2002-08-30 Kurzfassung GI Fachgruppentreffen Betriebsysteme im Herbst 2002

AspectIX: eine Middleware-Architektur zur Unterstützung nichtfunktionaler Eigenschaften verteilter Anwendungen

Franz J. Hauck[1], Rüdiger Kapitza[2], Hans Reiser[2], Andreas Schmied[1]

[1]*Abteilung Verteilte Systeme, Universität Ulm*

[2]*Lehrstuhl für Informatik 4, Universität Erlangen-Nürnberg*

Middleware-Systeme unterstützen die Entwicklung und den Betrieb verteilter Anwendungen. Dazu wird in der Regel ein Programmiermodell, verschiedene Entwicklungswerkzeuge und ein Laufzeitsystem bereitgestellt. Die Unterstützung nichtfunktionaler Eigenschaften verteilter Anwendungen muss sich in allen Middleware-Bestandteilen niederschlagen. Bei heutigen Systemen findet man hauptsächlich dedizierte Unterstützung verschiedener Eigenschaften, z.B. für Fehlertoleranz. *AspectIX* dagegen ist eine CORBA-konforme Middleware-Architektur, die generische Unterstützung für die meisten nichtfunktionalen Eigenschaften anbietet und damit nicht für neue und künftige Anforderungen erweitert werden muss. *AspectIX* basiert auf so genannten fragmentierten Objekten, einer Reihe von Middleware-Diensten und einem flexiblen Entwicklungswerkzeug.

1 Einführung

Verteilte Anwendungen werden heute meist auf der Basis von Middleware-Systemen entwickelt und betrieben. Middleware-Systeme bieten ein Programmiermodell an, in dem die Anwendung entwickelt wird (z.B. basierend auf verteilten Objekten). Für die Entwicklung der Anwendung werden Werkzeuge bereitgestellt (z.B. Stubgeneratoren). Die Anwendungsentwicklung wird erleichtert, dadurch dass die anwendungsinterne Kommunikation über Verteilungsgrenzen hinweg weitgehend transparent bleibt oder zumindest automatisch erzeugt wird. Während des Anwendungslaufs sorgt die Middleware für die Bereitstellung von Diensten für Benennung und Kommunikation (z.B. Namensdienste und eindeutige Identifikatoren).

Soll die Anwendung eine oder mehrere nichtfunktionale Eigenschaften besitzen, z.B. fehlertolerant gegenüber Rechnerausfällen sein, dann muss die Middleware entsprechend erweitert werden, da die nichtfunktionale Eigenschaft in der Regel die von der Middleware beeinflussten oder gar vollständig von ihr bereitgestellten Kommunikationsmechanismen betrifft. Im Falle von Fehlertoleranz werden redundante Anwendungsteile entwickelt, die als Ganzes adressiert und angesprochen werden müssen. Heutige Systeme, wie beispielsweise CORBA-basierte Systeme, tragen dem Rechnung, dadurch dass das Programmiermodell dediziert erweitert und spezielle Werkzeuge und Dienste angeboten werden. Der CORBA-Standard wurde daher für Fehlertoleranz und Echtzeitfähigkeit erweitert [3 (Kap. 23), 4].

Wir propagieren einige wenige Erweiterungen im Programmiermodell von CORBA und erreichen damit eine generische Unterstützung für eine Vielzahl nichtfunktionaler Eigenschaften, ohne dass das Programmiermodell der Middleware für neue nichtfunktionale Eigenschaften erweitert werden müsste. Durch drei generische Dienste können wiederum viele nichtfunktionale Eigenschaften zur Laufzeit unterstützt werden. Ein flexibles Entwicklungswerkzeug zur Code-Generierung muss für jede nichtfunktionale Eigenschaft und für Kombinationen solcher Eigenschaften speziell angepasst werden, kann dann aber den Entwick-

lungsprozess entscheidend vereinfachen und beschleunigen. Die von uns *AspectIX* getaufte Middleware-Architektur bleibt mit CORBA-Systemen interoperabel und wird zur Zeit in Java realisiert.

2 AspectIX Systemarchitektur

Das Programmiermodell der *AspectIX*-Middleware [1] orientiert sich an CORBA [3]. CORBA kennt verteilte Objekte, an deren Schnittstelle von überall her in einer verteilten Anwendung eine Operation aufgerufen werden kann. Die Schnittstelle wird in einer speziellen Sprache (IDL) beschrieben. Das verteilte Objekt kann in verschiedenen Programmiersprachen realisiert werden. In CORBA liegt die Objektimplementierung (der so genannte Servant) an einem bestimmten Ort. Von entfernten Orten wird ein Operationsaufruf durch lokale Stellvertreterobjekte vermittelt d.h. in Nachrichten umgewandelt, die am eigentlichen Aufenthaltsort für die Ausführung der Operation sorgen. Ergebnisse werden in die andere Richtung übertragen. Die ausgetauschten Nachrichten entsprechen denen eines entfernten Prozeduraufrufs (RPC). Die lokalen Stellvertreterobjekte (Stubs genannt) werden automatisch durch die CORBA-Middleware aus IDL erzeugt und instanziiert, sobald eine entfernte Objektreferenz lokal bekannt wird.

