von bernhard angerer

mehr zum thema:
www.syngo.com
www.siemensmedical.com

„SPACE BASED PROGRAMMING": J2EE BEKOMMT KONKURRENZ UND DAS AUS DEM EIGENEN LAGER.

„Space Based Programming" bezeichnet eine neue Art verteilte Anwendungen zu bauen, bei der das Abstraktionsniveau höher liegt als bei aktuell etablierten Methoden. Neben den Aspekten der Netzwerkkommunikation wird auch die Verteilung an sich vom Entwickler ferngehalten. Die Anwendung glaubt, auf einem Rechner zu sein, und kommuniziert ausschließlich über einen virtuellen Speicherbereich, was zu hervorragenden Eigenschaften führt und gleichzeitig die Menge an benötigtem Quellcode stark reduziert.

der autor

Dr. Bernhard Angerer (E-Mail: bernhard@angerer.com) ist unabhängiger Berater in New York und widmet sich ganz den Space-Technologien sowie deren zukünftigen Anwendungen

KORREKTURFAHNE

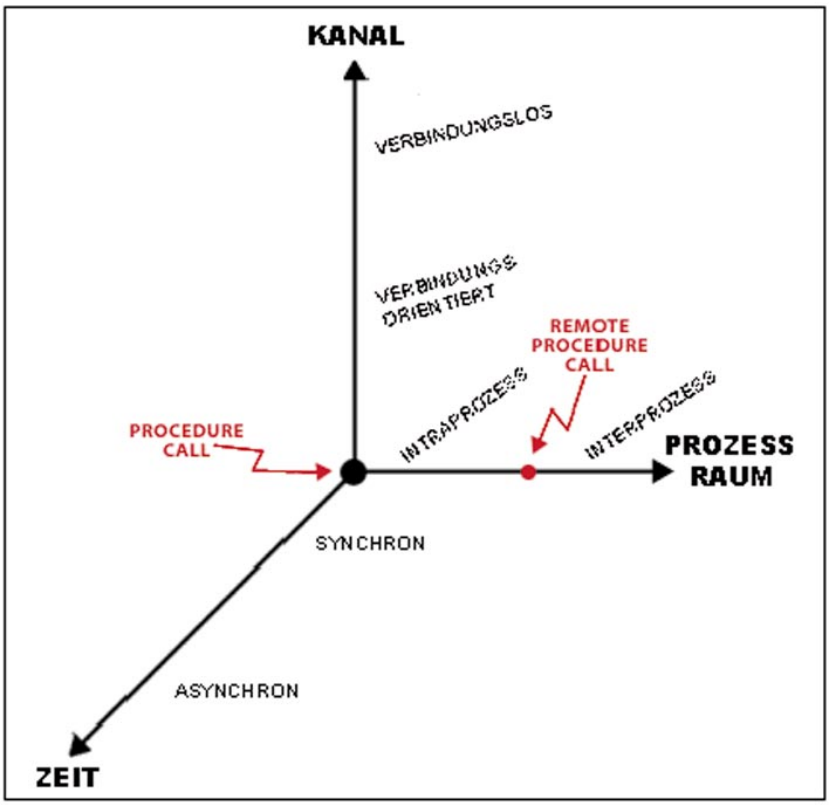
Der Ausdruck „Space Based Programming" bezeichnet eine neue Art und Weise, verteilte Anwendungen zu bauen. Die aktuell dominierenden Methoden sind durchwegs *RPC*-basisert (*Remote Procedure Call*) und finden ihren Niederschlag in CORBA, EJB oder COM/DCOM. Die Space-Architektur liefert hingegen ein überraschend einfaches Modell, das die RPC-Metapher vollständig ersetzt. Der minimalistische Ansatz, der dieser Architektur inhärent ist, führt zum einen zu sehr breit gefächerten Einsatzmöglichkeiten und zum anderen zu massiven Verbesserungen in Bezug auf Modularität, Skalierbarkeit und Reduktion an Quellcode.

