K.G. Saur Verlag, München, 2004

K. Herrmann, G. Mühl und K. Geihs

Self-Management – Potentiale, Probleme, Perspektiven

Klaus Herrmann ist Doktorand an der TU Berlin im Fachgebiet *Intelligente Netze und Management Verteilter Systeme*. Sein Diplom in Informatik erhielt er von der Johann Wolfgang Goethe-Universität in Frankfurt am Main. Anschließend arbeitete er als wissenschaftlicher Mitarbeiter in Frankfurt im Bereich Mobile Agenten und Netzmanagement. Seit seinem Wechsel an die Technische Universität Berlin arbeitet er an Middleware-Lösungen für die Selbstorganisation von Anwendungen in spontan vernetzten Systemen.

Gero Mühl erhielt 1998 das Diplom in Informatik und das Diplom in Elektrotechnik von der FernUniversität in Hagen. Im Jahr 2002 promovierte er an der Technischen Universität Darmstadt zum Dr.-Ing. mit dem Thema „Large-Scale Content-Based Publish/Subscribe Systems“. Seit Ende 2002 ist er wissenschaftlicher Assistent an der Technischen Universität Berlin am Fachgebiet von Prof. Geihs. Sein aktueller Forschungsschwerpunkt sind Algorithmen und Middleware für verteilte Systeme.

Prof. Dr. *Kurt Geihs* ist Inhaber der Telekom-Stiftungsprofessur Intelligente Netze und Management Verteilter Systeme an der Technischen Universität Berlin. Seine Forschungsinteressen liegen in den Bereichen Middleware, Management und Softwaretechnik für Verteilte Systeme. Themen aktueller Forschungsprojekte sind Dienstgüte in Middleware und komponentenbasierten Anwendungen, Agentensysteme für das Netzmanagement sowie Middleware für selbstorganisierende mobile, ad-hoc vernetzte Systeme. Vor dem Wechsel an die TU Berlin war er Professor an der Goethe-Universität Frankfurt und davor Wissenschaftler am IBM European Networking Center in Heidelberg. Weitere Informationen unter www.ivs.tu-berlin.de.

ZUSAMMENFASSUNG

Gordon Moores Gesetz vom exponentiellen Wachstum der Transistordichte pro Quadrat-Zoll hat seit 1965 die IT-Industrie geprägt. Mit der damit einhergehenden Explosion der Rechnerleistung wurde die Software immer leistungsfähiger, und man ist dazu übergegangen, Rechnersysteme zu vernetzen und Anwendungen zu verteilen. Eine Folge dieser Entwicklungen ist die rapide zunehmende Komplexität der modernen Informationstechnologie. 40 Jahre nach Moores Entdeckung droht eben diese Tatsache, dem bisherigen exponentiellen Wachstum natürliche Grenzen zu setzen. Moderne, vernetzte Rechnersysteme, wie sie in der Industrie weit verbreitet sind, sind schon heute zu komplex als dass sie auf manuellem Wege, d.h., durch menschliche Administratoren, in einem optimalen Betriebszustand gehalten werden können. Die Folgen sind eine unzureichende Ausnutzung vorhandener Ressourcen, wiederkehrende Fehlerzustände und Lücken in der Absicherung gegen mutwillige Angriffe auf die System-Integrität. Dies führt zu erheblichen finanziellen Mehraufwendungen bzw. Verlusten. Ein permanent überfordertes Administrationspersonal, trägt durch eigene Fehler ein Übriges bei.

Schenkt man den jüngst aufkeimenden Initiativen von IT-Giganten wie IBM, Microsoft und Sun Glauben, so heißt die Lösung dieser Misere *automatisiertes Management*. Vernetzte Rechnersysteme sollen sich auf lange Sicht selbst verwalten. Man erhofft sich hiervon ein effektiveres Management und eine Freistellung von Personal, welches sich dann um wichtigere Aufgaben kümmern kann.

In diesem Beitrag beleuchten wir den aktuellen Stand und die Perspektiven im Bereich des Self-Managements. Des Weiteren diskutieren wir offene Fragen, welche auf dem Weg zu selbstverwaltenden Systemen zu lösen sind.

1 EINLEITUNG

Die Entwicklung moderner IT-Landschaften ist durch eine Reihe von Trends bestimmt, welche die grundlegende Aufgabe des Management zunehmend erschweren:

- Die Größe einzelner Netzwerke und des gesamten Internet wächst rasant an.
- Die Heterogenität von Hard- und Software nimmt zu.
- Neue Vernetzungstechnologien halten Einzug: *Spontanvernetzung*, *Personal Area Networks*, *Wearable Computing*, etc.
- Mitarbeiter in Unternehmen werden zunehmend mobil und müssen in verstärktem Maße unterwegs auf Unternehmensdaten zugreifen.

– Die Abhängigkeiten zwischen Geschäftsprozessen und den darin involvierten Software-Produkten nehmen ständig zu.
– Verschiedene Technologien wie das Internet, Mobilfunksysteme und die Spontanvernetzung wachsen zusammen.
– Die uns umgebenden Rechnersysteme treten in zunehmendem Maße in den Hintergrund (*Ubiquitous Computing*, *Disappearing Computer*).
– Durch die sich beschleunigende technologische Entwicklung müssen auch Unternehmen in immer kürzeren Abständen mit Umstellungen im IT-Bereich reagieren.

