Seminar aus Informatik
Institut für Informationssysteme

Koordination verteilter Applikationen

Passiv vs. Aktiv: NSTP und PageSpace

1998-12-01

Markus Mazanec
Technische Universität Wien

Kurzfassung

Das Notification Service Transfer Protocol und PageSpace stellen zwei grundverschiedene Ansätze dar, wie verteilte Applikationen koordiniert werden können. NSTP ermöglicht synchrone Zusammenarbeit basierend auf einer zentralen Zustandsverwaltung. PageSpace verfolgt die Idee aktiver Agenten, die autonom und ortsunabhängig interagieren. Beide Varianten besitzen charakteristische Merkmale, die sie für unterschiedliche Verwendungszwecke prädestinieren.

1 Einleitung

Im Bereich der Koordination verteilter Applikationen in Netzwerken konnte sich noch kein Standard etablieren. Wirft man einen genaueren Blick auf die Landschaft vorhandener Entwürfe, kann man zwei Designrichtungen deutlich unterscheiden: passive und aktive Systeme. Während der Ansatz der passiven Systeme in die Richtung geht, Daten entgegenzunehmen und verteilt zur Verfügung zu stellen, sollen Teile eines aktiven Systems selbst applikationsspezifische Verarbeitungsaufgaben übernehmen. Im folgenden sollen typische Vertreter beide Konzepte illustrieren (NSTP als passiver Ansatz, PageSpace aktiv). Abschließend werden wesentliche Unterscheidungsmerkmale angeführt.

2 Beschreibung der Systeme

2.1 NSTP: Notification Service Transfer Protocol

2.1.1 Allgemeine Beschreibung

Das Notification Service Transfer Protocol beschreibt eine Infrastruktur, die es Applikationen erlaubt, in einem Netzwerk synchron zusammenzuarbeiten. Verglichen mit herkömmlichen Systemen mit dezentraler Semantik, wählten die Autoren von NSTP einen ungewöhnlichen Ansatz, für den Austausch von Daten bzw. die Kommunikation der Clients untereinander: Daten, die allen Clients zugänglich sein sollen, werden auf einem zentralen Server verwaltet. Clients können diese Daten lesen, ändern und hinzufügen. Über Änderungen der zentral abgelegten Daten

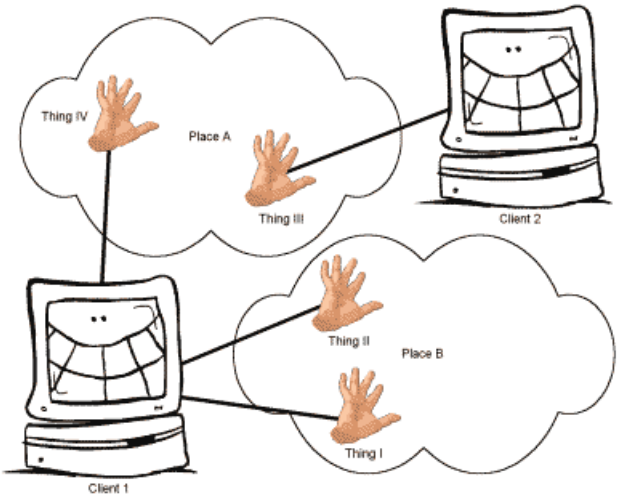
werden interessierte Clients vom Server informiert. Das Design sieht keine Integration bestehender Architekturen vor und ist von Programmiermodellen völlig unabhängig.

2.1.2 Modell: Things und Places

Die zentralen Konzepte in NSTP sind Things und Places.

Jedes Thing besitzt Attribute (attributes) und einen Wert (value). Die Attribute eines Things dienen der Identifikation inklusive Typbestimmung, beeinflussen die Behandlung des Thing im Server und sind nicht änderbar. Der Wegfall einer nachträglichen Änderungsmöglichkeit reduziert die Serverkomplexität und erspart die Implementierung eines fehleranfälligen Benachrichtigungsmechanismus für die Attributverwaltung. Der Wert eines Things beinhaltet die für den Client relevanten Daten. Things können jede Art applikationsspezifischer Daten aufnehmen, wobei diese Informationen auch lediglich der Kommunikation der Clients untereinander dienen kann und keine Benutzerdaten enthalten muß. Es besteht auch die Möglichkeit, einen Zustand in mehrere Things abzubilden. Werte werden vom Server lediglich als Bytemengen behandelt und nicht semantisch interpretiert. Ein Sperrmechanismus erlaubt den exklusiven Zugriff auf einen Wert und ermöglicht konsistente Verarbeitungsketten.