Die ersten Ansätze gehen auf Professor Gelernter und dessen Arbeiten in den frühen achtziger Jahren an der Yale University zurück. Dr. Gelernter (siehe [Gel92]) entwickelte eine Programmiersprache namens „Linda", um verteilte Anwendungen zu entwickeln. Linda bestand aus einer kleinen Anzahl an Operationen, kombiniert mit einem global verfügbaren persistenten Speicher, genannt der „Tuple-Space". Die Operationen in diesem System sind Teil einer Koordinierungssprache; sie liegen orthogonal zu herkömmlichen Programmiersprachen und können daher problemlos in bestehende Sprachen eingefügt werden. Das Ergebnis zeigte, dass mit dieser Architektur eine große Anzahl an Problemstellungen im Bereich der parallelen und verteilten Datenverarbeitung sehr gut implementierbar ist.

Verteilter, gemeinsam genutzter Speicher

Der *Space* bezeichnet einen virtuell verteilten, gemeinsam genutzten Speicher (*Shared Memory*). Die notwendige Programmierschnittstelle umfasst im Wesentlichen nur die vier Methoden *write*, *read*, *take* und *notify*. Mit *write* und *read* können Objekte in den Speicher geschrieben bzw. vom Speicher gelesen werden. *take* entfernt das Objekt nach der Lese-Operation und *notify* benachrichtigt über Änderungen im *Space*.

Dieser Ansatz liefert das einfachste Schema, das man sich bei der Entwicklung von verteilten Anwendungen wünschen kann. Bei entfernten Prozeduraufrufen kommunizieren und synchronisieren sich die Prozesse durch explizites Übergeben von Nachrichten. Diese Vorgehensweise skaliert meist nicht besonders gut (zumindest nicht ohne zusätzlichen Aufwand). Je mehr Gesprächspartner teilnehmen, desto komplizierter werden die Kommunikationswege. Bei der *space*-basierten



Implementierung werden alle Teilnehmer über selbigen synchronisiert. Es handelt sich um eine reine *Blackboard*-Kommunikation". Das bedeutet, dass die Gesprächspartner

- nichts voneinander wissen müssen (verbindungslos / anonym),
- nicht im selben Prozess oder auf der selben Maschine sein müssen (interprozess) und
- zeitlich vollkommen unabhängig voneinander sind (asynchron)

Diese vollständige Entkopplung in allen drei Freiheitsgraden (**siehe Abb. 1**) führt zu sehr schönen Ergebnissen auf dem Weg, flexible Systeme zu kreieren.

Folgende Eigenschaften gewährleistet der *Space* respektive der *Space*-Server (vgl. [Fre99]):

- **Gleichzeitiger transparenter Zugriff:** Der *Space* kümmert sich darum, dass mehrere Prozesse verteilt im Netzwerk gleichzeitig in transparenter Art und Weise Zugriff haben. Verteilte Datenstrukturen können so einfach erstellt werden, wobei die Details der Kommunikation vom Anwendungsentwickler vollkommen ferngehalten werden. Die tatsächliche Funktionsweise (Replikation bzw. Propagierung der Daten) hinter den Kulissen variiert dabei bei den verschiedenen Server-Implementierungen. So bietet Corso (**siehe Tabelle 1**) beispielsweise zwei Arten an Replikation (*on-demand*, d.h. auf Abruf, oder *pre-fetching*, d. h. vor dem eigentlichen Zugriff werden die Daten bereits geholt). „GigaSpaces" in der gerade verabschiedeten Version 2.0 bietet nun ebenfalls Replikationsmechanismen.
- **Persistenz:** Der *Space* besitzt einen Speichermechanismus, der sicherstellt, dass Objekte solange im *Space* zur Verfügung stehen, bis sie explizit von einem Prozess entfernt werden (auch wenn die Maschine zwischendurch neu gestartet wird). Das erlaubt beispielsweise die Implementierung einer Chat-Anwendung, in der die Teilnehmer nicht zur gleichen Zeit anwesend sein müssen, um eine Konversation zu führen. Darüber hinaus gibt es noch die sogenannte *lease-time*. Ist diese Zeit abgelaufen, wird ein Objekt automatisch aus dem Speicher entfernt.
- **Assoziativität:** Objekte werden mit Hilfe von sogenannten *associative lookups* Methode zum Lesen verlangt ein Objekt, dessen Felder wahlweise mit *null* (fungiert als Joker) oder bestimmten Werten initialisiert sind. So kann die *Space-Engine* mit Hilfe von Werte- und Typvergleichen das gesuchte Objekt zurückliefern.
- **Transaktionen:** Ein Transaktionsmodell stellt sicher, dass die Operationen mit dem *Space*atomar sind. Transaktionen werden sowohl für einzelne Operationen als auch über mehrere Operationen mit einem oder mehreren *Spaces* unterstützt.
- **Austausch von ausführbarem Code:** Objekte im *Space* befinden sich in serialisierter Form. d. h. es handelt sich um passive Daten. Sobald ein Objekt gelesen wird, können dessen Felder modifiziert oder Methoden aufgerufen werden. Handelt es sich bei der Zielsprache um Java, so sorgt der *Class-Loader* dafür, dass auch neue Objekte, die nicht bekannt sind, ausgeführt werden können.

