Selbstorganisation – Hype oder Möglichkeit

Thomas Pilger

Seminar Technische Informatik
Institut für Informatik, Freie Universität Berlin
Berlin, Deutschland
E-Mail: pilger@inf.fu-berlin.de

Zusammenfassung — In der Informatik wird der Ruf immer lauter nach alternativen Systemarchitekturen, da die Entwicklung immer weiter voranschreitet und immer komplexere Systeme verwaltet und gewartet werden müssen. Vor allem im Bereich der verteilten Systeme ist es wünschenswert dem Administrator, bzw. Benutzer so viel Organisation wie möglich zu ersparen. Dieses Paper beschäftigt sich damit, ob in dem Ansatz der Selbstorganisation ausreichend Potenzial steckt, damit davon zukünftig profitiert werden kann. Dazu wird auf verschiedene Möglichkeiten der Umsetzung, in Form verschiedener Systemarchitekturen, eingegangen und Messmöglichkeiten für Selbstorganisation vorgestellt. Abschließend findet eine Bewertung der Güte dieser Metriken statt und es wird ein Ausblick auf die Zukunft selbstorganisierender Systeme gegeben.

***Schlüsselbegriffe* — Selbstorganisation, Autonomic Computing, Organic Computing, Metriken**

I. Einführung

Die heutigen Computersysteme, vor allem im Bereich der verteilten eingebetteten Systeme, werden immer komplexer. Oftmals muss eine große Anzahl an Netzwerkelementen, so genannte Knoten, verwaltet, konfiguriert, gewartet und repariert werden. Außerdem soll das System so flexibel sein, dass Konfigurationsänderungen zur Laufzeit vorgenommen werden können, um z.B. Unterbrechungen des Betriebs zu vermeiden. Ein menschlicher Administrator wäre hier unter Umständen überfordert.

Auch in kritischen Systemen, wie bei einem Flugzeug oder in einem Kernkraftwerk ist es wünschenswert, bestimmte Wartungs- oder Reparaturarbeiten nicht von Menschenhand durchführen zu lassen, sondern zu automatisieren. Bei falscher Handhabung oder Flüchtigkeitsfehlern liegen hier die verhängnisvollen Konsequenzen auf der Hand.

Ein möglicher Ansatz für diese Automatisierung ist die Selbstorganisation dieser Systeme. Selbstorganisation bedeutet für uns so viel wie seinen eigenen Tagesablauf oder sein eigenes Leben zu planen und sich die vorhandene Zeit selbst einzuteilen. Selbst- und Zeitmanagement sind hier die Schlagwörter. Aber der Begriff der Selbstorganisation kann auch weiter gefasst werden, als ihn lediglich auf die zeitliche Ebene zu beschränken.

Wenn man z.B. den menschlichen Körper betracht, dann bekommt man eine Ahnung davon, was Selbstorganisation bedeuten kann. Unser Körper ist in dieser Hinsicht ein faszinierendes System. Er kann sich auf viele verschiedene Umgebungen von selbst einstellen, ohne dass man etwas aktiv dazu tun muss. Als Beispiel sei die sportliche Aktivität oder die Sauna gewählt. Sobald die Temperatur des Körpers ansteigt, wird dies bemerkt und (bei einem gesunden Körper) erhöht nun das vegetative Nervensystem automatisch den Puls und veranlasst eine vermehrte Flüssigkeitsbildung auf der Haut (Transpiration), um den Körper durch das Prinzip der Verdunstung abzukühlen, bis wieder ein ausgeglichener Zustand hergestellt ist.

Bei dieser sogenannten Thermoregulation passiert vereinfacht folgendes: Zunächst ändert sich der Zustand des Körpers, da seine Temperatur ansteigt. Dieser thermische Anstieg wird vom Nervensystem „beobachtet“ und „verarbeitet“. Es wird festgestellt, dass ein bestimmter Körpertemperaturwert überschritten wurde. Daher wird nun versucht, dieser Tatsache durch eine geeignete Kontrollmaßnahme, dem stärkeren Transpirieren, entgegenzuwirken. Jetzt ändert sich (hoffentlich) wieder der Zustand des Körpers und seine Temperatur nimmt ab. Falls der erwünschte Effekt aber nicht auftritt, müssen weitere Maßnahmen des vegetativen Nervensystems ergriffen werden. Dieser Kreislauf von Beobachtung, Verarbeitung und möglichen Ausübung einer Kontrollmaßnahme wird immer wieder durchschritten, so dass sich der Körper ständig selbst regulieren kann.

Dies ist nur ein Beispiel für die vielfältige Existenz von selbstorganisierenden Systemen in der Natur. Das Ziel ist es nun, dieses Prinzip auch auf die Welt der Computersysteme zu übertragen. Zu diesem Zweck benötigen wir eine präzise Definition des Begriffs der Selbstorganisation.

II. Definition von Selbstorganisation

Das erste Mal, dass der Begriff der Selbstorganisation abgedruckt erschien, war 1947 im Journal of General Psychology. Damals definierte W. Ross Ashby die Organisation in einem System als die Regel, die aktuelle Status in zukünftige Status überführt. Also muss ein selbstorganisierendes System seinen Status von selbst ändern können. (vgl. [1])

Seitdem hat sich der Begriff der Selbstorganisation sehr verändert und es haben sich mittlerweile verschiedenste Definitionen herausgebildet, die jeweils ein bestimmtes Merkmal besonders hervorheben.

Zum Beispiel besagt eine Definition, Selbstorganisation sei die Möglichkeit eines nicht ausgeglichenen Systems,

Strukturen und Muster in Abwesenheit von externer Kontrolle oder Manipulation zu entwickeln. (vgl. [2])

Wieder anders heißt es, dass Selbstorganisation ein Prozess ist, in dem die Organisation eines Systems spontan zunimmt, d.h. ohne dass diese Zunahme durch die Umgebung oder ein umgebendes oder anderes externes System vorgenommen wird. (vgl. [3])

Eine sehr ausführliche und genaue Definition gibt C. Lucas in seinem Self-Organizing Systems FAQ, welche beschreibt, dass das Wesentliche der Selbstorganisation sei, dass sich eine Systemstruktur durch die Interaktion der einzelnen Elemente herausbildet. Dies geschehe oft ohne expliziten äußeren Anstoß oder Einbeziehung äußerer Elemente. Dabei kann sich die Organisation im zeitlichen oder räumlichen entwickeln und stabiler oder vorübergehender Natur sein. (vgl. [4])

In der Regel haben selbstorganisierende Systeme vier Eigenschaften:

- Komplexität: Komplexität heißt in diesem Fall, dass die verschiedenen Elemente untereinander Beziehungen haben, die sich kontinuierlich ändern können. Auch die Elemente an sich können ständigen Änderungen unterworfen sein. Dadurch wird es natürlich schwer, das System komplett zu erfassen, oder irgendwelche Vorhersagen darüber zu machen.
- Selbstreferenz: Wenn das System selbständig eine Aktion durchführt, dann hat das Auswirkungen auf den Zustand des Systems und das wirkt wiederum auf die nächsten Aktionen des Systems ein.
- Redundanz: Alle Komponenten in einem selbstorganisierenden System können organisieren, gestalten und lenken, dadurch entsteht Redundanz.
- Autonomie: Das System verwaltet sich selbst, ohne eine Eingabe von außen, d.h. das System bestimmt selbst, welche Beziehungen und Interaktionen zwischen den Einzelkomponenten auftreten. (vgl. [5])

Zusammengefasst kann also gesagt werden, dass Selbstorganisation als eine Systementwicklung, eine Möglichkeit der Strukturbildung, ein Prozess und eine Entwicklung ohne äußere Einflüsse, definiert wird. Das Ergebnis von Selbstorganisation führt oft zu spontanen, unvorhergesehenen Effekten. Diese Effekte werden als Emergenz bezeichnet.