Places sind Container für Things. Da Things in unterschiedlichen Places den gleichen Namen haben können, ist eine eindeutige Identifikation grundsätzlich nur über Places möglich. Ein Client kann sich in mehreren Places gleichzeitig aufhalten. Der Server informiert Clients, falls sich der Wert eines Things ändert. Places stellen einen Zugriffskontrollmechanismus dar. Sie enthalten Informationen darüber, welche Clients Things ablegen dürfen und welche Things außerhalb des Place, dh. für Clients, die kein Thing in dem Place deponiert haben, sichtbar oder sogar beschreibbar sind.



Visualisierung zweier NSTP-Places

Things werden auch für interne Verwaltungsmechanismen wiederverwendet (predefined things). Zusätzliche Mechanismen, wie z.B. Kontrollnachrichten, sind somit nicht erforderlich. Über ein (typisch) oder mehrere Things wird das Erscheinungsbild (facade) eines Place gesteuert. Diese ausgezeichneten Things

enthalten die erforderlichen Daten, wie die Beschreibung des Place, eine URL wo für die Behandlung des Place erforderlicher Programmcode zu finden ist, oder den Programmcode selbst. Java und insbesondere JavaBeans bieten sich aufgrund ihrer Plattformunabhängigkeit für diese Art des "dynamic client enhancement" besonders an.

Places werden erstellt wenn ein Client dies beim Server anfordert. Dabei wird ein initiales Thing erstellt, um das herum der neue Place aufgebaut wird. Wann ein Place wieder zerstört wird, bleibt den Clients überlassen. Dafür sind mehrere Mechanismen vorgesehen, wobei "never" (der Place bleibt bis zur Client-initierten Zerstörung erhalten) und "when last user leaves" (der Place bleibt erhalten, solange sich Clients darin aufhalten) zu den wichtigsten zählen.

Die Anmeldung eines Clients an einem Place erfolgt über eine INIT-Nachricht, auf die der Server mit OK antwortet, die Anmeldung zurückweist, oder weitere Informationen anfordert. Für den Fall, daß weitere Informationen angefordert werden, kann sich ein längerer INIT-RESPONSE Dialog entwickeln. In diesem Fall handelt es sich dann um einen "proprietären" Authorisierungs- / Authentifizierungsmechanismus, dessen Ablauf von den jeweiligen Entwicklern definiert werden muß. Den Minimalfall stellt ein INIT-OK oder INIT-REJECT-Paar dar, das Anmeldeverfahren ist aber flexibel und kann an spezielle Anforderungen angepaßt werden.

Betritt ein Client einen existierenden Place (client enters place), wird ein "predefined thing", das "user thing", angelegt, das für die Verweildauer (client in place) bis zum Verlassen des Place (client leaves place) erhalten bleibt.

Things und Places sind nicht persistent. Ein Hardwaredefekt oder Softwarefehler führt dazu, daß Things und Places verloren gehen und unter Umständen nicht rekonstruiert werden können. Eine besondere Behandlung von Kommunikationsfehlern ist nicht erforderlich, da das Schließen einer Verbindung zum Client mit einer regulären Abmeldung gleichgesetzt wird.

Daten mit Echtzeitanforderung, wie z.B. Streaming Audio oder Video, kann ein NSTP-Server nicht behandeln. Derartige Aufgaben lassen sich über eigene Mechanismen abwickeln. Anstelle der zeitkritischen Daten wird nur ein Verweis auf einen dafür spezialisierten Server abgelegt.

2.1.3 Zugrundeliegende Designprinzipien

Beim Design von NSTP lag das Hauptaugenmerk darauf, das Protokoll so einfach und flexibel wie möglich zu halten. Damit der Server, auf dem die zentrale Zustandsverwaltung erfolgt, für eine möglichst breite Basis unterschiedlicher Anwendung eingesetzt werden kann, wurde auf komplexe und spezialisierte Mechanismen verzichtet ("keep it simple").

Nachrichten sind lokal (sie werden allen Clients mit Things innerhalb des Place zugestellt) und symmetrisch (der sendende Client erhält seine abgesendeten Nachrichten ebenfalls zugestellt).

Nachdem ein Client die Antwort auf eine Anfrage erhalten hat, kann er mit der

Verarbeitung fortfahren, ohne auf eine weitere Benachrichtigungen warten zu müssen.