All dies deutet darauf hin, dass die IT-Strukturen in großen Unternehmen noch um einiges komplexer werden, als dies heute schon der Fall ist. Die Frage, die sich angesichts dieser Entwicklung stellt, ist: Wie können derart vielschichtige Systeme noch effektiv verwaltet werden? Es ist nicht denkbar, dass menschliche Administratoren den Überblick über ein System behalten können, welches aus hunderten vernetzter Rechner und mobilen Klienten, sowie aus zahlreichen Anwendungs-Servern und Datenbanken besteht. Die Rolle des Menschen in diesem Prozess muss neu überdacht werden. Er kann nicht mehr eng gekoppelt und interaktiv in die Entscheidungsprozesse eingebunden werden (*Human-in-the-Loop*). Stattdessen sollte er sich darauf beschränken können, allgemeine Vorgaben (*Policies*) für die Steuerung eines Systems zu machen [Tenn00].

Diese Erkenntnis hat ein großes Interesse an Technologien für selbstverwaltende Systeme in Forschung und Industrie ausgelöst. Konzepte für die Entwicklung verlässlicher Systeme, welche Fehlerfälle kompensieren können und so für Robustheit und Stabilität sorgen, gibt es schon seit nahezu 40 Jahren [Aviz01]. Zur Gestaltung verlässlicher, fehlertoleranter Systeme, wurden vom IEEE bereits 1970 und von der IFIP 1980 entsprechende Gremien gegründet. Die jüngsten Bemühungen gehen jedoch in ihren Zielen deutlich weiter. Sie sind vom *Self-Management* als neuem Leitmotiv geprägt. Exemplarisch hierfür steht die 2001 ins Leben gerufene *Autonomic Computing Initiative* (im Weiteren mit ACI abgekürzt) von IBM [Gan03]. Diese umfasst eine weitreichende Vision, an deren Ende sich IT-Systeme komplett selbst verwalten sollen.

Die grundlegenden Konzepte der ACI sind nicht neu. Allerdings stellen sie eine gute begriffliche Basis dar und definieren ein aktuelles Referenzmodell, anhand dessen verschiedene Ansätze verglichen und eingeordnet werden können. Wir werden uns daher im Rest dieses Artikels an der ACI orientieren, um anhand ihrer Definitionen später offene Fragestellungen zu diskutieren.

2 SELF-MANAGEMENT IM SINNE DER ACI

Der Name *Autonomic Computing (AC)* ist durch das vegetative Nervensystem (engl.: *autonomic nervous system*) des Menschen inspiriert. Dieses System kontrolliert Körperfunktionen wie die Atmung und den Herzschlag, ohne dass der Mensch bewusst steuernd eingreifen muss. Genau dies ist die Hoffnung, die mit der ACI in Bezug auf die Komponenten eines komplexen Rechnersystems verbunden ist: Applikations-Server, Datenbanken und Kommunikations-Infrastrukturen sollen sich ohne Eingriff von außen verwalten. Die ACI unterteilt dieses Self-Management weiter in folgende Untergebiete:

– *Self-Configuration:* Automatische Neukonfiguration, um sich an veränderliche Umgebungen anpassen;
– *Self-Healing:* Aufdecken, Diagnose und Behebung von Störungen;
– *Self-Optimization:* Überwachung und Anpassung von Ressourcen im Sinne der betrieblichen und Kunden-bezogenen Anforderungen;
– *Self-Protection:* Antizipation, Identifikation und Schutz vor beliebigen Attacken.

Eine Umsetzung dieser Ziele geht nicht von heute auf morgen. Daher definiert IBM zusätzlich 5 Ebenen des Managements. Ebene 1 (*Basic Level*) beschreibt den Zustand des manuellen Managements einzelner Komponenten, in dem sich heutige Systeme befinden. Die Ebenen 2-4 definieren Zwischenzustände auf dem Weg hin zu völlig autonomen Systemen auf Ebene 5 (*Autonomic Level*). Die IT-Technologie soll sich in einem evolutionären Prozess bis hin zur Ebene 5 bewegen.

Die grundlegenden Bausteine eines autonomen Systems (*Autonomic System*) sind autonome Elemente (*Autonomic Elements*). Ein solches Element besteht aus einem verwalteten Element (*Managed Element*) und einem autonomen Manager (*Autonomic Manager*). Das verwaltete Element ist eine einzelne Ressource oder ein Zusammenschluss mehrerer Ressourcen und wird von seinem autonomen Manager verwaltet. Dabei arbeitet in dem autonomen Element ein geschlossener Regelkreis (Abb. 1). Geschlossene Regelkreise sind das Herzstück der ACI, da sie theoretisch in der Lage sind, ohne Eingriff von außen (geschlossen) das System so zu regeln, dass es immer im Sollzustand bleibt.

