



Universität Ulm

Fakultät für Informatik
Verteilte Systeme

Flexible und adaptive Replikation in verteilter objektbasierter Middleware

Technischer Bericht

Hans P. Reiser¹
Rüdiger Kapitza²
Jörg Domaschka¹
Franz J. Hauck¹

¹ Abteilung Verteilte Systeme, Universität Ulm

² Informatik 4, Universität Erlangen-Nürnberg

Nr. VS-R06-2006

27.01.2006

Flexible und adaptive Replikation in verteilter objektbasierter Middleware

Hans P. Reiser¹, Rüdiger Kapitza², Jörg Domaschka² und Franz J. Hauck¹

¹ Abteilung Verteilte Systeme, Universität Ulm

{hans.reiser, franz.hauck}@uni-ulm.de

² Lehrstuhl für Verteilte Systeme und Betriebssysteme, Universität Erlangen-Nürnberg

{kapitza, sijodoma}@informatik.uni-erlangen.de

Zusammenfassung. Verteilte objektbasierte Middleware bietet oft nur unzureichende Unterstützung für fehlertolerante Replikation und ist den Herausforderungen von dynamischen, selbstorganisierenden Systemen nicht gewachsen. In der AspectIX-Middleware wurden basierend auf fragmentierten Objekten Konzepte entwickelt, um Objektreplikation zu vereinfachen und flexibel zu gestalten. Bei der Entwicklung von replizierten Objekten unterstützt ein Werkzeug für automatische Codegenerierung die Optimierung der Konsistenzmechanismen auf Grund von semantischen Annotationen. Innerhalb der Objektimplementierung wird dabei auch Unterstützung für die konsistente Behandlung von Nichtdeterminismus, Multithreading und externen Interaktionen geboten. Im Vergleich zu bestehenden, meist starren und inflexiblen Ansätzen wird es gestattet, eingesetzte Mechanismen über einen weiten Bereich bedarfsgerecht auszuwählen. Dabei wird für alle angebotenen Varianten eine dynamische Rekonfiguration zur Laufzeit unterstützt; ebenso werden Mechanismen zur autonomen Selbstoptimierung angeboten.

1 Überblick

Die Entwicklung fehlertoleranter Anwendungen in verteilten Systemen ist eine komplexe Aufgabe. Die Unterstützung des Entwicklers durch Middleware-Funktionalität ist daher erstrebenswert. Es existieren verschiedene Ansätze zur Integration von Replikationsmechanismen in vorhandene Middleware-Systeme wie zum Beispiel CORBA. Oft wird die Middleware direkt auf nichtportable Art modifiziert (z.B. Orbix+Isis [2], Electra [10]); eine portable Variante, die allerdings weniger transparent für Client-Anwendungen ist, verwendet lokale Dienstobjekte oberhalb der Middleware zum Zugriff auf Replikatgruppen (z.B. OpenDREAMS/OGS [3]). Transparent und Middleware-unabhängig ist das Abfangen entfernter Aufrufe und deren Weiterleitung an eine Replikatgruppe unterhalb der Middleware (z.B. Eternal [12]), was man allerdings durch Betriebssystemabhängigkeit erkauft.

In aktuellen Publikationen (z.B. [1], [4]) wird zu Recht die fehlende Flexibilität bei derartigen existierenden Systemen beklagt. Einige der Probleme beschränken die Eignung für ubiquitäre, adaptive oder selbstorganisierende Systeme. Ein erstes Problem ist die mangelnde Interoperabilität zwischen einzelnen Fehlertoleranz-Implementierungen – in der CORBA-Welt, in der die Interoperabilität zwischen Plattformen unterschiedlicher Hersteller eine große Rolle spielt, eine unerfreuliche Einschränkung. Bei der Implementierung eines replizierten Objekts werden dem Entwickler oft gravierende Schranken auferlegt. Beispielsweise wird strikter Determinismus verlangt (d.h. nichtdeterministische Systemaufrufe verboten), oder nur eine strikt sequentielle Ausführung von Methoden erlaubt (d.h. kein Multithreading unterstützt). Ein sicher ebenso gravierendes Problem ist, dass existierende Systeme kaum in der Lage sind, semantisches Wissen über das replizierte Objekt zu nutzen. Auch das führt zu Effizienzeinbußen: Wenn beispielsweise Methoden nur lesend auf ein Objekt zugreifen oder dieses auf konfliktfreie Weise modifizieren, könnte man diese Methoden problemlos nebenläufig ausführen. Noch gravierender ist generell