Produktname	Hersteller	URL
GigaSpaces 2.0	GigaSpaces, Herzelia, Israel	http://www.gigaspaces.com
IntaSpaces	IntaMission, Windor, UK	http://www.intamission.com
TSpaces	IBM, Almanaden Research Center, USA	http://www.almaden.ibm.com/cs/TSpaces/
Corso Release 3.0	Tecco, Wien, Österreich	http://www.tecco.at

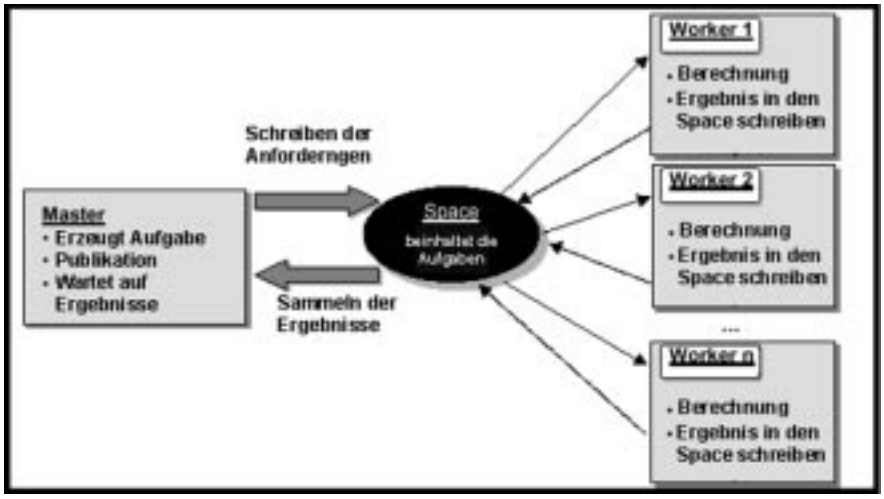
Tabelle 1: Die vier am Markt verfügbaren Server

Die Einfachheit des grundlegenden Paradigmas ist bestechend. Es kann sehr schnell überblickt werden und das Lernen von komplexen Schnittstellen fällt weg. Die minimalistische Programmierschnittstelle führt zu stark reduziertem Aufwand bei der Realisierung. Durch die Entkopplung von Sender und Empfänger werden die Anwendungsprotokolle einfacher, flexibler und stabiler. Dies unterstützt das Bauen von großen Anwendungen, das Analysieren und Testen (die Komponenten sowie die verteilte Koordination können getrennt betrachtet werden) sowie Wiederverwendung (Komponenten können leicht ausgetauscht werden, solange sie dem „Gesprächskontext" folgen).

Das Entwurfsmuster „Master Worker"

Das Entwurfsmuster *Master Worker* (siehe [Fre99c]) zeigt sehr schön, auf welche Art und Weise beispielsweise eine Web-Anwendung mit dem *Space*-Ansatz entwickelt wird.