Verarbeitungsschritte werden soweit wie möglich den Clients übertragen (work offloading), sodaß der Server ausschließlich für das Placemanagement zuständig ist. Clients übernehmen aber keine Serverfunktionalität.

Die Reduktion der Verarbeitung am Server und eingeschränkte Komplexität des Protokolls erleichtert die Behandlung später hinzukommender Clients und die Realisierung des Sperrmechanismus. Ausschließlich der Wert eines Things ist änderbar. Veränderliche Attribute würden eine Überwachung durch den Server und Benachrichtigungen bei auftretenden Modifikationen erforderlich machen. Da Things nicht interpretiert werden, ist eine Adaption des Servers für neue Applikationen nicht erforderlich.

NSTP kennt keine Multi-Place Operationen, welche den Benachrichtigungsmechanismus deutlich komplexer ausfallen lassen würden. Allerdings existieren atomare Multi-Thing Operationen, deren Gültigkeitsbereich auf einen Place beschränkt ist.

2.2 PageSpace

2.2.1 Allgemeine Beschreibung

PageSpace ist eine Plattform für die Unterstützung verteilter Aktivitäten (Applikationen) im World Wide Web. Das Modell integriert vorhandene Technologien zu einem Konzept, dessen integraler Bestandteil die Ausführung aktiver Inhalte (Agenten) ist. Gleichzeitig bezeichnet man mit PageSpace auch das Kollektiv agierender Agenten und die Umgebung in der diese Aktionen stattfinden. Die jeweils anzuwendende Bedeutung des Wortes ergibt sich im folgenden aus dem Kontext.

Agenten arbeiten auf verschiedenen Servern im World Wide Web verteilt und kommunizieren untereinander. Der Benutzer kann über einen darauf spezialisierten Agenten Kontakt mit anderen Agenten im PageSpace aufnehmen, diesen Aufträge erteilen, Informationen einholen und ablegen.

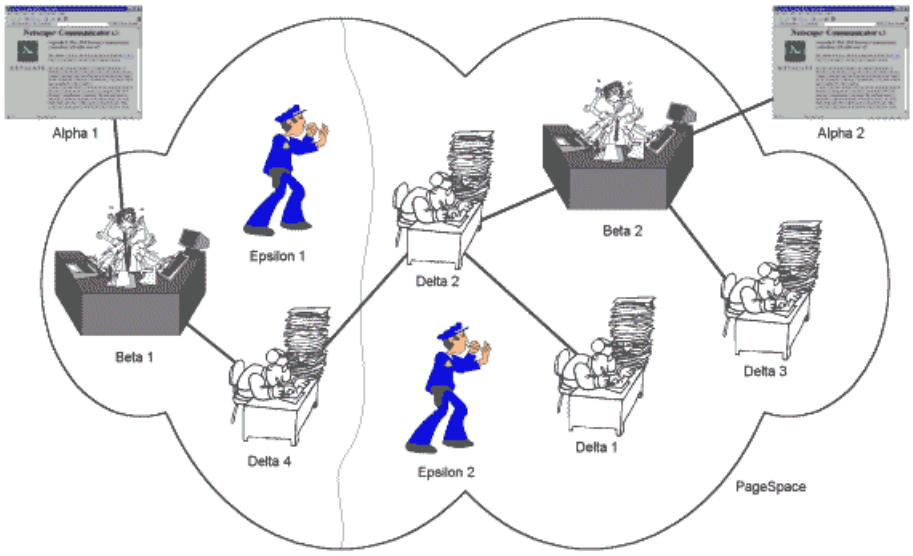
2.2.2 Modell: Agents, Environments und Gateways

PageSpace kennt keine zentrale Koordinationsinstanz. Verteilte PageSpace-Applikationen laufen und kommunizieren untereinander auf unterschiedlichen Prozessor-, Netzwerk- und Betriebssystemarchitekturen. Transparenz der Netzwerkarchitektur wird durch die Nutzung vorhandener Internettechnologie erreicht. Für die Kommunikation wird HTTP verwendet, Java ermöglicht eine hardware- und betriebssystemunabhängige Programmierung. Die vorgesehenen Koordinationsmechanismen nehmen Anleihe bei darauf spezialisierten und etablierten Sprachen (Linda, Laura und Shade).

Beginn- und Endzeitpunkte einer Zusammenarbeit sind PageSpace unbekannt. Agenten können dem PageSpace zu jedem beliebigen Zeitpunkt beitreten oder diesen verlassen.

