Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung AW1

Maik Weindorf

Autonomic Computing (AC)

Fachbereich Elektrotechnik und Informatik
Department of Electrical Engineering and Computer Science

Maik Weindorf

Thema der Ausarbeitung

Autonomic Computing

Stichworte

Autonomic Computing (AC), self-x, CHOP, Touchpoint, Autonomic Manager, Microreboots, Crash-Only Software

Kurzzusammenfassung

Die zunehmende Komplexität von Computersystemen und die ebenfalls zunehmende wirtschaftliche Abhängig von diesen Systemen, machen rein manuelle Kontrolle, Steuerung, Fehlerbehebung und Konfiguration praktisch unmöglich.
Hinter dem von IBM geprägten Begriff ”Autonomic Computing” verbirgt sich eine Initiative, die genau dieses Problem aufgreift.
In dieser Ausarbeitung wird beschrieben, welche Anforderungen an ein Autonomic Computing System bestehen und wie der ”Reifegrad” eines solchen Systems beurteilt werden kann. Darüber hinaus werden verschiedene Konzepte vorgestellt und kritisch betrachtet, die alle das Ziel verfolgen, Aspekte von Autonomic Computing in Computersystemen zu verwirklichen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Die zunehmende Komplexität von Computersystemen in Unternehmen - sowohl auf Hardware–, als auch auf Softwareseite - macht rein manuelle Kontrolle, Steuerung, Fehlerbehebung und Konfiguration praktisch unmöglich. Hinter dem von IBM geprägten Begriff "Autonomic Computing" verbirgt sich eine Initiative, die genau dieses Problem aufgreift.

Autonomic Computing stellt dabei eine Analogie zum Zentralen Nervensystem (engl. "autonomic central nervous system") des Menschen dar. Das Zentrale Nervensystem hat die Aufgabe, lebenswichtige Körperfunktionen zu überwachen, zu steuern und diese an sich ändernde Bedingungen anzupassen. Z.B. steigt die Herzfrequenz, wenn man sich schneller bewegt oder die Iris zieht sich bei starkem Lichteinfall zusammen. Diese Körperfunktionen müssen nicht aktiv durch den Verstand kontrolliert und gesteuert werden. Die Frage lautet nun: "Können Computersysteme ebenfalls ohne aktives Eingreifen (durch Menschen) kontrolliert und gesteuert werden?".

In dieser Ausarbeitung werden verschiedene Konzepte vorgestellt und kritisch betrachtet, die alle das Ziel verfolgen, Aspekte von Autonomic Computing in Computersystemen zu verwirklichen. Dabei wird nicht in allen Konzepten "Autonomic Computing" als Begriff verwendet. Die verfolgten Ziele sind jedoch immer denen des Autonomic Computing ähnlich.

"Autonomic Computing" wurde als Titel für diese Ausarbeitung gewählt, da die gleichnamige Initiative von IBM für die Praxis die größte Relevanz besitzt und die mit dem Begriff verbundene Analogie, bzw. Methapher auch für alle anderen Konzepte passend ist.

Kapitel 2

Grundlagen

Dieses Kapitel beschäftigt sich damit, welche Anforderungen an ein Autonomic Computing System bestehen und wie der "Reifegrad" eines solchen Systems beurteilt werden kann. Weiterhin wird die Relevanz des Themas dargelegt.

2.1 Historie

- 2001: IBM startet Autonomic-Computing-Initiative und veröffentlicht das Autonomic Computing Manifesto
- 2003: Autonomic Computing Workshop auf dem 5th Annual International Workshop on Active Middleware Services (AMS 2003)
- 2004: 1st IEEE International Conference on Autonomic Computing (ICAC 2004)

Diese Entwicklung zeigt, dass Autonomic Computing zunehmend an Relevanz gewinnt. Nicht zuletzt bedingt durch die nun jährlich stattfindende IEEE Konferenz gibt es inzwischen zahlreiche akademische Arbeiten zum Thema "Autonomic Computing". Weiterhin sind fast alle großen IT Unternehmen (z.B. Microsoft, HP und Sun) der Autonomic-Computing-Initiative beigetreten. Somit sind die Voraussetzungen für die Entwicklung von Hersteller übergreifenden Standards geschaffen.

