In: KiVS 2007 - Kommunikation in Verteilten Systemen - Industriebeiträge, Kurzbeiträge und Workshops,
VDE Verlag, Seite 255-266, Editors: T. Braun; G. Carle; B. Stiller, März 2007

Ein Ansatz für das Service Level Management in dynamischen Architekturen

Markus Schmid

Fachhochschule Wiesbaden – University of Applied Sciences,
Labor für Verteilte Systeme,
Kurt-Schumacher-Ring 18, D-65197 Wiesbaden
schmid@informatik.fh-wiesbaden.de,
http://wwwvs.informatik.fh-wiesbaden.de

Zusammenfassung. Der Beitrag skizziert zunächst die Notwendigkeit der Umsetzung von Service Level Management in aktuellen, dynamischen Software-Architekturen und beschreibt dabei bestehende Herausforderungen. Eine Kombination von Selbstmanagement- und Selbstorganisationsprinzipien werden als mögliche Lösungsansätze dargestellt, und ein entsprechender, modularer Architekturansatz wird präsentiert. Die Architektur umfasst ein Framework zum lokalen Selbstmanagement von Softwarekomponenten über standardisierte Schnittstellen für Sensorik und Aktorik, und einen Kommunikationsansatz zur Selbstorganisation von verteilten Managern anhand von Kriterien, die aus bestehenden Service Level Management-Anforderungen abgeleitet werden. Der aktuelle Umfang eines auf diesem Architekturansatz beruhenden prototypischen Frameworks wird ebenfalls dargestellt. Der Beitrag schließt mit einem Ausblick auf notwendige Erweiterungen und zukünftige Arbeiten.

1 Einleitung

Mit steigender Komplexität von IT-Anwendungen und IT-Prozessen und der damit einher gehenden zunehmenden Kritikalität von IT-gestützten Diensten für den Unternehmenserfolg bildet die Implementierung eines wirksamen *Service Level Managements (SLM)* [1,2] eine Notwendigkeit für wettbewerbsfähige Unternehmen. SLM beschäftigt sich mit der Einhaltung von Dienstgütevereinbarungen für IT-Services, die oftmals auf einer Reihe von Hardware- und Softwarekomponenten aufbauen. Solche Dienstgütevereinbarungen werden formal als *Service Level Agreements (SLAs)* zwischen Anbieter und Nutzer eines Dienstes vereinbart. Als SLA bezeichnet man einen Vertrag, der Dienstgüteparameter – wie z.B. Performance- oder Verfügbarkeitskriterien für Funktionen eines Dienstes – in Form von *SLA-Parametern* spezifiziert und für diese so genannte *Service Level Objectives (SLOs)* als einzuhaltende Zielwerte festlegt. Zur Laufzeit ist es Aufgabe des SLM, die definierten SLA-Parameter mit ihren SLOs permanent zu überwachen, Übertretungen zu registrieren und ggf. korrigierend einzugreifen. Dies erfordert allerdings eine geeignete Sensorik-Infrastruktur und für Eingriffe entsprechende Aktorik-Schnittstellen der unter Management stehenden Komponenten.

In: KiVS 2007 - Kommunikation in Verteilten Systemen - Industriebeiträge, Kurzbeiträge und Workshops, VDE Verlag, Seite 255-266, Editors: T. Braun; G. Carle; B. Stiller, März 2007

Zur Umsetzung eines SLAs bedarf es der Auswahl der Hard- und Softwarekomponenten, die sich an der Vertragserfüllung beteiligen sollen, sowie der Konfiguration der SLM-Infrastruktur zur Überwachung der relevanten Parameter. In der Regel spezifizieren SLAs jedoch die vom Provider zur Verfügung zu stellende Infrastruktur nicht im Detail, sondern beschränken sich hier auf die Charakterisierung des Leistungsumfangs und der Güte des zu erbringenden Dienstes. Auf diese Weise erreicht der Provider eine gewisse Flexibilität in der Umsetzung des Dienstes und muss keine internen Details über Geschäftsabläufe oder Infrastruktur preisgeben.

Bei der Umsetzung eines SLAs müssen damit auf Providerseite allerdings noch detaillierte Diensteabbildungen spezifiziert werden, was ein automatisiertes Deployment verkompliziert. Eine Voraussetzung für ein (teil-)automatisiertes Deployment ist die Formulierung des SLA in einer maschinell auswertbaren Syntax, beispielsweise in den SLA-Beschreibungssprachen SLAng [3] oder WSLA [4]. Zudem existieren mittlerweile unterschiedliche Ansätze zum Management und Deployment von SLAs, beispielsweise SweetDeal [5], Fresco [6] und [7]. Neben der Komplexität einer Abbildung beliebiger SLAs mit ihren Definitionen auf ausführende Ressourcen und das notwendige SLM, erweist sich die aktuell starke Zunahme hochdynamischer Umgebungen als Problem für die genannten Frameworks zur SLA-Implementierung.

