

# Evolution in verteilten Systemen – der evolutionäre Trader

*Markus Völter, MATHEMA AG*

[markus.voelter@mathema.de](mailto:markus.voelter@mathema.de)

*Markus Völter ist Consultant, Trainer und Architekt bei der MATHEMA AG in Ulm, deren Ulmer Geschäftsstelle er leitet. Er beschäftigt sich schwerpunktmäßig mit verteilten Systemen und Komponententechnologien. Desweiteren ist er Autor mehrerer Muster und Mustersprachen sowie Artikeln in diversen Fachzeitschriften.*

## Abstract

In verteilten Systemen werden üblicherweise Dienste angeboten und von Klienten verwendet. Meist fehlt allerdings eine Rückkopplung von Dienst-Benutzern zu den Anbietern, um eine Verbesserung der Dienste oder eine Anpassung an sich ändernde Anforderungen zu ermöglichen. Das Paper beschreibt einen an der biologischen Evolution angelehnten Ansatz, wie eine derartige Rückkopplung und eine damit verbundene Weiterentwicklung von Diensten mit Hilfe eines modifizierten Traders realisiert werden kann.

## Evolution in der Biologie

Evolution stellt in der Biologie den Motor der Weiterentwicklung dar. Praktisch die gesamte biologische Entwicklung beruht auf evolutionären Prinzipien. Wie unsere Umwelt zeigt, ist dies ein Konzept welches auf Lange Sicht bemerkenswert gut funktioniert. Dabei ist insbesondere interessant, dass keine zentrale Stelle benötigt wird, um den Evolutionsprozess zu "steuern". Evolution ist selbstregelnd.

Um Evolution zu verstehen, sollen im Folgenden einige Prinzipien und Einflussgrößen eingeführt werden.

### Mutation

Unter Mutation versteht man die zufällige Änderung der Genstruktur eines Lebewesens. Solche Änderungen können hervorgerufen werden durch "Kopierfehler" im Rahmen der Fortpflanzung, durch Krankheiten, etc.. Änderungen im Genotyp (also der genetischen Struktur eines Individuums) haben nicht automatisch Änderungen im Phänotyp (also dem externen Erscheinungsbild) zur Folge, da für eine Änderung des Phänotyps üblicherweise Änderungen mehrere Gene nötig sind.

Wenn Mutation stattfindet, ist es völlig unklar, ob sich diese Änderung der Genstruktur auf Lange Sicht durchsetzt oder nicht. Es ist auch nicht klar, ob diese Änderung von Vorteil oder von Nachteil ist, oder ob sie überhaupt einen Einfluss hat. Ob sich eine

Änderung langfristig durchsetzt hängt davon ab, wie fit das Lebewesen bezüglich seiner Umwelt ist.

### **Umwelt**

Eine Spezies, also eine Menge gleichartiger Individuen, lebt immer in einer bestimmten Umwelt. Die Lebewesen müssen mit der Umwelt zurecht kommen. Die Umwelt ändert sich üblicherweise im Laufe der Zeit und die Spezies muss sich durch Evolution an die sich ändernde Umwelt anpassen, um zu überleben. Diese Anpassung basiert üblicherweise auf Mutation und Selektion. Man nennt den Einfluss der Umwelt auch Selektionsdruck.

### **Fitness**

Fitness definiert die Eignung eines Lebewesens, mit der gegebenen Umwelt zurecht zu kommen. Die Fitness wird bestimmt vom Phänotyp des Lebewesens, der wiederum indirekt durch den Genotyp bestimmt wird. Man erkennt die Fitness an der Überlebensquote (bzw. der Fortpflanzungswahrscheinlichkeit bzgl. des Genpools der Population).

### **Population**

Eine Population ist eine Menge (phänotypisch) „gleicher“ Lebewesen, deren interne genetische Struktur aber leicht unterschiedlich ist. Änderungen am Genotyp einer Population treten auf entweder an einem Individuum (üblicherweise durch Krankheit) oder durch Fortpflanzung bei der folgenden Generation. Wichtige (und dauerhafte) Änderungen manifestieren sich durch eine Änderung der Genfrequenz, d.h. eine bestimmte Genstruktur kommt in einer Population öfter vor als andere.

