»Self Healing« Systeme

Philip Eder Ortwin Probst

Seminar für Informatik bei Prof. Dr. Pree, 2009

Gliederung

Gliederung

Von Anpassung zu Selbstheilung

»Amazon erweitert seinen Cloud-Computing-Dienst EC2 um neue Funktionen wie Monitoring, Auto Scaling und Load Balancing. Damit können auf EC2 aufgesetzte Systeme automatisch je nach Traffic skalieren.«
(`http://www.golem.de/0905/67173.html` 18.05.2009 / 9:37)

- Selbstheilung betrifft hauptsächlich die Anwendun selbst und nicht nur die Infrastruktur (Cloud)

Computer Grids and Clouds.

- Grid von Computer Resourcen (Speicher, Datenbank, Durchsatz, ...)
- ermöglichen: Bereitstellen, Zugreifen, Teilen, Hinzufügen, Entfernen und Verwalten von Resourcen
- bieten: Spezielle Services zur automatischen Anpassung von dynamischen Anwendungen

Anforderungen an Self Healing Systeme.

- Anpassungsfähigkeit an unterschiedliche Situationen
- Transparenz bzw. kein Einfluss auf die Awendung

1. Fehlervorbeugung (z.B. Ressourcemanagement)
2. Fehler- und Ursachen-Erkennung
3. Fehlerbehebung

Gliederung

»Self Healing« und »Assisted Healing«.

Definition

»Self Healing« kann als Eigenschaft eines Software-Designs definiert werden, wobei ein System selbst feststellen kann, ob es korrekt funktioniert, und bei einer Fehlfunktion mit (bzw ohne) Fremdeingreifens die notwendigen Schritte gesetzt werden, um die „normale“ Funktionalität wieder herzustellen[1].

Gliederung

Systemzustände erkennen und definieren.

- Für die Entwicklung von effektiven Selbstheilungsstrategien muss festgestellt werden können, ob das System »gesund« oder »krank« ist.
- Wie definiert man aber den »normalen« Systemzustand?
  - Korrekter Betrieb leitet sich aus der Spezifikation ab.
  - Probleme: Vollständige Spezifikation ist nicht jedem bekannt bzw wird für jeden User anders zu definieren sein.

ADL zur Systembeschreibung.

- ADL = Architectural Description Language
  - Berühmtester Vertreter ist die Unified Modeling Language (UML)
- Begutachtung des Laufzeitverhaltens eines Systems mit »Darwin«-ADL zusammen mit der Sprache alloy

Gliederung

Fehlervorbeugung »Vorsorge«.

- Viele Vorgänge aus der Natur wurden als Vorbild benutzt.
  - Zellteilung
  - Ameisen, Vogelschwärme
- Fehlervorbeugung mittels Redundanzen
  - Selfassembling

Gliederung

Selfassembling

- Große Menge von identischen autonomen Agenten (auf jedem Agenten läuft das gleiche Programm).
- Diese Agenten interagieren nur lokal miteinander.
- Ziel ist es, verschiedenste Formen zu bilden, ohne die Agentenprogramme modifizieren zu müssen.
- Formen werden mit der sogenannten Origami Shape Language (OSL) spezifiziert.

Wichtige Überlegungen

- Wie kann das globale Ziel auf lokale Interaktionen der identisch programmierten Agenten übersetzt werden?
- Wie kann man ein robustes und »vorhersehbares« Verhalten von an sich unverlässlichen Komponenten erreichen?
  - einzelne Agenten können ausfallen
  - Selfhealing des Systems

Basiskonfiguration

- Programmierbares Blatt, das aus tausenden zufällig verteilten Agenten besteht.
- Agenten können so untereinander koordiniert werden, um das Blatt entlang von Geraden falten zu können.
- Kommunikation passiert nur lokal unter den Agenten
  - nur in einem kleinen Umkreis
  - physische Berührung der Agenten bei einer Faltung