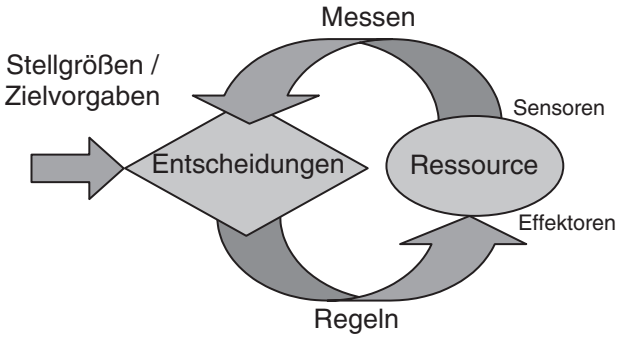

Abb. 1 Geschlossener Regelkreis eines autonomen Elements (adaptiert aus [Gan03])

3 AKTUELLE HERAUSFORDERUNGEN

Die Vision des Self-Management ist weitreichend und umfasst viele Teilprobleme, welche heute noch nicht gelöst sind. Zwar sind einzelne Mosaiksteine vorhanden: Beispielsweise existieren eine Reihe von Verfahren und Techniken zur Instrumentierung und Überwachung der Ressourcen [DG03]. Auch kann man den dezentralen Einsatz autonomer Agenten als eine Vorstufe des Self-Management ansehen [HG01]. Aber an vielen Stellen fehlen Antworten auf grundlegende Fragen des Self-Management. In diesem Abschnitt werden wir die offenen Probleme erörtern und einen Überblick über die forschungsrelevanten Themen geben.

Im Mittelpunkt stehen dabei geschlossene System-Management-Regelkreise und die offenen Fragen, welche auf dem Weg zu ihrer Realisierung beantwortet werden müssen. Das Konzept des Regelkreises (s. Abb. 1) ist aus vielen technischen Bereichen hinreichend bekannt und begegnet uns auch im All-

tag häufig. So handelt es sich bei einer Heizung mit Thermostat (bestehend aus Temperatursensor und gekoppeltem Durchflussregler) und beim Antiblockiersystem im Auto um *geschlossene Regelkreise*, d.h. um Regelkreise mit Rückkopplung. Diese arbeiten autonom ohne Eingriffe von außen. Im Gegensatz hierzu fehlt bei *offenen* Regelkreisen die Rückkopplung. Ein Beispiel hierfür ist die manuelle Regelung des Durchflusses bei einer Heizung. Die Regelung wird hierbei allein durch Eingriffe von außen erreicht. Streng genommen hat man es hier mit einer Steuerung und nicht mit einer Regelung zu tun. (Geschlossene) Regelkreise sind also in der Lage gemäß bestimmter Vorgaben (Zieltemperatur, Bremsdruck) und gegebener Rahmenbedingungen (aktuelle Umgebungstemperatur, Bodenbeschaffenheit, Radumdrehung), durch Beobachtung des Ist-Zustandes (mit Sensoren) und entsprechende Anpassungen (durch Effektoren), den vorgegebenen Soll-Zustand (definiert anhand von Stellwerten) zu erreichen und zu erhalten. Dies ist in gewissem Maße selbst dann möglich, wenn äußere Störgrößen (wie etwa ein offenes Fenster) auf die Regelung einwirken.

Das Streben nach selbstverwaltenden Rechnersystemen läuft auf die Anwendung desselben Prinzips hinaus. Allerdings sind moderne, vernetzte Rechnersysteme weitaus komplexer als die hier aufgeführten Beispiele. Ein Thermostat und das ABS sind auf ganz bestimmte Aufgaben festgelegt, bei denen die möglichen Ist- und Soll-Zustände, sowie die notwendigen Anpassungen komplett spezifiziert sind. Dies ist im Allgemeinen bei Rechnersystemen nicht der Fall. Die Kernprobleme sind dabei vor allem:

- die problemabhängige Beschreibung des Systemzustands und die potentiell sehr große Menge an Zustandsdaten;
- die Spezifikation der Stellgrößen und der gewünschten Soll-Zustände;
- die unvollständige Beschreibung der Systemreaktion auf Veränderungen der Eingaben und Systemparameter;
- die Fähigkeit des Managementsystems, das System an den gewünschten Zustand anzunähern, ungewünschte Oszillationen zu vermeiden und zwischen konkurrierenden Anforderungen zu vermitteln;
- die Fähigkeit des Managementsystems, zu lernen, unbekannte Situationen zu bewältigen und sich selbst an Veränderungen anzupassen;
- die verwobene Struktur komplexer Rechnersysteme und die damit verbundene Schwierigkeit einzelne Managementaufgaben zu isolieren.

3.1 Einheitliche Definitionen

Bevor wir diese technischen Kernprobleme näher beleuchten, wenden wir uns zunächst einer sehr grundlegenden Problematik zu: Bisher fehlt bei der Begriffsbildung und der Einordnung technischer Lösungen im Bereich des Self-Management die notwendige Klarheit. Zwar bietet die Definition der vier Teilbereiche der ACI eine sinnvolle Kategorisierung; bei näherer Betrachtung verschwimmen die Unterschiede jedoch. So führen z.B. Self-Healing, Self-Optimization und Self-Protection immer auch zu einer Änderung der Konfiguration. Die angewendeten Mechanismen könnten also auch der Self-Configuration zugeordnet werden. Die Anpassung eines Systems durch eine automatische Konfiguration dient wiederum immer der Optimierung. Wie grenzen sich die vier Teilbereiche also voneinander ab? Die Klärung dieser Frage ist im Hinblick auf eine wissenschaftliche Diskussion in einem größeren Rahmen unabdingbar, da ansonsten die Fülle unterschiedlicher Ansätze kaum vergleichbar ist.

