Realisierung eines adaptiven Replikationsmanagers mittels J2EE-Technologie

Heiko Niemann[1], Wilhelm Hasselbring[2], Michael Hülsmann[2], Oliver Theel[2]

1. OFFIS
   Escherweg 2
   26121 Oldenburg
   heiko.niemann@offis.de
2. Carl von Ossietzky Universität Oldenburg
   Fachbereich Informatik
   26111 Oldenburg
   {hasselbring|michael.huelsmann|theel}
   @informatik.uni-oldenburg.de

Kurzfassung: Die Integration in heterogenen Systemlandschaften bedeutet häufig eine Kopplung zur Replikation von Daten lokaler, autonomer Systeme. Dabei soll einerseits Datenmanipulation unter Wahrung der Konsistenz erfolgen, andererseits sollen die beteiligten Systeme i.A. ihre Autonomie beibehalten. Dieser Konflikt der Anforderungen wird durch bekannte Replikationsverfahren nur begrenzt gelöst: Je nach Strategie wird ein Kompromiss zwischen Konsistenz und Autonomie eingegangen. Hier kann ein konfigurierbarer, adaptiver Replikationsmanager Abhilfe schaffen, der eine Kombination bildet aus synchroner Replikation, die eine hohe Konsistenz bei eingeschränkter Verfügbarkeit garantiert, und asynchroner Replikation, die bei eingeschränkter Konsistenz den Systemen eine größere Autonomie belässt. Die Konfiguration des Replikationsmanagers wird durch ein Regelsystem gesteuert, dass Faktoren wie Tageszeit oder Performanz berücksichtigt. Im vorliegenden Papier wird eine Realisierung des adaptiven Replikationsmanagers auf Basis der J2EE-Technologie vorgestellt. Sie bietet sich an, da hier z.B. Transaktions- und Nachrichtendienste genutzt werden können sowie Möglichkeiten zur technischen Anbindung der beteiligten Systeme zur Verfügung stehen.

1 Einleitung

Gegenüber einer Datenreplikation in einem homogenen Systemumfeld, in dem z.B. die Replikation von gleichen Tabellen einer relationalen Datenbank vom Datenbankmanagementsystem selbst übernommen wird, stellt die Replikation in einem heterogenen, autonomen Systemumfeld höhere Ansprüche an die Umsetzung einer geeigneten Strategie. Hier werden zusätzlich Fragen wie die Interoperabilität der Systeme, lokale Schemata oder Semantik der Daten aufgeworfen. Nachdem die zu replizierenden Daten und die involvierten Speichersysteme identifiziert wurden, sind folgende Aufgaben von einer Datenreplikation in diesem Kontext zu bewältigen:

1. Bestimmung einer Replikationsstrategie
2. Auswahl der Transaktionsverarbeitung
3. Technische Anbindung der beteiligten Systeme
4. Integration der lokalen Schemata

Der hier vorgestellte *adaptive Replikationsmanager (ARM)* dient zur Umsetzung dieser vier Aufgaben. Ein Schwerpunkt des ARM bildet naturgemäß die Replikationsstrategie (siehe Punkt 1), die auf Grund unterschiedlicher Konsistenz- und Autonomieanforderungen der heterogenen, autonomen Systeme *konfigurierbar* und *adaptiv* sein soll. Damit einergehend sind die eingesetzten Transaktionskonzepte und die technische Anbindung der beteiligten Systeme relevant. Die J2EE-Technologie unterstützt gerade diese Punkte 2) und 3) durch die entsprechenden Dienste innerhalb einer einheitlichen

Architektur. Da die beteiligten Systeme in einer heterogenen Systemlandschaft i.A. über unterschiedliche Schemata verfügen, ist eine Schemaintegration nötig (siehe Punkt 4). Beim Vorgang der Schemaintegration werden eher semantische Fragen berücksichtigt [SK92], die wir im vorliegenden Papier nicht behandeln.

In Abschnitt 2 werden Replikationsverfahren und Transaktionskonzepte als Grundlagen vorgestellt. Anschließend wird der Aufbau und die Funktionalität des ARM beschrieben: Die Architektur des ARM wird in Abschnitt 3 gezeigt. Abschnitt 4 diskutiert eine Implementierung auf Basis der J2EE-Technologie. In Abschnitt 5 wird das Regelsystem vorgestellt, das der Arm für seine Arbeitsweise (siehe Abschnitt 6) nutzt. Die einzelnen Komponenten des ARM zeigt Abschnitt 7. Abschließend werden in Abschnitt 8 verwandte Arbeiten diskutiert und in Abschnitt 9 ein kurzes Fazit gezogen.

2 Grundlagen

2.1 Replikationsverfahren

In [BD96] werden Klassifizierungskriterien für Replikationsverfahren, die üblicherweise benutzt werden, anhand von drei Fragestellungen diskutiert:

1. Wie erfolgt die replikationsübergreifende Synchronisation der Zugriffsoperationen, d.h. welches Korrektheitskriterium wird verwendet? Dies führt zu *syntaktischen* bzw. *semantischen* Verfahren.
2. Wie werden die Replikate aktualisiert? Es können *synchrone* bzw. *asynchrone* Verfahren unterschieden werden.
3. Wie sieht die Konsistenzsicherung im Fehlerfall aus, z.B. bei Netzpartitionierung? Diese Frage erlaubt eine Einteilung in *pessimistische* bzw. *optimistische* Verfahren.