Beispiel: Falten einer Tasse

- Aus dem Blatt soll eine Tasse gefaltet werden
- Beschreibung der verschiedenen Faltungen, die zur Tassenform führen (mit OSL)
- Basiselemente von OSL
  - Punkte
  - Linien
  - Regionen
- Agentenoperationen
  - Gradient
  - Nachbarn abfragen
  - Agent-zu-Agent Kontakt
  - Polaritätsumkehrung
  - Faltung

Basiskonfiguration

- Das Blatt sei mit 4 Eckpunkten (c1-c4) und 4 Begrenzungslinien (e12-e41) initialisiert.
- Linien können verwendet werden, um Regionen zu definieren
- Regionen begrenzen Faltungen auf bestimmte Ebenen

OSL Cup programm

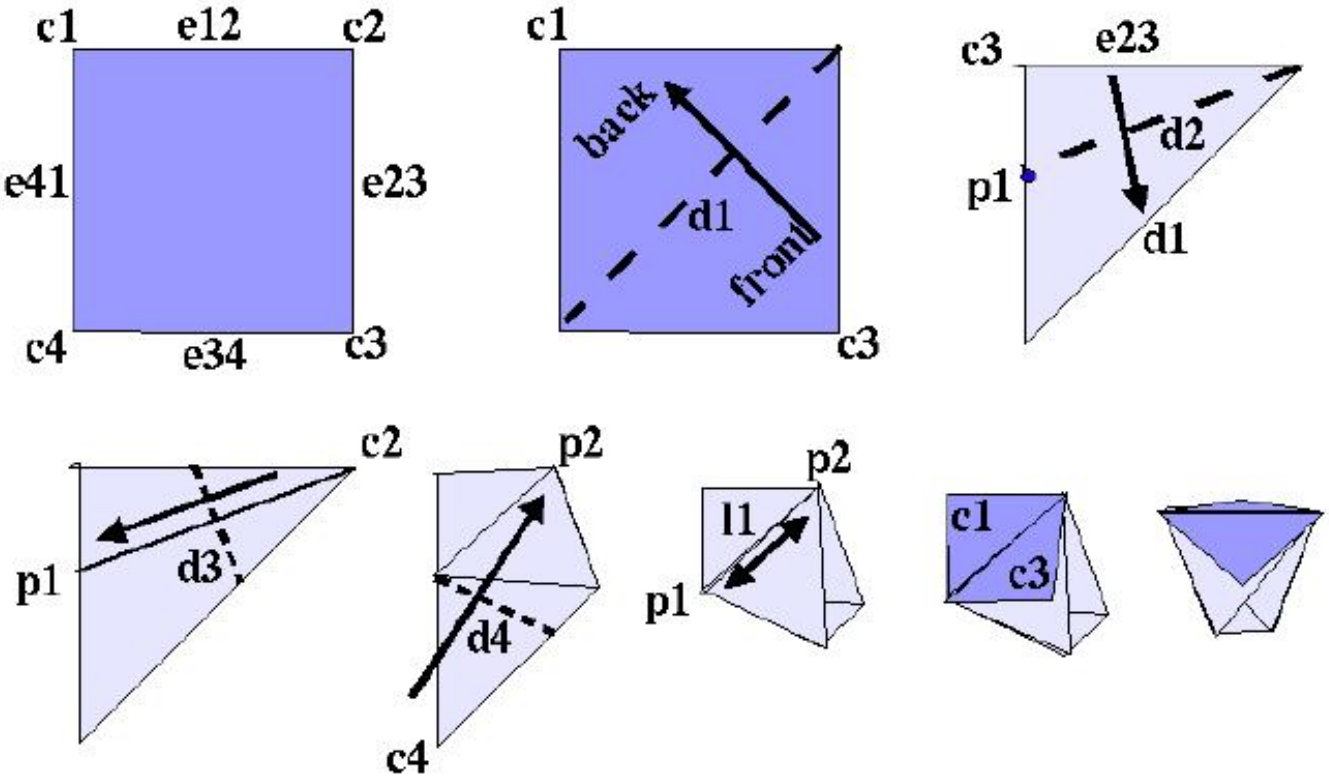
```
(define d1 (axiom2 c3 c1))
(define front (create-region c3 d1))
(define back (create-region c1 d1))
(execute-fold d1 apical c3)
(define d2 (axiom3 e23 d1))
(define p1 (intersect d2 e34))
(define d3 (axiom2 c2 p1))
(execute-fold d3 apical c2)
(define p2 (intersect d3 e23))
(define d4 (axiom2 c4 p2))
(execute-fold d4 apical c4)
(define l1 (axiom1 p1 p2))
(within-region front (execute-fold l1 apical c3))
(within-region back (execute-fold l1 basal c1))
```