Emergenz kann auf das lateinische Verb emergo zurückgeführt werden und bedeutet im Allgemeinen „auftauchen“ oder „sich herausarbeiten“. (vgl. [6, S.7])

Emergenz bedeutet so viel wie das Auftauchen einer Eigenschaft, die vorher nicht als ein funktionales Charakteristikum des Systems beobachtet wurde. Generell werden nur Eigenschaften, die auf einer höheren Ebene auftreten als emergent betrachtet. Zum Beispiel ist ein Automobil eine emergente Eigenschaft seiner untereinander verbundenen Teile. Diese emergenten Eigenschaften verschwinden aber, wenn die Einzelteile getrennt und zu einem Stapel aufgeschichtet werden. (vgl. [4])

Mit den Worten des griechischen Philosophen Aristoteles kann man Emergenz auch als „Das Ganze ist mehr als die Summe seiner Teile.“ definieren.

Da Emergenz unerwartet auftritt, kann es natürlich passieren, dass unerwünschte Effekte entstehen. Daher muss man versuchen, die erwünschten emergenten Effekte in selbstorganisierenden Systemen zu nutzen und die unerwünschten zu vermeiden.

Betrachten wir nun zwei Ansätze für die Umsetzung selbstorganisierender Systeme, den Ansatz des Autonomic Computing und des Organic Computing.

III. Stand der Dinge

A. *Autonomic Computing (AC)*

Dieser Ansatz wurde von IBM entwickelt. Paul Horn definierte 2001 den Begriff des Autonomic Computing als „computer systems that regulate themselves much in the same way our autonomic nervous system regulates and protects our bodies.” [7]

Laut IBM muss das neue Computerparadigma aus Nutzersicht folgende Grundeigenschaften besitzen:

- Flexibilität: Das System muss in der Lage sein, Daten von überall her zu bekommen und zu verarbeiten.
- Erreichbarkeit: Die Natur eines autonomen Systems ist es, immer erreichbar zu sein.
- Transparenz: Das System muss seine Aufgaben erfüllen und sich an die Bedürfnisse des Benutzers anpassen, ohne ihn in komplizierte Feinheiten einzubeziehen. (vgl. [8])

Um so ein System umzusetzen und um den Begriff des Autonomic Computing zu verfeinern, definierte IBM acht Elemente, die ein autonomes System charakterisieren:

1. Ein autonomes System muss detaillierte Informationen über sich und alle seine Komponenten haben.
2. Es muss sich selbst konfigurieren und rekonfigurieren können, um mit unterschiedlichen und wechselnden Umgebungen fertig zu werden.
3. Ein autonomes Computersystem muss ständig versuchen seine Arbeitsweise zu optimieren, um dadurch seine vorgegebenen Ziele zu erreichen.
4. Ein autonomes System muss etwas Ähnliches wie Heilung besitzen. Es muss sich von Fehlfunktionen erholen können, Probleme erkennen können und sie beheben können.
5. Ein autonomes Computersystem muss alle Arten von Angriffen entdecken, identifizieren und sich dann davor schützen können.
6. Es muss seine Umgebung und den Kontext seiner Aktivität kennen und dabei die besten Regeln zur Interaktion mit den Nachbarn finden und sich an die Gegebenheiten anpassen können.
7. Ein autonomes System kann nicht in einer hermetisch abgeschlossenen Umgebung sein. Es muss, per Definition, trotz seiner Fähigkeit sich selbst zu

managen, in einem heterogenen Umfeld bestehen und offene Standards implementieren können.

8. Ein autonomes System muss die optimal benötigten Ressourcen bereitstellen können und sich an den Benutzer anpassen, ohne den Benutzer in das komplexe System mit einzubeziehen. (vgl. [9])

Aus diesen Elementen können nun sogenannte Self-x-properties abgeleitet werden:

- Self-Configuration: Automatische Konfiguration der einzelnen Komponenten, die vorgegebenen Richtlinien folgt.
- Self-Healing: Automatische Findung, Analyse und Korrektur von Fehlern und Fehlfunktionen. Dadurch wird das ganze System robuster und unempfindlicher gegen Systemausfälle.
- Self-Optimization: Automatische Beobachtung und Kontrolle von Ressourcen, um diese optimal einzusetzen.
- Self-Protection: Das System verteidigt sich automatisch gegen böswillige Angriffe und warnt früh vor Systemfehlern. (vgl. [10, S.43, Table 1])

Eine weitere Self-x-Eigenschaft ist die der Self-Explanation (Selbsterklärung), denn ein Mensch, der ein autonomes System warten und pflegen muss, braucht unbedingt genaue Informationen über den aktuellen Systemstatus.

Da es schwierig ist, ein komplettes Autonomic Computing System mit all diesen Eigenschaften auf einmal zu erstellen, wurden von IBM fünf hierarchisch geordnete „Reifegrade“ im „IBM Autonomic Computing Adoption Model“ festgelegt:

- Level 1: Basic
  In diesem Level werden alle Systemelemente unabhängig voneinander von IT-Fachleuten betrieben und gewartet. Die meisten heutigen Systeme sind auf diesem Level.
- Level 2: Managed
  Hier werden System Management Werkzeuge benutzt, um Informationen über verschiedene Systemelemente des Systems zu sammeln und zusammenzufassen, so dass weniger administrative Arbeit benötigt wird.
- Level 3: Predictive
  Das System beobachtet und verknüpft Daten, um zu versuchen, gewisse Systemstrukturen zu erkennen. Dadurch kann es den Administratoren Vorschläge unterbreiten, welche Einstellungen sie vornehmen sollten.
- Level 4: Adaptive
  Zusätzlich zur Sammlung und Verknüpfung der Daten kann das System auch Aktionen auf Basis der vorhandenen Informationen ausführen.
- Level 5: Autonomic
  Vollständig integrierte Systeme und Komponenten werden dynamisch durch Regeln und Richtlinien gesteuert. (vgl. [11])

Um ein Computersystem zu entwerfen, muss man dieses zunächst planen und eine geeignete Architektur erstellen. IBM hat für Autonomic Computing eine Architektur namens