PageSpace unterscheidet verschiedene Arten von Agenten. Ein Agent ist eine aktive Entität, die auf einem Server ("im Netz") ausgeführt wird. Agenten nutzen Dienste anderer Agenten und stellen anderen Agenten Dienste zur Verfügung. Jede Art von Agenten ist auf einen speziellen Aufgabenbereich spezialisiert. Dadurch wird es z.B. möglich, daß Benutzer nicht permanent mit dem PageSpace verbunden sein müssen, um Aufgaben abarbeiten zu lassen.

Alpha interfaces (oder Agenten) und Beta Agenten verbinden den Benutzer mit dem PageSpace. Alphas stellen das Interface dar, über das der Benutzer mit seinem Beta innerhalb des PageSpace kommunizieren kann.
Delta Agenten sind aktive Prozesse, die Verarbeitungsaufgaben übernehmen. Sie können als "Bewohner" des PageSpace gesehen werden. Deltas können sich innerhalb des PageSpace, der über mehrere Server verteilt sein kann, bewegen.
Gamma Environments (oder Agenten) übernehmen Koordinations- und Synchronisationsfunktionen.
Epsilon Agenten liefern Deltas Informationen über ihren Aufenthaltsort und ihre Rechte (wie z.B. die Möglichkeit Deltas zu kreiern). Auf jedem Server läuft genau ein Epsilon. Epsilons haben das Recht Deltas zu starten oder zu stoppen. Sie können Deltas von Server zu Server transportieren. Um Deltas nach einem Systemfehler wieder zum Laufen bringen zu können, können Epsilons Checkpoints von Deltas aufbewahren. In der Standardkonfiguration ist der Checkpointmechanismus allerdings nicht aktiviert.
Zeta gateways (oder Agenten) ermöglichen anderen Agenten den Zugriff auf externe Daten (Daten außerhalb des PageSpace). So kann zB. ein Zeta Agenten zum Einsatz kommen, der Zugriff auf CORBA-Objekte nehmen kann und diese Daten anderen Agenten zur Verfügung stellt.



Visualisierung des PageSpace

Es gibt mehrere Möglichkeiten, wie ein Delta von einem Server zu einem anderen gelangen kann. Deltas können eine diesbezügliche Anfrage an ihren Epsilon Agenten stellen, ein dritter Agent wünscht den Transfer (unverbindlich) oder ein anderer Agent mit entsprechenden Befugnissen (Epsilon) befiehlt den Ortswechsel. Die Übertragung wird von Epsilons durchgeführt (der Epsilon Agent der Ausgangsmaschine überträgt Delta zum Epsilon der Zielmaschine), die den

gesamten PageSpace maschinenabhängig adressieren können und auf den Code von Deltas Zugriff haben.

2.2.3 Anforderungen an das zugrundeliegende Design

- Alle Agenten haben eine durchgängige Zugriffsmöglichkeit auf den gemeinsamen Informationsbereich.
- Agenten können sich gegenseitig aurufen und Daten indirekt über den gemeinsamen Informationsbereich austauschen.
- Dialoge können lang und komplex ausfallen.
- Agenten haben Zustände und sind persistente Objekte.

2.2.4 Fehlertoleranz

Da PageSpace Persistenz kennt, kommt dem Thema Fehlertoleranz besondere Bedeutung zu. Im folgenden sei kurz auf die Auswirkungen von Fehlern der beteiligten Entitäten eingegangen.

Fehler von Alphas beeinflussen Betas und andere Agenten nicht. Nach dem erneuten Start des Interfaces durch den User kann erneut Kontakt mit dem zuständigen Beta aufgenommen werden.

Eine Störung eines Servers hat zur Folge, daß die Zustände aller Agenten auf diesem Server verloren gehen. Beim Hochfahren des Systems wird der Epsilon neu gestartet, der Deltas aufgrund eventuell vorhandener Checkpoints zu neuem Leben erwecken kann. Wird dieser Mechanismus verwendet, muß sichergestellt werden, daß es zu keinen Inkonsistenten mit Agenten auf einem anderen Server kommt.

Das Kommunikationssystem muß sicherstellen, daß keine Nachrichten verloren gehen.

Fehlerhaftes Verhalten von Betas und Deltas kann sich auf den verursachenden Agenten und die mit ihm in Kontakt stehenden Agenten beschränken, aber auch zu einem Totalausfall des Servers führen.