[2]

Implementierung von Axiomen

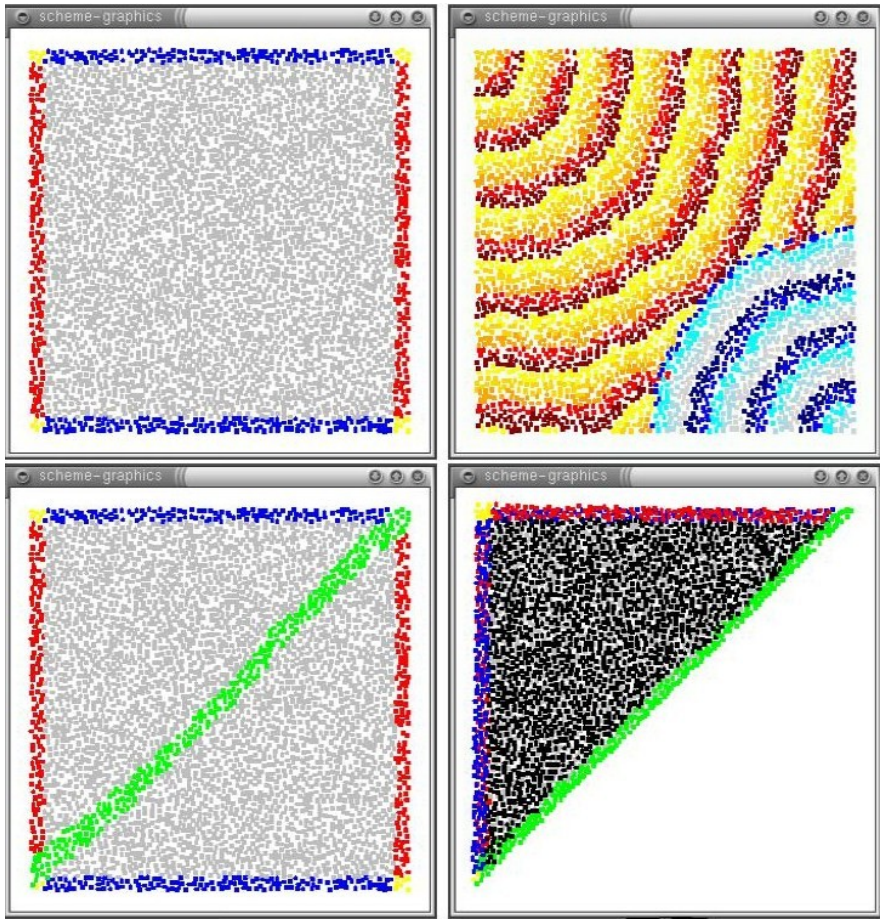
- Axiome werden unter Verwendung von Primitiven definiert
- Die erste Operation im Faltungsvorgang erzeugt die Diagonale d1 von den Punkten c1 und c3 unter Verwendung des axiom2
  - Das Blatt wird also so gefaltet, dass c3 auf c1 liegt
- Eine der wichtigsten Primitiven ist die Gradient-Primitive
  - Analogie: Gradient einer chemischen Konzentration ausgehend von einer Quelle
- Gradient-Primitive dient zur
  - Messung von Distanzen
  - Feststellung von lokalen Richtungen