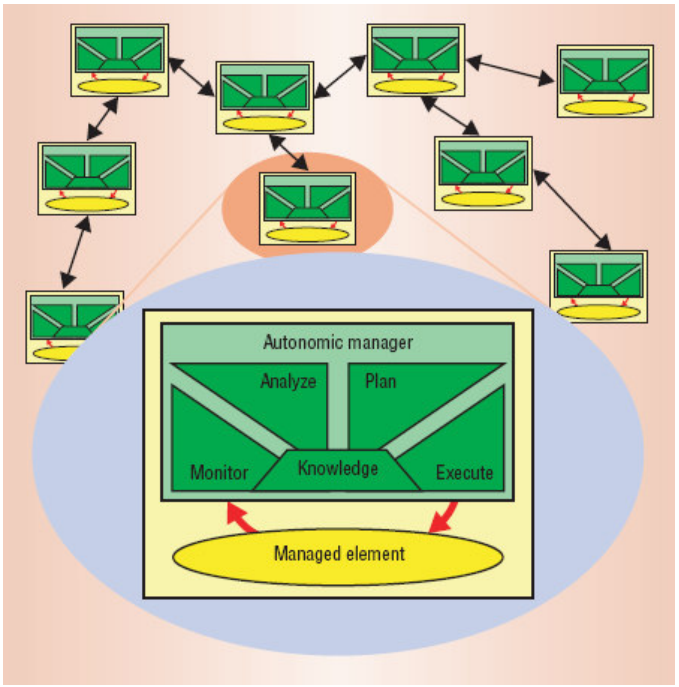

Abbildung 1. Struktur eines Autonomen Elementes [10, S.44]

MAPE (Monitor, Analyse, Plan, Execute) entworfen, wie in Abbildung 1 illustriert.

Autonome Systeme bestehen aus vielen autonomen Elementen. Diese Elemente können wiederum Ressourcen beinhalten oder Dienste für den Menschen bzw. andere autonome Elemente anbieten. Typischerweise besteht ein autonomes Element aus einem oder mehreren Managed Elements und einem Autonomic Manager.

Der Autonomic Manager ist das Herzstück der Architektur und ist für die Ausführung der einzelnen Schritte Monitor, Analyse, Plan und Execute zuständig. Dabei kann er auf ein Knowledgemodul zugreifen, das eine Historie und aktuelle Daten enthält, z.B. die aktuell benötigten Regeln für die Analyse.

Einen Prozessablauf kann man sich so vorstellen: Der Autonomic Manager beobachtet mit seinem Monitormodul das Managed Element und sammelt Informationen durch verschiedene Sensoren, verarbeitet sie nach bestimmten Regeln aus dem Knowledgemodul und reagiert auf bestimmte Ereignisse. Wenn so ein Ereignis auftritt, dann wird die Information darüber an das Analysemodul übergeben. Hier wird nun das Ereignis analysiert. Wenn eine Änderung erforderlich wird, wird alles an das Planmodul weitergegeben, aber das Analysemodul merkt sich den Sachverhalt und beobachtet, ob das gewünschte veränderte Verhalten des Systems auch eintritt. Das Planmodul erstellt einen zum Problem passenden Änderungsplan. Nun wird dieser an das Executemodul weitergegeben. Dieses führt die gewünschten Änderungen dann aus, indem es die Aktionen an so genannte Effektoren weitergibt, die die eigentliche physikalische Ausführung durchführen. Außerdem aktualisiert das Executemodul das Knowledgemodul. Dieser Prozess findet

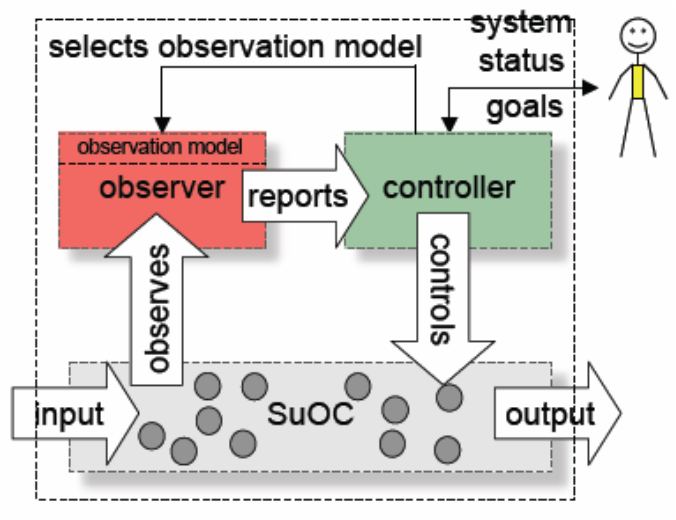

Abbildung 2. Observer/Controller – Architektur [12, S.11]

ständig statt, so dass das Managed Element beständig korrigiert wird.

Vergleicht man MAPE mit der Observer/Controller-Architektur (siehe Absatz III B 1), so besteht praktisch der Observer aus Monitor- und Analysemodul und der Controller aus Plan- und Executemodul. (vgl. [12, S.30-38])

B. Organic Computing (OC)

2002 gab es einen Workshop zum Thema Zukunftsthemen der technischen Informatik, in dem die Ergebnisse mehrerer Workshops, die im Verlaufe des Jahres durchgeführt worden sind, zusammengefasst wurden. In den Workshops wurden natürliche Mechanismen, die auch in der IT zu gebrauchen sind, gesucht und es hat sich der Begriff des Organic Computing herausgebildet. (vgl. [13, S.3])

Es wurden wie beim Autonomic Computing die natürlichen Mechanismen, wie Selbstheilung und Selbstorganisation aus der Natur abgeschaut.

Auch hier ist es nötig, Systemarchitekturen zu entwerfen, so dass man OC- Systeme umsetzen kann. Wir werden nun 3 davon genauer betrachten.

1) Observer/Controller

Die Observer/Controller- Architektur wurde an den Universitäten Karlsruhe und Hannover entwickelt und ist die Architektur, die im OC am häufigsten zu finden ist. In Abbildung 2 wird die Struktur dieser Architektur anschaulich dargestellt.

In dieser Architektur gibt es folgende Komponenten, die zusammenarbeiten, bzw. Beziehungen untereinander haben:

- System under Observation and Control (SuOC): Das System, dass zu überwachen und kontrollieren ist, besteht aus vielen interagierenden, sogenannten Agenten.
  Input: Eingaben können System- oder Umgebungsparameter sein, aber auch Informationen anderer komplexer Systeme.
  Output: Ausgaben können Kontrollsignale sein, der aktuelle Status des Systems oder wiederum Informationen für andere Systeme.
- Observer: Der Observer ist für die Überwachung des SuOC zuständig und soll das zukünftige (emergente) Verhalten beobachten, bewerten und voraussagen. Welche Attribute der Observer beobachten soll wird im Observation Model festgelegt. Hier wird auch festgelegt, wie die Attribute analysiert werden sollen und nach welchen Kriterien die Vorhersage geschehen soll. Die aufbereiteten Daten werden dann dem Controller zur Verfügung gestellt.
- Controller: Der Controller interpretiert die Informationen des Observers und versucht daraufhin das Verhalten des SuOC so zu beeinflussen, dass die von außen eingegebenen Systemziele erreicht werden. Dabei ist es seine Aufgabe, erwünschtes emergentes Verhalten herbeizuführen und unerwünschtes zu vermeiden bzw. zu unterbrechen, indem er durch seine Kontrollparameter in das System eingreift. Im besten Fall erfüllt der neue Zustand des Systems, der durch das Eingreifen des Controllers entsteht, alle geforderten Systemziele. Ansonsten sind weitere Zyklen nötig, um zum gewünschten Ziel zu gelangen. Bei der Analyse und Auswahl der besten Vorgehensweise kann der Controller auf eine Historie zugreifen und dadurch auch bewerten, ob eine Annäherung an das eingegebene Systemziel erfolgt ist, oder nicht. (vgl. [12, S.7-10])