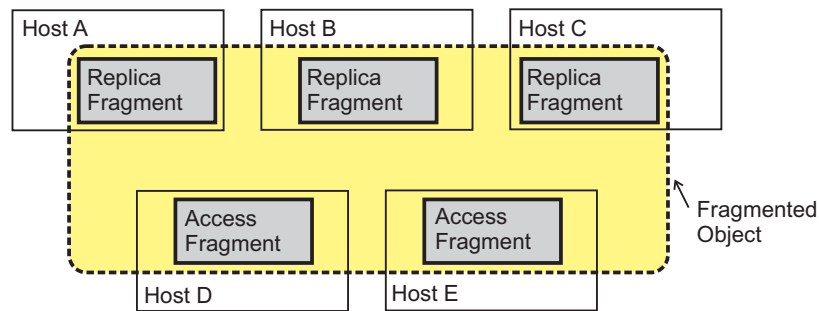


Abb. 1. Replikation mit fragmentierten Objekten

die Starrheit existierender Systeme. Oft wird nur eine bzw. nur wenige Strategien (z.B. aktive und passive Replikation) angeboten. Wünschenswert wäre eine bedarfsgerechte Anpassung der Mechanismen, z.B. um genau das gewünschte Fehlermodell (Rechnerausfälle, Netzwerkausfälle mit Partitionierung, gezielt „boshafte“ Fehlverhalten/byzantinische Fehler) zu tolerieren oder um die Konsistenzmechanismen an die vorhandene Netzinfrastruktur (LAN, WAN, selbstorganisierendes P2P-Overlay) anzupassen.

Die Fehlertoleranz-Unterstützung in der AspectIX-Middleware trägt zur Lösung einiger dieser problematischen Punkte bei. Als Grundlage bietet AspectIX ein fragmentiertes Objektmodell an [15]. Dies macht es möglich, den Code zum Zugriff auf Replikatgruppen und zur Konsistenzwahrung konzeptionell als Teil des replizierten Objekts zu betrachten, anstelle diesen als Teil der Middleware-Infrastruktur anzusehen. Erreicht wird so eine Portabilität zwischen allen Plattformen, welche fragmentierte Objekte unterstützen. Der notwendige Fragmentcode wird dabei weitgehend automatisch durch einen Code-Generator erzeugt. Die Implementierung des replizierten Objekts läßt sich mit semantischen Annotationen versehen, welche bei der Codeerzeugung zur Optimierung verwendet werden. Zudem unterstützt der Codegenerator die Behandlung von Nichtdeterminismus und Multithreading in aktiv replizierten Objekten. Darüberhinaus wird eine Vielzahl an konfigurierbaren Konsistenzmechanismen innerhalb eines modularen Gruppenkommunikationssystems bereitgestellt. Diese erlauben nicht nur eine flexible Anpassung an die Bedürfnisse des Entwicklers, sondern unterstützen auch eine dynamische Rekonfiguration zur Laufzeit und Mechanismen zur autonomen Selbstoptimierung.

Im folgenden Abschnitt 2.1 wird zunächst unsere Replikationsarchitektur basierend auf fragmentierten Objekten vorgestellt. Abschnitt 2.2 beschreibt die Referenzierung von replizierten Objekten. Semantische Annotationen von Methoden werden in Abschnitt 2.3 diskutiert, die flexiblen Mechanismen für Objektimplementierungen in Abschnitt 2.4. Schließlich stellt Abschnitt 2.5 das flexible und rekonfigurierbare Gruppenkommunikationssystem AGC kurz vor. Abschnitt 3 fasst nochmal die wesentlichen Beiträge unserer Architektur zusammen.