Faltungsvorgang



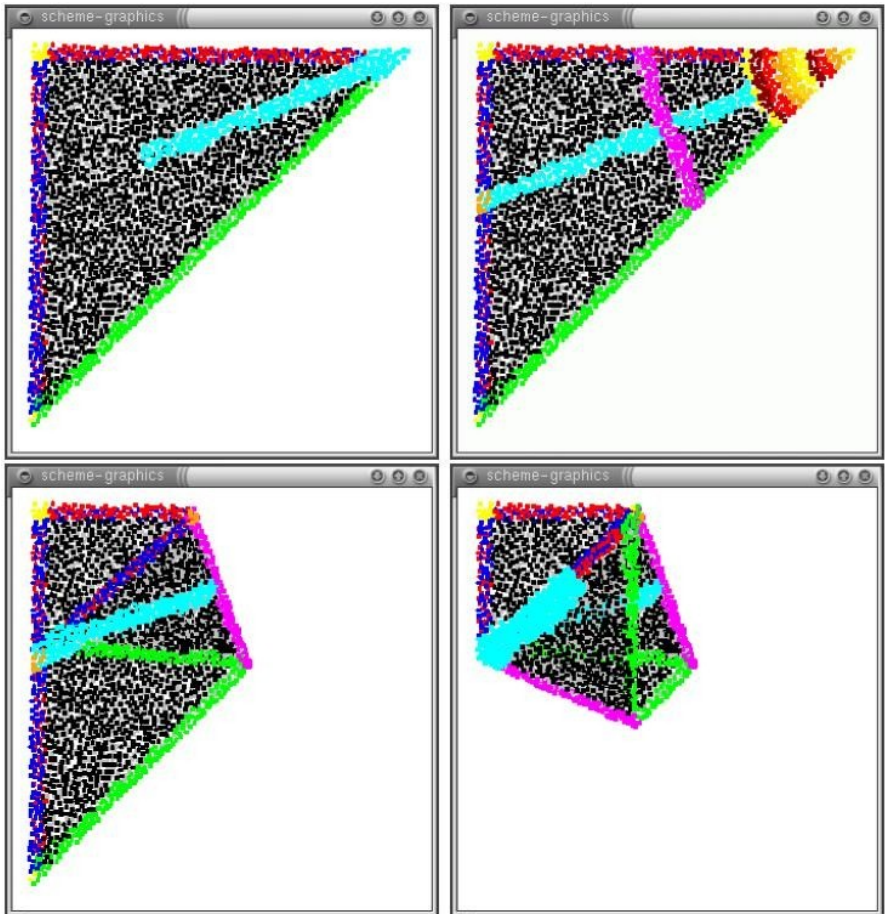
[2]

Simulation (1/3)