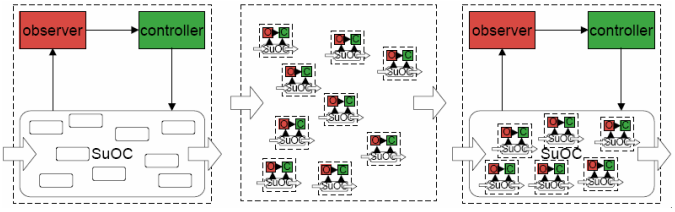

Abbildung 3. Observer/Controller Realisationsszenarios [12, S.21]

Außerdem kann man die Observer/Controller- Architektur in drei verschiedene Realisierungszenarien untergliedern, und zwar in eine zentrale, eine verteilte und eine Multi-Level-Architektur, die in Abbildung 3 veranschaulicht sind. Dabei bedeutet zentral, dass man nur einen Observer/Controller für das ganze System hat, verteilt heißt, dass jedes Subsystem einen eigenen Observer/Controller hat und Multi-Level bedeutet beide Architekturen zusammen, d.h. einen Observer/Controller für jedes Subsystem und einen für das Ganze System. Natürlich ist es auch denkbar, dass es Hybridarchitekturen davon gibt. (vgl. [12, S.21])

2) Autonomous Middleware for Ubiquitous eNvironments (AMUN)

Die Autonomous Middleware for Ubiquitous eNvironments (AMUN) stellt an sich eine Erweiterung und Verfeinerung der Observer/Controller- Architektur dar. Sie wurde für das Smart Doorplate- Projekt entworfen. (vgl. [12, S.40])

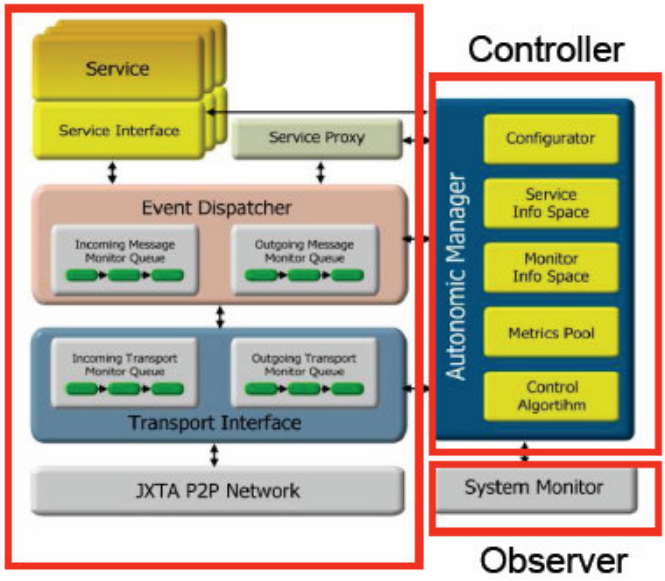

Abbildung 4. AMUN- Architektur
[12, S.47]

In diesem Projekt der Universität Augsburg stellen Türschilder „aktuelle Informationen über Büroinhaber dar, können in deren Abwesenheit bestimmte Handlungen übernehmen und weisen Besuchern den Weg zum gegenwärtigen Aufenthaltsort des Büroinhabers über ein Location-Tracking System.“ [14]

Abbildung 4 illustriert die Architektur von AMUN. Sie beinhaltet folgende Komponenten:

- Transport Interface: Das Transport Interface ist die Schnittstelle, die für die Entkopplung der Nachrichtenübermittlung zuständig ist.
- Event Dispatcher: Der Event Dispatcher stellt die Nachrichten an das richtige System zu, d.h. er entscheidet, ob die Nachricht lokal oder von einem Dienst bearbeitet werden muss. Ein Dienst muss sich daher bei ihm registrieren, so dass er weiß, dass dieser Dienst vorhanden ist.
- Service Interface: Dies ist die Schnittstelle, über die sich die Dienste an das System koppeln können. Somit können sie die Nachrichten ebenfalls über den Event Dispatcher erhalten. Falls ein Dienst seinen Knoten wechselt, leitet der Service Proxy die Nachricht weiter.
- Autonomic Manager: Hier werden alle Informationen zentral gesammelt. Der Autonomic Manager besteht aus dem Konfigurator, der für die Konfiguration der Dienste auf einem Knoten zuständig ist, dem Kontrollalgorithmus, welcher den aktuellen Status mithilfe der Metrik aus dem Metrikpool auswertet und die Dienste rekonfiguriert und dem Service und Monitor Info Space, die Speicherplatz für aktuelle Statusinformationen bereit halten. Der Autonomic Manager entspricht sozusagen dem Controller.
- System Monitor: Zuständig für die Überwachung des Systems. Der System Monitor entspricht sozusagen dem Observer.(vgl. [12, S.40-46])

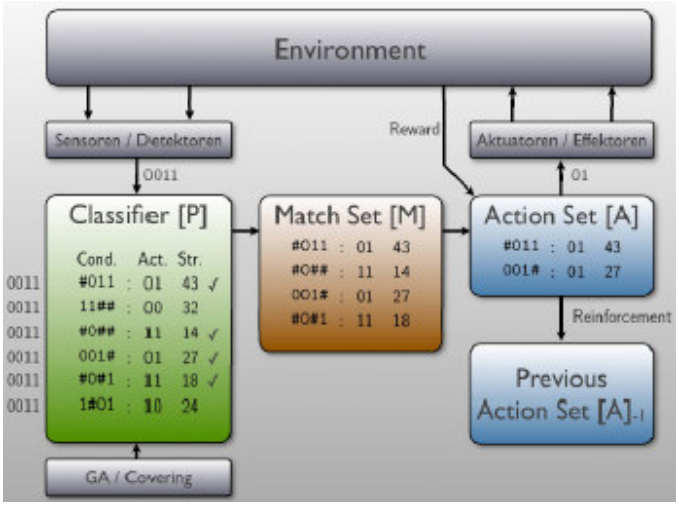

Abbildung 5. ZCS- Architektur
[16]

3) Zeroth Level Classifier System (ZCS)

Ebenfalls eine Erweiterung der Observer/Controller-Architektur stellt das Zeroth Level Classifier System (ZCS) dar, das in Abbildung 5 veranschaulicht ist.