Soll ein verteiltes Objekt nichtfunktionale Eigenschaften besitzen, so steht die automatische Instanziierung von Stubs und die Kommunikation über RPC dem meist entgegen. Soll beispielsweise Fehlertoleranz über Replikation realisiert werden, so sollte ein repliziertes Objekt wie ein nicht repliziertes auftreten. Im Fault-Tolerant-CORBA Standard werden daher Gruppenadressen zur Adressierung mehrerer Einzelobjekte eingeführt und dem Middleware-Entwicklern die Möglichkeit gegeben, Multicast-Protokolle für die Kommunikation einzusetzen.

AspectIX dagegen führt so genannte fragmentierte Objekte ein, bei denen auch die lokalen Stellvertreter als Teil des verteilten Objekts betrachtet werden. Welche Stellvertreter instanziiert werden wird dem Objekt und damit letztlich dem Objektentwickler überlassen. In der Objektreferenz wird codiert, welche Implementierung der lokale Stellvertreter haben soll. Diese wird dann, falls notwendig, beim Austausch von Objektreferenzen über Rechnergrenzen hinweg automatisch geladen.

Somit kann der lokale Stellvertreter Teile der Objektimplementierung mit sich tragen, z.B. ein Replikat oder einen intelligenten Stellvertreter bilden, der mit mehreren anderen Replikaten kommunizieren kann. Aus diesem Grund wird der Stellvertreter ein Fragment genannt. Fragmente werden gleichwertig mit der ursprünglichen Servantimplementierung behandelt und können miteinander komplexe verteilte Objekt bilden. Denkbar sind

- Replikatfragmente und intelligente Stellvertreterfragmente (Fehlertoleranz und Skalierbarkeit),
- Serverfragment und spezielle Stellvertreterfragmete für Multimediadaten-Kommunikation (Rechtzeitigkeit, Bandbreitenreservierung usw.),
- hierarchisch angeordnete Serverfragmente mit Datenpartitionierung (Skalierbarkeit),
- Server- und Stellvertreterfragmente mit verschlüsselter Datenkommunikation (Sicherheit),
- Agenten- und Stellvertreterfragmente für mobile Agentensysteme (Mobilität),

und einige mehr. Schließlich könnten Internetdienste, die heute als viele einzelne Serversysteme gesehen werden, konzeptionell zu einem verteilten fragmentierten Objekt zusammengefasst werden, z.B. DNS, WWW oder E-mail.

Da die Objektreferenz im Gegensatz zu CORBA nicht in jedem Fall die reale Kommunikationsadresse eines zentralen Kommunikationspartners (des Servants) enthalten kann, sind in *AspectIX* verschiedene Varianten von Objektreferenzen vorgesehen:

- *Standard CORBA:* die Objektreferenz enthält eine Kommunikationsadresse, mit der ein RPC-basierter Operationsaufruf an einem zentralen Serverfragment durchgeführt werden kann.
- *Client-Server:* die Objektreferenz enthält eine Kommunikationsadresse, mit der ein zentrales Serverfragment angesprochen werden kann, jedoch nicht notwendigerweise mit einem RPC. Stattdessen sind Echtzeit-, Multimediaprotokolle usw. möglich.
- *Peer-to-Peer:* die Objektreferenz enthält lediglich eine eindeutige Objektidentifikation, über die bei einem externen Ortsdienst eine oder mehrere Kommunikationsadressen von zu dem Objekt gehörigen Fragmenten erfragt werden können. Die Kommunikationsadressen können zeitlich veränderlich sein und damit mobile Fragmente abbilden.

Ähnlich wird zwischen verschiedenen Varianten der Objektreferenz unterschieden, die festlegen, auf welche Weise die Implementierung des lokalen Fragments ermittelt wird:

- *Standard CORBA:* es wird ein CORBA-konformer Stellvertreter instanziiert.
- *Statisch:* die Objektreferenz enthält eine explizite Angabe der Implementierung, z.B. einen Klassennamen bzw. einen eindeutigen Bezeichner für einen Implementierungstyp. Letzterer wird dann von einem so genannten Dynamic-Loading-Service in eine lokale Implementierung umgewandelt. Dies hat den Vorteil, dass Implementierungen in verschiedenen Sprachen geschrieben sein können und die jeweils passende Variante automatisch ausgewählt wird.
- *Dynamisch:* die Objektreferenz enthält lediglich einen eindeutigen Entscheidungsbezeichner, der von einem externen Entscheidungsdienst in einen Implementierungstyp aufgelöst werden kann. Der Entscheidungsdienst (vgl. [2]) kann dann lokale Eigenschaften (z.B. Last- und Leistungsparameter, Vertraulichkeit usw.) berücksichtigen und eine geeignete lokale Implementierung auswählen.