Das bekannteste Korrektheitskriterium ist wohl die *One-Copy-Serialisierbarkeit (1SR*, siehe Punkt 1): Eine nebenläufige Ausführung von verteilten Transaktionen ist *one-copy-serialisierbar*, wenn sie mit einer seriellen Ausführung dieser Transaktionen auf einer nicht replizierten Datenbank äquivalent ist. In diesem Sinne soll unter *Konsistenz* verstanden werden, dass eine Replikationsstrategie nach einem Korrektheitskriterium wie z.B. 1SR arbeitet. Die Replikationsstrategien, die ein serialisierbares Korrektheitskriterium verwenden, werden den *syntaktischen* Verfahren zugeordnet. Bei den *semantischen* Verfahren wird Wissen über die Semantik der Zugriffsoperationen bzw. Transaktionen zusätzlich ausgenutzt.

Falls z.B. durch Netzpartitionierung (siehe Punkt 3) nicht mehr alle Replikate erreicht werden können, kann ein Replikationsverfahren entweder die Verfügbarkeit der Daten zu Gunsten der Konsistenz einschränken (*pessimistische Verfahren*) oder eine Weiterverarbeitung zulassen und erst einmal Inkonsistenzen in Kauf nehmen (*optimistische Verfahren*). Diese Inkonsistenzen werden auch als *Konflikte* bezeichnet, wobei *Lesekonflikte* (Lesen veralteter Daten) und *Änderungskonflikte* (zwei Systeme ändern den gleichen Wert eines replizierten Objekts jeweils lokal, d.h. ihr eigenes Replikat) auftreten können. Bei der Zusammenführung müssen die Konflikte erkannt und behandelt werden.

Auf Punkt 2 der Klassifizierungsmöglichkeiten wird in [BD96] nicht weiter eingegangen. In diesem Papier wollen wir uns jedoch auf diesen Punkt konzentrieren. Es soll analog zur Kommunikation von Systemen zwischen *synchroner* und *asynchroner* Replikation unterschieden werden, um einen Fokus auf die Interaktion der Systeme zu legen:

- Synchrone Replikation bei hoher Konsistenzanforderung
- Asynchrone Replikation bei hoher Autonomieanforderung

***Synchrone Replikation*:** Mit synchroner Replikation ist die kontrollierte Änderung der notwendigen Replikate innerhalb einer Transaktion gemeint. Für das ROWA-Verfahren (Read One, Write All) [TG82] sind das alle Replikate, für die Voting-Verfahren (siehe z.B. [Gi79]) sind das die Replikate, die zu einem Quorum gehören. Synchrone Replikation genügt hohen Konsistenzanforderungen, aber es müssen Einschränkungen hinsichtlich Autonomie, d.h. Verfügbarkeit und Performanz, hingenommen werden. Wir werden hier auch quasi-synchrone Replikation (Abschnitt 2.2) mit parallelen Sagas betrachten.

***Asynchrone Replikation*:** Bei der asynchronen Replikation werden die Replikate zeitversetzt aktualisiert. Beim Peer-To-Peer-Verfahren (siehe z.B. [GRR98]) können alle Kopien geändert werden, die Aktualisierung wird asynchron vorgenommen und es werden somit bewusst Konflikte in Kauf genommen. Bei der Zusammenführung müssen dann die möglicherweise aufgetretenen Konflikte geeignet behandelt werden. Das Peer-To-Peer-Verfahren belässt den Systemen eine hohe Autonomie, da nur geringe Einschränkungen für das Bereitstellen des Propagierungsauftrags auftreten. Es nimmt aber Einschränkungen hinsichtlich der Konsistenz in Kauf, d.h. das Korrektheitskriterium 1SR wird nicht erfüllt.

2.2 Transaktionskonzepte

Um einen ordnungsgemäßen Dienst zu gewährleisten, müssen die Replikationsverfahren nach den Prinzipien der Transaktionsverarbeitung implementiert werden (zu verschiedenen Konzepten siehe z.B. [GR93]). Im Folgenden soll dargestellt werden, welche Transaktionskonzepte für uns in Frage kommen:

- 2PC- / XA-Protokoll [Gr78] für die synchrone Replikation XA-fähiger Systeme.
- Sagas [GS87] für die *quasi-synchrone* Replikation nicht XA-fähiger Systeme.
- Queued Transactions [BN97] für die asynchrone Replikation.

***2-Phasen-Commit-(2PC-)Protokoll/XA-Protokoll*:** Das 2PC-Protokoll basiert auf dem Grundgedanken, dass alle an einer globalen Transaktion T beteiligten Systeme sich darauf einigen, ob T durchgeführt oder abgebrochen wird [Da96]. Die X/Open, ein Konsortium von DBMS-Herstellern, hat Systemkomponenten und Schnittstellen für verteilte Transaktionsverarbeitung spezifiziert, die auf dem 2PC-Protokoll fußt. Die lokalen Systeme stellen einen Ressourcenmanager bereit, der das sogenannte XA-Protokoll unterstützt. Hierüber steuert ein globaler Transaktionsmanager die verteilte Transaktion. Das XA-Protokoll wird von vielen Systemherstellern unterstützt, so dass diese Form der Verarbeitung zu einem defacto-Standard geworden ist.