Weitere Begriffe, die sich die Wissenschaft zu eigen gemacht hat sind *Self-Adaptation* und *Self-Organization*. Diese könnten als Oberbegriffe des Self-Management aufgefasst werden. Weiterhin ist es notwendig, eine Verbindung zu den klassischen Bereichen der verlässlichen und fehlertoleranten Systeme [Aviz01] herzustellen. Hier ist auch das Prinzip der Selbststabilisierung [Dij74] zu nennen. Dieses stellt sicher, dass ein System aus jedem fehlerhaftem Zustand letztendlich wieder in einen korrekten Zustand überführt wird, falls kein neuer Fehler auftritt. Hierfür ist es nicht einmal notwendig, dass das System erkennt, dass ein Fehler aufgetreten ist. Häufig ist die Erkennung eines Fehlers der schwierigste Schritt zu dessen Beseitigung. In den angesprochenen Gebieten existieren bereits Begriffe und Mechanismen, welche im Self-Management nutzbringend angewendet werden können. So hat es sich zum Beispiel bewährt, Ressourcen nur temporär durch zu erneuernde Nutzungsverträge (*Leases*) zu vergeben, um die dauerhafte Blokkierung von Ressourcen durch fehlerhafte Nutzer zu vermeiden. Diese Methode findet häufig im Bereich der Speicherbereinigung Anwendung.

3.2 Systembeschreibung und Interoperabilität

Für die autonome Verwaltung eines komplexen, heterogenen Systems benötigt man zunächst geeignete Mittel, um sowohl das Gesamtsystem, als auch seinen jeweiligen Ist- und Soll-Zustand zu beschreiben. Gegenstand einer solchen Beschreibung sind unter anderem:

- Die System-Architektur;
- Management-Policies;
- Ein Regelwerk für die Ableitung konkreter autonomer Eingriffe aus bestimmten Systemzuständen;
- Service-Level-Agreements und QoS-Verträge, welche die Anforderungen an das System festlegen.

Für jeden dieser Bereiche gibt es zwar schon Beschreibungsmöglichkeiten. Es fehlen allerdings herstellerübergreifende Standards, welche die Interoperabilität zwischen verschiedenen autonomen Teilsystemen sicherstellen. Im wissenschaftlichen Umfeld gibt es Ansätze, adaptive Systeme auf der Grundlage einer Architekturbeschreiung zu entwickeln [Orie99, Mage02, Garl03]. Im kommerziellen Sektor baut Microsoft momentan sehr stark auf seine *Dynamic Systems Initiative* [MS03], welche auf dem *System Description Model* als Kernstück aufbaut. Statische Beschreibungen reichen im Hinblick auf autonome Systeme nicht aus. Vielmehr ist es notwendig, dass nicht nur die verwalteten Systeme selbst adaptiert werden, sondern auch die Regeln, nach denen dies geschieht. Damit ein System sich überhaupt autonom verwalten kann, muss es im laufenden Betrieb lernen können, und diese Anforderung hat auch Auswirkungen auf die Gestaltung entsprechender Beschreibungstechniken.

Neben der Systembeschreibung müssen zusätzlich die Instrumentierung, die Schnittstellen und das Datenformat für die Erfassung relevanter System-Daten standardisiert sein. Hier will man z.B. auf der *Application Response Measurement (ARM) API* der Open Group [Open01] und dem *Common Information Model (CIM)* der DMTF [DMTF99] aufsetzen [Gan03].

3.3 Lernende Systeme

Die Qualität, welche ein menschlicher Administrator in das Management einbringt, ist seine Fähigkeit, unbekannte Situationen zu erfassen und zu handhaben. Er kann auch neu auftretende Fehler kategorisieren, beheben und aus ihnen lernen. Wenn der Mensch letztendlich aus dem Managementkreislauf verschwinden und nur noch als Entscheidungsträger für die Stellgrößen fungieren soll, muss das autonome System selbst lernfähig sein. Es muss aus Erfahrung lernen, seine Leistung stetig verbessern und sich an neue Situationen anpassen. Die Tatsache, dass neue Innovationen die Betreiber von Rechnersystemen in immer kürzeren Abständen zu Umstellungen zwingen, erschwert die Anpassung des Management-Systems enorm. Das Management muss sich also nicht nur auf ein statisches System einstellen, sondern es muss sich auch schnell an Veränderungen anpassen können. Die dynamische Anpassung an unbekannte Situationen und das Lernen aus Erfahrung sind Intelligenzleistungen, welche bisher in künstlichen Systemen nicht in dem Umfang nachgeahmt werden können, wie dies im Sinne des Self-Management notwendig erscheint.