In der Praxis werden die genannten Parameter in der CORBA IOR (Interoperable Object Reference) in Form spezieller Profile codiert, die es bei entsprechender Vorkehrung auch Standard-CORBA-Systemen ermöglichen, mit einem *AspectIX*-Objekt zu interagieren.

3 Anwendungsentwicklung

Anwendungsentwicklern ist es nun möglich, in einem verteilten Objekt vollständig die Implementierung nichtfunktionaler Eigenschaften zu kapseln. Die jeweilige Middleware entfernter Objektbenutzer muss lediglich die Instanziierung lokaler Fragmente unterstützen. Anwendungsentwickler entwerfen unter Umständen mehrere Fragmenttypen für einen Objekttyp und entscheiden wie diese miteinander interagieren können. Bei der Objektinstanziierung ist dafür zu sorgen, dass die Implementierungen der Fragmente entweder jeweils lokal vorliegen oder dynamisch ladbar sind. Der Entscheidungsdienst und der Dynamic-Loader-Service sind entsprechend zu initialisieren. Der Ortsdienst im Falle von mehreren miteinander kommunizierenden Fragmenten oder bei mobilen Fragmenten kann dagegen von den Fragmenten selbst initialisiert werden.

Um die Anwendungsentwicklung für einzelne nichtfunktionale Eigenschaften zu unterstützen stellt *AspectIX* das Application-Developer-Kit (ADK) bereit. Hierbei handelt es sich um ein Source-Code-Transformationswerkzeug auf der Basis von Java. Das ADK subsummiert die Funktion des Stubgenerators von Standard-CORBA-Systemen. Neben der Schnittstellenbeschreibung in IDL kann eine Java-basierte Objektimplementierung eingespeist werden. Nach Auswahl vorkonfigurierter Varianten wandelt das ADK die Objektimplementierung in eine Reihe von Fragmenttypen um, die zusätzlich eine bestimmte Menge nichtfunktionaler Eigenschaften realisiert. Das ADK kann aber auch einfache CORBA-konforme Stubs generieren.

In einem ersten Experiment wurde eine nicht fehlertolerante, zentrale Objektimplementierung in drei verschieden Fragmenttypen konvertiert: intelligenter Stellvertreter, intelligenter Stellvertreter mit Datencache und schließlich ein Replikatfragmenttyp. Der Transformationsprozess wurde einmalig beschrieben und kann auf alle Objektimplementierungen angewandt werden, die bestimmte Voraussetzungen erfüllen. So muss es beispielsweise möglich sein, transiente von persistenten Objektdaten zu unterscheiden und zu benennen, um den Zustandsaustausch bei Replikation zu steuern.

Der Transformationsprozess selbst ist modular aufgebaut und kann teilweise wiederverwendet werden. Die Komposition einzelner Transformationsprozesse zu Kombinationen, welche eine Menge von nichtfunktionalen Eigenschaften beisteuern, ist Gegenstand der Forschung.

4 Zusammenfassung

AspectIX führt das Konzept der Fragmente ein, bei dem lokale Stellvertreter je nach Anforderung des Objektentwicklers bzw. des Objekts instanziiert werden. Dies geschieht für einen Objektnutzer völlig transparent und gewährt dem verteilten Objekt einen lokalen Implementierungsanteil. Dieser kann dann zusammen mit den anderen verteilten Fragmenten desselben Objekts viele verschiedene nichtfunktionale Eigenschaften realisieren.

Die Entwicklung von Fragmenttypen eines Objekttyps wird durch ein Entwicklungswerkzeug vereinfacht. Die Umwandlung einer zentralen Objektimplementierung ohne nichtfunktionale Eigenschaften kann durch einen einmalig beschriebenen Transformationsprozess automatisch durchgeführt werden.

5 Referenzen

[1] F. Hauck, U. Becker, M. Geier, E. Meier, U. Rastofer, M. Steckermeier: AspectIX: a quality-aware, object-based middleware architecture. *Proc. of the 3rd IFIP Int. Conf. on Distrib. Appl. and Interop. Sys.* – DAIS (Krakau, 17–19. Sep. 2001), Kluver, 2001.

[2] F. Hauck, E. Meier, U. Becker, M. Geier, U. Rastofer, M. Steckermeier: A middleware architecture for scalable, QoS-aware and self-organizing global services. *Proc. of the 3rd IFIP/GI Int. Conf. on Trends towards a universal service market* – USM (München, 12–14. Sep. 2000), LNCS 1890, Springer, 2000. pp. 214–229.

[3] Object Mgmt. Group: *The Common Object Request Broker, architecture and specification.* Ver. 3.0, OMG Doc. formal/02-06-01. Framingham, Mass., Juli 2002.

[4] Object Mgmt. Group: *Real-Time CORBA.* Ver. 1,1, OMG Doc. formal/02-08-02, Framingham, Mass., Aug. 2002.