2 Replikation in der AspectIX-Middleware

2.1 Replikation mit fragmentierten Objekten

Fragmentierte Objekte sind ein Konzept für objektbasierte verteilte Systeme, das vom klassischen Stub/Server-Prinzip abweicht. Ursprünglich wurden fragmentierte Objekte in FOG [11] und Globe [6] verwendet; AspectIX setzt dieses Objektmodell in einem CORBA-Umfeld um

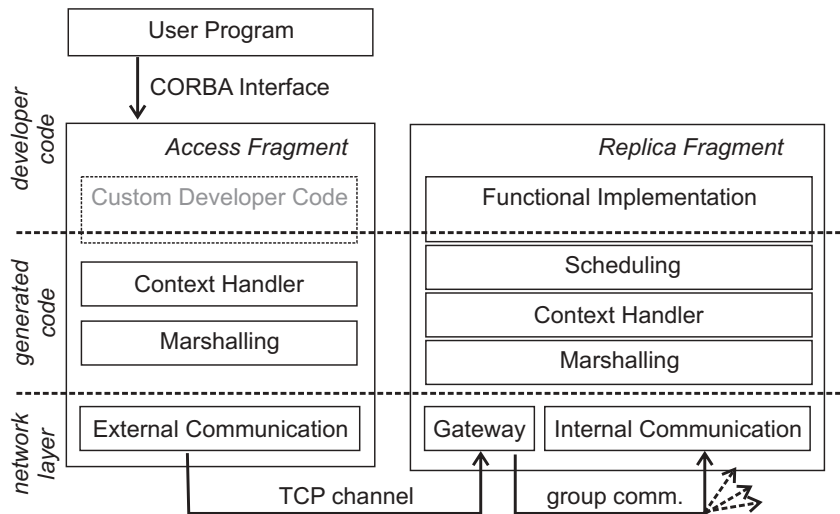


Abb. 2. Architektur-Überblick

[15, 5]. Konzeptionell besteht ein fragmentiertes Objekt (FO) aus einer Menge von verteilt vorhandenen Fragmenten. Jeder Knoten, der auf ein FO zugreift, benötigt dazu ein lokales Fragment. Fragmente haben nach außen eine Schnittstelle, welche der eines klassischen Stubs entspricht. Anders als in traditionellen Systemen erlaubt es die FO-Middleware, für ein fragmentiertes Objekt völlig transparent jeweils objektspezifischen Code zu laden. Dieses Laden von Code wird in AspectIX durch einen dynamischen Ladedienst für Programmcode (DLS, [7]) auch in heterogenen Systemen unterstützt. Durch einen Mechanismus zum dynamischen Austausch von Fragmentimplementierungen eignet sich die Plattform auch gut für adaptive Dienste [8].

Ein fehlertoleranter Dienst lässt sich im FO-Modell als ein verteiltes Objekt darstellen, das aus mehreren Replikats- und Zugriffsfragmenten besteht (siehe Abbildung 1). Die Entwicklung eines replizierten Dienstes durch ein FO erfolgt durch die Definition der Schnittstelle in CORBA IDL, der Implementierung des funktionalen Teils des Dienstes, und der automatischen Generierung von Replikats- und Zugriffscode. Bei der Generierung können auch semantische Informationen, die zusätzlich zur Schnittstelle definiert werden, ausgewertet werden. [16]

Der interne Aufbau der beteiligten Fragmente ist in Abbildung 2 dargestellt. Auf Client-Seite steht ein Interface identisch zu dem eines CORBA-Objekts zur Verfügung. Das generierte Zugriffs-Fragment kann direkt vom Entwickler bereitgestellten, objektspezifischen Code enthalten (siehe 2.3). Der *Context Handler* fügt Kontextinformationen zur eindeutigen Identifizierung des Aufrufs hinzu. Auf Netzebene wird ein total geordnetes Gruppenkommunikationssystem als Basis für aktive Replikation verwendet. Im aktuellen Prototyp wird die Gruppenkommunikation nur zwischen den Replikaten selbst eingesetzt; clientseitige Zugriffsfragmente wenden sich an einen auf allen Replikaten vorhandenen *Gateway*, um Nachrichten von dort per Gruppenkommunikation in totaler Ordnung an die Gruppe zu versenden. Alle eintreffenden Nachrichten werden von *Internal Communication* an die darüberliegende Schicht weitergegeben. Dort durchlaufen sie das (De-)Marshalling sowie den *Context Handler*; letzter ist u.a. für die Unterdrückung von doppelten Ausführungen anhand der ID zuständig, falls ein Aufruf wegen Ausfällen wiederholt werden muss. Schließlich werden Nachrichten an die Warteschlange des Schedulers weitergegeben. Dieser ermöglicht eine kontrollierte, deterministische Ausführung der Aufrufe mit mehreren Threads (siehe Abschnitt 2.4).