Sagas ([GS87], auch „verkettete Transaktion“) gehören zu den offen-geschachtelten Transaktionen. Eine *Saga-Transaktion* besteht aus einer Folge von einzelnen Transaktionen $T_1,...,T_n$, die jeweils bei erfolgreicher Beendigung „committed“ werden, d.h. ihre Änderungen sichtbar machen. Für Sagas treffen isolierte Zurücksetzbarkeit und Serialisierbarkeit im Sinne der ACID-Eigenschaften nicht mehr zu. In unserem Fall sollen Sagas genutzt werden, um nicht XA-fähige Systeme an der Replikation teilnehmen zu lassen. Durch Überlappung der Einzeltransaktionen T_i (siehe Parallel Sagas [GS87]) kann eine scheinbar zeitgleiche Verarbeitung erzielt werden, d.h. eine quasi-synchrone, aber nicht echt synchronisierte Verarbeitung. Die Aktualisierung der Replikate kann so erfolgen, dass in jeder Einzeltransaktion T_i jeweils genau ein Replikat geändert wird. In jeder Einzeltransaktion T_i wird somit eine lokale, „flache“ Transaktion ausgeführt. Diese spezielle Variante des Sagas-Konzepts wollen wir im weiteren durch den Index R kennzeichnen, also $\boldsymbol{Sagas_R}$. Bei Abbruch einer Einzeltransaktion T_i wird gemäß dem Forward Recovery neu aufgesetzt

Queued Transactions: In [BN97] wird das Queued Transaction Processing Modell anhand des Client/Server-Modells erläutert. Dabei zerfällt die globale Transaktion in drei Teiltransaktionen, die jeweils den ACID-Eigenschaften unterliegen. T_1: Ein Client stellt eine Anfrage in die *request queue*. T_2: Der Server bearbeitet die Anfrage und erteilt eine Antwort in die *reply queue*. T_3: Der Client holt die Antwort und bearbeitet sie. Die Queued Transactions können wie folgt für die asynchrone Replikation verwendet werden (hier als $\boldsymbol{QT_R}$ bezeichnet): Ein Client stellt eine *Replikationsanforderung*, d.h. eine Änderungsanforderung aller Replikate in eine „Replica Queue". Ein Server holt die Anforderung aus der Replica Queue und aktualisiert die Replikate. Auf das Schreiben einer Antwort und die folgende Bearbeitung der Antwort kann verzichtet werden. Im fehlerfreien Fall ist die Transaktion erfolgreich abgeschlossen, im Fehlerfall wird ein Konfliktmanagement für die weiteren Schritte aktiviert.

3 Architektur des adaptiven Replikationsmanagers

Um sowohl die Vorteile der synchronen Replikation (Konsistenz) als auch der asynchronen Replikation (Autonomie) zu nutzen, ist eine Kombination der jeweiligen Verfahren nötig. In Abhängigkeit der Replikationsanforderungen und der Zustände der beteiligten Systeme kann zwischen den Verfahren gewechselt bzw. ein Mischverfahren verwendet werden, wobei eine dynamische Anpassung an veränderte Bedingungen erfolgen soll. Dabei werden grundsätzlich Lesezugriffe auf veraltete Daten gemäß der Peer-To-Peer-Replikation toleriert. Eine derartige *adaptive Replikationsstrategie* [NH02] kann von einem ARM (adaptiver Replikationsmanager) umgesetzt werden, der gemäß der in Abbildung 1 gezeigten Architektur aufgebaut ist. Die ARM realisiert durch die transaktionsgesicherte Verarbeitung aller Replikationsanforderungen ein Verfahren nach dem Korrektheitskriterium *Konvergenz* [Le97]. Konvergenz besagt, dass alle Replikate eines replizierten Objekts letztendlich den gleichen Wert annehmen und dieser Wert derjenige ist, der durch die letzte, abgeschlossene Transaktion geschrieben wurde.