3 Gegenüberstellung

Ob der passive oder der aktive Ansatz für eine konkrete Implementierung weiter verfolgt werden soll, muß von Fall zu Fall entschieden werden. Die skizzierten Lösungsansätze lassen deutlich die unterschiedlichen Möglichkeiten und Grenzen erkennen. Einige zentrale Aspekte seien hier herausgestrichen:

Komplexität: PageSpace weist konzeptuell eine wesentlich höhere Komplexität auf als NSTP, ist leistungsfähiger aber anfälliger für Implementierungsfehler.

Synchronität: Während NSTP ausschließlich für synchrone Kommunikation geeignet ist, kann im PageSpace auch asynchron zusammengearbeitet werden.

Persistenz/Zustandssicherheit: NSTP kennt keinerlei Persistenz, während in PageSpace ein Checkpointmechanismus vorgesehen wurde.

Sicherheit: Mobile Agenten bergen hohe und z.T. noch ungelöste Sicherheitsrisiken in sich. In NSTP existieren keine aktiven Inhalte, daher können Sicherheitsüberprüfungen vergleichsweise einfach im Server implementiert werden.

Performance/Skalierbarkeit: Der Ansatz von NSTP geht in die Richtung, das Protokoll so einfach wie möglich zu gestalten, um auch höhere Clientzahlen zufriedenstellend realisieren zu können. Dennoch ist durch die zentrale Zustandsverwaltung eine obere Grenze der Belastbarkeit gegeben. Der PageSpace kann um Server erweitert werden, bei Performanceengpässen können Epsilons Deltas auf weniger belastete Server transferieren. Der Ressourcenbedarf ist aufgrund der aktiven Objekte wesentlich höher als bei NSTP.

4 Abschließende Bemerkungen

NSTP und PageSpace lösen das Problem der Koordination verteilter Applikationen ausgehend von völlig unterschiedlichen Ansätzen. Während NSTP den Weg einer zentralen Zustandsverwaltung geht und das Hauptziel lautet, die Gesamtkomplexität möglichst gering zu halten, um Applikationen eine flexible, standardisierte Basis für die Zusammenarbeit zu bieten, setzt PageSpace auf aktive Objekte, die eine Reihe von Herausforderungen in sich bergen. Diesen begegnet man mit etablierten Mechanismen, von denen Erfahrungswerte vorliegen. Beide Ansätze verfügen über unterschiedliche Möglichkeiten und Beschränkungen. Für viele Anwendungen verteilter Applikationen dürfte PageSpace einen Overkill darstellen, der zudem einen wesentlich größerer Entwicklungsaufwand mit sich bringt. Der Einsatz von PageSpace macht bei Anwendungen, die autonom agierende Agenten als wesentlichen Bestandteil beinhalten, Sinn.

5 Literatur

Day, M., J.F. Patterson and D. Mitchell, The Notification Service Transfer Protocol (NSTP), Computer Networks and ISDN Systems 29(9): 905-915, September 1997

Ciancarini, P., A. Knoche, R. Tolksdorf, and F. Vitali, PageSpace: an architecture to coordinate distributed applications on the Web: Proc. 5th International World Wide Web Conference, http://www5conf.inria.fr/fich_html/papers/P5/Overview.html

Fielding, R., U.C. Irvine, J. Gettys, J.C. Mogul, H. Frystyk, T. Berners-Lee, Hypertext Transfer Protocol - HTTP/1.1, Proposed Standard RFC 2068, http://www.w3.org/Protocols/rfc2068/rfc2068

Frivold, T.J., R.E. Lang, and M.W. Fong, Extending WWW for synchronous collaboration, 2nd World Wide Web Conference '94: Mosaic and the Web, http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/CSCW/frivold/frivold.html

Liskov, B., A. Adya, M. Castro, D. Curtis, M. Day, R. Gruber, S. Ghemawat, A.C. Myers, and L. Srira, Safe and efficient sharing of persistent objects in Thor, ACM 1996 Conference on Management of Data (SIGMOD),

ftp://ftp.pmg.lcs.mit.edu/pub/thor/thor.ps

Patterson, J.F., M. Day, and J. Kucan, Notification servers for synchronous groupware, ACM 1996 Conference on Computer Supported Cooperative Work, pp. 122-129, http://www.lotus.com/lotus/research.nsf/

Roseman, M. and S. Greenberg. Building real time groupware with Groupkit, a groupware toolkit, ACM Transactions on Computer Human Interaction, March 1996, http://www.cpsc.ucalgary.ca/projects/grouplab/papers/

Leupold, M. und M. Mazanec, JavaBeans, http://www.service.at/javabeans/