### **Selektion**

Basierend auf der Fitness überleben Lebewesen mit einer bestimmten Genstruktur (Genotyp) eher als andere. Für das Überleben ist der Phänotyp verantwortlich, d.h. die nach aussen sichtbaren Konsequenzen des Genotyps. Fittere Individuen bringen ihre Gene mit höherer Wahrscheinlichkeit in die nächste Generation ein, da sie länger leben als weniger fitte.

Selektion sorgt also dafür, dass die für eine bestimmte Umwelt besser geeigneten Lebewesen „bevorzugt“ werden, und die anderen verdrängt werden und letztlich aussterben.

## **Definition des Evolutionsbegriffs**

In der Biologie kann Evolution also definiert werden als: *Veränderung des Genpools einer Population im Laufe von Generationen*. Verallgemeinert bedeutet dies *dass Evolution die Änderung der Eigenschaften in einer Menge von Entitäten im Laufe von Generationen darstellt*. Schrittweise:

- ? Eigenschaften mutieren zufällig
- ? Individuen werden selektiert aufgrund Fitness bzgl. der Umwelt; (Selektionsdruck)

Dadurch entwickeln sich Populationen weiter Für mehr Informationen zum Thema Evolution siehe [1].

## Beispiel: Biston Betularia (Englische Motte)

Ein bekanntes Beispiel für die Wirkungsweise der Evolution in der Biologie ist die englische Motte zur Zeit der Industrialisierung. Ende des 19. Jahrhunderts gab es von dieser Motte zwei Arten: dunkle und helle. Die Frage, ob ein Exemplar einer Motte dunkel oder hell zur Welt kommt, hängt von einem einzigen Gen ab. 1848 waren weniger als 2% dunkel, alle anderen waren hell.

1898, nach dem die Industrialisierung eingesetzt hatte, waren in industrialisierten Gebieten 95% aller Motten dunkel, in ländlichen Gebieten waren es deutlich weniger. Diese Veränderung des Genotyps bezüglich der Population der Motten ist ein Beispiel für Evolution. Ein Exemplar der Motte hat seine Farbe nie geändert – Änderungen traten nur bezüglich der Wahrscheinlichkeit des Gens im Pool auf.

Der Grund für die Veränderung war, dass die Bäume durch den Rauch der Industrieanlagen dunkler wurden. Vögel konnten die hellen Motten also besser sehen, und fraßen daher mehr helle Motten. Die dunklen hatten größere Überlebenschancen und somit auch mehr Nachfahren. Im Laufe von Generationen änderte sich also die Genstruktur der Population.

## Softwaresysteme und Evolution

Üblicherweise wird Software für eine definierte Systemlandschaft bzw. für definierte Anforderungen entwickelt. Diese Systemlandschaft und Anforderungen stellen die Umwelt aus Sicht der Software dar. In traditionellen Ansätzen wird zunächst die Umwelt analysiert, und dann für diese Umwelt ein möglichst ideales System entwickelt.

### Probleme des traditionellen Ansatzes

Da man üblicherweise nicht beim ersten Versuch das bestmögliche System erzielt, werden iterative Prozesse verwendet, bei denen man versucht, durch “asymptotische Annäherung” an die Umwelt dieser in einem akzeptablen Kostenrahmen möglichst gut gerecht zu werden. Allerdings wird auch bei iterativen Prozessen immer noch *ein* System für *eine* bestimmte Umwelt entwickelt.

Wenn sich nun die Anforderungen oder die Systemlandschaft ändern bekommt man das klassische Moving-Target-Problem: Die Anforderungen ändern sich schneller als das System fertig gestellt wird – mit der Folge, dass das System nie fertig wird; es bleibt instabil, und die ewig dauernde Analyse ist aufwendig und teuer.