Das ZCS ist ein lernendes System, in dem eine Menge von Bedingungs- Aktions- Regeln, so genannte Classifier, das System kontrollieren. Jeder Classifier hat außerdem einen Gewichtungswert, der beschreibt, wie gut die Regel ist, die sogenannte Stärke. Diese Classifier kann man sich also als Tupel aus Bedingung, Aktion und Stärke vorstellen. Die Bedingungen und Aktionen sind binär codiert. Neben Nullen und Einsen in einer Bedingung, kann auch der Platzhalter # benutzt werden, der dann als 0 oder 1 gewertet wird. Daher kann es für die gleiche Bedingung auch mehrere passende Classifier geben, die die gleiche Aktion beinhalten.

Das ZCS hat Detektoren, die Informationen aus der Umgebung auswerten, d.h. bestimmte Parameter messen. Wenn ein Ereignis eintritt, dass ein Eingreifen erfordert, wird ein sogenanntes Match Set erstellt. Die von der Bedingung her passenden Classifier werden in das Match Set verschoben. Falls kein Classifier zu dem eingehenden Ereignis passt, d.h. falls keine Bedingung der Classifier dazu passt, erstellt Covering einen neuen Classifier, dessen Bedingung zum eingetroffenen Ereignis passt. Die Aktion des neuen Classifiers wird zufällig gewählt und seine Stärke anhand der durchschnittlichen Stärke aller vorhandenen Classifier ermittelt. Demnach ist es also nicht möglich, ein leeres Match Set zu erhalten und der Algorithmus kann normal weiter abgearbeitet werden.

Nun findet die Auswahl einer Aktion nach der Roulette-Wheel- Methode statt. Hierbei wird der Stärkewert der Classifier herangezogen, d.h. es werden alle Stärken der Classifier im Match Set zusammengerechnet, die die gleiche Aktion beinhalten. Nun wird diese Zahl durch die Summe der Stärken aller im Match Set enthaltenen Classifier dividiert. Dies ergibt dann die Auswahlwahrscheinlichkeit der Aktion. Z.B. seien c_1 bis c_5 passende Classifier im Match Set mit den

Stärken 1 bis 5. In c_1 bis c_3 steht Aktion A und in c_4 und c_5 Aktion B. Die Wahrscheinlichkeit, dass Aktion A ausgewählt wird ist also (1+2+3)/(1+2+3+4+5) = 6/15 = 40% und für B beträgt sie (4+5)/(1+2+3+4+5) = 9/15 = 60%. Die ausgewählten Classifier mit der gleichen Aktion werden nun aus dem Match Set in das Action Set verschoben.
Jetzt wird die gewählte Aktion an die Attraktoren weitergegeben, die die Umgebung mit dieser Aktion beeinflussen.
Außerdem gibt ein Belohnungssystem für die Classifier, so dass starke und gute Classifier noch stärker werden und die schwachen und schlechten verworfen werden. Zum Schluss werden die bewerteten Classifier wieder zurück in die Menge aller Classifier verschoben. Dadurch findet ein Optimierungsprozess statt, der in der jeweils besten Aktion für eine bestimmte Bedingung resultiert. (vgl. [15, S.2-8])

IV. Vergleich von AC und OC

AC und OC sind grundsätzlich gleich, da bei beiden Ansätzen versucht wird, aus der Natur abgeschaute Self-x-properties auf Computersysteme zu übertragen.

Allerdings wird beim Autonomic Computing eine größere Betonung auf die Unabhängigkeit des Systems gelegt. Ein Level 5 System soll alleine durch vorher definierte globale Richtlinien und Regeln gesteuert werden. Das System soll automatisch agieren, um optimal sein Ziel zu erreichen. Es soll sich selbst vor Angriffen schützen, Fehler entdecken und reparieren. Dadurch werden Systemausfälle minimiert und den Systemadministratoren viel Arbeit abgenommen. Vor allem soll der Umgang mit einem AC System für den Benutzer so leicht wie möglich gemacht werden.

Organic Computing ist hingegen etwas allgemeiner gefasst. Hier liegt die Betonung nicht unbedingt auf der Benutzerfreundlichkeit. Die globalen Richtlinien werden teilweise ebenfalls vorab eingestellt, aber in manchen Architekturen ist es auch möglich diese Richtlinien dynamisch zur Laufzeit, mit Hilfe eines Interface, einzustellen, bzw. zu verändern. Dies ist beispielsweise in der Observer/Controller-Architektur explizit vorgesehen.

Durch die Verwendung des Observer/Controller- Musters sind alle hier vorgestellten Architekturen sehr ähnlich zueinander und organisieren sich selbst durch den Kreislauf von Beobachtung, Analyse und Kontrolle. Dabei optimieren sich („lernen“) die Systeme auf verschiedene Arten. Bei MAPE und der Observer/Controller- Architektur kann das Kontrollmodul auf eine Historie zugreifen, um zu entscheiden, ob die ausgeführten Aktionen zum Ziel führten und dadurch die nächsten Aktionen bestimmen. Im ZCS wird die Optimierung des Systems dagegen durch das Belohnungssystem für die Classifier erreicht.

Um zu messen, wie gut sich Computersysteme selbst organisieren, müssen dafür Bewertungskriterien gefunden werden.

V. Metriken für Selbstorganisation

Viele Ansätze zur Messung von Selbstorganisation basieren auf der Gleichung der informationstheoretischen Entropie von Claude Elwood Shannon. Entropie wird folgendermaßen definiert:

Gegeben sei ein Alphabet mit N Zeichen. Die Wahrscheinlichkeit, mit der ein Zeichen auftritt sei p_i ($1 \leq i \leq N$). Dann lautet die Entropie nach Shannon

$$H = -\sum_{i=1}^{N} p_i \cdot \mathrm{ld}(p_i) \qquad (1)$$

Die Entropie nach Shannon ist sozusagen der durchschnittliche Informationsgehalt eines Zeichens oder anders gesagt, es ist die Anzahl an Bits (durch den Logarithmus Digitalis bedingt), die benötigt werden, um ein Zeichen des Alphabetes zu beschreiben. Am größten wird die Entropie, wenn alle Zeichen mit der gleichen Wahrscheinlichkeit auftreten. (vgl. [17])

Der Begriff „Entropie“ stammt aus der statistischen Thermodynamik und definiert dort so etwas wie die „Unordnung“ eines Systems.