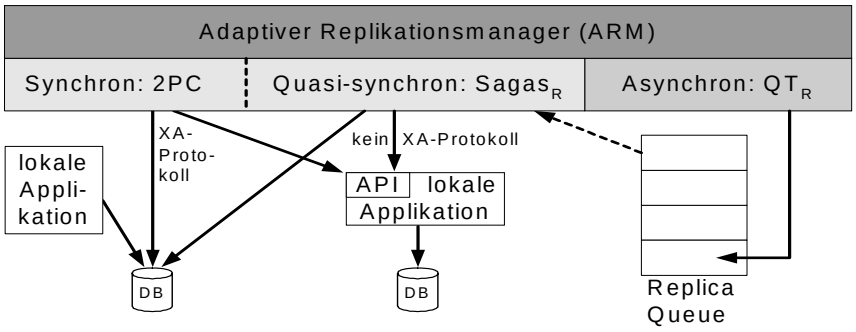


Abbildung 1 Architektur des konfigurierbaren, adaptiven Replikationsmanagers

Der ARM verarbeitet Replikationsanforderungen anhand eines parametrisierten Regelsystems (Abschnitt 5), wobei die Konfiguration dynamisch den aktuellen Gegebenheiten angepasst wird. Auf Grund der aktuellen Einstellungen werden diejenigen Systeme bestimmt, die synchron bzw. asynchron aktualisiert werden (Abschnitt 6). Die synchrone Replikation wird transaktional entweder gemäss dem 2PC durchgeführt, sofern die beteiligten Systeme das XA-Protokoll unterstützen, oder gemäss dem Sagas$_R$-Konzept, sofern das XA-Protokoll nicht unterstützt wird (letzteres bedeutet eine quasi-synchrone Replikation). Die asynchrone Replikation nach dem QT$_R$-Konzept geschieht dadurch, dass die

Replikationsanforderung je Replikat zunächst in eine „Replica Queue“ eingefügt wird. Zu einem späteren Zeitpunkt wird die zurückgestellte Anforderung gemäß dem Prinzip verteilter Transaktionen bearbeitet, d.h. erstens die Anforderung aus der Replica Queue holen und zweitens ein beteiligtes System aktualisieren. Hierfür wird wiederum der Mechanismus der synchronen Transaktionsverarbeitung genutzt (gestrichelter Pfeil in Abbildung 1).

Der Zugriff auf die operativen Systeme hängt davon ab, ob auf die Datenquelle direkt oder über Schnittstellen zugegriffen wird, d.h. ob mit einem direkten Datenbankzugriff die Aktualisierung erfolgen kann oder ob die Datenänderung mittels eines entfernten Prozeduraufrufes auf die Schnittstellen (APIs) einer lokalen Anwendung geschieht. In [RMB00] wird zwischen *Datenbankintegration*, d.h. ein Zugriff auf die Datenquelle erfolgt parallel zu den lokalen Anwendungen, und *Applikationsintegration*, d.h. ein Zugriff erfolgt über die Schnittstellen der lokalen Anwendungen, unterschieden. Letzteres ist immer dann erforderlich, wenn keine direkte Datenmanipulation an der zugrunde liegenden Datenquelle erlaubt ist oder diese zu komplex ist.

4 Realisierung mittels J2EE-Technologie

Die J2EE-Technologie (Java 2 Enterprise Edition [JNB01]) bietet sich aus folgenden Gründen für die Realisierung des ARM an:

- Sie umfasst einen Anwendungsserver mit Unterstützung für synchrone und asynchrone Kommunikation: Ein Enterprise JavaBeans Server (EJB-Server) verwaltet und unterstützt die EJB-Anwendungen durch sogenannte Container für EJBs. Verschiedene Dienste, wie z.B. ein Transaktionsdienst (Java Transaction Service, JTS) und ein Nachrichtendienst (Java Message Service, JMS), können bei Bedarf eingebunden werden.
- Sie ermöglicht Zugriffe auf operative Systeme (Legacy-Systeme): Für Zugriffe auf Datenbanken steht die Java Database Connectivity (JDBC) zur Verfügung. Die Java Connector Architecture (JCA) spezifiziert Zugriffe auf Anwendungen bzw. deren Schnittstellen.
- Sie unterstützt Nutzungs-Schnittstellen und -Programme: Sowohl mit der Java 2 Standard Edition (J2SE) als auch der Java 2 Micro Edition (J2ME) können Client-Anwendungen entwickelt werden. Java Server Pages (JSP) bzw. Java Servlets unterstützen die Entwicklung von Web-Anwendungen. Die Realisierung der Client-Anwendungen wird im vorliegenden Papier nicht betrachtet.

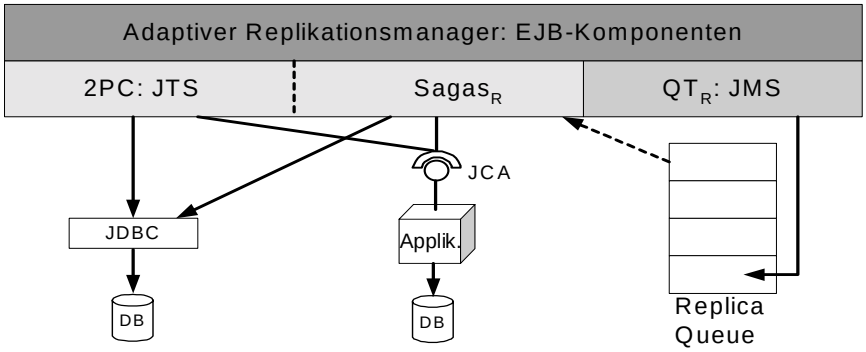


Abbildung 2 Realisierung des adaptiven Replikationsmanagers mittels J2EE-Technologie

Der ARM in Abbildung 2 wird als EJB-Anwendung realisiert und kommt auf einem J2EE-Server zum Einsatz (Deployment). Die Umsetzung der Transaktionsverarbeitung im synchronen Fall erfolgt über JTS (inkl. XA-Protokoll). Falls das $Sagas_R$-Konzept verwendet werden muss, ist eine eigene Implementierung nötig. Das QT_R-Konzept für den asynchronen Fall wird über den Nachrichtendienst JMS (inkl. XA-Protokoll) realisiert, wobei die Replica Queue als Topic (Publish/Subscribe-Verfahren) oder Queue (Point-To-Point-Verfahren) gemäß der JMS-Terminologie gestaltet wird [MC01].