Das in diesem Beitrag vorgestellte Framework unterstützt SLM in komplexen, hochdynamischen Umgebungen durch die Implementierung einer selbstorganisierenden Management-Infrastruktur und ergänzt damit klassische Ansätze zum Management und Deployment von SLAs.

Zunächst wird in Abschnitt 2 auf dynamische Software-Umgebungen eingegangen um die weiteren Ausführungen zu motivieren. Abschnitt 3 gibt einen Überblick über Selbstmanagement und Selbstorganisation. Diese Prinzipien definieren die Grundlage für die in Abschnitt 4 vorgestellte Architektur zum SLM in dynamischen Umgebungen. Abschnitt 5 beschreibt die prototypische Implementierung dieser Architektur und Abschnitt 6 gibt einen Ausblick auf zukünftige Arbeiten.

2 Dynamische Software-Architekturen

Kommunikation zwischen einzelnen Anwendungsteilen wird in unternehmensweiten, geschäftskritischen verteilten Anwendungen normalerweise durch standardisierte Middleware-Technologien wie z.B. die *Common Object Request Broker Architecture (CORBA)* [8] oder das *Java 2 Enterprise Framework (J2EE)* [9] realisiert. Traditionell werden zudem in derartigen Systemen Präsentation, Geschäftslogik und Datenhaltung in unterschiedlichen Strängen (Tiers) implementiert, zwischen denen klare Schnittstellen definiert werden. Für die Definition von SLAs ist ein solches Vorgehen von Vorteil, da Dienstgüteanforderungen die das Gesamtsystem betreffen, an definierten Einsprungpunkten im Präsentations- oder Geschäftslogik-Strang gemessen werden können.

In: KiVS 2007 - Kommunikation in Verteilten Systemen - Industriebeiträge, Kurzbeiträge und Workshops, VDE Verlag, Seite 255-266, Editors: T. Braun; G. Carle; B. Stiller, März 2007

Höhere Anforderungen an Flexibilität und Dynamik von Geschäftsanwendungen resultieren aber zunehmend in der Auflösung dieser starren Schnittstellen und der Sichtweise einer Anwendung als temporäre Verknüpfung unterschiedlicher, voneinander unabhängiger Dienste. Eine solche Architektur bezeichnet man als *Service Oriented Architecture (SOA)*.

Die Dienste einer SOA bieten standardisierte, plattform- und programmiersprachenunabhängige Schnittstellen, kapseln aber ihre Implementierung. Einzelne Dienste werden durch Workflow-Definitionen verknüpft, die zur Laufzeit durch eine generische *Workflow-Engine* interpretiert werden.

Zu unterst findet man in der üblichen technischen Realisierung einer SOA die von Geschäftskomponenten genutzten Ressourcen des Systems. Diese Geschäftskomponenten offerieren über standardisierte Schnittstellen bestimmte Dienste, die unter Umständen zusammen mit anderen Diensten einen so genannten *Composite Service* realisieren können. Solche Dienste bilden die IT-Schnittstelle einer Abteilung im Unternehmen. Zur Realisierung einer Anwendung werden Dienste zu temporären Workflows komponiert, auf die der Benutzer in der Regel über ein Portal zugreift. SLAs werden im SOA-Bereich sowohl auf Service- als auch auf Workflow-Ebene definiert. Durch Modifikation von Workflows lässt sich die Unternehmens-Software-Architektur einfach an Änderungen in der Unternehmensorganisation anpassen.

3 Selbstmanagement und Selbstorganisation

Traditionelle Enterprise-Management-Systeme wie *IBM Tivoli*, *CA Unicenter* oder *HP OpenView* wurden ursprünglich als Frameworks konzipiert, bestehen mittlerweile aber aus einer Vielzahl spezialisierter Module. Sie fokussieren allerdings eher auf das Management statischer Strukturen – SLM von hochdynamischen Umgebungen und sich ändernden Strukturen wird nur unzureichend unterstützt [10].

Unter dem Begriff *Selbstmanagement* fasst man Ansätze zur autonomen Rekonfiguration, Fehlerbehebung und Optimierung, bzw. Reduktion der nach außen hin sichtbaren Komplexität von Softwaresystemen zusammen. Solche Eigenschaften eines selbstmanagenden Systems bezeichnet man auch als *Self-X*-Properties [11]. Im Vergleich zu klassischen Managementansätzen, bei denen ein Administrator letztendlich Managemententscheidungen verantwortet, wird beim Selbstmanagement eine Managemententscheidung durch einen Regelungsalgorithmus herbeigeführt.