Verschiedene Methoden und Techniken sind mit dem Ziel angetreten, Software für eine sich schnell ändernde Umwelt produzieren zu können. Einige Beispiele:

### Frameworks

Hier wird eine Anwendung (bzw. eine Anwendungsfamilie) in feste Teile und variable Teile unterteilt. Die variablen Anteile können für jede konkrete Anwendung des Frameworks angepasst werden. Der Nachteil hierbei ist allerdings, dass diese variablen Stellen von vornherein bekannt sein müssen, um sie entsprechend flexibel zu gestalten. (siehe auch [2])

## Systemfamilien

Hier ist das Ziel der Entwicklung nicht nur ein System, sondern eine Familie von Systemen. Jede dieser Anwendungen kann eine leicht andere, aber immer noch statische Umwelt haben. Auch hier das Problem: Die Unterscheidungsmerkmale müssen vorher bekannt sein, eine umfangreiche Domänen-Analyse ist daher Voraussetzung.

## eXtreme Programming

XP versucht durch einen Prozess, der das Ändern und Anpassen der Software an sich ändernde Umgebungen durch regelmäßiges Testen zu "entschärfen". (siehe auch [3])

## "Mob Software"

Richard P. Gabriels Vortrag auf der OOPSLA 2000 (siehe [4]) beschäftigte sich u.a. Mit dem Thema, ob es überhaupt möglich ist, Software zu planen und gezielt zu designen. Er stellte die Hypothese auf, dass es besser sei, wenn man Anarchie walten lasse, um zu „schönen“ Ergebnissen zu kommen.

Als Beispiel nannte er die Entwicklung von Linux: Viele Leute arbeiten „unkontrolliert“ an der Weiterentwicklung, das Endergebnis entsteht aus dieser „chaotischen“ Entwicklung durch „Selektion“ durch die Umwelt (also die Benutzer).

## Service Evolution

Der evolutionäre Trader wie er im Folgenden beschrieben wird ermöglicht die Evolution von Services in verteilten Systemen. Dazu ist es zunächst nötig zu definieren, was in diesem Zusammenhang unter einem Service zu verstehen ist: *Ein Service ist eine Softwarekomponente in einem verteilten System, die einen durch ein oder mehrere Interfaces genau definierten Dienst erbringt.* Beispiele für Dienste die dieser Definition genügen wären Java's RMI Objekte, CORBA Objekte [5], Enterprise Java Beans [6], sowie Jini Services [7]. Meistens sind derartige Dienste zustandslos – dies ist allerdings keine zwingende Voraussetzung.

Wenn man diese Definition eines Service zugrunde legt, dann bleiben noch eine ganze Reihe von Dingen die im Rahmen des Interfaces nicht definiert sind. Da wäre zunächst einmal die Semantik der Operationen. Diese kann entweder durch OCL oder durch Prosatext definiert werden. In den meisten Fällen sind sich die Nutzer und die Implementierer eines Dienstes über diese Semantik einig, sonst könnte der Dienst nicht verwendet werden. Es gibt allerdings noch eine ganze Reihe weiterer Aspekte, die durch ein Interface nicht definiert werden:

- ? Reaktionszeit
- ? Genauigkeit
- ? Zuverlässigkeit
- ? Verfügbarkeit
- ? etc.

Diese Dinge lassen sich unter der Begriff *Quality of Service* oder auch *nicht-funktionale Anforderungen* zusammenfassen. Damit dürfte klar sein, was im Rahmen dieses Papers unter Service Evolution verstanden wird: *Service Evolution ist die evolutionäre Verbesserung der Qualität eines Services aus Sicht der Clients.*

Dabei bedeuten:

- ? evolutionär: schrittweise, ungeplante Veränderung einer Menge von Service-Instanzen
- ? Verbesserung: Ein Selektionsverfahren sorgt dafür, dass die Service später besser an die Umwelt angepasst sind als vorher.
- ? Qualität des Services: Alle Freiheitsgrade, die die Spezifikation des Services noch ermöglicht, z.B. Performanz, Genauigkeit, Zuverlässigkeit, ...