2.2 Anforderungen an AC Systeme

Ein System muss verschiedene Eigenschaften aufweisen, um als "autonom" betrachtet werden zu können. Laut IBM's "Autonomic Computing Manifesto" [IBM, 2001] sind dies folgende 8 Punkte:

1. Ein Autonomic Computing System muss sich selbst ”kennen”. D.h. es muss ein detailliertes Wissen über seine einzelnen Komponenten, seinen aktuellen Zustand und seine Verbindungen zu anderen Systemen haben.
2. Ein Autonomic Computing System muss sich selbst konfigurieren und sich ändernden (evtl. unvorhersehbaren) Bedingungen anpassen können.
3. Ein Autonomic Computing System erreicht nie einen ”Status Quo”. Es sucht ständig nach Möglichkeiten, sich selbst zu optimieren.
4. Ein Autonomic Computing System muss sich selbst ”heilen”. Es muss Probleme, bzw. potentielle Probleme erkennen können und Wege finden, diese zu umgehen oder zu beheben.
5. Ein Autonomic Computing System muss sich selbst ”schützen”. Hinter diesem Punkt stecken Überlegungen zu Security. Das System muss in der Lage sein, verschiedenste Typen von Angriffen zu erkennen und sich gegen diese zu schützen.
6. Ein Autonomic Computing System muss seine Umgebung kennen. D.h. es muss sich in eine Systemlandschaft integrieren und in der Lage sein, mit anderen Systemen zu interagieren.
7. Ein Autonomic Computing System kann keine proprietäre Lösung sein. Es muss in der Lage sein, in einer heterogene Systemlandschaft zu agieren. Dies ist nur sinnvoll durch die Verwendung von offenen Standards zu realisieren.
8. Ein Autonomic Computing System muss den Benutzern gegenüber seine Komplexität verbergen. Die Benutzer sollen sich nur darum kümmern müssen, ”was” für Ziele erreicht werden sollen, das Autonomic Computing System soll sich darum kümmern, ”wie” diese Ziele erreicht werden können.

Aus diesen Anforderungen ergeben sich 4 Kerneigenschaften für Autonomic Computing, die sog. self-x oder CHOP Eigenschaften: self-optimizing, self-healing, self-protecting, und self-configuring.

2.3 Levels of Autonomic Computing Maturity

Aus den Anforderungen unter 2.2 wird schnell klar, dass die Entwicklung eines Autonomic Computing Systems, sowie dessen Eingliederung in eine bestehende (historisch gewachsene) Systemlandschaft eines Unternehmens nicht triviale Probleme sind. Zudem ist ”Autonomic Computing” derzeit ein aktives Forschungsgebiet, in dem noch viele Probleme und Fragen ungelöst sind. Bis hin zu völlig autonomen Systemen im Sinne von 2.2 ist es noch ein weiter Weg (siehe [Ganek und Corbi, 2003] und [Sterritt und Hinchey, 2005]). Jedoch werden zunehmend autonome Eigenschaften in Produkten implementiert (siehe 3.4) und

viele Unternehmen erkennen zunehmend das Potential von Autonomic Computing und beginnen Produkte mit autonomen Eigenschaften einzusetzen. Folgende 5 Punkte bieten eine Möglichkeit zur Einteilung und Bewertung des "Reifegrads" von Autonomic Computing:

- Basic (Level1): manuelle Analyse und Problembehebung
- Managed (Level2): zentralisierte Tools, manuelle Maßnahmen
- Predictive (Level3): Cross-Resource Korrelation und Führung
- Adaptive (Level4): System Monitors erkennen Korrelationen und führen Maßnahmen aus
- Autonomic dynamisches Management auf Basis von Geschäftsregeln

[Ganek und Corbi, 2003] [1]