Abbildung 1 stellt traditionelle Managementinfrastrukturen und Selbstmanagement einander gegenüber. In Teil a) der Abbildung erfolgt eine klare Trennung zwischen Management- und Anwendungskomponenten, in Teil b) werden Anwendungskomponente und Selbstmanager als Teile eines sich selbst managenden Systems (oder einer sich selbst managenden Komponente) betrachtet, eine klare Trennung ist nicht notwendigerweise gegeben. Nach außen kann eine solche sich in Grenzen selbst managende Komponente wiederum in eine weitere Managementumgebung eingebunden werden, die dann lediglich mit der nach außen hin

In: KiVS 2007 - Kommunikation in Verteilten Systemen - Industriebeiträge, Kurzbeiträge und Workshops,
VDE Verlag, Seite 255-266, Editors: T. Braun; G. Carle; B. Stiller, März 2007

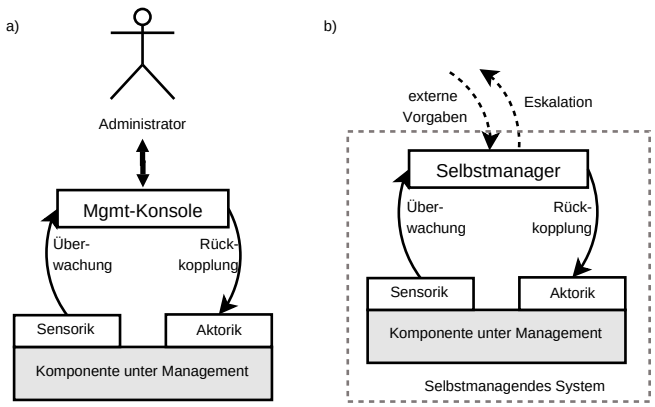

Abb. 1. Traditionelles Management vs. Selbstmanagement

sichtbaren reduzierten Komplexität der Komponente umgehen muss und nicht notwendigerweise Kenntnis vom Selbstmanagementbestandteil derselben haben muss. Einen Überblick aktueller Ansätze und Fragestellungen im Bereich des Selbstmanagements gibt [12].

Selbstmanagement-Regelkreise können jedoch nur in strukturell stabilen Systemen etabliert werden, sie eignen sich weniger zur Herausbildung von Strukturen im Zusammenspiel unterschiedlicher Komponenten. Eine solche Strukturbildung wird auch als *Selbstorganisation* bezeichnet und kann in der Realität in unterschiedlichsten Systemen (physikalische, biologische, soziale Systeme, Märkte, ...) beobachtet werden. Durch die Übertragung von Prinzipien der Selbstorganisation auf technische Systeme erhofft man sich einen vereinfachten Umgang mit steigender Komplexität, verbesserte Skalierbarkeit sowie erhöhte Toleranz gegenüber Fehlern und Ausfällen einzelner Subsysteme. Bei in der Natur vorkommenden selbstorganisierenden Systemen kann man jedoch teils massive Redundanzen beobachten (z.B. im Nervensystem), die bisher beim Design technischer Systeme aus Kostengründen weitgehend vermieden werden.

Wir sind zuversichtlich, dass es durch geschickten Einsatz von Selbstmanagement in Verbindung mit Selbstorganisationsmechanismen möglich ist, die in Abschnitt 1 beschriebene Lücke in Bezug auf (Re-)Konfiguration eines SLM-Systems in dynamischen Architekturen zu schließen.

4 SLM-Ansatz für dynamische Architekturen

Für das SLM in dynamischen Architekturen (vgl. Abb. 1 b)) verfolgen wir einen zweistufigen Ansatz: Auf der lokalen Ebene setzt ein Selbstmanager ein ihm vorgegebenes Managementziel auf der ihm zugeordneten Anwendungskomponente um. Auf der oberen Ebene etablieren die unterschiedlichen Manager unter Nutzung von Techniken der Selbstorganisation eine SLM-Struktur, die der Architektur der Anwendungsumgebung folgt und so in der Lage ist, mit einem Kunden

In: KiVS 2007 - Kommunikation in Verteilten Systemen - Industriebeiträge, Kurzbeiträge und Workshops, VDE Verlag, Seite 255-266, Editors: T. Braun; G. Carle; B. Stiller, März 2007

vereinbarte SLA-Bestandteile geeignet auf einzelne beteiligte Komponenten herunterzubrechen und deren Einhaltung zu überwachen.

4.1 Lokales Selbstmanagement

Auf lokaler Ebene werden selbstmanagende Komponenten durch einen Management- und einen zugehörigen Geschäftsanteil (in Abb. 1 Komponente unter Management) gebildet. Der Managementanteil einer solchen Komponente beobachtet die Geschäftskomponente durch geeignete Sensorik-Schnittstellen und greift in das Verhalten derselben über Aktorik-Schnittstellen ein. Sowohl Sensorik als auch Aktorik sind notwendigerweise anwendungsspezifisch zu implementieren, weshalb sich ein modularer Ansatz zur Realisierung des Managementanteils anbietet. Der eigentliche Regelungsalgorithmus repräsentiert das formalisierte Managementwissen eines Administrators.

[13] beschreibt die prototypische Umsetzung eines lokalen SLM-Selbstmanagementszenarios am Beispiel des Managements eines dynamischen Clusters von JBoss-Applikationsservern basierend auf der durchschnittlichen Antwortzeit eingehender Requests. Als Regelungsalgorithmus kommt hier ein endlicher Automat zum Einsatz. Die in [13] beschriebene Architektur bietet jedoch noch keine Möglichkeiten zur Koordination unterschiedlicher Manager und kann deshalb nur eingeschränkt in komplexeren Architekturen eingesetzt werden.