Diao et al. haben gezeigt, dass man die Konfiguration eines Datenbanksystems für e-Commerce-Anwendungen während des laufenden Betriebes mittels eines Regel-basierten Ansatzes im Hinblick auf die Antwortzeit optimieren kann [Diao03]. Hierbei werden die Abhängigkeiten der Konfigurationsparameter untereinander mit einer Regelbasis beschrieben. Allerdings stellt dies nur den ersten Schritt dar. Es ist notwendig, dass sich auch die Anpassungs-Regeln selbst an veränderliche Bedingungen adaptieren, um den komplexen Wechselwirkungen zwischen den Anwendungen innerhalb eines Gesamtsystems, welches sich in einem ständigen Wandel befindet, gerecht zu werden. Wie soll das System reagieren, wenn neben e-Commerce-Systemen weitere Anwendungen mit zusätzlichen Anforderungen hinzukommen? Wie verhält sich das Optimierungs-System, wenn neben der Antwortzeit später z.B. auch der Speicherverbrauch optimiert werden soll? Im Fall dieser gegenläufigen Optimierungskriterien sind die rein Antwortzeit-orientierten Regeln nicht mehr adäquat. Eine einfache Ergänzung durch speicheroptimierende Regeln verfehlt ihr Ziel, da beide Optimierungsziele nicht unabhängig voneinander betrachtet werden können. Die Frage, wie ein selbstverwaltendes System in diesem Fall reagieren soll, bleibt offen. Es ergibt sich ein neues Optimierungsproblem bezüglich der Anpassung der Regelbasis.

3.4 Verwobene Systeme

In vielerlei Hinsicht ist der Vergleich mit dem menschlichen Körper und dessen vegetativen Nervensystem, der als Leitmotiv der ACI dient, durchaus zutreffend. Ein vernetztes Rechnersystem besitzt eine Komplexität, welche eher mit der des menschlichen Körpers als mit der eines Thermostats oder eines Antiblockiersystems vergleichbar ist. Hieraus ergeben sich weitreichende Konsequenzen.

Unser Körper befindet sich in einem fein austarierten Gleichgewichtszustand, in dem sich alle Teilsysteme dem gemeinsamen Ziel des Überlebens unterordnen. Um dies sicherzustellen, hat die Natur unserem Körper eine enorme Komplexität verliehen, die es uns erlaubt, auf die unterschiedlichsten Einflüsse angemessen zu reagieren. Unser Immunsystem ist z.B. in der Lage viele unbekannte Krankheitserreger zu erkennen, zu bekämpfen und sich auf einen erneuten Befall einzustellen. Es handelt sich also um einen sehr komplexen geschlossenen Regelkreis, welcher wiederum aus unzähligen untergeordneten Regelkreisen (Temperaturregulierung, Stoffwechsel, Atmung, etc.) besteht. Hierbei greifen alle Teilsysteme ineinander und interagieren auf eine sehr komplexe und bisher weitgehend unverstandene Weise. Der Versuch einzelne Teilsysteme zu isolieren, um sie komplett zu verstehen, scheitert in letzter Konsequenz, da die entscheidenden Wechselwirkungen (vgl. Abb. 2) mit anderen Systemen nicht mehr betrachtet werden. Dies ist das Wesen eines komplexen Systems: Das Ganze ist mehr als die Summer seiner Teile, und ein klassisch reduktionistisches Vorgehen bei der Analyse führt nicht zwingend zum Verständnis des Gesamtsystems.

Auch moderne, vernetzte Rechnersysteme besitzen eine *verwobene Struktur*. Gemeint ist hier nicht die physikalische Vernetzung der Einzelrechner, sondern die vielfältigen Abhängigkeiten zwischen den einzelnen Subsystemen, wie z.B. dem Betriebssystem, dem Kommunikationsnetz und den einzelnen Anwendungen, deren Daten sowie deren Subkomponenten. Das Problem, dass sich aufgrund dieser Abhängigkeiten das Gesamtsystem anders verhält, als man dies bei Betrachtung der einzelnen Komponenten erwartet, ist wohlbekannt. Änderungen an einem Subsystem haben meist Auswirkungen auf andere Subsysteme.

Hieraus entsteht ein Grundproblem für das Self-Management. Nach der Vorstellung der IBM-Visionäre kann ein *Autonomic System* tausende von Regelkreisen besitzen [IBM03]. Es ist vorstellbar, dass verschiedene Aspekte eines einzelnen Subsystems von unterschiedlichen Regelkreisen geregelt werden. Im Sinne der Modularität und der Trennung der Verantwortungen ist es weiterhin sinnvoll, dass jeder Hersteller einer Hardware- oder Software-Komponente für die Implementierung der hierfür notwendigen Regelkreise verantwortlich ist. Aufgrund der verwobenen Struktur des Gesamtsystems werden sich diese Regelkreise jedoch gegenseitig beeinflussen. Änderungen, welche an einem Subsystem durchgeführt werden, ziehen automatisch Änderungen an anderen Subsystemen nach sich, da deren Regelkreise Veränderungen in ihren Sensordaten feststellen und reagieren.

Ein Teil dieser Problematik spiegelt sich auch in einer Anekdote aus den siebziger Jahren (des letzten Jahrhunderts) wider, nach der die Operateure eines bestimmten Großrechner-Betriebssystems öfter feststellen mussten, dass das System nur noch mit der eigenen Verwaltung beschäftigt war: Das Seitenflattern der Hauptspeicherverwaltung und andere Verwaltungsvorgänge führten zu einer so hohen CPU-Auslastung, dass keine Nutzlast mehr verarbeitet wurde.