Die operativen Systeme werden im Fall einer Datenbankintegration über JDBC [SS00] angebunden, im Fall einer Applikationsintegration erfolgt der Zugriff gemäss JCA (siehe z.B. [SSN02]). Im letzteren Fall wird für das betreffende System ein Ressourcenadapter benötigt, über den die individuellen Schnittstellen angesprochen werden. Sowohl über JDBC als auch JCA können XA-fähige und nicht XA-fähige Ressourcen angebunden werden. Clients, die z.B. in J2SE oder J2ME implementiert sind, können Replikationsanforderungen über RMI (Remote Methode Invocation) an den ARM übergeben.

5 Das Regelsystem des adaptiven Replikationsmanagers

Wir definieren zunächst den Begriff der sogenannten *Replikationseinheit (RE)*: Eine Replikationseinheit ist entweder ein System, das eine Replikationsanforderung stellt, eine Relation oder ein Tupel.

Die hier vorgestellte adaptive Replikationsstrategie kombiniert die synchrone und asynchrone Replikationsstrategie wie folgt: Zunächst wird von synchroner Replikation ausgegangen, um eine hohe Konsistenz zu erreichen. Zwischen synchroner und quasi-synchroner Replikation wird hier nicht unterschieden, weil dadurch nur die technische Anbindung der Systeme beeinflusst wird. Falls ein oder mehrere Systeme die Autonomie der anderen gefährden bzw. zu stark einschränken, wird für diese Systeme dynamisch zur asynchronen Replikation gewechselt. Nach einer Prüfung oder einem festgelegtem Zeitintervall wird wieder zur synchronen Replikation zurückgewechselt. Der Wechsel zwischen synchroner und asynchroner Replikation ist konfigurierbar, d.h. dem Replikationsmanager werden Regeln vorgegeben, anhand derer er automatisch Anpassungen an die Replikationsstrategie vornimmt.

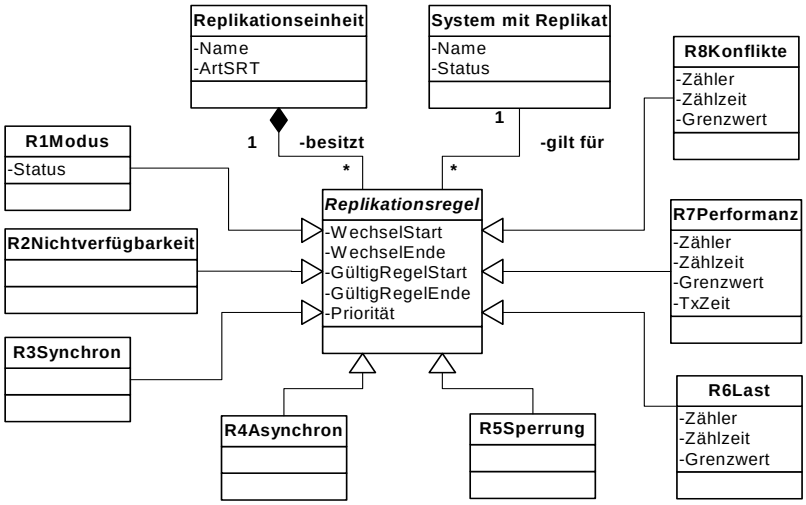


Abbildung 3 vereinfachtes Klassendiagramm des Regelsystems

Je RE können Regeln vergeben werden. Eine Regel gilt genau für ein System mit Replikat. Für jedes System mit Replikat gibt es eine ausgezeichnete Regel, die besagt, ob das System grundsätzlich synchron aktualisiert wird (z.B. ein „Mastersystem“), grundsätzlich asynchron aktualisiert wird (z.B. ein Data Warehouse) oder wechselfähig ist.

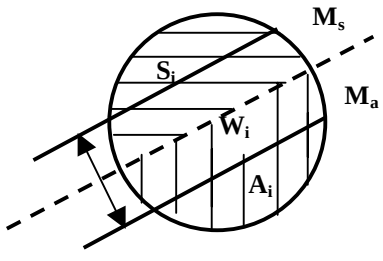
Das Regelsystem in Abbildung 3 als UML-Klassendiagramm [RJB99] ist wie folgt definiert: Anhand des Attributes ArtSRT wird für eine RE festgehalten, ob es sich um ein System, eine Relation oder ein Tupel handelt. Auch die Spezifikation einer Default-Einstellung ist möglich, da nicht für alle Systeme bzw. Relationen individuelle Regeln vorgegeben werden müssen. Eine RE besitzt mehrere Replikationsregeln (abstrakte Klasse). Eine Replikationsregel gilt für Systeme mit Replikat und kann zu speziellen Regeln abgeleitet werden. Diese speziellen Regeln legen fest, wann zwischen synchroner und asynchroner Replikation gewechselt wird bzw. ob für ein System grundsätzlich synchron oder asynchron repliziert werden soll. Die speziellen Regeln lauten (siehe auch Abbildung 3):