Grundlegende Voraussetzung für die Koordination unterschiedlicher Managerinstanzen ist die Vergleichbarkeit des bisherigen Managementerfolges einzelner Manager. Dieser bezieht sich beim SLM auf das für den jeweiligen Manager vorgegebene Ziel. Ein solches Ziel hängt jedoch von den vereinbarten SLA-Parametern ab. Betrachtet man beispielsweise “maximale Antwortzeit”, “minimaler Durchsatz”, “Verfügbarkeit” oder “Security” als Beispiele möglicher Vereinbarungen von Dienstgütekriterien, so wird klar, dass in Bezug auf die genannten Ziele eine Definition von Managementerfolg notwendigerweise vollkommen unterschiedlich ist. Um zu einer einheitlichen Definition zu kommen, wird die Betrachtung von Managementzielen zunächst auf *reelle Größen* und *ganze Zahlen* eingeschränkt; so lässt er sich auf einer allgemeinen Skala normieren und als Distanzmaß durch einen *Abweichungsgrad* vom Managementziel darstellen.

Neben dem Abweichungsgrad geht in die Betrachtung der Managementleistung – im Folgenden *Compliance* in Bezug auf ein Managementziel genannt – als weiterer Faktor der *Aufwand* ein, den ein Manager treiben muss, um den erreichten Abweichungsgrad einzuhalten. Somit ergibt sich die Compliance als ein Tupel aus *Abweichungsgrad* und *Aufwand*, jeweils gemittelt über einen bestimmten Zeitraum.

$$Compliance := (Abweichungsgrad,\ Aufwand)$$

Die Compliance kann nicht auf einen skalaren Wert reduziert werden, ermöglicht es aber, Managementerfolg und dafür notwendigen Aufwand unterschiedlicher Komponenten zu vergleichen. Niedrige Werte für beide Komponenten sagen aus, dass ohne großen Managementaufwand die veranschlagten Ziele eingehalten

In: KiVS 2007 - Kommunikation in Verteilten Systemen - Industriebeiträge, Kurzbeiträge und Workshops, VDE Verlag, Seite 255-266, Editors: T. Braun; G. Carle; B. Stiller, März 2007

werden können, ein niedriger Abweichungsgrad bei hohem Aufwand besagt, dass der Manager zwar sein Ziel erreicht, aber wohl keine Reserven für weitergehende Belastungen mehr zur Verfügung hat, dass die "Stellschrauben" sich fast am Anschlag befinden. Ein hoher Abweichungsgrad bei geringem Aufwand deutet auf einen Konfigurationsfehler oder eine noch nicht abgeschlossene Nachregelung hin.

Die Herleitung der zur Berechnung der Compliance verwendeten Kenngrößen aus dem Zustand des unter Management stehenden Systems ist ebenfalls notwendigerweise abhängig vom eingesetzten Regelungsalgorithmus und den sonstigen Systemeigenschaften und kann deshalb nicht generisch erfolgen.

4.2 Selbstorganisation von Managern

Zur Erfüllung übergeordneter Managementziele, die mehrere Komponenten des Systems betreffen, ist die Koordination unterschiedlicher Selbstmanager nötig. Ziel ist es, die für eine solche Koordination erforderlichen Strukturen selbstorganisierend aufzubauen und permanent entsprechend den sich ändernden Gegebenheiten in einer dynamischen Softwarearchitektur anzupassen.

Koordinierungsbedarf für Manager tritt potentiell in unterschiedlichen Situationen auf, die sich durch Berührungspunkte der von ihnen verwalteten Komponenten ergeben: (1) gemeinsam zu erfüllende Dienstgüteanforderungen (SLA-Parameter und darauf definierten SLOs), (2) Inhaltliche Abhängigkeit durch Geschäftsprozesse und (3) Sharing von Ressourcen und dabei auftretende Konflikte.

Verfahren zum Aufbau von Kommunikationsstrukturen müssen diese Berührungspunkte berücksichtigen. Im hier vorgestellten Architekturansatz gruppieren sich Selbstmanager zur Kommunikation entsprechend dieser Kriterien, wobei für das SLM insbesondere (1) von Bedeutung ist; (2) und (3) charakterisieren Situationen mit potentiellen Auswirkungen auf (1).