[1] siehe auch Abbildung 2.1

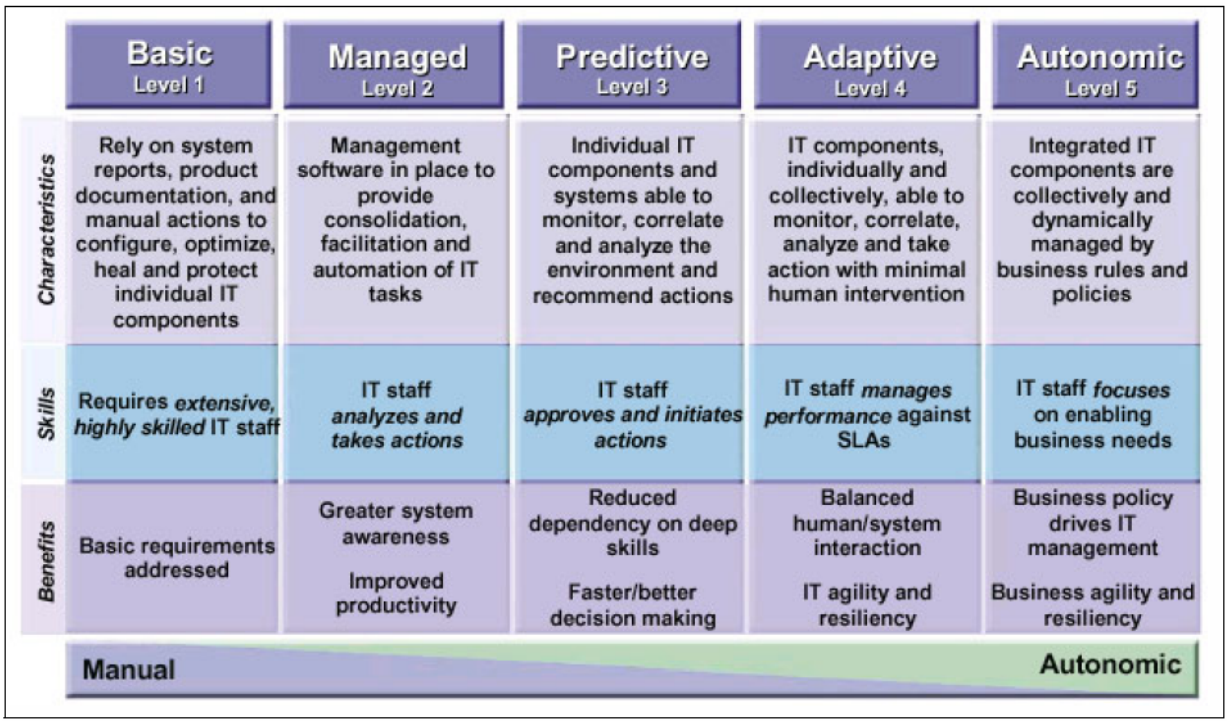

Abbildung 2.1: Levels of Autonomic Computing Maturity [Manoel et al, 2005]

Kapitel 3

Aktuelle Entwicklungen

In diesem Kapitel werden verschiedene Konzepte vorgestellt, die alle das Ziel verfolgen, Aspekte von Autonomic Computing in Computersystemen zu verwirklichen. Der Schwerpunkt liegt dabei auf dem Konzept von IBM, da dieses die größte Relevanz besitzt.

3.1 IBM: Autonomic Computing

Im Folgenden wird ein Architektur Konzept von IBM vorgestellt, welches aufzeigt, wie ein Autonomic Computing System aufgebaut werden könnte.

Abbildung 3.1 zeigt die Gesamtarchitekur. Nachfolgend werden die einzelnen Komponenten der Architektur vorgestellt und beschrieben.

Managed Resources Eine Managed Resource ist ein Hard- oder Softwarekomponente, die "gemanaged" werden kann. Infrage kommt hier praktisch jede "konventionelle" Systemkomponente (z.B. Server, Datenbank, Application Server, Anwendungen, ect.). Die Managed Resources können dabei durchaus über Fähigkeiten zum Selbsmanagement ("self managing") verfügen.