1) Zustandsentropie

H. Parunak und S. Brueckner stellen in [18] eine Bewertungsmethode vor, die auf der Gleichung der Shannonschen Entropie basiert. Sie definieren ihre Art der Entropie als

$$S = -\sum_{i=1}^{N} p_i \cdot \log(p_i) \qquad (2)$$

wobei i die verschiedenen Zustände (Anzahl der Zustände = N) durchläuft, in die das System im nächsten Schritt kommen kann und p_i die Wahrscheinlichkeit ist, das System im nächsten Schritt im Zustand i zu finden. S wird am größten, wenn alle Zustände, die das System hat, mit der gleichen Wahrscheinlichkeit erreicht werden können, das heißt, wenn keine Bewertung der Zustände stattgefunden hat. Wenn nämlich jeder Systemzustand mit der gleichen Wahrscheinlichkeit eintritt, wird nichts selbstorganisiert. Wenn jedoch ein Zustand wahrscheinlicher auftritt als ein anderer, so muss eine gewisse Logik stattgefunden haben, um eine Gewichtung der Zustände vorzunehmen. Somit muss es also eine gewisse Selbstorganisation gegeben haben. Eine Abnahme der so abgewandelten Entropie S bedeutet also eine Zunahme an Selbstorganisation und andersherum.
Die Autoren normalisieren jetzt noch die Gleichung mit dem größtmöglichen erreichbaren Logarithmus, also mit log(N). Wir erhalten nun die Gleichung

$$S' = -\sum_{i=1}^{N} p_i \cdot \frac{\log(p_i)}{\log(N)} \qquad (3)$$

Durch diese Normalisierung erreicht man eine prozentuale Bewertung der Unordnung eines Systems. Nun ist es auch irrelevant, welcher Logarithmus verwendet wird. (vgl. [18,

S.2-4])

2) Statistische Komplexitätsdichte

Eine weitere Möglichkeit, Selbstorganisation zu bewerten, wird in [19] und [20] vorgeschlagen. Hier wird Selbstorganisation als ein spontaner Anstieg der Komplexität definiert und daher wird eine Möglichkeit der Errechnung der so genannten „statischen Komplexitätsdichte“ C_μ beschrieben. Diese Idee basiert ebenfalls auf der Gleichung der Shannonschen Entropie.

Es wird zunächst ein beliebiges Element, das aus seiner aktuellen Konfiguration besteht, betrachtet. Nun werden alle vergangenen Konfigurationen, die die aktuelle Konfiguration des Elementes beeinflussen konnten, betrachtet. Man kann sich dieses wie einen Lichtkegel vorstellen, der in die Vergangenheit scheint. Je weiter man zurückgeht, desto mehr Konfigurationen haben die aktuelle beeinflusst. Durch die Informationen aus der Vergangenheit können die zukünftigen Konfigurationen vorhergesagt werden. Man kann sich die möglichen zukünftigen Konfigurationen auch als einen Lichtkegel in die Zukunft vorstellen, da man bei der Vorhersage immer unsicherer wird, welche Konfiguration des Elements auftreten wird.

Jetzt wird eine lokale Statistik des Elements definiert (wie eine Historie), die eine Zusammenfassung aller vergangenen Konfigurationen, die dieses Element beeinflussen konnten, beinhalten. Wenn man jetzt alle einzelnen vergangenen Konfigurationen betrachtet, gibt es diverse, die auf denselben zukünftigen Zustand, mit derselben Auftrittswahrscheinlichkeit, hinweisen. Daher fasst man diese äquivalenten Konfigurationen zusammen, um eine sogenannte „minimal ausreichende Statistik“ zu erhalten.

Die Größe dieser Statistik beschreibt nun die Anzahl der Informationen, die benötigt werden, um das System vorherzusagen. Anhand derer wird nun mit Hilfe der Gleichung der Shannonschen Entropie die Bewertung vorgenommen, indem i in Gleichung (1) nun alle Konfigurationen durchgeht, die in der minimal ausreichenden Statistik stehen. p_i ist hier die Wahrscheinlichkeit dafür, dass diese Konfiguration als nächstes auftritt. Dies wird statistische Komplexitätsdichte C_μ genannt. Im Gegensatz zur ersten Metrik wird hier der Wert nicht normiert, sondern ergibt eine absolute Zahl.

Ein System hat sich zwischen t_1 und t_2 selbst organisiert, wenn die statistische Komplexitätsdichte gestiegen ist, d.h. wenn $C_\mu(t_1) < C_\mu(t_2)$ gilt. Außerdem darf diese Steigerung nicht nur aus äußerem Eingreifen resultieren. (vgl. [19, S.4-6])

3) Externe vs. Interne Kontrolle

In [21] stellen Çakar et al. eine Messung der Selbstorganisation vor, die die Menge an externer Kontrolle, die benötigt wird, um die eigene Struktur und/oder Verhalten zu ändern, misst. Hierbei gehen die Autoren von der Observer/Controller- Architektur aus.

Um ein System zu verändern, muss man seine Struktur, bzw. sein Verhalten verändern. Die aktuellen Werte von Struktur und Verhalten werden in der Konfiguration zusammengefasst. Die Menge aller möglichen Konfigurationen, die vom Observer unterschieden werden können, wird Konfigurationsraum genannt. Die Größe des Konfigurationsraums wird mit der Anzahl der benötigten Bits für seine Abbildung beschrieben. Diese Anzahl an Bits wird als Variabilität V definiert (das ist sozusagen der duale Logarithmus der Größe des Konfigurationsraums). Die Struktur und das Verhalten eines System under Observation and Control (SuOC) werden durch Parameter von außen bestimmt und dem Controller muss dafür ein Interface zur Verfügung gestellt werden. Die Breite dieses Kontrollinterface beträgt c Bits. Es ist klar, dass immer $c \leq V$ gelten muss, da man nicht mehr von außen ändern kann, als innerhalb veränderbar ist, sondern eher weniger Einstellmöglichkeiten hat als möglich sind.

Nun werden bei der Observer/Controller- Architektur zwei verschiedene Konfigurationsräume und Kontrollinterface benötigt und zwar jeweils einen Konfigurationsraum und ein Interface für das SuOC (Interface für Zugriffsmöglichkeiten des Controllers), sowie für den Controller (Interface für äußere Zugriffsmöglichkeiten, z.B. Einstellung des Systemziels). Daher wird eine interne Variabilität V_i und interne Kontrollbandbreite c_i definiert für das SuOC und dementsprechend eine externe Variabilität V_e und Kontrollbandbreite c_e für den (internen) Controller.

Das heißt, wenn $V_e < V_i$, dann kann man von Selbstorganisation sprechen. Daher wird die Differenz der beiden Variabilitäten als Komplexitätsreduktion R eines selbstorganisierenden Systems definiert:

$$R = V_i - V_e \tag{4}$$

Praktisch heißt das, dass die Differenz der beiden Variabilitäten die Anzahl aller möglichen, aus Selbstorganisation resultierenden, Konfigurationen repräsentiert. Des Weiteren wird der statische Grad der Selbstorganisation definiert als Quotient aus der Komplexitätsreduktion und der internen Variabilität:

$$S = \frac{R}{V_i} = \frac{V_i - V_e}{V_i} \quad (0 \leq S \leq 1) \tag{5}$$

Der statische Grad der Selbstorganisation ist das Verhältnis zwischen allen möglichen selbstorganisierten Konfigurationen und allen möglichen Konfigurationen des SuOC überhaupt. Dieses Verhältnis bewertet sozusagen prozentual den Anteil der möglichen selbstorganisierten Zustände an der Summe aller möglichen Zustände.