In der Konsequenz bedeutet dies, dass ein Service durch eine Menge von individuellen Service-Instanzen repräsentiert wird (Population), sich jede Instanz verändern können muss, und es muss eine Art Rückmeldung (Selektion) durch die Umgebung erfolgen. Die Service-Instanzen wissen dabei nicht, warum sie von der Umwelt bevorzugt oder benachteiligt werden.

## Der evolutionäre Trader

Ein Trader ist ein Service, welcher einem Client Referenzen auf Dienstinstanzen liefert, basierend auf einer vom Client gelieferten Dienstbeschreibung. Abbildung 2 verdeutlicht dies.

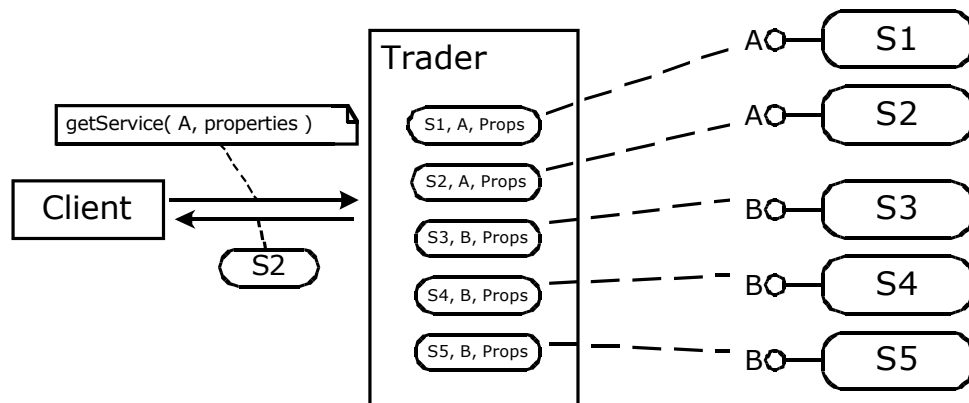


Abbildung 1: Funktionsweise eines Traders: Der Client fordert einen Dienst an, welchen er durch eine Menge von Properties beschreibt. Der Trader liefert eine passende Instanz zurück

Die vom Client angegebene Beschreibung besteht aus dem Interface des Dienstes, und einer Menge an Properties die den Dienst näher beschreiben. Auch „komplexere“ Queries sind prinzipiell möglich, da die Properties eine beliebig komplexe Suchanfrage enthalten können. Falls mehrere Instanzen auf eine Suchanfrage passen, entscheidet der Trader, welche Instanz zurückgeliefert wird. Dies hat die folgenden Vorteile:

- ? Entkopplung der Provider von ihren Benutzern
- ? Neue Dienstprovider können dynamisch registriert und zurückgezogen werden
- ? Lastverteilung ist möglich (Round Robin, Basierend auf Lastauskunft)
- ? Basis für Failover: Wenn Dienst stirbt, dann kann sich der Client (mit derselben Beschreibung) einen neuen holen

Das Konzept eines Traders findet u.a. Anwendung in CORBA's Trading Service und in Jini's Lookup Service. Um evolutionäre Konzepte im Trader verwenden zu können,

müssen die im ersten Teil erläuterten Prinzipien aus der Biologie irgendwie umgesetzt werden. Dies geschieht folgendermaßen:

- ? Population: Mehrere Implementierungen (Provider) pro Dienst
- ? Umwelt: Clients und deren Anforderungen
- ? Selektion: Rückmeldung von Clients, Policies des Traders
- ? Mutation und Weiterentwicklung: Veränderung einiger Dienstimplementierungen

## **Funktionsweise**

Im Folgenden soll die Funktionsweise des evolutionären Traders schrittweise erläutert werden.