Touchpoint Ein Touchpoint kann als Wrapper für eine Managed Resource verstanden werden. Er stellt eine definierte Schnittstelle (Sensor/Effector) zur Verfügung, über die die Managed Resource kontrolliert und beeinflusst werden kann. [1]

[1] siehe auch Abbildung 3.2

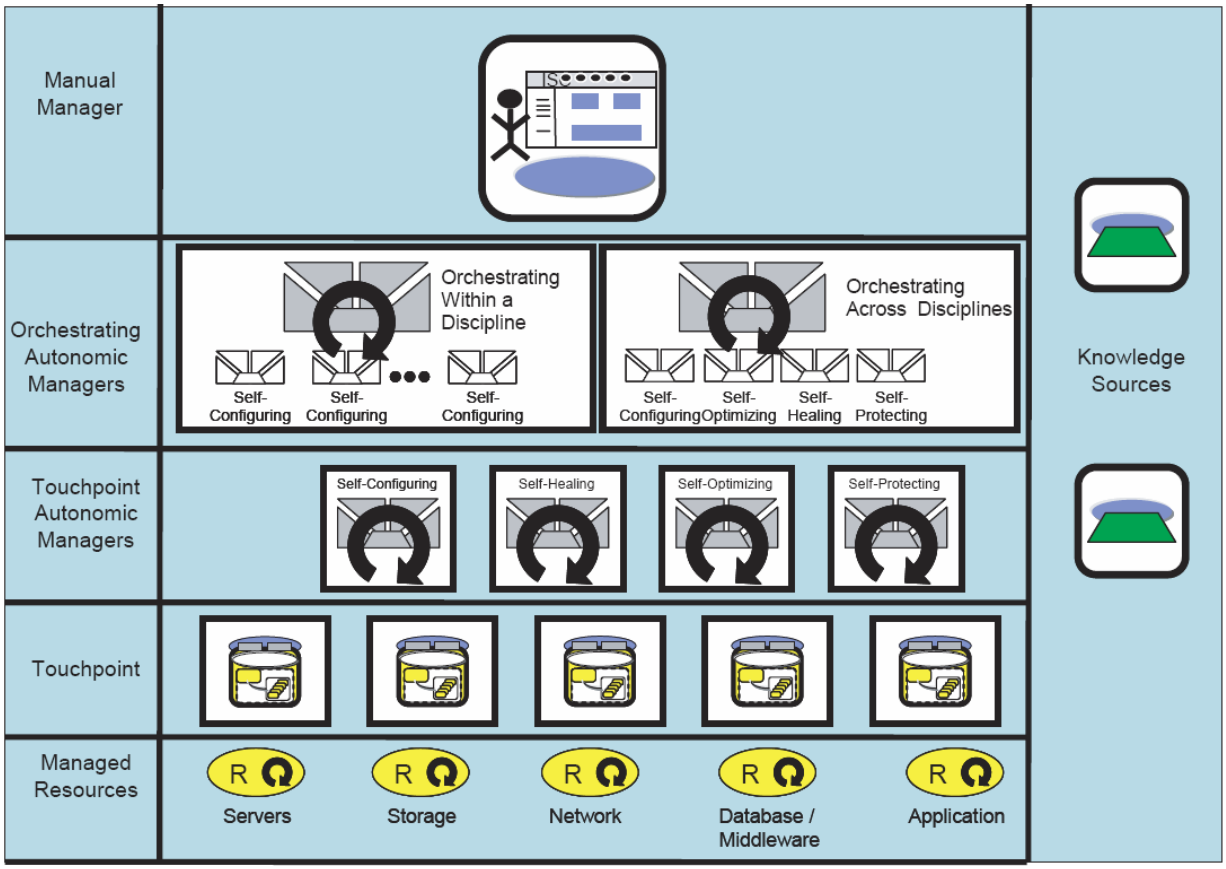

Abbildung 3.1: Autonomic Computing Architektur [IBM, 2005]