- R1 Das System (Attribut Status) wird synchron bzw. asynchron aktualisiert oder es ist wechselfähig.
- R2 Asynchrone Replikation bei Nichtverfügbarkeit.
- R3 Synchrone Replikation im gegebenen Zeitintervall.
- R4 Asynchrone Replikation im gegebenen Zeitintervall.
- R5 Sperrung im gegebenen Zeitintervall. Die Anforderung wird zwar angenommen, aber erst nach Ende des Zeitintervalls asynchron ausgeführt.
- R6 Wechsel zur asynchronen Replikation, wenn in einem gegebenen Zeitintervall zu viele Replikationsanforderungen auftreten.
- R7 Wechsel zur asynchronen Replikation, wenn in einem gegebenen Zeitintervall die Performanz zu Wünschen übrig lässt.
- R8 Synchrone Replikation im gegebenen Zeitintervall wegen zu großer Anzahl von Konflikten.

Die Regeln R2 bis R8 können je System mehrfach gesetzt werden, z.B. eine Sperrung gemäß R5 von 10:00 Uhr bis 11:30 Uhr und 15:00 Uhr bis 16:15 Uhr. Da die Regeln R3 und R8 einen Wechsel von asynchron zu synchron begründen, die übrigen Regeln einen Wechsel von synchron zu asynchron, können Konflikte innerhalb der Regeln auftreten. Durch Vergabe von Prioritäten kann festgelegt werden, welche Regel den Vorzug erhält.

6 Die Arbeitsweise des adaptiven Replikationsmanagers

Die Abbildung 4 zeigt, wie der ARM die Systeme mit Replikat in die Gruppen synchron und asynchron zu aktualisierender Systeme einteilt:



$$S_i \subseteq M_s \subseteq (S_i \cup W_i)$$
$$A_i \subseteq M_a \subseteq (A_i \cup W_i)$$

Abbildung 4 Aufteilung der Systeme mit Replikat

Die Menge **I** sei die Menge der REen und die Default-Einstellung, **M** sei die Menge der Systeme mit Replikat, $\mathbf{S_i}$ die Menge der Systeme, die synchron aktualisiert werden, $\mathbf{A_i}$

die Menge der Systeme, die asynchron aktualisiert werden und $\mathbf{W_i}$ die Menge der wechselfähigen Systeme, mit $\mathbf{i \in I}$. S_i, A_i und W_i partitionieren M.

Wenn eine Replikationsanforderung gestellt wird, teilt der ARM anhand der Regeln und der aktuellen Situation die beteiligten Systeme in zwei Mengen (siehe Abbildung 4): $\mathbf{M_s}$ sei die Menge der Systeme, die synchron aktualisiert werden, und $\mathbf{M_a}$ sei die Menge der Systeme, die asynchron aktualisiert werden. M wird durch M_s und M_a partitioniert. M_a und M_s werden für jede Replikationsanforderung neu bestimmt und können somit variieren (in Abbildung 4 durch den Doppelpfeil dargestellt). Zu einem bestimmten Zeitpunkt hängen M_a und M_s einerseits vom Regelsystem (Konfiguration) und andererseits vom sich dynamisch ändernden System ab (Adaption).

7 Die Komponenten des adaptiven Replikationsmanagers

Der ARM kann in einzelne Komponenten gegliedert werden, die mehr oder weniger unabhängig bestimmte Aufgaben bewältigen. In Abbildung 5 ist dargestellt, aus welchen Komponenten (gemäß der J2EE-Terminologie EJB-Komponenten) der ARM besteht.

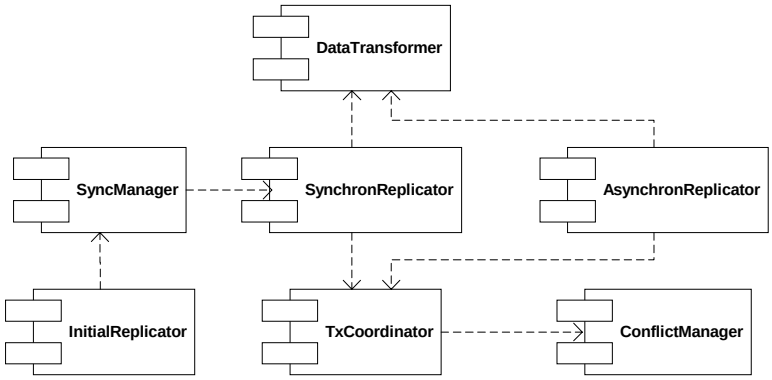


Abbildung 5 Komponenten des ARM, dargestellt als UML-Komponenten-Diagramm [RJB99]

Die einzelnen Komponenten erfüllen folgende Aufgaben:

