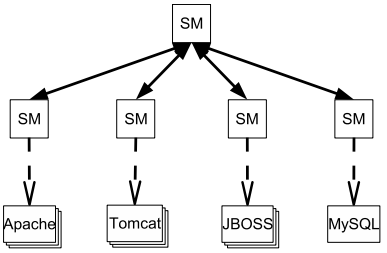

Abbildung 10: Hierarchische Selbstmanager-Kooperation im eBusiness-Testbed

4 Ausblick

Schon die Realisierung von kooperativem Selbstmanagement in dem hier beschriebenen statischen Szenario gestaltet sich relativ komplex.

Stellen traditionelle verteilte Anwendungen mit ihrer über die Zeit stark angestiegenen Zahl beteiligter Komponenten bereits hohe Anforderungen an Managementsysteme, so steigen diese für das Management von Geschäftsprozessen innerhalb einer SOA-Umgebung weiter:

- SOA-Geschäftsprozesse können Komponenten unterschiedlicher administrativer Domänen umfassen. Ferner entscheidet sich die Zusammensetzung der Prozesse häufig erst zur Laufzeit. Unter Umständen ist es zu Beginn noch nicht möglich, den Prozess vollständig zu definieren, Änderungen und Erweiterungen der Definition müssen zur Laufzeit möglich sein. Gleiches gilt im Falle einer Rekonfiguration nach Ausfall einer Komponente. Ein traditionelles Management ist aufgrund der hohen Dynamik in der Regel nicht möglich. Um einen unternehmensübergreifenden Geschäftsprozess unterstützen zu können, muss sicherlich Kooperation von Managern (entsprechend Abbildung 10) praktiziert werden.

- Managementziele ergeben sich im SOA-Kontext aus der Definition des Geschäftsprozesses und vereinbarten SLAs. In einer dynamischen SOA-Umgebung werden solche Vereinbarungen mit Unterdienstleistern ggf. kundenspezifisch und erst zur Laufzeit geschlossen. Das Management-System muss in der Lage sein, diese sich dynamisch ergebenden Abhängigkeiten zu erfassen und zu repräsentieren. Die Ableitung konkreter Managementziele für die beteiligten Selbstmanager kann hier nicht mehr durch formalisiertes Expertenwissen statisch geleistet werden.

Es besteht allerdings die Hoffnung, dass durch die im SOA-Umfeld verwendeten Web Services und die damit erreichte Homogenisierung der zu verwaltenden Umgebung die Formalisierung von Managementwissen erleichtert wird. Untersucht werden Möglichkeiten zur Auswertung der formalisiert vorliegenden Prozessbeschreibung (BPEL-Dokument) zu einem modellbasierten, generischen Vorgehen für die Ableitung lokaler Managementziele.

Mit der Entwicklung einer Web-Services-Schnittstelle für den hier vorgestellten Selbstmanager, die das Absetzen von SOAP-Aufrufen für Managementzwecke ermöglicht, wurde bereits der Grundstein zum Management von SOA-Architekturen gelegt. Nächste Schritte bestehen in der Definition eines Kooperations-Frameworks für Selbstmanager im SOA-Umfeld. Schwerpunkte werden hierbei Kooperationsmechanismen für Selbstmanager und Konfliktauflösungsstrategien sein.

Literatur

[1] Krafzig, Dirk ; Banke, Karl ; Slama, Dirk: *Enterprise SOA*. Prentice Hall, 2004

[2] Lewis, Lundy: *Managing Business and Service Networks*. Kluwer Academic / Plenum Publishers, 2001

[3] Sturm, Rick ; Morris, Wayne ; Jander, Mary: *Foundations of Service Level Management*. Sams Publishing, 2000

[4] Kephart, Jeffrey O. ; Chess, David M.: The Vision of Autonomic Computing. In: *IEEE Computer* (2003), Jan, S. 41–50

[5] Hewlett Packard: *Geschäftliche Agilität: Die Adaptive Enterprise Strategie von HP*. 2003. – http://www.hp.com/go/adaptive

[6] Diao, Y. ; Eskesen, F. ; Froehlich, S. ; Hellerstein, J. L. ; Spainhower, L. F. ; Surendra, M.: Generic Online Optimization of Multiple configuration Parameters With Application to a Database Server. In: *Proceedings of the fourteenth IFIP/IEEE Workshop on Distributed systems: Operations and Management (DSOM)*, 2003

[7] Liu, X. ; Zhu, X. ; Singhal, S. ; Arlitt, M.: Adaptive Entitlement Control of Resource Containers on Shared Servers. In: *Proceedings of the 9th IFIP/IEEE Int. Symp. on Integrated Network Management (IM)*, 2005, S. 163 – 176

[8] Herrmann, K. ; Muehl, G. ; Geihs, K.: Self-Management - Potentiale, Probleme, Perspektiven. In: *PIK - Praxis der Informationsverarbeitung und Kommunikation* (2004), Nr. 27 / 2, S. 74–79

[9] Debusmann, M. ; Kroeger, R.: Widening Traditional Management Platforms for Managing CORBA Applications. In: *Proceedings of the third IFIP WG 6.1 Int. Conf. on Distributed Applications and Interoperable Systems (DAIS)*, 2001

[10] Debusmann, M. ; Kroeger, R. ; Kullmann, T. ; Lindner, D. ; Piwek, T. ; Weyer, C.: Eine Managementlösung zur Integration CORBA-basierter Anwendungen in bestehende Managementplattformen. In: *PIK - Praxis der Informationsverarbeitung und Kommunikation* (2001), Nr. 24 / 4, S. 194 – 201

[11] The OpenGroup: *Application Response Measurement (ARM) Issue 4.0, V2 - C Binding*. 2004. – http://www.opengroup.org/management/arm/

[12] Distributed Management Task Force, Inc.: *COMMON INFORMATION MODEL (CIM) SPECIFICATION*. 2.2. 1999. –

http://www.dmtf.org/standards/documents/CIM/DSP0004.pdf

[13] Debusmann, Markus ; Schmidt, Marc ; Schmid, Markus ; Kroeger, Reinhold: Unified Service Level Monitoring using CIM. In: *Proceedings of the 7th Int. Enterprise Distributed Object Computing Conference (EDOC)*, 2003, S. 76 – 85

[14] Debusmann, M. ; Schmid, M. ; Kroeger, R.: Model-Driven Self-Management of Legacy Applications. In: *Proceedings of the 5th IFIP WG 6.1 Int. Conf. on Distributed Applications and Interoperable Systems (DAIS)*, 2005, S. 56 – 67

[15] Debusmann, Markus: *Modellbasiertes Service Level Management verteilter Anwendungssysteme*, Universität Kassel, Diss., 2004

[16] Colouris, George ; Dollimore, Jean ; Kindberg, Tim: *Verteilte Systeme - Konzepte und Design, Kap. 11.* 3. Auflage. Addison-Wesley, 1994