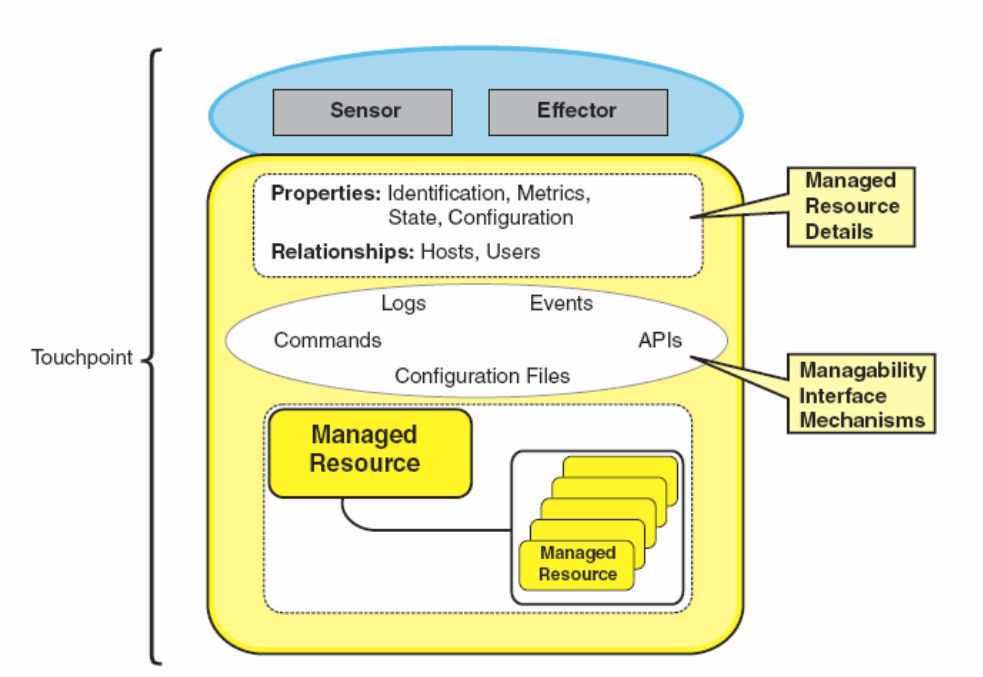

Abbildung 3.2: Touchpoint [IBM, 2005]

Autonomic Manager Autonomic Managers (AM) sind die wichtigsten Komponenten dieser Architektur. In ihnen steckt die eigentliche "Intelligenz" des Systems. Sie bieten folgende Funktionen:

- Monitor: Sammelt Informationen über Ressourcen, die vom AM verwaltet werden. Die gesammelten Informationen werden aggregiert und gefiltert. Die so aufbereiteten Informationen (Symptome) werden an die "Analyze" Funktion übergeben.
- Analyze: Analysiert die Symptome und entscheidet, ob Änderungen notwendig sind. (basierend auf Wissen aus Knowledge Source)
- Plan: Erstellt einen Plan, wie die geforderten Änderungen umgesetzt werden können. Dies können einfache Befehle, bis hin zu komplexen Workflows sein.
- Execute: Führt die im Plan festgelegten Änderungen aus. Dies erfolgt über die Effector Schnittstelle der verwalteten Ressourcen.

Zusammen stellen diese Funktionen einen "Control Loop" dar. Es gibt einen ständigen Zyklus von messen des "Ist" Zustandes, Änderungsmaßnahmen zur Erreichung eines "Soll" Zustandes und wiederum über die Messung des "Ist" Wertes eine Kontrolle der Auswirkungen dieser Änderungsmaßnahmen. Dies macht wieder die Analogie von Autonomic Computing und zentralem Nervensystem deutlich. Außerdem finden sich Parallelen zur klassischen Regelungstechnik.