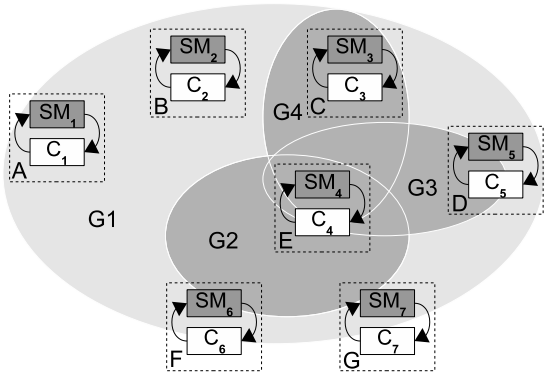

Abb. 2. Gruppierung von Selbstmanagern nach unterschiedlichen Kriterien

In: KiVS 2007 - Kommunikation in Verteilten Systemen - Industriebeiträge, Kurzbeiträge und Workshops, VDE Verlag, Seite 255-266, Editors: T. Braun; G. Carle; B. Stiller, März 2007

Abbildung 2 zeigt eine mögliche Gruppierung von Selbstmanagern (SM_i), die jeweils eine Anwendungskomponente (C_i) verwalten. Im System kommunizieren alle Manager miteinander in einer globalen Gruppe (in der Abbildung als *G1* bezeichnet). Diese Gruppe dient lediglich zur Etablierung weiterer Gruppen und somit zur Herausbildung problemspezifischer Kommunikationsstrukturen. Beispielsweise könnte die Gruppe *G2* aus Komponenten bestehen, die am gleichen Workflow teilnehmen, *G3* könnte Komponenten umfassen, die auf dem gleichen Host laufen (potentieller Ressourcenkonflikt), und die in *G4* gruppierten Komponenten unterliegen einer gemeinsamen Dienstgüteanforderung.

Dynamische Softwarearchitekturen erfordern es, dass Komponenten existierenden Gruppen zur Laufzeit beitreten bzw. diese jederzeit verlassen können, so dass die Kommunikation innerhalb der Gruppen zunächst auf Basis von Gruppenkommunikationsmechanismen [14] abgewickelt wird. In allen oben genannten Fällen erfolgt je nach Verfahren eine Reorganisation der Gruppenstruktur, die die Grundlage für später in der Gruppe zu fällende strategische Entscheidungen bildet. Im Folgenden wird dieser Prozess als Strukturbildung bezeichnet. Strukturbildung in einer Gruppe kann beispielsweise auf Basis der Verwendung von Election-Algorithmen [15] durchgeführt werden.

Basierend auf einer solchen Gruppenkommunikationsinfrastruktur werden dann Koordinationsverfahren definiert. Diese gehen von einer momentan stabilen und allen Mitgliedern bekannten Gruppenstruktur aus, was durch die unterlagerten Kommunikationsprozesse sicherzustellen ist. In der Praxis alternieren Phasen der Struktur- und Koordination.

Ein Koordinationsverfahren muss unterschiedlichen Anforderungen genügen, um die in (1) bis (3) beschriebenen Situationen abdecken zu können. Prinzipiell kann es sowohl zu Kooperationen als auch zu Konkurrenzsituationen zwischen Komponenten kommen, dazwischen sind beliebige Abstufungen denkbar. Wir unterscheiden anhand der Situationen (1) bis (3) somit prinzipiell Mechanismen zur Konfliktauflösung und Mechanismen zum Transfer von Ressourcen bzw. Optionen auf Ressourcennutzung.

Konfliktauflösung kann beispielsweise durch Konsensusalgorithmen oder Eskalation und damit durch Entscheidung eines in der zuvor gebildeten Hierarchie höher stehenden Managers erfolgen. Ein Transfer von Ressourcen(-optionen) kann auf unterschiedlichste Arten geschehen: Ressourcen können beispielsweise zwischen Komponenten versteigert (Auktion) oder verkauft (Markt) werden, auch das Verschenken von Ressourcen ist denkbar. Als Alternative zum Ressourcentransfer kann der Transfer von Aufgaben gesehen werden, hier können prinzipiell die gleichen Mechanismen eingesetzt werden. Ziel ist jeweils die Selbstorganisation eines Systems im Hinblick auf das Erfüllen einer bestimmten SLM-Anforderung.

Dieser Beitrag konzentriert sich ausschließlich auf Mechanismen zum Transfer von Ressourcen, beispielhaft werden im Weiteren Auktionen, Markt/Basar-Verfahren und Adaptionen sozialer Modelle vorgestellt.

Auktions- und Marktprotokolle [16] eigenen sich zum Scheduling von Tasks oder Prozessen oder zur Ermittlung eines Gewinners in Konkurrenzsituationen,

In: KiVS 2007 - Kommunikation in Verteilten Systemen - Industriebeiträge, Kurzbeiträge und Workshops, VDE Verlag, Seite 255-266, Editors: T. Braun; G. Carle; B. Stiller, März 2007

sie können aber auch in kooperativen Szenarien (Workflows) zum Einsatz kommen, um anderen Teilnehmern überschüssige Ressourcen zugänglich zu machen.

Als Gegenwert für zu veräußernde Ressourcen kommt in beiden Fällen eine virtuelle Währung zum Einsatz. Über die initiale Verteilung des Vermögens auf einzelne Komponenten können diese priorisiert werden, eventuell empfiehlt sich eine automatisierte Verteilung gemäß dem Gewicht einer Komponente im Workflow.

Zur Durchführung einer Auktion wird eine zentrale Komponente, der Auktionator benötigt. Hier dürfen von den Gruppenmitgliedern Angebote abgegeben werden, die dann gemäß dem Auktionsprotokoll verarbeitet werden. Initiiert wird eine Auktion immer durch ein Gruppenmitglied, das überschüssige Ressourcen versteigern (steigende Auktion) oder eine Aufgabe vergeben möchte (fallende Auktion).