2.2 Referenzen auf fragmentierte Objekte

Die Zugriffsfragmente werden durch die AspectIX-Middleware automatisch geladen, sobald sich ein Client an das fragmentierte Objekt bindet. Diese Bindeoperation kann explizit erfolgen, oder implizit beim Empfang einer Objektreferenz bei *call-by-reference*-Semantik. Objektspezifische Informationen in der globalen Referenz (IOR) werden zusammen mit Plattforminformationen verwendet, um über den Codelade-Dienst den gewünschten Code zu laden. Dieser wertet dann Informationen in der Referenz aus, um die Replikatgruppe zu kontaktieren. In einem dynamischen System können Replikate auf andere Rechner migriert werden oder neue Replikate gestartet werden; entsprechend ist die IOR des Objekts zu aktualisieren. Neben dem bekannten FT-CORBA-Konzept der Versionierung von Referenzen mit Versionsüberprüfung und Aktualisierung bei allen Methodenaufrufen lassen sich in AspectIX Referenzen (ähnlich *Leases*) mit einer Gültigkeitsdauer versehen, innerhalb deren Clienten sich um eine Aktualisierung kümmern müssen, selbst wenn sie zwischenzeitlich nicht mit dem Objekt interagieren [9].

2.3 Berücksichtigung semantischer Annotationen

Bei der Generierung von Code für Zugriffs- und Replikatsfragmente wird nicht nur die Schnittstellendefinition in CORBA IDL und die vom Entwickler bereitgestellte Objektimplementierung herangezogen. Es lassen sich auch semantische Objekteigenschaften spezifizieren, die zur Optimierung der Replikationsstrategie verwendet werden.

- `readonly`: Eine `readonly`-Methode ändert den Replikatzustand nicht. Bei aktiver Replikation müssen solche Methoden nicht bei allen Replikaten ausgeführt werden. In der *Gateway*-Komponente werden diese daher direkt, unter Umgehung der Gruppenkommunikation, an die darüberliegende Schicht weitergegeben.
- `parallelizable(methodlist)`: Die spezifizierte Parallelisierbarkeit von Methoden kann im deterministischen Thread-Scheduler verwendet werden, um zwei Bearbeitungsthreads parallel auszuführen, ohne dass die Konsistenz verletzt werden kann.
- `local`: Bei einer so gekennzeichneten Methode wird deren Programmcode direkt im Zugriffsfragment plaziert. Damit lassen sich Methoden, die nicht auf den replizierten Objektzustand zugreifen, direkt auf Client-Seite bereitstellen.
- `intercepted`: Bei einer so gekennzeichneten Methode kann zusätzlich zur funktionalen Implementierung auf Replikatseite Programmcode bereitgestellt werden, der direkt auf Clientseite vor bzw. nach Aufruf der Replikatmethode ausgeführt werden kann. Damit lassen sich Mechanismen wie clientseitiges Caching oder die Akkumulierung von mehreren Methodenaufrufen des Clients zu einem gemeinsamen entfernten Aufruf zu den Replikaten realisieren. Auf diese Weise sind Mechanismen, die in anderen Systemen mit „smart proxies“ realisiert werden, auch in unserer Architektur problemlos möglich.

Derzeit werden diese Annotationen als `#pragma`-Anweisungen direkt in der IDL-Definition festgelegt. Auf Basis des flexibel erweiterbaren IDL-Codegenerators IDLflex [17] entstand hierzu ein Werkzeug zur Codeerzeugung für Replikate.

2.4 Flexibilisierung für Objektentwickler

Bei der Erzeugung von Fragmentcode lassen sich zudem weitere Code-Modifikationen vornehmen. Ein wichtiges Beispiel hierzu ist die Unterstützung von Multithreading in aktiv replizierten

Objekten. Hier ist es wichtig, eine deterministische Reihenfolge beim Scheduling und damit beim Zugriff auf den gemeinsamen Objektzustand sicherzustellen.