Autonomic Manager können zudem hierarchisch angeordnet werden. D.h. Autonomic Manager können andere Autonomic Manager verwalten. Die übergeordneten Autonomic Manager werden dabei als ”Orchestrating Autonomic Manager” bezeichnet, die untergeordneten Autonomic Manager als ”Touchpoint Autonomic Manager”. Allerdings spielen Orchestrating Autonomic Manager derzeit kaum eine Rolle, da noch nicht klar ist, wie abstrakte Ziele eines Orchestrating Autonomic Manager in konkrete Ziele für Touchpoint Autonomic Manager überführt werden können. Selbst IBM nennt keine konkreten Anwendungsbeispiele.

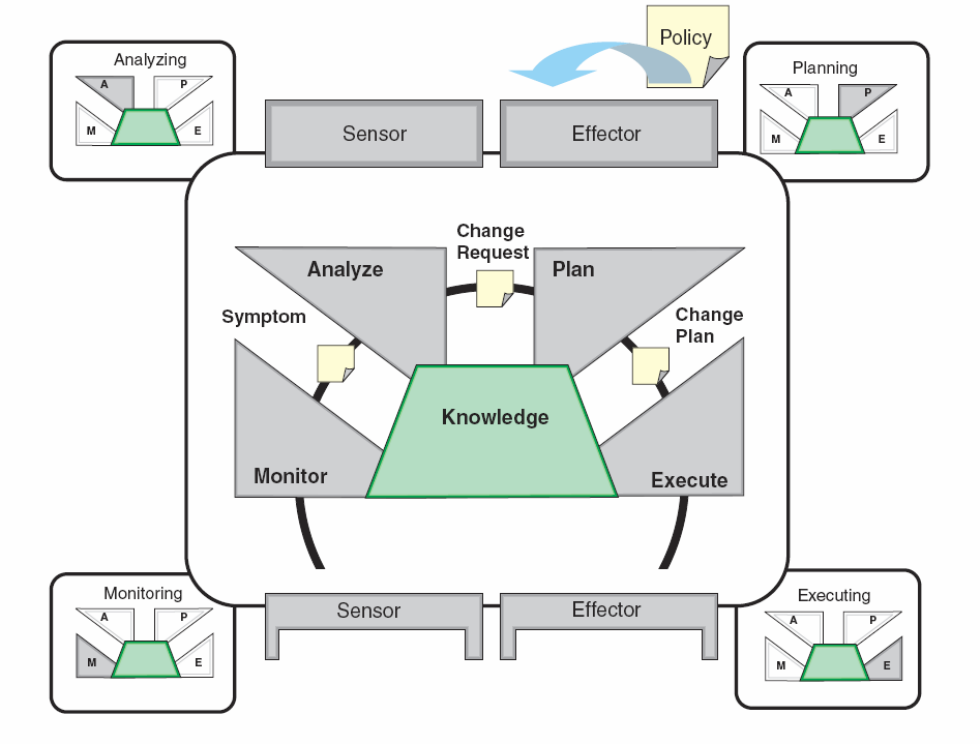

Abbildung 3.3: Autonomic Manager [IBM, 2005]

Knowledge Source Eine Knowledge Source (bzw. Knowledge Service oder K-Service) ist eine Art Registry oder Datenbank, in der z.B. Symptome, Policies oder Pläne gespeichert werden können. Dieses gespeicherte Wissen wird für die verschiedenen Funktionen der Autonomic Manager genutzt. Dabei wird das Wissen der Knowledge Sources ständig durch die Autonomic Manager erweitert.

3.2 Berkeley/Stanford: Recovery-Oriented Computing

Ein gemeinsames Forschungsprojekt der University of California at Berkeley und der Stanford University beschäftigtr sich mit dem Design fehlertoleranter Systeme. Dabei wird davon ausgegangen, dass sich Fehler in komplexen Computersystemen nie völlig vermeiden lassen. Ziel ist nun, auftretende Fehler so schnell und einfach wie möglich zu kompensieren.