Da sich ein selbstorganisierendes System aber im laufe der Zeit verändert, verändern sich auch die Anzahlen der aktiven internen und externen Kontrollbits. Die Anzahl der zur Zeit t aktiven internen Kontrollbits wird als $v_i(t)$ definiert. Entsprechend wird $v_e(t)$ für die Anzahl der zur Zeit t aktiven externen Kontrollbits definiert. Es muss immer gelten $v_i(t) \leq c_i$ ($v_e(t) \leq c_e$), da nur maximal so viele Kontrollbits aktiv sein

können, wie auch im Interface vorhanden sind. Zur Zeit t ist die Komplexitätsreduktion $v_i(t) - v_e(t)$.
Um die Kontrolle in einem System über die Zeit besser abschätzen zu können, wird vorgeschlagen die internen und externen Kontrollflüsse zu vergleichen, d.h. wie das Verhältnis zwischen interner und externer Kontrolle zwischen zwei Zeitpunkten t_1 und t_2 ist. Daher wird eine dynamische Komplexitätsreduktion r definiert als:

$$r = \int_{t_1}^{t_2} (v_i(t) - v_e(t))dt \quad (6)$$

Dadurch werden sozusagen zwischen den Zeitpunkten t_1 und t_2 alle aktiven internen Kontrollbits, d.h. alle Selbstkontrolle, aufsummiert und davon dann die Summe aller externen aktiven Kontrollbits, d.h. alle externe Kontrolle, die in der Zeit auftritt, abgezogen. Daraus folgt, durch dividieren aller internen aktiven Kontrollbits, jetzt der dynamische Grad der Selbstorganisation:

$$s = \frac{r}{\int_{t_1}^{t_2} v_i(t)dt} = \frac{\int_{t_1}^{t_2} (v_i(t) - v_e(t))dt}{\int_{t_1}^{t_2} v_i(t)dt} \quad (7)$$

s ist sozusagen der prozentuale Anteil der Selbstorganisation in der Zeit zwischen t_1 und t_2. (vgl. [21, S.3-5])

VI. Bewertung der Metriken

Alle hier vorgestellten Metriken wurden für spezielle Arten von selbstorganisierenden Systemen entwickelt. Daher ist es teilweise schwierig bis unmöglich diese Metriken auf alle selbstorganisierenden Systeme anzuwenden, bzw. anzupassen. Beispielsweise wurde die dritte Metrik, in der die interne und externe Kontrolle verglichen wird, explizit für die Observer/Controller- Architektur entworfen. Die beiden anderen Metriken (Messung der Zustandsentropie und der statistischen Komplexitätsdichte) sind hingegen allgemeiner gehalten und beziehen sich beide auf die verschiedenen möglichen Systemzustände / Systemkonfigurationen.

Bei allen Metriken kann man den Grad der Selbstorganisation nur zur Laufzeit bestimmen. Lediglich durch den statischen Grad der Selbstorganisation (siehe Gleichung (5)) kann über die Selbstorganisation eines Systems eine Aussage getroffen werden, bevor man dieses in Betrieb nimmt. Aber diese Aussage ist nicht unbedingt genau, da ja erst zur Laufzeit klar wird, welche Kontrollbits benutzt werden. Dadurch kann man damit nur die jeweils höchst mögliche Anzahl an internen und externen Kontrollbits miteinander vergleichen. Also ist der statische Grad der Selbstorganisation für eine genaue Quantifizierung von Selbstorganisation nicht zu gebrauchen.

Bei der ersten Metrik (Zustandsentropie) werden die Wahrscheinlichkeiten, mit der Systemzustände auftreten, verwendet, um Selbstorganisation zu messen. Hierbei erhält man einen relativen Wert, der als Vergleichsmaß für die Betrachtung verschiedener Systeme genommen werden kann. Dies ist praktisch, da man dadurch verschiedene Systeme miteinander vergleichen kann. Allerdings ist es bei dieser Metrik so, dass zu Anfang keine Selbstorganisation gemessen werden kann, da dort die Wahrscheinlichkeiten der Systemzustände alle gleich verteilt sind. Diese Metrik greift also erst dann richtig, wenn eine gewisse Zeit vergangen ist. Es kann sozusagen nicht von Anfang an ein repräsentabler Grad der Selbstorganisation bestimmt werden, sondern erst nach einer gewissen Eingewöhnungszeit. Ein weiteres Problem besteht bei dieser Metrik darin, dass es auch passieren kann, dass im System an irgendeinem Punkt alle zukünftigen Systemzustände wieder gleich wahrscheinlich sind. Das würde nach dieser Metrik heißen, dass die Selbstorganisation abgenommen hat. Das muss aber nicht der Fall sein, es könnte lediglich ein anderes Systemziel angegeben worden sein, so dass das Kontrollmodul keine andere Möglichkeit hat als die vorher nicht so häufig benutzten Zustände auszuwählen. Damit würde aber die Unordnung steigen und die Metrik hätte versagt.

Bei der zweiten Metrik (Statistische Komplexitätsdichte) wird die Größe der kleinsten Historie, die nötig ist, um den nächsten Zustand vorherzusagen, herangezogen, um Selbstorganisation zu messen. Hier erhält man ebenfalls ein Vergleichsmaß, das aber einem absoluten Wert entspricht. Je größer der Wert ist, desto mehr wurde das System selbst organisiert. Hier ist es aber wie bei der ersten Metrik. Es liegt in der Natur dieser Metrik, dass dieser Wert zur Laufzeit des Systems immer höher steigt, da zu Anfang keine Historie vorhanden ist. Diese baut sich erst nach und nach auf und man kann ebenfalls wie bei der ersten Metrik erst nach einer Eingewöhnungszeit den Grad der Selbstorganisation messen. Bei dieser Metrik gibt es das Problem, dass fälschlicherweise keine Selbstorganisation gemessen wird, in einer ähnlichen Form. Wenn hier das Systemziel neu festgelegt wird, kann es passieren, dass aus allen früheren Konfigurationen nur eine mögliche Vorhersage getroffen werden kann. Dadurch würde die minimal ausreichende Historie erheblich schrumpfen und der Wert der Selbstorganisation würde sehr klein werden, obwohl das System sich genauso viel selbst organisiert wie vorher.

Bei der dritten Metrik (Externe vs. Interne Kontrolle) wird der externe Kontrollinformationsfluss mit dem internen verglichen, um Selbstorganisation zu messen. Ein Problem hier ist der spezielle Ansatz dieser Metrik, da hiermit nur Systeme, die nach der Observer/Controller- Architektur entworfen sind, also unter anderem ein Eingabeinterface besitzen, verglichen werden können. Beim Autonomic Computing würde diese Metrik z.B. immer einen Selbstorganisationsgrad von 100% ergeben, da vorher das Systemziel eingegeben wird und das System dann komplett von selbst agiert. Hier könnte man natürlich die vorher eingegebenen Systemziele als Input ansehen. Diese würden aber nicht zur Laufzeit verändert werden, was aber gerade die Stärke dieser Metrik (Messung des Verhältnisses der sich über die Zeit verändernden aktiven Kontrollbits) zunichte machen

würde. Für Systeme, die auf der Observer/Controller-Architektur basieren, besteht aber ein ähnliches Problem, da wenn ein Systemziel einmalig eingegeben wird, die Selbstorganisation nach und nach auf nahezu 100% kommen würde. Daher ist diese Metrik, im Gegensatz zu den anderen Metriken, eher für Systeme geeignet, in denen sich das Systemziel des Öfteren ändert, oder man benutzt sie, um zu messen, wie schnell der Anteil der Selbstorganisation in einem System ansteigt, dass heißt, wie schnell der Controller das eingegebene Systemziel umsetzt, d.h. wie viele Bits er dafür aktiviert. Bei dieser Metrik, wie bei allen anderen auch, steigt die Selbstorganisation mit der Zeit an, wenn die Systemziele nicht verändert werden.

