Einsatz von Entwurfsmustern bei der Entwicklung eines föderierten Krankenhausinformationssystems mit CORBA

Dr. Wilhelm Hasselbring & Peter Ziesche
Universität Dortmund, Informatik 10 (Software-Technologie), D-44221 Dortmund
E-mail: {hasselbring|ziesche}@ls10.informatik.uni-dortmund.de

Zusammenfassung

Verteilte Krankenhausinformationssysteme sind komplexe Systeme von Systemen, die eine flexible Software-Architektur benötigen. Im vorliegenden Beitrag präsentieren wir die Architektur eines föderierten Krankenhausinformationssystems und berichten über unsere Erfahrungen mit dem Einsatz von Entwurfsmustern für die Strukturierung und Programmierung der CORBA-basierten Kommunikationsschnittstelle für dieses System.

1 Einleitung

Die Aufgabe eine Krankenhausinformationssystems (KIS) ist, das medizinische und administrative Personal bei der Arbeit zu unterstützen [SPFW90, WZB$^+$96]. Integrierte Anwendungssoftwaresysteme für Krankenhäuser, die eine ausreichende Rechnerunterstützung für das gesamte KIS bieten, sind nicht verfügbar [WiZi95], auch wenn einige Anbieter dieses versprechen. Es ist auch aus wirtschaftlichen Gründen sinnvoll, jeweils optimale Anwendungssoftwareprodukte als weitgehend autonome Anwendungssysteme in den einzelnen Abteilungen zu installieren, die dann jeweils nur Teile des KIS unterstützen. Typische Beispiele sind Anwendungssysteme für die Patientenaufnahme und -abrechnung, die Leistungserfassung in Laboratorien oder Ambulanzen, und die Auswertung von Therapieergebnissen zur Qualitätskontrolle und zur medizinischen Forschung.

So entstehen KIS, die durch Heterogenität in Hard- und Software gekennzeichnet sind. Im Sinne einer effektiven und effizienten Unterstützung der Arbeitsabläufe im Krankenhaus müssen die Teilsysteme miteinander integriert werden, um mehrfache Eingaben von Daten und Inkonsistenzen zwischen verschiedenen Teilsystemen zu vermeiden. Sinnvoll ist somit ein modulares System als *Föderation* interoperabler und kooperativer Teile, wobei die Autonomie der Teilsysteme weitgehend bewahrt werden sollte. Die Konsistenzsicherung von Daten, die in verschiedenen Teilsystemen redundant gespeichert sind, ist hier ein zentrales zu lösendes Problem [Hass97].

Die konsistente Replikation von Daten erhöht aber auch die Zuverlässigkeit: falls ein Teilsystem ausfällt, können die anderen Teilsysteme weiterarbeiten, solange sie nicht auf Daten aus ausgefallenen Systemen zugreifen müssen. Eine Replikation wird sich ohnehin kaum vermeiden lassen: kommerzielle Produkte können nicht dahingehend verändert werden, daß sie sich Daten, die auch in anderen Systemen gespeichert sind, über das Netzwerk holen, statt sie aus der lokalen Datenbank zu laden. Die Replikation von Daten erlaubt zusätzlich einen effizienten lesenden Zugriff auf die lokalen