Der Ablauf eines Marktprotokolls erfordert keine zentrale Instanz sondern geschieht direkt zwischen den beteiligten Komponenten. Initiiert wird ein Kauf/ Verkauf entweder durch ein Gruppenmitglied, das überschüssige Ressourcen oder Aufgaben verkaufen möchte oder durch ein Gruppenmitglied, das auf der Suche nach zusätzlichen Ressourcen oder Aufgaben ist. Interessierte Gruppenmitglieder geben Angebote ab, der Verkäufer entscheidet sich für einen Käufer. Dieser Vorgang entspricht im Wesentlichen dem Vorgehen bei Auktionen, bietet aber die Möglichkeit, dass Gruppenmitglieder unterschiedliche Verkaufsalgorithmen implementieren, in die noch andere Bewertungskriterien einfließen (z.B. steht ein Bieter in Konkurrenz zu mir?). Als Strategie einzelner Komponenten wird in beiden Fällen eine Gewichtungsfunktion verwendet, die Streben nach Gewinnmaximierung, Einhalten der individuellen Managementziele und Sicherheitsreserven für unvorhergesehene Änderungen kombiniert.

Protokolle zur sozialen Interaktion arbeiten statt mit einer Währung mit dem sozialen Status eines Gruppenmitglieds [17]. Sie eignen sich potentiell zur gerechten Verteilung von Ressourcen, wenn diese teilbar gehandelt werden und können in kooperativen Szenarien (Workflows) zum Einsatz kommen, um anderen Teilnehmern überschüssige Ressourcen zugänglich zu machen.

Initiiert wird eine Weitergabe (Verschenken) durch ein Gruppenmitglied, das überschüssige Ressourcen in die Gruppe einbringen möchte. Interessierte Gruppenmitglieder bekunden ihr Interesse, nach Verstreichen einer gewissen Frist bekommt ein Interessent die Ressource zugeteilt. Resultat ist eine gewisse soziale Bindung zwischen Schenkendem und Beschenktem oder eine generelle Erhöhung des sozialen Status eines Schenkenden. Zur Ermittlung der Erhöhung kommt eine Gewichtungsfunktion zum Einsatz, die z.B. die Größe der eingebrachten Ressource oder auch den bisherigen sozialen Status mit einbeziehen kann. Bei der Bewertung der Interessenten kann deren sozialer Status oder auch deren *Compliance* neben anderen Kriterien herangezogen werden.

In: KiVS 2007 - Kommunikation in Verteilten Systemen - Industriebeiträge, Kurzbeiträge und Workshops, VDE Verlag, Seite 255-266, Editors: T. Braun; G. Carle; B. Stiller, März 2007

5 Selbstmanagement-Framework

5.1 Überblick

Die in Abschnitt 4 beschriebene selbstorganisierende SLM-Architektur wird derzeit prototypisch im modularen Self-Manager-Framework des Labors für Verteilte Systeme realisiert.

Das Framework unterstützt sowohl die Realisierung von konkreten Management-Kreisläufen für unterschiedliche Komponenten als auch die Kommunikation verschiedener Manager mit dem Ziel der Selbstorganisation.

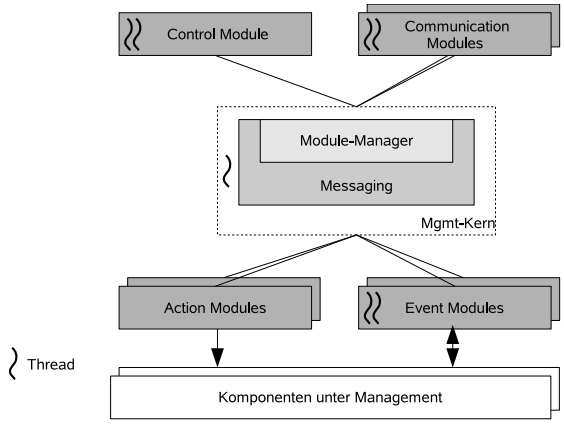

Abb. 3. Prinzipielle Architektur des modularen Selbstmanagers

Abbildung 3 zeigt die prinzipielle Architektur eines mit Hilfe des Frameworks implementierten Managers, angelehnt an [18]. Ein Selbstmanager besteht intern aus fünf wesentlichen Elementen: Zentrale Komponente ist der `Module Manager`. Von diesem bereitgestellte `Adapter` dienen der Anbindung von Erweiterungsmodulen, die Sensorik, Aktorik, Logik und Möglichkeiten zur Kommunikation mit anderen Selbstmanagern bereitstellen. Ein Messaging-Subsystem innerhalb des `Module Managers` ist für die interne Weiterleitung von Nachrichten zuständig.