1. Der Client stellt eine Anfrage an der Trader, wobei er den benötigten Dienst durch das Interface und eine Menge von Properties spezifiziert.
2. Der Trader liefert eine passende Referenz zurück. Zusätzlich wird ein VotingTicket (also eine Art Wahrscheinlichkeit, die die Berechtigung zur Abgabe einer den Dienst beurteilenden Stimme repräsentiert) zurückgegeben.
3. Wie beim normalen Trader verwendet der Client den Dienst, ohne den Trader zu benötigen.
4. Danach wendet sich der Client an den Trader und bewertet den verwendeten Dienst. Dazu verwendet er das vorher erhaltene VotingTicket. Die Bewertung findet auf einer vorgegebenen Skala (z.B. 0-100) statt. Es bleibt dem Client überlassen, nach welchen Kriterien er den Dienst bewertet.
5. Der Trader speichert die Bewertung der einzelnen Dienstimplementierungen ab.
6. Der Trader erzeugt sich Statistiken über die Bewertungen der einzelnen Dienstimplementierungen. Dabei können z.B. ältere Votings im Laufe der Zeit "vergessen" werden, damit eine "schlechte Vergangenheit" einen in der Zwischenzeit verbesserten Dienst nicht auf Dauer belastet.
7. Bei zukünftigen Anfragen der Clients liefert der Trader die Dienstimplementierungen mit der besten Bewertung zurück.
8. Der Trader informiert die Dienstimplementierungen regelmäßig über deren eigene Bewertungen, und über die Bewertungen der anderen Anbieter (anonymisiert).
9. Die Services können sich nun verändern, wenn sie wissen, dass ihre Qualität verglichen mit anderen schlecht ist. Sie wissen aber nicht wie sie sie verbessern können; sie müssen irgendwelche Annahmen treffen.
10. Bei einer Anfrage von Client liefert der Trader manchmal auch den nicht-besten Dienst zurück; dieser kann sich ja in der Zwischenzeit verbessert haben!

Weiterhin ist zu beachten: Wenn sehr viele Clients den besten Service als Antwort auf ihre Anfrage bekommen, wird dessen Belastung möglicherweise höher und sein Rating damit schlechter. Andere Services bekommen dadurch wieder eine Chance.