Microreboots and Crash-Only Software Eine Möglichkeit, um ein System wieder in einen wohldefinierten Zustand zu bringen, ist ein Reboot. In der Regel dauert der Reboot eines komplexen Systems allerdings einige Zeit. Microreboots übertragen dieses Konzept auf Applikationskomponenten. Ziel dabei ist, eine deutliche Zeitersparnis gegenüber einem vollen Reboot zu erreichen.

Bei Crash-only Software wird schon das Design darauf ausgelegt, Systemteile neu starten zu können und plötzliche Ausfälle zu kompensieren. Darüber hinaus wird ein Crash-only-Programm dann nicht mehr wie üblich terminiert, sondern stets durch einen herbeigeführten Absturz beendet. Wichtig ist dabei die saubere Trennung zwischen Applikationslogik und Applikationsdaten, damit letztere nicht in Mitleidenschaft gezogen werden. [Stanford, 2006] [TU Berlin, 2006]

3.3 Carnegie Mellon University: Architecture Based Languages and Environments

Project Rainbow Das Rainbow-Projekts beschäftigt sich mit der automatisierten und dynamischen Adaptierung komplexer Systeme an sich ändernde Bedingungen. Kernaufgaben dabei sind die Ermittlung von Systemeigenschaften zur Laufzeit, der Vergleich dieser ermittelten Eigenschaften (Systemzustand) mit Vorgaben und Reaktion auf Abweichungen. Die Architektur steht dabei im Mittelpunkt. Anders als üblich gibt es bei diesem Konzept Modellmanipulationen zur Laufzeit. [CMU, 2006] [TU Berlin, 2006]

3.4 Produkte, Technologien und Standards

Wie in Kapitel 2.3 beschreiben, ist völliges Autonomic Computing (Level 5) eine bisher noch nicht gemeisterte Herausforderung. Jedoch gibt es inzwischen durchaus Produkte, in die die bisherigen Forschungsergebnisse einfließen. Eine detaillierte Betrachtung der verfügbaren Produkte würde allerdings den Rahmen dieser Ausarbeitung sprengen. Zu nennen sind an dieser Stelle zumindest: IBM: Autonomic Computing Toolkit, IBM: Tivoli Produkte, IBM:

WebSphere Produkte, IBM: DB2, HP: Utility Data Center Tool, SUN Microsystems: N1 Vision und Computer Associates: Sonar. [2]

Die Schwierigkeiten bei der Realisierung von Autonomic Computing Systemen begründen sich u.a. durch den Mangel an geeigneten Technologien und Standards. Besonders in den letzten Jahren verbessert sich diese Situation jedoch ständig, Als Folge der zunehmenden Verbreitung geeigneter Technologien und Standards wie z.B. Web Services, Java Management Extensions (JSR3), Logging API Specification (JSR47), Java Agent Services (JSR87) und Portlet Specification (JSR168). [IBM, 2005]

[2]Für detailliertere Informationen sei an dieser Stelle auf die Seiten der Hersteller verwiesen.

Kapitel 4

Bewertung und Ausblick

4.1 Bewertung

Autonomic Computing ist eine große Herausforderung und ein spannendes Thema. Es gibt in diesem Bereich noch viele ungelöste Probleme. Besondere Herausforderungen entstehen durch "dumme" und legacy Systeme, die nur schwer (bzw. eingeschränkt) in eine Autonomic Computing Architektur integriert werden können. Völliges Autonomic Computing (Level 5) wird in absehbarer Zeit nicht zu realisieren sein. Trotzdem bietet Autonomic Computing ein hohes Potential und wird langfristig die herkömmlichen Mechanismen für den Betrieb großer Computersysteme ablösen oder zumindest stark beeinflussen. Eine Entwicklung in diese Richtung zeichnet sich bereits seit einigen Jahren ab und erhält zusätzlichen Aufschwung durch die Verbreitung neuer Technologien und Standards, die für Autonomic Computing geeignet sind. (siehe Kapitl 3.4)