Hierzu stellt die AspectIX-Replikationsarchitektur einen geeigneten Scheduler bereit [14]. Bei der Fragmenterzeugung wird nativer Java-Synchronisierungscode transparent für den Entwickler durch Instruktionen ersetzt werden, die mit dem deterministischen Scheduler der Middleware interagieren. Durch diesen Ansatz läßt sich Determinismus bei Multithreading erreichen, ohne dass die Ablaufumgebung (JVM, Betriebssystem) modifiziert werden muss, und ohne dass Objektentwickler manuell geeignete Maßnahmen selbst implementieren müssen.

2.5 Konsistenz-Strategien auf Gruppenkommunikationsebene

Gruppenkommunikation ist ein wichtiger Baustein für fehlertolerante Replikation. Um den Anforderungen bezüglich Flexibilität und dynamischer Anpassbarkeit an unsere Replikationsarchitektur gerecht zu werden, wurde das AspectIX-Gruppenkommunikationssystem (AGC) entwickelt [13]. Das AGC basiert auf verteilten Einigungsalgorithmen zur Erzielung einer totalen Ordnung von Nachrichten und ist zur Laufzeit rekonfigurierbar.

Unser Design verwendet einen politikbasierten Mechanismus für eine dynamische Rekonfiguration zur Laufzeit ohne Beeinträchtigung des Kommunikationsdienstes. Rekonfigurationen erlauben eine Optimierung auf beste Effizienz in „guten“ Situationen oder auf geringste Reaktionszeiten auf Fehler; verschiedene Fehlermodelle wie z.B. crash-stop, crash-recovery oder byzantinische Fehler können ausgewählt werden. Darüber hinaus lassen algorithmische Varianten und veränderliche Parameter (wie z.B. Timeouts zur Erkennung von Ausfällen) eine Optimierung bezüglich der Kommunikations-Infrastruktur (z.B. LAN, WLAN, P2P-Overlay) zu.

Während Vorgaben wie das Fehlermodell nur explizit durch einen Administrator verändert werden können, eignen sich mehrere Parameter auch zur dynamischen Selbstoptimierung. Durch Entscheidungsregeln lässt sich abhängig von Informationen wie z.B. Netzlast, Latenzen zwischen den Replikaten oder Replikatanzahl die jeweils optimale Konfiguration autonom auswählen.

3 Zusammenfassung

Dieses Papier hat einen Überblick über die Architektur zur fehlertoleranten Replikation gegeben, welche im Rahmen des AspectIX-Projekts an der Universität Ulm und der Universität Erlangen-Nürnberg entwickelt wurde. Bei dieser Architektur wird insbesondere das Ziel verfolgt, eine Plattform anzubieten, die sich für weitverteilte, ubiquitäre, dynamische, und selbstorganisierende Systeme besser eignet als bisher existierende Architekturen. Hierzu werden drei wesentliche Beiträge gemacht:

Erstens erlaubt es die Basisarchitektur aus fragmentierten Objekten sehr einfach, objektspezifisch generierten Code bei Klienten und bei Replikaten direkt zu laden. Dabei unterstützt die zugrundeliegende Middleware eine transparente Rekonfigurierung zur Laufzeit durch koordinierten Code-Austausch und bietet ein Konzept für Referenzen in dynamischen Systemen.

Zweitens wird ein Werkzeug zur automatischen Generierung von Fragmentcode bereitgestellt. Dieses ist in der Lage, semantische Annotationen (wie z.B. „readonly“-Methode) auszuwerten und damit die Replikationsmechanismen zu optimieren. Auch bietet es Unterstützung für die Behandlung von Nichtdeterminismus und Multithread-Scheduling in aktiv replizierten Objekten.

Drittens wird eine Variantenvielfalt von Konsistenzstrategien auf Gruppenkommunikationsebene bereitgestellt; diese umfasst u.a. das Fehlermodell („crash-stop“ bis hin zu byzantinischen Fehlern) und die Optimierung auf das gegebene Kommunikationsnetz (z.B. LAN, WAN, P2P-Overlay). Dabei ist nicht nur eine einmalige bedarfsgerechte Auswahl möglich, sondern auch eine dynamische Rekonfiguration sowie in bestimmten Parametern eine Selbstoptimierung (z.B. in Form von Reaktion auf Überlast-Situationen).