Betrachten wir als Beispiel das System aus [Diao03], welches die Antwortzeit einer Datenbank optimiert. Nehmen wir an, dass ein wesentliches Mittel hierfür die Vergrößerung des im Speicher gehaltenen Datenpuffers sei. Nehmen wir weiter an, dass parallel zu diesem *Antwortzeit-Regelkreis* ein *Speicher-Regelkreis* im Betriebssystem existiere, der eine allgemeine Speicherverknappung feststellen und über eine Standardschnittstelle beliebige Anwendungen auffordern kann, ihren Datenpuffer zu verringern. Dieses in Abb. 2 dargestellte Szenario kann dazu führen, dass das System aus Antwortzeit-Regelkreis, Speicher-Regelkreis und Datenbank durch die indirekte Wechselwirkung zwischen beiden Regelkreisen in Schwingung gerät. Dies geschieht dann, wenn beide abwechselnd die Größe des Datenbank-Puffers erhöhen und wieder verringern. Es ist sogar möglich, dass auf Grund des zusätzlichen Aufwan-

des beim Ein- und Auslagern des Pufferinhaltes die Antwortzeit der Datenbank entgegen dem Optimierungsziel des Antwortzeit-Regelkreises schlechter wird. Der Regelkreis könnte hieraus ableiten, dass eine weitere Erhöhung des Puffervolumens notwendig ist. Die Folge wäre, dass sich das System „aufschaukelt“ und schließlich durch diesen indirekten Rückkopplungseffekt „umkippt“.

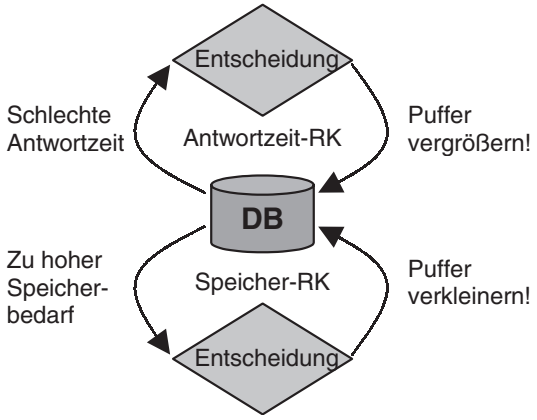

Abb. 2 Indirekte Wechselwirkung zwischen zwei Regelkreisen

Dieses einfache Beispiel zeigt, dass es notwendig ist, Wechselwirkungen in die Entwicklung einzelner Regelkreise mit einzubeziehen. Dies wirft zwei weitere Probleme auf:

1. In einem System mit tausenden von Regelkreisen sind die möglichen Wechselwirkungen nicht zu überblicken.
2. Die Kapselung der einzelnen Regelkreise muss aufgebrochen werden. Sie können nicht mehr unabhängig voneinander entwickelt werden.

Unsere beiden Regelkreise sollten in der Lage sein, „sich zu einigen“: Die anfänglich getrennten Optimierungsvorgaben müssen zugunsten eines „Kompromisses“ zwischen Speicherverbrauch und Antwortzeit aufgegeben werden (vgl.: Abschnitt *Lernende Systeme*). Wie kann dies erreicht werden? Müssen die Regelkreise einander kennen und explizit verhandeln, oder gibt es die Möglichkeit einer indirekten gegenseitigen Anpassung? Skaliert diese Anpassung mit der Anzahl der Regelkreise? Wie können Abhängigkeiten behandelt werden, welche sich über mehrere Indirektionen erstrecken? Wie kann sichergestellt werden, dass die gemeinsamen Optimierungsziele aller Regelkreise das System noch in einen „sinnvollen“ Zustand überführen? Wenn viele Regelkreise miteinander interagieren, welche Auswirkungen haben dann Fehler in ihrer Programmierung?

Die verwobene Struktur moderner Rechnersysteme und die Wechselwirkungen zwischen autonomen Regelkreisen stellen aus unserer Sicht das entscheidende Problem bei der Umsetzung der Ziele des Self-Management dar. Die Lösung der hier gestellten Fragen ist nur in Kooperation mit Forschern aus anderen Disziplinen wie z.B. der Biologie, der Soziologie und der Komplexen Systeme möglich.

4 LÖSUNGSANSÄTZE

IBM hat durch sein Engagement bei der ACI einen wichtigen Anstoß und Beitrag geleistet. So stellt die vorgeschlagene Architektur und die Begriffsbildung, auch wenn sie noch verfeinert und konkretisiert werden muss, eine Schärfung des Problembewusstseins und gute Grundlage dar. Allerdings werden hier vor allem die Fragen nach der Lernfähigkeit und nach dem Umgang mit der Verwobenheit selbstverwaltender Systeme nicht betrachtet. Die grundlegende Natur dieser offenen Probleme macht es notwendig, über sehr viel tiefgreifendere und teilweise radikale Paradigmenwechsel nachzudenken. In diesem Abschnitt werden wir kurz einige Ansätze präsentieren, welche Antworten auf die diskutierten Fragen liefern könnten.

4.1 Regelkreisorientierte Programmierung

Shaw hat schon 1995 ein alternatives Programmierparadigma vorgeschlagen, welches sich an Prozess-Regelkreisen orientiert [Shaw95]. Hierbei wird ein Problem nicht im Sinne der Objektorientierung in Komponenten zerlegt. Vielmehr werden die Prozesse eines Systems mittels entsprechender Abstraktionen abgebildet. So gibt es Prozessdefinitionen, Kontrollalgorithmen, Prozessvariablen, Soll-Werte und Sensoren. Weiterhin trennt dieses Paradigma den Prozess von externen Störungen in den Systemparametern. Diese prinzipielle Herangehensweise ist im Hinblick auf die Umsetzung Regelkreis-basierter Systeme, wie sie in der ACI definiert sind, viel versprechend.