- Der SynchronReplicator nimmt Replikationsanforderungen entgegen, bearbeitet sie und liefert einen Rückgabewert. Die Bearbeitung umfasst die synchrone Replikation sowie das Einstellen in die Replica Queue.
- Der AsynchronReplicator verwaltet die Replica Queue, d.h. die asynchronen Replikationsaufträge werden für die jeweiligen Systeme ausgeführt.
- Der DataTransformer ist für die Transformation von Daten zum globalen und zu den lokalen Datenschemata im Sinne der Schemaintegration verantwortlich.
- Der TxCoordinator steuert die transaktionale Verarbeitung des ARM mittels des Transaktionssystems des J2EE-Servers. Für das $Sagas_R$-Konzept wird eine eigene Implementierung benötigt. Konflikte werden an den ConflictManager übergeben.
- Der ConflictManager ist sowohl für die Erkennung von Konflikten als auch für die Behandlung der aufgetretenen Konflikte zuständig.
- Der SyncManager synchronisiert ein anfragendes System, z.B. ein mobiles Gerät, mit den zentralen Systemen.
- Der InitialReplicator dient der erstmaligen Erstellung einer replizierten Datenquelle oder der Aktualisierung von „Downloads“ der mobilen Geräten.

Nur der SynchronReplicator kann einen Wechsel von synchron zu asynchron bewirken. Der AsynchronReplicator sorgt ausschließlich für den Wechsel von asynchron zu synchron, wobei der Wechsel nur dann erfolgen kann, wenn für das betreffende System keine Aufträge in der Replica Queue anstehen.

8 Verwandte Arbeiten

Zum Thema Replikation sind eine Vielzahl von Arbeiten erstellt worden, so dass an dieser Stelle nur ein kleiner Ausschnitt betrachtet werden soll. Einen Schwerpunkt auf Autonomie setzen die optimistischen Verfahren. Gerade hier finden neuerdings auch mobile Geräte Berücksichtigung. Vertreter von Systemen, die nach diesen Konzepten entwickelt wurden, sind CODA [Sa02], Bayou [TTP95] oder Rumor [GRR98].

In [LH00] wird eine konfigurierbare Replikation speziell bei mobilen Applikationen diskutiert. Lenz [Le97] stellt in seiner Dissertation eine adaptive Datenreplikation vor, die pessimistische und optimistische Ansätze kombiniert. Es werden sogenannte Konsistenzinseln gebildet, deren Replikate alle den gleichen, aktuellen Wert haben. Nicht zur Konsistenzinsel gehörende Replikate können zeitverzögert aktualisiert werden.

Fragen zum Konfliktmanagement und zum Datenabgleich (reconciliation) tauchen bei Netzpartitionierung auf. Zur Schreib/Schreib-Konflikterkennung wird ein klassisches Verfahren in [PPR83] vorgestellt. In diesem optimistisch-syntaktischen Ansatz sollen Inkonsistenzen am replizierten Datenbestand erkannt und lokalisiert werden. Ein weiteres klassisches Verfahren ist das sogenannte optimistische Protokoll [Da84]. Es pflegt einen sogenannten precedence graph, mittels dem partitionsbezogene Konflikte erkannt und beseitigt werden können. In [GAB83] werden diese Probleme bei der Zusammenführung dadurch gelöst, dass im Datensatz bestimmte Regeln abgelegt werden, die bei Konflikten eine Behandlung ermöglichen.

In [Ha97] wird die Replikation über die Schemata föderierter Datenbanksysteme erläutert, d.h. hier wurde insbesondere ein Schwerpunkt auf die verschiedenen Schemata in verteilten Systemen gelegt.

9 Zusammenfassung und Ausblick

In heterogenen, autonomen Informationssystemen kann die Replikation von Daten durch rein synchrone bzw. asynchrone Verfahren nicht optimal gelöst werden. Ein adaptiver Replikationsmanager (ARM), der durch Kombination der beiden Varianten eine konfigurierbare, adaptive Replikationsstrategie implementiert, kann eine optimalere Lösung erreichen, insbesondere dann, wenn zusätzlich Funktionalität wie z.B. das Konfliktmanagement unterstützt wird. Diese Vorteile werden durch die Abhängigkeit vom ARM bzw. dem System, auf dem der ARM installiert ist, erkauft. Eine Unabhängigkeit von einem zentralen System kann durch Clustern vom ARM erreicht werden. Für die Implementierung bietet sich die Java-Technologie J2EE mit den entsprechenden Diensten an.

In diesem Beitrag wird die Architektur des ARM sowie eine Realisierung auf Basis der J2EE-Technologie gezeigt. Die Umsetzung der adaptiven Replikationsstrategie wird vom ARM über ein hier vorgestelltes Regelsystem gewährleistet. Weiterhin werden die einzelnen Komponenten und Funktionalitäten des ARM betrachtet, sowie die technische Anbindung der beteiligten Systeme diskutiert.