Anfragen der Clients (z.B. *Queries* in verschiedenen Archiven) werden vom Web-Server bzw. dem *Master* asynchron mit Hilfe von Anfrage-Objekten in den *Space* publiziert. Darauf reagieren dann die verantwortlichen *Worker*-Prozesse, welche die Anfragen parallel abarbeiten und die Ergebnisse mit Hilfe von Antwort-Objekten wiederum in den *Space* zurückgeben (**siehe Abb. 2**). Beim Web-Server werden Benachrichtigungen ausgelöst und das Ergebnis kann in eine entsprechende Benutzerschnittstelle verpackt zum Client geschickt werden.



KORREKTURFAHNE

KORREKTURFAHNE

Durch die Abwicklung der Kommunikation über den *Space* und die damit verbundene 100-prozentige Entkopplung skaliert diese Lösung sehr gut. Bei einer großen Anzahl an gleichzeitigen Anfragen kann die Last durch das Hinzufügen von weiteren Web-Server- und *Worker*-Rechnern beliebig erweitert werden, ohne dass teure Spezialhardware gekauft werden muss.

Jeder *Worker* entspricht einem Adapter zum jeweiligen Archiv. Kommt ein neues Archiv hinzu, muss nur dieser Adapter neu programmiert werden. Dieser kann dem laufenden System hinzugefügt werden, ohne die Anwendung zu stoppen. Diverse Industrieprojekte berichten mit Freude von dieser Fähigkeit, Komponenten während der Laufzeit auszutauschen.

Verteiltes Caching

Eine weitere Eigenschaft – dem *Space* nahezu inhärent – ist „automatisches" verteiltes Caching. Die *Worker*-Prozesse werden dabei so ausgebaut, dass sie eine Liste mit allen innerhalb eines gewissen Zeitraumes gestellten Anfragen mit entsprechenden Antwort-Objekten verwalten. Die Resultate (Antwort-Objekte) bleiben dabei einfach im *Space* stehen. Wiederholte Anfragen werden vom *Worker* identifiziert und das entsprechende Antwort-Objekt wird sofort zurückgegeben.

Bei Abwägung von Kosten und Nutzen ist es häufig schwierig, einen richtigen Lastausgleich in die Systeme einzubauen. Der *Space* bietet auch hier einen „natürlichen" Zugang. Jeder *Worker* weiß ja über die Auslastung seiner Maschine bescheid. Das heißt, ein *Space*-Proxy kann dafür sorgen, dass nur Anfragen entgegengenommen werden, wenn genügend Ressourcen zur Verfügung stehen. Die Last verteilt sich damit automatisch auf die *Worker*-Prozesse bzw. *Worker*-Maschinen, ohne dass dabei zusätzliche Informationen (die Informationen über die Last) übertragen werden müssen.

Die Vorteile einer *space*-basierten Lösung lassen sich wie folgt zusammenfassen:

- schnellere Entwicklungszeiten,
- hohe Skalierbarkeit durch Hinzufügen von günstiger Standard-Hardware,
- Entfernung der Last vom Web-Server und Verteilung auf die parallel arbeitenden *Worker*,
- Entkopplung der Schnittstellen,
- „automatisches" Caching und
- einfache Integration von heterogenen Landschaften.

Drei Viertel weniger Quellcode

Aktuelle Projekte zeigen – den ersten Punkt betreffend – erstaunliche Ergebnisse (vgl. [Sag01]). Bill Rawlings (CTO bei der Firma Lockheed Martin) berichtet, dass er die Menge an Quellcode gegenüber klassischen *3-Tier*-Projekten um mehr als drei Viertel reduziert (vgl. [Raw01]). Es liegt auf der Hand, welche Auswirkungen auf Dauer und Kosten der Projekte daraus erwachsen.

Im Wesentlichen sind dafür folgende Faktoren verantwortlich:

- *Blackboard-basierte Architektur:* Der *Space* bietet eine neue Art an Nachrichtenbus, der über die ganze (heterogene) Landschaft hinweg die gleiche Schnittstelle bietet. Prozesse werden angestoßen und gleichzeitig synchronisiert.
- *Anwendungsweite Live-Objekte:* Aus Gründen der Skalierbarkeit ist oft die Datenbank der einzige Ort, an dem Statusinformationen dauerhaft gehalten werden (*stateless programming*). Mit dem *Space* wird das Design sehr vereinfacht, da sowohl technische als auch anwendungsbezogene Klassen ihre Informationen einfach im *Space* ablegen können. Das Problem des *impedance mismatch* wird verringert.
- *Minimalistische Programmierschnittstelle:* Die Einfachheit der *Space*-Schnittstelle ist ein Schlüssel für dessen hervorragende Eigenschaften. Schnittstellenbeschreibungen (Stichwort „CORBA IDL") sind in einem *Space*-System nicht notwendig, da die Kommunikationspartner nichts voneinander wissen müssen – weder zur Übersetzungs- noch zur Laufzeit.