Gefahren Trotz der vielen potentiellen Vorteile, sollte Autonomic Computing auch kritisch betrachtet werden. In einem funktionierenden Autonomic Computing System würden praktisch keine technischen Experten mehr benötigt. Die Folge wäre somit ein Verlust von technischem Know-How und eine extrem hohe Abhängigkeit vom Autonomic Computing System.

was ist wirklich neu? Bei der Beschäftigung mit Autonomic Computing drängt sich schnell die Frage auf, was an den Konzepten von Autonomic Computing eigentlich neu ist. Fehlererkennung, Fehlertoleranz und Selbstkonfiguration sind keine neuen Themen. Besonders aus dem Server und DB Bereich sind Recovery Mechanismen, Load-Balancer, etc. schon lange bekannt. Eine wirkliche Neuerung ist jedoch die globale Sicht auf Unternehmen. Autonomic Computing betrachtet nicht nur einzelne Systemkomponenten, sondern das Zusammenspiel vieler Komponenten. (keine Insellösungen)

4.2 Ausblick für das AW2 Projekt

Autonomic ist ein sehr großes und komplexes Themengebiet. Im Rahmen des Projekts wird es nicht möglich sein, eine komplette Autonomic Computing Infrastruktur umzusetzen. Daher werden für das Projekt einige interessante Teilaspekte von Autonomic Computing ausgewählt und bearbeitet.

Thematisch passt Autonomic Computing in jedes der AW2 Projekte. Die Entscheidung ist jedoch für das Projekt "Pervasive Gaming Framework" gefallen. Im Rahmen dieses Projektes sollen besonders "self healing" und "self configuration" Funktionalitäten umgesetzt werden. Geplant ist dafür die Realisierung eines Knowledge Services und einfacher Autonomic Manager. Weiterhin sollen bei der Architektur des Frameworks standardisierte Schnittstellen für Autonomic Computing vorgesehen werden. Die zu entwickelnden Komponenten sollen u.a. folgende Aufgaben erfüllen können:

- Ausfälle oder Fehler von Servern kompensieren
- Ausfälle oder Fehler von Clients kompensieren
- Automatische Updates der verschiedenen Softwarekomponenten (automatisiertes Change Management)

Zusätzlich soll untersucht werden, welche Anwendungsmöglichkeiten sich für Orchestrating Autonomic Manager im Umfeld des Projekts bieten.

Literaturverzeichnis

[Ganek und Corbi, 2003] A.G. Ganek, T.A. Corbi: **The dawning of the autonomic computing era**, IBM Systems Journal, 42(1), (2003).

[Sterritt und Hinchey, 2005] Roy Sterritt, Mike Hinchey: **Autonomic Computing Panacea or Poppycock?**, Proceedings of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS05).

[Manoel et al, 2005] Edson Manoel, Morten Jul Nielsen, Abdi Salahshour, Sai Sampath K.V.L., Sanjeev Sudarshanan: **Problem Determination Using Self-Managing Autonomic Technology**, IBM Redbook, (2005)

[IBM, 2001] ibm.com: **Autonomic Computing**, (2001). Im Internet zu finden unter <http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf> (Juli 2006)

[IBM, 2005] ibm.com: **An architectural blueprint for autonomic computing.**, (2005). Im Internet zu finden unter <http://www03.ibm.com/autonomic/blueprint.shtml> (Juli 2006)

[Stanford, 2006] stanford.edu: **Microrecovery & Microreboot, Crash-Only Software**, (2006). Im Internet zu finden unter <http://crash.stanford.edu/> (Juli 2006)

[CMU, 2006] cmu.edu: **Rainbow Architecturebased Adaptation of Complex Systems**, (2006). Im Internet zu finden unter <http://www.cs.cmu.edu/~able/rainbow/> (Juli 2006)

[TU Berlin, 2006] tu-berlin.de: **Seminar Autonomic Computing**, (2006). Im Internet zu finden unter <http://uebb.cs.tuberlin.de/lehre/2006SSautonomiccomputing/> (Juli 2006)