Ein explorativer Prototyp zur Validierung der Ergebnisse wurde auf Basis des J2EE-Servers Bea Weblogic (www.bea.com) implementiert. Als Systeme mit Replikat wurden

die XA-fähigen Datenbanken Cloudscape (www.ibm.com), Oracle (www.oracle.com) und MS-SQL-Server (www.microsoft.com) mit identischem Schema eingesetzt. Des weiteren wurde eine Anbindung des nicht XA-fähigen SAP R/3 Systems (www.sap.com) über einen Ressourcenadapter gemäß der Java Connector Architecture (JCA) untersucht. Zukünftige Arbeitspakete beinhalten eine genauere Analyse des Regelsystems und formale Korrektheitsprüfungen, z.B. zur Deadlock- und Lifelock-Erkennung, sowie ein Monitoring der Arbeitsweise des ARM. Als Anwendungsbereich werden Krankenhausinformationssysteme betrachtet [NHW02].

Literaturverzeichnis

[BD96] Beuter, T.; Dadam, P.: Prinzipien der Replikationskontrolle in verteilten Systemen. In: Informatik Forschung und Entwicklung 11 (4), 1996, S.203-212.

[BN97] Bernstein, P.; Newcomer, E.: Principles of Transaction Processing. Morgan Kaufmann, 1997.

[Da84] Davidson, S.: Optimism and Consistency in Partitioned Distributed Database Systems. In: ACM Transactions on Computer Systems 9 (3), 1984, S.456-481.

[Da96] Dadam, P.: Verteilte Datenbanken und Client/Server-Systeme. Springer-Verlag, 1996.

[GAB83] Garcia-Molina, H.; Allen, T.; Blaustein, B. T. et al.: Data-Patch: Integrating Inconsistent Copies of a Database After a Partition. In:Symposium on Reliability in Distributed Software and Database Systems, Clearwater Beach, FL, 1983 S.38-44.

[Gi79] Gifford, D. K.: Weighted Voting for Replicated Data. In:ACM Symposium on Operating Systems Principles (SIGOPS), Pacific Grove, California, 1979 S.150-159.

[Gr78] Gray, J.: Notes on Database Operating Systems. In R. Bayer et al. (ed): Operating Systems: an Advanced Course, Heidelberg, Springer Lecture Notes in Computer Science, 1978.

[GR93] Gray, J.; Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993.

[GRR98] Guy, R.; Reiher, P.; Gunter, M. et al.: Rumor: Mobile Data Access Through Optimistic Peer-To-Peer Replication. In:International Conference on Conceptual Modeling (ER), Singapore, 1998 S.254-265.

[GS87] Garcia-Molina, H.; Salem, K.: Sagas. In:ACM SIGMOD, San Francisco, 1987 S.249-260.

[Ha97] Hasselbring, W.: Federated integration of replicated information within hospitals. In: International Journal on Digital Libraries 1 (3), 1997, S.192-208.

[JNB01] Juric, M.; Nagappan, R.; Basha, J. et al.: Professional J2EE EAI. WROX, 2001.

[Le97] Lenz, R.: Adaptive Datenreplikation in verteilten Systemen. Teubner, Leipzig 1997.

[LH00] Lubinski, A.; Heuer, A.: Configured Replication for Mobile Applications. In:Proc. of Baltic DB & IS, Vilnius, Litauen, 2000 S.1-5.

[MC01] Monson-Haefel, R.; Chappel, D. A.: Java Message Service. O'Reilly & Associates, 2001.

[NH02] Niemann, H.; Hasselbring, W.: Adaptive Replikationsstrategie für heterogene, autonome Informationssysteme. In:14. Workshop über Grundlagen von Datenbanken, Fischland, 2002 S.81-85.

[NHW02] Niemann, H.; Hasselbring, W.; Wendt, T. et al.: Kopplungsstrategien für Anwendungssysteme im Krankenhaus. In: Wirtschaftsinformatik 44 (5), 2002, S.425-434.

[PPR83] Parker, D. S.; Popek, G.; Rudisin, G.: Detection of Mutual Inconsistency in Distributed Systems. In: IEEE Transactions on Software Engineering 9 (3), 1983, S.240-247.

[RJB99] Rumbaugh, J.; Jacobson, I.; Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley, 1999.

[RMB00] Ruh, W.; Maginnis, F.; Brown, W.: Enterprise Application Integration: A Wiley Tech Brief. John Wiley & Sons, 2000.

[Sa02] Satyanarayanan, M.: The Evolution of CODA. In: ACM Transactions on Computer Systems 20 (2), 2002, S.85-124.

[SK92] Sheth, A. P.; Kashyap, V.: So Far (Schematically) yet So Near (Semantically). In:Database Semantics Conference on Interoperable Database Systems, Lorne, Victoria, Australia, 1992 S.283-312.

[SS00] Saake, G.; Sattler, K.-U.: Datenbanken und Java. JDBC, SQLJ und ODMG. dpunkt-Verlag, 2000.

[SSN02] Sharma, R.; Stearns, B.; Ng, T.: J2EE Connector Architecture and Enterprise Application Integration. Addison Wesley, 2002.

[TG82] Traiger, I. L.; Gray, J.; Galthier, C. A. et al.: Transactions and consistency in distributed database systems. In: ACM Transactions on Database Systems 7 (3), 1982, S.323-342.

[TTP95] Terry, D. B.; Theimer, M. M.; Petersen, K. et al.: Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In:Symposium on Operating Systems Principles, Copper Mountain Resort, Colorado, 1995 S.172-183.