Über diese zentrale Kernarchitektur hinaus wird im AspectIX-Projekt auch eine weitergehende Infrastruktur zur verteilten Lokalisierung und Verteilung von Diensten und zur verteilten Ressourcenverwaltung in dynamischen verteilten Systemen entwickelt [8]. Weitere Informationen zum AspectIX-Projekt sind unter <http://www.aspectix.org/> verfügbar.

Literatur

1. Ken Birman. Can web services scale up? *Computer*, 38(10):107–110, 2005.
2. Kenneth P. Birman and Robbert Van Renesse. *Reliable Distributed Computing with the ISIS Toolkit*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1993.
3. Pascal Felber. *The CORBA Object Group Service: A Service Approach to Object Groups in CORBA*. PhD thesis, École Polytechnique Fédérale de Lausanne, Switzerland, 1998. Number 1867.
4. Pascal Felber and Priya Narasimhan. Experiences, strategies, and challenges in building fault-tolerant CORBA systems. *IEEE Trans. Comput.*, 53(5):497–511, 2004.
5. Franz J. Hauck, Rüdiger Kapitza, Hans P. Reiser, and Andreas I. Schmied. A flexible and extensible object middleware: CORBA and beyond. In *Proc. of the Fifth Int. Workshop on Software Engineering and Middleware*, 2005.
6. Philip Homburg, Leendert van Doorn, Maarten van Steen, Andrew S. Tanenbaum, and Wiebren de Jonge. An object model for flexible distributed systems. In *Proceedings of the 1st Annual ASCI Conference*, pages 69–78, 1995.
7. Rüdiger Kapitza and Franz J. Hauck. DLS: a CORBA service for dynamic loading of code. In *Proc. of the OTM'03 Conferences (DOA, Sicily, Italy, Nov. 3-7, 2003)*, number 2888 in LNCS. Springer, 2003.
8. Rüdiger Kapitza, Franz J. Hauck, and Hans P. Reiser. Decentralized, adaptive services: The AspectIX approach for a flexible and secure grid environment. In *Proc. of the GSEM 2004 Conferences (GSEM, Erfurt, Germany, Nov., 2004)*, number 3270 in LNCS. Springer, 2004.
9. Rüdiger Kapitza, Hans P. Reiser, and Franz J. Hauck. Stable, time-bound references in context of dynamically changing environments. In *MDC'05: Proc. of the 25th IEEE Int. Conf. on Distributed Computing Systems - Workshops (ICDCS 2005 Workshops)*, 2005.
10. Silvano Maffei. Adding group communication and fault-tolerance to CORBA. In *Proceedings of the Conference on Object-Oriented Technologies, (Monterey, CA), USENIX*, pages 135–146, 1995.
11. Mesaac Makpangou, Yvon Gourhant, Jean-Pierre Le Narzul, and Marc Shapiro. Fragmented objects for distributed abstractions. In T. L. Casavant and M. Singhal, editors, *Readings in distributed computing systems*, pages 170–186. IEEE Computer Society Press, 1994.
12. Luise E. Moser, P. M. Melliar-Smith, and Priya Narasimhan. Consistent object replication in the eternal system. *Theor. Pract. Object Syst.*, 4(2):81–92, 1998.
13. Hans P. Reiser, Udo Bartlang, and Franz J. Hauck. A reconfigurable system architecture for consensus-based group communication. In *Proc. of the 17th IASTED Int. Conf on Parallel and Distributed Systems (Phoenix, AZ, USA, Nov 14-16, 2005)*, 2005.
14. Hans P. Reiser, Franz J. Hauck, and Rüdiger Kapitza. Deterministic multithreading for replicated CORBA applications, 2006. submitted for publication.
15. Hans P. Reiser, Franz J. Hauck, Rüdiger Kapitza, and Andreas I. Schmied. Integrating fragmented objects into a CORBA environment. In *Proc. of the Net.ObjectDays (Erfurt, Germany)*, 2003.
16. Hans P. Reiser, Rüdiger Kapitza, Jörg Domaschka, and Franz J. Hauck. Fault-tolerant replication based on fragmented objects, 2006. submitted for publication.
17. Hans P. Reiser, Martin Steckermeier, and Franz J. Hauck. IDLflex: a flexible and generic compiler for CORBA IDL. In *Proc. of the Net.ObjectDays (Erfurt, Germany, Sep. 10-13, 2001)*, 2001.