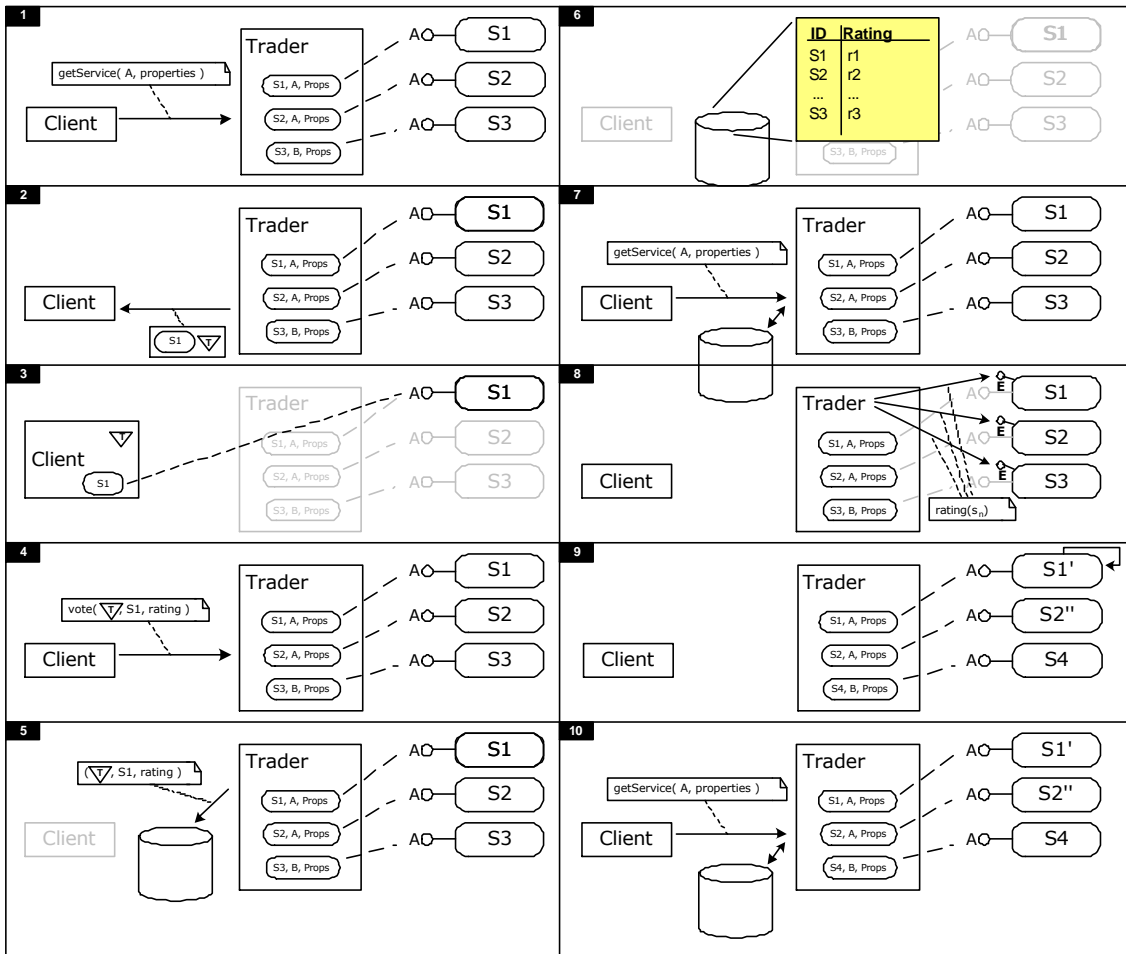


Abbildung 2: Funktionsprinzip des evolutionären Traders

## Vor- und Nachteile des evolutionären Traders

Zunächst hat der evolutionäre Trader alle Vorteile des normalen Traders, diese sollen hier nicht wiederholt werden. Zusätzlich sind die folgenden Vorteile feststellbar:

- ? Kein Koordinationsaufwand bezüglich der Art der Bewertung. In dezentralen verteilten Systemen kann sich diese Koordination zu einem ernsthaften Problem entwickeln.
- ? Sehr lose Kopplung
- ? Services entwickeln sich zufällig weiter: Chance für unerwartete Veränderung
- ? Dynamische, schrittweise Anpassung an eine sich verändernde Umwelt
- ? Clients bestimmen die Qualitäten der Services

Nachteile sind natürlich auch zu verzeichnen: Je nach Szenario, findet nur eine langsame Anpassung an die sich verändernde Umwelt statt. Weiterhin kann nicht garantiert werden, dass jeder Client bei jeder Anfrage den besten Service bekommt. Potentiell besteht auch die Möglichkeit, dass sich die Qualität der „Dienste“, zumindest zeitweise, nach unten verändert. Durch entsprechende Policies im Server muss verhindert werden, dass sich ein „schwingendes System“ einstellt.

## Mögliche Anwendungsbeispiele

### Dienst zur Matrizenberechnung

Der Dienst kann man große Matrizen multiplizieren, addieren, etc. Große Matrizenoperationen bieten viel Potential für Optimierungen und sind insofern ein gutes Beispiel. Als Qualitätsaspekte sind insbesondere Genauigkeit und Performanz zu nennen. Als Rating kann eine Zahl zwischen 0 und 100 verwendet werden. Die Weiterentwicklung der Dienste kann durch Verbesserung dieser Aspekte durch Programmierung stattfinden.

### Radarstellungen

Eine Radarstellung bietet einen genau definierten Dienst: Luftlagedaten für ein bestimmtes Gebiet. Überschneidungen der vom Radar abgedeckten Gebiete haben zu Folge, dass es für ein bestimmtes Gebiet mehrere Dienstanbieter (also Radarstellungen) gibt. Diese liefern aber aufgrund ihrer Position, des Wetters oder durch Störungen unterschiedlich gute Qualität. Die Leitstände die diese Daten verwenden bewerten die Dienste.