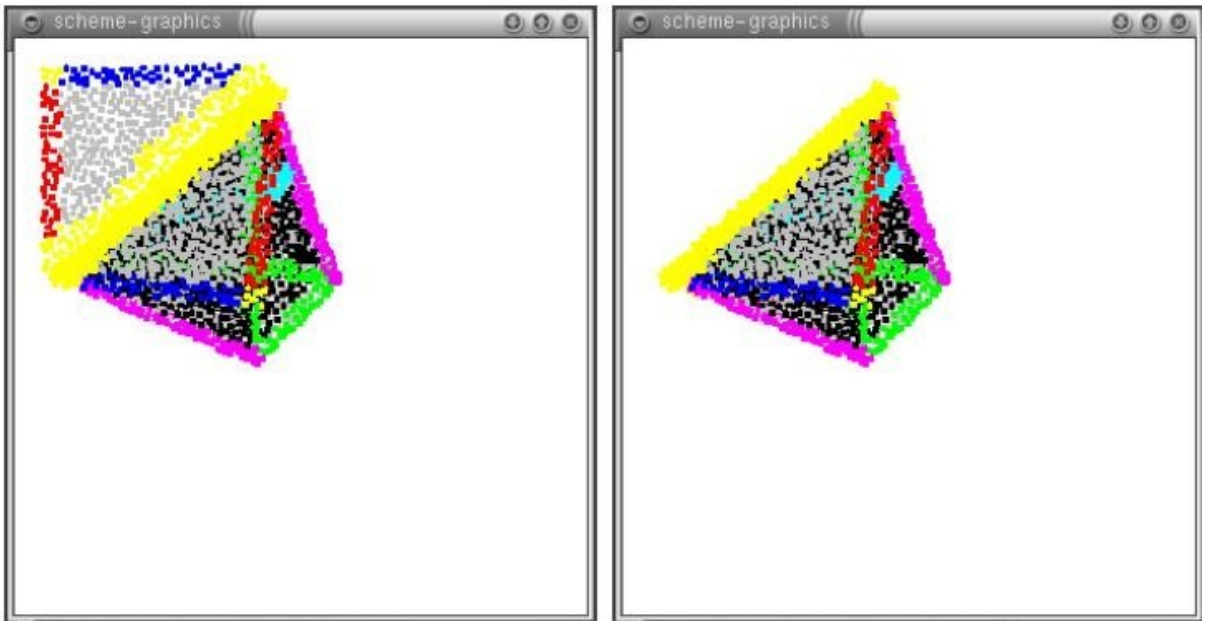
[2]

Simulation (2/3)



[2]

Simulation (3/3)



[2]

Interessante Eigenschaften

- Global-to-local Kompilierung
  - Abstrakte Beschreibung des globalen Problems, das auf lokale Interaktionen der Agenten übersetzt wird
- Selfhealing Eigenschaften
  - Tassenform bleibt erhalten, auch wenn einige Agenten "sterben" sollten –>selfrepair
- Skalierungsunabhängig, ohne dass Modifikationen notwendig werden
  - Agentprogramme arbeiten auch bei einer steigenden Anzahl von Agenten korrekt –> Vergrößerung des Blattes
- Verschiedenste Formen können gebildet werden, ohne Modifikation der Agentenprogramme