Für die Kommunikation mit der zu managenden Komponente sind die in Abbildung 3 im unteren Bereich dargestellten `Action Modules` und `Event Modules` zuständig. Im Zusammenspiel mit einem den Managementalgorithmus beherbergenden `Control Module` stellen diese bereits eine vollständige lokale Lösung zur Realisierung von Selbstmanagement dar. Die in der Abbildung ebenfalls dargestellten `Communication Modules` dienen der Kommunikation von Managern zur Koordination ihrer Ziele.

Im folgenden wird nun zunächst näher auf die Module zur Realisierung eines lokalen Selbstmanagers eingegangen, im Anschluss daran werden die Möglichkeiten zur Manager-Kommunikation erläutert.

In: KiVS 2007 - Kommunikation in Verteilten Systemen - Industriebeiträge, Kurzbeiträge und Workshops, VDE Verlag, Seite 255-266, Editors: T. Braun; G. Carle; B. Stiller, März 2007

5.2 Lokales Selbstmanagement

Der Selbstmanager kann durch `Event Modules` ergänzt werden, 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 1 lässt sich durch `Event Modules` (Push-Modell) oder `Action Modules` (Pull-Modell) realisieren. Zur Realisierung von Aktorik werden ebenfalls sowohl `Event Modules` als auch `Action Modules` eingesetzt. In einem einfachen Szenario können Aktoren z.B. Betriebssystemaufrufe zum Terminieren bzw. Erzeugen von Prozessen implementieren. Sensorik und Aktorik können mittels geeigneter Module aber auch generische Schnittstellen zur Kommunikation mit einer Anwendungsklasse nutzen (z.B. für Application Response Measurement (ARM) oder Java Management Extensions (JMX) instrumentierte Anwendungen).

`Control Modules` stellen in Verbindung mit formalisiertem Managementwissen das "Gehirn" eines Selbstmanagers dar. Sie werden zeitgesteuert oder aufgrund eingehender interner Nachrichten aktiv. `Control Modules` treffen Management-Entscheidungen und erzeugen als Ergebnis wiederum interne Nachrichten, die an andere Module des Selbstmanagers weitergeleitet werden, wodurch beispielsweise mittels der spezifischen Aktorik eines Moduls das unter Management stehende System rekonfiguriert werden kann.

Das Framework stellt momentan beispielhaft ein `Control Module` bereit, das in der Lage ist, beliebige mit dem von IBM Research entwickelten *Agent Building and Learning Environment (ABLE)* Toolkit [19] erstellte Regelungsalgorithmen zu integrieren. Das ABLE Toolkit stellt komplexe Konstrukte wie z.B. neuronale Netze zur Verfügung, kann aber problemlos durch eigene Komponenten erweitert werden.

ABLE verfügt über einen grafischen Editor zur Konfiguration der Komponenten, der von einem Experten zur Formalisierung des Managementwissens eingesetzt werden kann. Hierbei 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.

5.3 Kommunikationsplattform für Manager

Zur Selbstorganisation von Manager-Gruppen stellt das Framework eine modulare Basis bereit, die derzeit primär auf die Unterstützung von Auktions- und Marktverfahren zum Aushandeln lokaler Teil-SLOs ausgelegt ist, aber zukünftig leicht für andere Verfahren adaptiert und erweitert werden kann.

Kommunikationsinhalt sind Teile von zuvor auf die Einzelkomponenten heruntergebrochenen SLOs eines globalen SLA. Diese lokalen Teil-SLOs bilden ein Subset von WSLA und sind auf in WSLA definierte SLOs abbildbar. Dies

In: KiVS 2007 - Kommunikation in Verteilten Systemen - Industriebeiträge, Kurzbeiträge und Workshops, VDE Verlag, Seite 255-266, Editors: T. Braun; G. Carle; B. Stiller, März 2007

ermöglicht eine einfache Überprüfung der Einhaltung zuvor in WSLA definierter SLAs. Das Framework verwendet JXTA als unterlagerte Kommunikationsinfrastruktur.

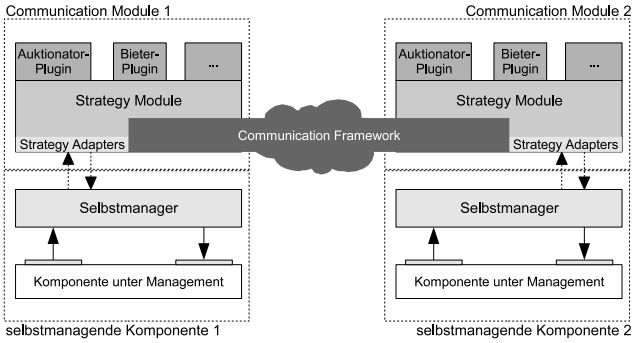

Abb. 4. Anbindung eines Kommunikationsmoduls an den Selbstmanager

Abbildung 4 zeigt die Architektur des Selbstmanager-Kommunikationsmoduls. Zentrales Element ist ein `Strategy Module`, das über *Control-Module*-spezifische `Strategy Adapter` mit dem Selbstmanager kommunizieren kann. Über die `Strategy Adapter` wird die aktuelle Compliance (vgl. Abschnitt 4.1) des Managers ermittelt. Basierend auf der Compliance entscheidet das Framework über die lokale Koordinationsstrategie, beispielsweise eine Beteiligung des Selbstmanagers an Auktionen oder Verkäufen. In Abbildung 4 ist beispielhaft eine Konfiguration für Auktionen dargestellt. Hier übernimmt ein Auktionator-Plugin das Durchführen von Auktionen, falls der Selbstmanager aus der Gruppe zur Durchführung der Auktionen bestimmt wurde, ein Bieter-Plugin realisiert das Bieten auf Ressourcen. Zur Partizipation an Marktprotokollen oder Protokollen zur sozialen Interaktion müssten lediglich diese Plugins geeignet ersetzt werden.