Eine Schwäche aller drei Metriken ist, dass lediglich die Selbstorganisation an sich bewertet wird, aber nicht die Qualität dieser. Es könnte genauso gut passieren, dass das, was das System selbständig tut, in eine unerwünschte Richtung geht und vom eigentlichen Systemziel abweicht. Dies kann man aber anhand dieser Metriken nicht erkennen.

VII. Fazit – Hype oder Möglichkeit?

Das Problem der schlechten quantitativen und qualitativen Bewertbarkeit von Selbstorganisation bleibt leider bestehen. Was man als Mensch einfach sehen kann, lässt sich hier, wie in vielen Fällen, nicht so einfach auf ein Computersystem übertragen.

Obwohl es schon diverse Ansätze zur Umsetzung von selbstorganisierenden Systemen gibt, fehlen heute noch immer produktive Systeme, die zeigen, dass die Ideen auch wirklich umsetzbar sind. Vereinzelt gibt es zwar Teilumsetzungen, wie das Smart Doorplate- Projekt oder bestimmte Server von IBM, aber das Konzept wurde bisher nicht als Ganzes realisiert.

Von daher kann man die Idee der Selbstorganisation momentan schon als Hype bezeichnen, wenn man bedenkt, dass seit 2001, also innerhalb von 7 Jahren sehr viel in Bewegung gesetzt wurde. Es wurde eigens für OC ein neuer Forschungsschwerpunkt eingerichtet. In der Organic Computing Initiative (OCI) sind weltweit mehr als 20 verschiedene Universitäten vertreten und außerdem auch Vertreter von z.B. Siemens oder der NASA. (vgl. [22])

Trotzdem steckt großes Potenzial in selbstorganisierenden Systemen. Die Notwendigkeit eines Umdenkens besteht allemal, nicht zuletzt durch die starke Zunahme von komplexen eingebetteten und verteilten Systemen und den steigenden Ansprüchen der Konsumgesellschaft. Aber ob dadurch auch der Traum von einer sogenannten künstlichen Intelligenz erfüllt werden kann, also ob es wirklich lernende Systeme geben wird, muss sich noch zeigen. Heute sind wir jedenfalls noch weit davon entfernt.

Literatur

[1] C. R. Shalizi, „W. Ross Ashby", 1999
URL: http://cscs.umich.edu/~crshalizi/notebooks/ashby.html
Stand: 12.06.2008

[2] M. Naylor, „Glossary: Self-organisation", 2007
URL: http://www.geos.ed.ac.uk/homes/mnaylor/Glossary.html
Stand: 12.06.2008

[3] F. Heylighen: „Self-organisation", in: F. Heylighen, C. Joslyn and V. Turchin (editors): *Principia Cybernetica Web* (Principia Cybernetica, Brussels), 1997
URL: http://pespmc1.vub.ac.be/SELFORG.html
Stand: 12.06.2008

[4] C. Lucas, „Self-Organizing Systems (SOS) FAQ"
URL: http://www.calresco.org/sos/sosfaq.htm
Stand: 12.06.2008

[5] Gilbert J.B Probst, „Selbstorganisation - Ordnungsprozesse in sozialen Systemen aus ganzheitlicher Sicht", Verlag Paul Parey, Berlin, Hamburg, 1987 (4 Eigenschaften von Selbstorganisation)

[6] S. Stein, „Emergenz in der Softwareentwicklung – Bereits verwirklicht oder Chance?", Diplomarbeit, Hochschule für Technik und Wirtschaft Dresden (FH), Fachbereich Informatik / Mathematik, 2004

[7] IBM, „Autonomic Computing – Overview"
URL: http://www.research.ibm.com/autonomic/overview/
Stand: 12.06.2008

[8] IBM, „Autonomic Computing – Overview – The Solution"
URL: http://www.research.ibm.com/autonomic/overview/solution.html
Stand: 12.06.2008

[9] IBM, „Autonomic Computing – Overview – The 8 Elements"
URL: http://www.research.ibm.com/autonomic/overview/elements.html
Stand: 12.06.2008

[10] J. O. Kephart, D. M. Chess, „The Vision of Autonomic Computing", Computer magazine, 36(1), S.41-50, Januar 2003

[11] IBM, „About IBM Autonomic Computing – Get started – Adoption Model"
URL: http://www-03.ibm.com/autonomic/about_get_model.html
Stand: 12.06.2008

[12] S. Mostaghim, H. Schmeck, „Organic Computing – Chapter 3: Architecture of OC", Vorlesungsfolien der Vorlesung "Organic Computing" im SoSe 2007, Universität Karlsruhe, April 2007

[13] C. Müller-Schloer, H. Schmeck und T. Ungerer, „Antrag auf Einrichtung eines neuen DFG-Schwerpunktprogramms", Februar 2004

[14] Universität Augsburg, „Smart Doorplate – Zusammenfassung"
URL: http://www.informatik.uni-augsburg.de/lehrstuehle/sik/forschung/ubicomp/smartdoorplate/
Stand: 12.06.2008

[15] S. W. Wilson, „ZCS A Zeroth Level Classifier System", Evolutionary Computation, *2*(1), S.1-18, 1994

[16] W. Trumler, Vorlesungsfolien der Vorlesung „Organic Computing" im SoSe 2007, Universität Augsburg, April 2007

[17] H. Föll, „Entropie und Information",
URL: http://www.tf.uni-kiel.de/matwis/amat/mw1_ge/kap_5/advanced/t5_3_2.html
Stand: 12.06.2008

[18] H. Parunak, S. Brueckner, „Entropy and Self-Organization in Multi-Agent Systems", Proceedings of the International Conference on Autonomous Agents, S. 124-130, 2001

[19] C. Shalizi, K. Shalizi, „Quantifying Self-Organization in Cyclic Cellular Automata", 2005

[20] C. Shalizi, K. Shalizi, R. Haslinger, „Quantifying Self-Organization with Optimal Predictors", Physical Review Letters, vol. 93, no. 11, S. 1-4, September 2004

[21] E. Çakar et al., „Towards a Quantitative Notion of Self-organisation", Proceedings of the 2007 IEEE Congress on Evolutionary Computation, S.4222-4229, 2007

[22] Leibniz Universität Hannover, „Organic Computing Initiative – People"
URL: http://www.organic-computing.de/
Stand: 12.06.2008