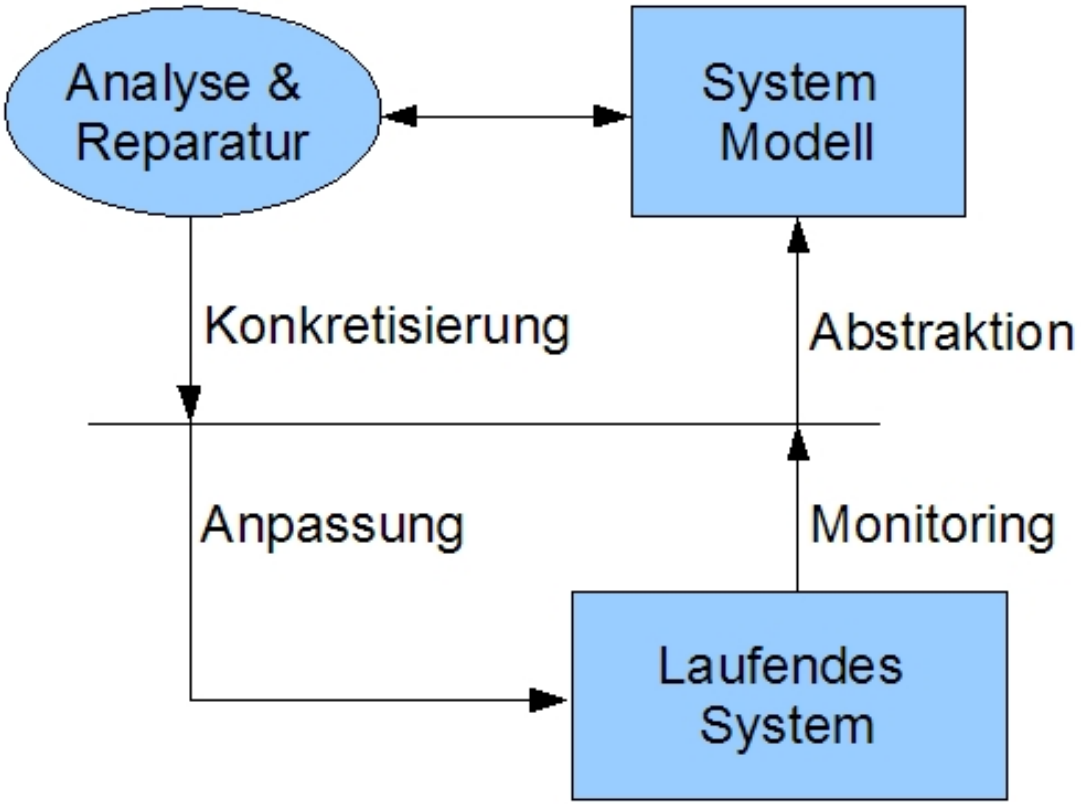
Gliederung

Grundlegende Strategie

Fehlererkennung Beansprucht 75% der 'Genesungszeit' (TellMe Networks)

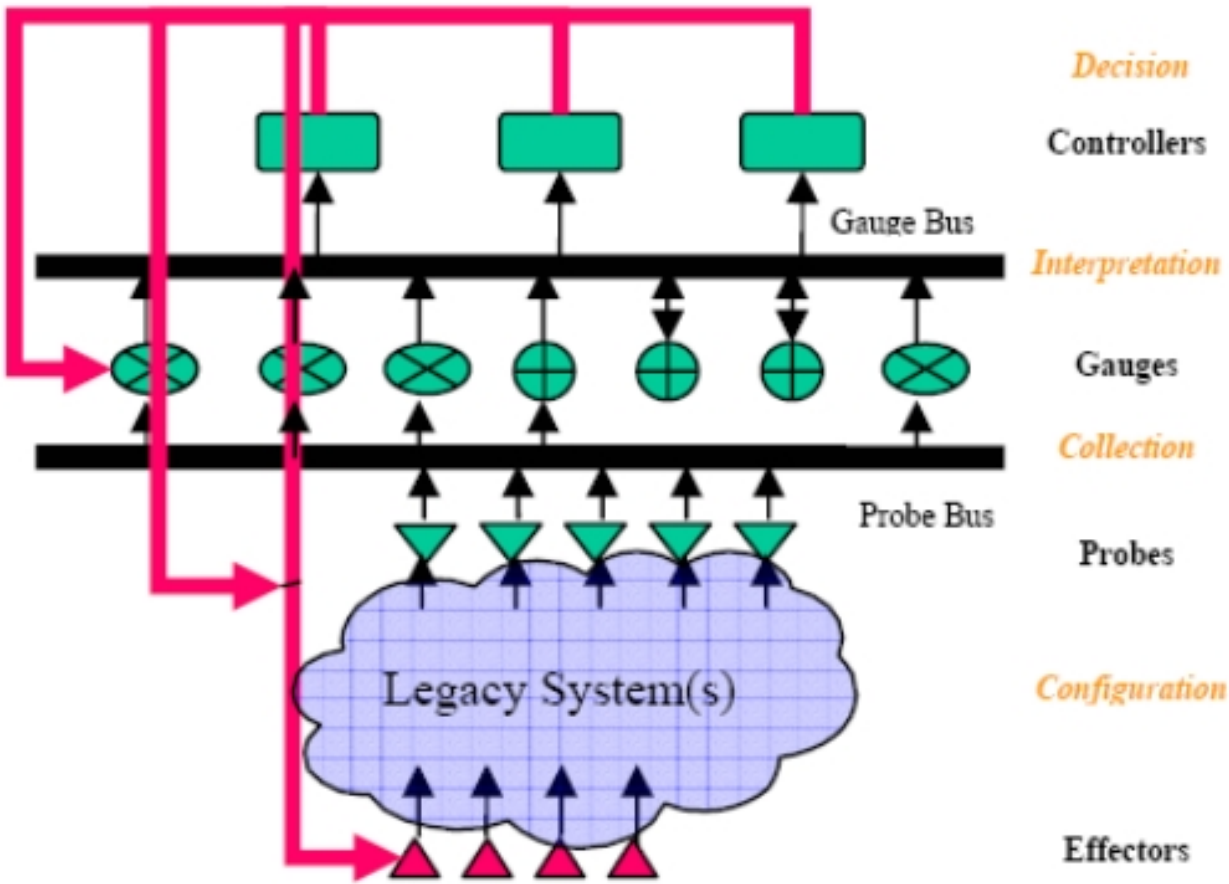
- Datensammlung
- Ermittlung des Basisverhalten
- Feststellen und Klassifizieren von Abweichungen zum Basisverhalten