6 Ausblick

Derzeit wird die vorgestellte Architektur in Bezug auf die Eignung von ansteigenden Auktionen zur Verteilung von Teil-SLOs innerhalb einer Gruppe evaluiert. Nächste Schritte umfassen die Erweiterung des Frameworks um zusätzliche Verfahren und ein Vergleich der Effizienz dieser Verfahren. Zudem sollen die derzeit noch sehr restriktiven Annahmen in Bezug auf mögliche Kommunikationsinhalte schrittweise durch entsprechende Ergänzungen im Framework erweitert werden. Für den Einsatz in größeren Umgebungen ist auch eine detaillierte Betrachtung von Konfliktauflösungsstrategien und deren Umsetzung auf unterschiedlichen Ebenen des Frameworks notwendig.

In: KiVS 2007 - Kommunikation in Verteilten Systemen - Industriebeiträge, Kurzbeiträge und Workshops, VDE Verlag, Seite 255-266, Editors: T. Braun; G. Carle; B. Stiller, März 2007

Literaturverzeichnis

1. Sturm, R., Morris, W., Jander, M.: Foundations of Service Level Management. SAMS Publishing (2000)
2. Lewis, L.: Managing Business and Service Networks. Kluwer Academic Publishers (2001)
3. Lamanna, D., Skene, J., Emmerich, W.: SLAng: A Language for Defining Service Level Agreements. In: In Proc. of the 9th IEEE Workshop on Future Trends in Distributed Computing Systems - FTDCS 2003. (2003) 100–106
4. Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. Journal of Network and Systems Management **11** (2003) 57–81
5. Grosof, B.N., Poon, T.C.: SweetDeal: representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In: Proceedings of the 12th international conference on World Wide Web, ACM Press (2003) 340–349
6. Ward, C., Buco, M.J., Chang, R.N., Luan, L.Z., So, E., Tang, C.: Fresco: a Web services based framework for configuring extensible SLA management systems. (2005) 237–245 vol.1
7. Debusmann, M.: Modellbasiertes Sevice Level Management verteilter Anwendungssysteme. PhD thesis, University of Kassel (2005) (in German).
8. Object Management Group: Common Object Request Broker Architecture: Core Specification. (2004)
9. Sun Microsystems, Inc.: Java 2 Platform Enterprise Edition Specification, v1.4. (2003) Final Release, http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf.
10. Garschhammer, M., Schiffers, M.: Integrated IT-Management in Large-Scale, Dynamic, and Multi-Organizational Environments. In: Proceedings of the 12th Annual Workshop of HP OpenView University Association, Hewlett Packard (2005)
11. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36** (2003) 41–50
12. Herrmann, K., Muehl, G., Geihs, K.: Self-Management - Potentiale, Probleme, Perspektiven. PIK - Praxis der Informationsverarbeitung und Kommunikation (2004) 74–79
13. Debusmann, M., Schmid, M., Kroeger, R.: Model-Driven Self-Management of Legacy Applications. In Kutvonen, L., Alonistioti, N., eds.: 5th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2005), Athens, Greece, June 2005, IFIP, Springer (2005) 56–67
14. Chockler, G.V., Keidar, I., Vitenberg, R.: Group communication specifications: a comprehensive study. ACM Comput. Surv. **33** (2001) 427–469
15. Garcia-Molina, H.: Elections in distributed computer systems. IEEE Transactions on Computers **C-31** (1982) 48–59
16. Wellman, M., Walsh, W., Wurman, P., Mason, M.J.: Auction protocols for decentralized scheduling. Games and Economic Behavior **35** (2001) 271–303
17. Gu, D., Welch, L.R., Bruggeman, C., Shelly, R.: The applicability of social models for self-organizing real-time systems. In: Parallel and Distributed Processing Symposium, 2003. Proceedings. International. (2003)
18. Debusmann, M., Kroeger, R., Kullmann, T., Lindner, D., Piwek, T., Weyer, C.: Eine Managementlösung zur Integration CORBA-basierter Anwendungen in bestehende Managementplattformen. PIK - Praxis der Informationsverarbeitung und Kommunikation (2001) 194 – 201
19. Bigus, J.P., Schlossnagle, D.A., Pilgrim, J.R., Mills, W.N., Diao, Y.: ABLE: A Toolkit for Building Multiagent Autonomic Systems. IBM Systems Journal (2002)