4.2 Weiche Systeme und Homöostase

Shaw geht auch auf die Notwendigkeit ein, Systeme weich und flexibel zu gestalten [Shaw02]. Rigide Definitionen der gewünschten Systemzustände und die Forderung nach genau verifizierbarer Korrektheit entsprechen nicht mehr den Bedingungen, unter denen moderne Systeme arbeiten. Da das Wissen über die konkreten Arbeitsbedingungen meist unvollständig ist, tendieren Systeme, welche rigiden Definitionen unterliegen dazu *brüchig* zu werden. Die Autorin führt daher den Begriff der *hinreichenden Korrektheit* ein. Die Definitionen von Ist- und Soll-Zuständen sollten bewusst unpräzise gehalten werden, um genügend Raum für akzeptable Zustände zu schaffen. Weiterhin regt Shaw an, *homöostatische* Systeme zu bauen, welche sich permanent in Richtung eines akzeptablen Zustandes bewegen. Steuernde Eingriffe werden hierbei nicht als Reaktion auf einen diskreten Übergang eines Systems in einen „schlechten“ Zustand aufgefasst, sondern als andauernder Hintergrundprozess (vgl. selbststabilisierende Systeme). Ein einfaches Beispiel für ein homöostatisches System ist die *Garbage Collection* einer Java VM. Obwohl Shaw keine konkreten Mechanismen und Lösungen anbietet, gibt sie eine Richtung für die Forschung auf dem Gebiet des Self-Managements vor.

4.3 Der Körper als Vorbild

Forrester et al. beschreiben einen neuartigen Ansatz für die Entwicklung von Sicherheitssystemen, die auf Prinzipien basieren, welche denen des Immunsystems ähneln [Forr97]. Die Fähigkeit des Immunsystems, körpereigene von körperfremden Substanzen zu unterscheiden und auf der Basis dieser Entscheidung angemessen zu reagieren, wird anhand von Systemaufrufsequenzen nachgebildet. Mittels einer Datenbank, in der solche Sequenzen für „freundliche“ und „feindliche“ Aktionen abgelegt sind, kann das vorgestellte System Angriffe mit einer Genauigkeit zwischen 1% und 20% feststellen. Zwar ist das System, nicht zuletzt wegen der mangelnden Genauigkeit, noch weit von einem praktischen Einsatz entfernt, jedoch demonstriert es anschaulich den Nutzen eines bionischen Ansat-

zes für die „Selbstverteidigung“ eines selbstverwaltenden Systems.

4.4 Selbststrukturierung

Nagpal et al. zeigen einen Ansatz zur Selbststrukturierung zweidimensionaler Formen, welche aus einer großen Anzahl autonomer Akteure bestehen [Nag03]. Das Ergebnis hat zunächst keine praktische Relevanz im Bereich des Self-Managements. Es zeigt jedoch einen Ansatz zur abstrakten Beschreibung eines gewünschten Systemzustandes, welche automatisch auf einfache Regeln für die einzelnen Akteure herunter gebrochen wird. Wir halten dies für ein Schlüsselresultat, welches auch die Suche nach Beschreibungsmechanismen im Bereich des Self-Management (vgl. Abschnitt *Systembeschreibung und Interoperabilität*) stimulieren kann. Tatsächlich ist der Schritt von der Systemebene zur Komponentenebene in komplexen, selbstorganisierenden Systemen ein bisher wenig verstandenes Grundproblem bei der ingenieurstechnischen Umsetzung.

4.5 Zellen-orientierte Programmierung

George et al. zeigen einen rudimentären Ansatz für das Zellenorientierte Programmieren [Geor02]. Hierbei werden Strukturen aus einzelnen Komponenten (Zellen) erzeugt. Diese Strukturen besitzen die Fähigkeit zur Selbstheilung, also zum Erhalt der vorgegebenen Struktur. Der Mechanismus, der ihnen dies erlaubt, basiert auf der Diffusion von Chemikalien. Die Zellen emittieren diese, nehmen ihre Konzentration wahr und reagieren darauf. Die Kopplung zwischen den Zellen ist sehr lose und die Kommunikation verläuft indirekt und lokal. Auch dies ist zunächst ein sehr grundlegendes Resultat, welches nicht direkt praktisch einsetzbar ist. Allerdings zeigt es einen möglichen biologischen Ansatz zur Konstruktion selbstheilender Systeme, welcher in der Zukunft an Bedeutung gewinnen könnte.

5 ZUSAMMENFASSUNG UND AUSBLICK

Die steigende Komplexität vernetzter Rechnerumgebungen führt zu einem rasant wachsenden Aufwand für deren Management. Dies treibt sowohl die Forschung als auch die Industrie an, selbstverwaltende Systeme zu entwickeln. Obwohl in jüngster Zeit vor allem in der Industrie ein verstärktes Engagement in diese Richtung zu beobachten ist, und obwohl es schon seit einigen Jahrzehnten verwandte Ansätze in der Forschung gibt, stehen wir immer noch ganz am Anfang dieser Entwicklung. In diesem Artikel haben wir die wichtigsten Fragen erörtert. Die offenen Probleme sind dabei hauptsächlich:

- Die mangelnde Klarheit in der Definition von Self-Management;
- das weitgehend unverstandene Phänomen der Verwobenheit und der daraus entstehenden Komplexität;
- das Fehlen allgemeiner Konzepte für die Entwicklung lernfähiger Systeme;
- die Notwendigkeit, herstellerübergreifende Standardmechanismen zur Beschreibung von Systemen, Regeln und Policies zu finden, welche die Eigenschaften selbstverwaltender Systeme berücksichtigen.