Modell Basierte Anpassung



[3]

Beispiel einer Monitoring Infrastruktur



[4]

Gliederung

Statisch vordefinierte Regeln

- Für jeden (bekannten) Fehler ist eine entsprechende Korrekturmaßnahme festgelegt.

Ein auf Lernen basierter Ansatz

- Vordefinierte Reparaturmaßnahmen (vollständig!)

1. Iteratives Probieren der Korrekturen
2. Wahl der besten Korrekturmaßnahme
3. Merken der besten Maßnahme für den Fehler

Gliederung

Reparaturmaßnahmen

- Verständigung des Administrators
- Systemneustart / Reboot
- Microreboot

Microreboot

Definitionen

Fein granulare Reboots auf Komponentenbasis

Voraussetzungen:

- Persistenzschicht von Anwendungslogik gekapselt
- Ausfallsichere dauerhafte Daten und Sitzungen
- Lose Kopplung zwischen den Komponenten
- Sichere Referenzen außerhalb der Komponente

Gliederung

Grid Computing

- Heterogene Netzwerke und SW-Module setzen anpassbare Software voraus.
- Cheng et al beschreibt Softwarearchitektur für Grid Computing, die zur Laufzeit gewartet werden kann und Selbstheilungseigenschaften aufweist

Service Discovery Systeme

- Service Discovery Systeme bieten eine konsistente Sicht auf verteilte Komponenten unter sich verändernden Netzwerkbedingungen
- Für derartige Systeme werden verschiedene Selfhealing-Strategien verwendet
  - Fehlererkennung, Wiederherstellung und Fehlervorbeugung

Reflective Middleware

- Middleware = Software-Layer, der die Heterogenität der zugrundeliegenden Netzwerke, Hardware, Betriebssysteme,... versteckt
- Performancesteigerung für manche Applikationen, wenn sie Zugriff auf die "versteckten" Informationen haben
- Beispiel Client-Server Architektur
  - Lastverteilung wird verbessert, wenn dem System Informationen über momentane Ressourcenbelegung zur Verfügung stehen
- Auch hier Selfhealing-Aspekte wichtig
  - Ausfall von Komponenten
  - Rekonfiguration bei Änderungen der Abhängigkeiten zwischen den Komponenten

Weiterführende Literatur I


Debanjan Ghosh und Raj Sharman und H. Raghav Rao und Shambhu Upadhyaya
Self-healing systems — survey and synthesis
2006.


Radhika Nagpal und Attila Kondacs und Catherine Chang
Programming Methodology for Biologically-Inspired Self-Assembling Systems
2003.


David Garlan und Bradley Schmerl
Model-based Adaptation for Self-Healing Systems.
WOSS '02, 27–32, 2002

Weiterführende Literatur II


Giuseppe Valetto und Gail Kaiser
A Case Study in Software Adaption.
WOSS '02, 73–78, 2002