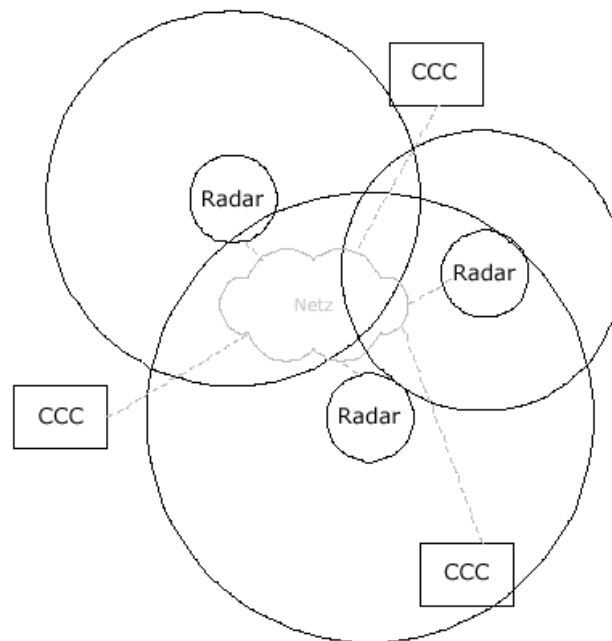


Abbildung 3: Radar-Beispiel. CCC's sind die Clients, die Radardaten nutzen

Eine Verbesserung der Qualität kann erfolgen durch

- ? Verwendungen eines anderen Bandes
- ? Reparatur
- ? Verlegung

# **Erweiterungsmöglichkeiten**

## **Ressourcenmanagement**

Wenn die verschiedenen Dienste unter der Kontrolle des Traders laufen, können den schlechteren Diensten die Ressourcen beschränkt werden, z.B. durch Reduktion der Anzahl der Handler Theresa, der Thadden Priorität, etc. Damit wird erreicht, dass schlechte Dienstimplementierungen wirklich aussterben. Im Extremfall kann der Dienst deinstalliert werden.

## **Subventionen**

Der Trader kann den schlechten Diensten mit Absicht bessere Ratings zukommen lassen, um im Falle von Ausfällen des besten noch ein Backup zu haben. Damit werden schlechte Abbildung Dienste "subventioniert". Dies ist insbesondere im Zusammenhang mit Ressourcenmanagement interessant.

## **Benachrichtigung bei Änderung der Implementierungen**

Der Trader liefert manchmal auch die nicht-optimalen Dienstimplementierungen zurück, da die sich potentiell geändert haben können. Der Fortgang der Evolution ist sehr langsam. Um dies zu beschleunigen, können neue (oder geänderte) Services den Trader über die Änderung benachrichtigen, damit dieser den neuen gezielt mehr Chancen geben kann.

## **Zusammenfassung**

Das Paper zeigt wie Prinzipien aus der biologischen Evolution in verteilten Systemen eingesetzt werden können, um Systeme zu bauen, die sich im Laufe der Zeit an sich möglicherweise ändernde Anforderungen anpassen können. Da keine Absprache über die Semantik des Protokolls nötig ist, eignet sich dieser Ansatz ideal für sehr lose gekoppelte Systeme, beispielsweise im P2P Bereich.

Der nächste Schritt ist nun die Implementierung eines Prototypen, der die vorgestellten Konzepte validiert.

## **Danksagungen**

Vielen Dank an Bernd Kolb, mit dem ich das Thema ursprünglich diskutiert habe – diese Diskussion ist die Basis für dieses Paper. Weiterhin gilt der Dank Jens Tränkle und Wolfgang Jäkel, die mir beim biologischen Teil zur Seite standen.

## Literatur und Ressourcen

- [1] Grundlagen der Evolution: <http://www.talkorigins.org/origins/faqs-evolution.html>
- [2] Cetus-Links.org, Frameworks, [http://www.cetus-links.org/oo\\_frameworks.html](http://www.cetus-links.org/oo_frameworks.html)
- [3] eXtreme Programming, xProgramming.com, <http://www.xprogramming.com/>
- [4] Richard P. Gabriel, Mob Software, <http://www.laputan.org/gabriel/mob.pdf>
- [5] Object Management Group, CORBA, <http://www.corba.org/>
- [6] JavaSoft, Enterprise JavaBeans, <http://java.sun.com/products/ejb/index.html>
- [7] Sun, The Jini Technology, <http://www.sun.com/jini/>