Die Hersteller

Sun hat mit den „JavaSpaces" (siehe [Fre99b] und [Fle01]) eine Referenzimplementierung vorgestellt, die auf Jini (siehe [Sin00]) aufbaut und einen rudimentären *Space* zur Verfügung stellt. Die beiden Hersteller GigaSpaces und IntaMission folgen diesem Standard. „TSpaces" von IBM ist ebenfalls in Java implementiert, geht aber eigene Wege. Diese Lösung zielt unter anderem darauf ab, Datenbankfunktionalität in den *Space* einzubauen. So wird eine einfache Abfragesprache angeboten, um eine große Anzahl an Objekten mit dem *Space* einfach verwalten zu können. TSpaces ist kein offizielles Produkt der IBM, befindet sich im Forschungsstatus, kann jedoch erworben werden. Die Firma Tecco aus Wien hat ebenfalls eine eigene Implementierung entwickelt, wobei das ursprüngliche <I>Tuple-Space<P>-Modell in etliche Richtungen weiterentwickelt wurde. So können beispielsweise neben der assoziativen Suche die Objekte im *Space* direkt referenziert werden, was zum einen besser skaliert und zum anderen einem *Garbage Collector* die Arbeit ermöglicht. „Corso" läuft nativ auf Windows, Unix und Linux und bietet Sprachanbindungen für Java, C/C++, Visual Basic und Oracle Forms. Der Laufzeit-Kernel umfasst weniger als ein MB.

Fazit

Die Schönheit der *Space*-Architektur besteht in ihrer Einfachheit und gleichzeitigen Mächtigkeit. Im Vergleich zu anderen Modellen für die verteilte Softwareentwicklung bietet dieser Ansatz in den meisten Fällen einfacheres Design, Reduzierung von Entwicklungs- und Debugging-Aufwand und resultiert in Anwendungen, die robuster und einfacher in der Wartung sowie Integration sind.

In Anbetracht der aktuellen Marketingstrategien von Sun & Co. bleibt freilich abzuwarten, wie lange es dauern wird, bis die J2EE-Gemeinde aufmerksam wird. Führt man sich die stark steigende Komplexität (Stichwörter „mobile" oder „P2P") zukünftiger Szenarien vor Augen und glaubt man an die Möglichkeit, dass der Informationsfluss zwischen Entwicklern und Entscheidern in manchen Firmen doch besser funktioniert als vermutet, dann könnte sich das Bild in absehbarer Zeit radikal ändern. ■

Literatur & Links

[Fle01] R. Flenner, Jini and Javaspaces Application Development, Sams 2001

[Fre99a] E. Freeman. S. Hupfer, Make room for JavaSpaces – Part 1, JavaWorld 1999 (siehe http://www.javaworld.com)

[Fre99b] E. Freeman, S. Hupfer, K. Arnold, JavaSpaces Principles, Patterns and Practice, Addison-Wesley 1999

[Fre99c] E. Freeman, Make room for JavaSpaces – Part 2, JavaWorld 1999 (siehe http://www.javaworld.com)

[Gel92] D. Gelernter, Mirrorworlds, Oxford University Press, 1992

[Raw01] B. Rawlings – M. Lockheed, JavaSpace Mailinglist, Item 003081 und 001588 (siehe http://archives.java.sun.com/archives/javaspaces-users.html)

[Rup02] Ruple White Paper – A Loosely Coupled Architecture Ideal for the Internet, RogueWave Software 2002

[Sag01] D. Sag, Jini as an Enterprise Solution, O'Reilly Network 2001

[Sin00] L. Sing, Professional Jini, Wrox Press 200

KORREKTURFAHNE