Zur Lösung der vor uns liegenden Probleme müssen unserer Meinung nach unkonventionelle Wege beschritten werden, da klassische Mechanismen des Designs, der Entwicklung und des Einsatzes verteilter Systeme nicht weiterführen. Im letzten Teil dieses Beitrages haben wir daher einige Ansätze erläutert, die wegweisend sein könnten.

Die Hauptaufgabe von Forschung und Industrie ist es nun, auf Basis einer allgemein akzeptierten Definition von Self-Management die existierenden Ansätze zusammenzuführen und zu einer gemeinsamen Lösung zu verbinden. Der Weg zu dieser Lösung ist jedoch noch lang, da man auf einigen Schlüsselgebieten noch immer in der Grundlagenforschung steckt.

6 LITERATUR

[Aviz01] A. Avizienis; J.-C. Laprie; B. Randell: Fundamental Concepts of Dependability. Research Report N01145, LAAS-CNRS, April 2001.

[Diao03] Y. Diao; F. Eskesen; S. Froehlich; J.L. Hellerstein; L.F. Spainhower; M. Surendra: Generic Online Optimization of Multiple Configuration Parameters With Application to a Database Server. In: 14th IFIP/IEEE Workshop on Distributed Systems: Operations and Management (DSOM 2003), Heidelberg, October 2003.

[DG03] M. Debusmann; K. Geihs: Efficient and Transparent Instrumentation of Application Components using an Aspect-oriented Approach. In: 14th IFIP/IEEE Workshop on Distributed Systems: Operations and Management (DSOM 2003), Heidelberg, October 2003.

[Dij74] E.W. Dijkstra: Self-stabilizing systems in spite of distributed control. Communications of the ACM, 17(11): 643-644, 1974.

[DMFT99] Distributed Management Task Force: Common Information Model (CIM) Specification Version 2.2. DSP0004, June 1999)

[Forr97] S. Forrest; S.A. Hofmeyr; A. Somayaji: Computer immunology. Communications of the ACM, v. 40 n. 10, p. 88-96, October 1997.

[Gan03] A.G. Ganek; T.A. Corbi: The dawning of the autonomic computing era. IBM SYSTEMS JOURNAL, VOL 42, NO 1, 2003.

[Garl03] D. Garlan; S. Cheng; B. Schmerl: Increasing System Dependability through Architecture-based Self-repair. In: Architecting Dependable Systems, R. de Lemos, C. Gacek, A. Romanovsky (Eds.), Springer-Verlag, 2003.

[Geor02] S. George; D. Evans; L. Davidson: A biologically inspired programming model for self-healing systems. In: Proceedings of the First ACM SIGSOFT Workshop on Self-Healing Systems (WOSS '02), Charleston, SC, November 2002.

[HG01] K. Herrmann; K. Geihs: Integrating Mobile Agents and Neural Networks for Proactive Management. In: IFIP International Working Conference on Distributed Applications and Interoperable Systems (DAIS '01), Krakow, Poland, 2001. Chapman-Hall.

[IBM03] IBM Corporation: An Architectural Blueprint for Autonomic Computing. IBM White Paper, April 2003.

[Mage02] I. Georgiadis; J. Magee; J. Kramer: Self-Organising Software Architectures for Distributed Systems. In: Proceedings of the First ACM SIGSOFT Workshop on Self-Healing Systems (WOSS '02), Charleston, SC, November 2002.

[MS03] Microsoft Cooperation: Microsoft Dynamic Systems Initiative. White Paper, October 2003.

[Open01] The Open Group: Application Response Measurement Issue 3.0 Java Binding. Open Group Technical Standard CO14, October 2001.

[Orie99] P. Oriezy; M.M. Gorlick; R.N. Taylor; G. Johnson; N. Medvidovic; A. Quilici; D. Rosenblum; A. Wolf: An Architecture-Based Approach to Self-Adaptive Software. IEEE Intelligent Systems 14(3): 54-62, May/June 1999.

[Nag03] R. Nagpal; A. Kondacs; C. Chang: Programming Methodology for Biologically-Inspired Self-Assembling Systems. AAAI Spring Symposium on Computational Synthesis: From Basic Building Blocks to High-Level Functionality, March 2003.

[Shaw02] M. Shaw: ‚Self-Healing‘: Softening Precision to Avoid Brittleness. In: Proceedings of the First ACM SIGSOFT Workshop on Self-Healing Systems (WOSS '02), Charleston, SC, November 2002, pp. 111-113.

[Shaw95] M. Shaw: Beyond Objects: A Software Design Paradigm Based on Process Control. ACM Software Engineering Notes, Vol 20, No 1, January 1995.

[Tenn00] D.L. Tennenhouse: Proactive Computing. Communications of the ACM 43, No. 5, 43-50. May 2000.