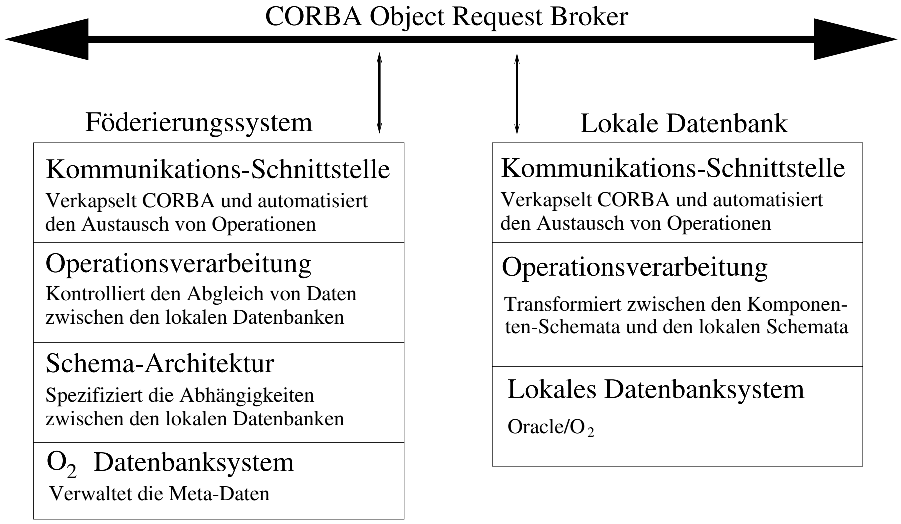

Abbildung 1: Die Schichten in den Hauptkomponenten unserer CORBA-basierten Architektur.

Daten, ohne ein Netzwerk zu belasten. Änderungsoperation könnten allerdings aufwendiger werden. In KIS werden Patientendaten jedoch nur sehr selten geändert; zumeist gibt es Ergänzungen.

Für die Kopplung von Teilsystemen in Krankenhäusern werden häufig Kommunikationsserver eingesetzt [LOPH96]. Kommunikationsserver unterstützen den Austausch von Nachrichten zwischen Teilsystemen. Die Integrität der Daten kann durch einen Kommunikationsserver jedoch nicht direkt unterstützt werden. Die Aktionen, die z.B. zur Aufrechterhaltung von Konsistenz zwischen Daten in verschiedenen Teilsystemen nötig sind, müssen durch die entsprechenden Teilsysteme initiiert werden. Die Teilsysteme müssen selbst dafür sorgen, daß die Inhalte von empfangenen Nachrichten geeignet in den lokalen Datenbanken abgespeichert werden. Mit einer Kopplung, die auf Kommunikationsservern basiert, ist es auf Integrationsebene nicht bekannt, *wo* Daten *wie* gespeichert und repliziert sind. Das sieht bei der Integration über ein Föderierungssystem anders aus, da es hier Abbildungen zwischen den Teilen der lokalen Datenbankschemata, die die gleichen Informationen bezeichnen, gibt.

2 Integration durch ein Föderierungssystem

In einem föderierten Datenbanksystem werden sowohl Applikationen auf lokalen Datenbanken als auch globale Applikationen, die auf mehrere lokale Datenbanken zugreifen können, unterstützt, wobei die Autonomie der Teilsysteme weitgehend bewahrt wird [ShLa90]. Die Integration der Datenbestände erfolgt über die Datenbankschemata. Das Föderierungssystem kann auch die Konsistenz replizierter Daten, die in verschiedenen lokalen Datenbanken gespeichert sind, sichern. Diese Aufgabe, für die im folgenden ein Lösungsansatz präsentiert wird, ist besonders wichtig in KIS.

Wie in Abb. 1 dargestellt, besteht unsere Architektur aus mehreren Komponenten. Das zentrale Föderierungssystem ist für den Datenaustausch zwischen den beteiligten lokalen Datenbanken zuständig, von denen in Abb. 1 nur eine dargestellt ist. Die Kommunikation zwischen diesen Komponenten wird über einen *CORBA Object Request Broker* (ORB) [MoZa95] abgewickelt. Die CORBA-spezifischen Mechanismen werden durch eine Kommunikations-Schnittstelle gekapselt, die den Komponenten eine einfach zu benutzende Schnittstelle bietet (siehe Abschnitt 4). Alle Komponenten haben eine Schichtenarchitektur. Die jeweils oberste Schicht bildet die Kommunikations-Schnittstelle. Die Operationsverarbeitung übernimmt u.a. die Konvertierungen zwischen dem kanonischen Datenmodell des Föderierungssystems und dem lokalen Datenmodell (siehe Abschnitt 3).

Der Datenaustausch zwischen den lokalen Datenbanken (über das Föderierungssystem) erfolgt auf der Basis von Datenbankoperationen. Die Kommunikations-Schnittstelle einer lokalen Datenbank sendet eine Operation zusammen mit den benötigten Daten an das Föderierungssystem, das daraus Operationen für andere Datenbanken generiert. Die Kommunikations-Schnittstelle sorgt für eine einfache Handhabung der Operationen. Die Schema-Architektur und die Meta-Daten des Föderierungssystems werden in Abschnitt 3 beschrieben.

Zur Zeit verwenden wir die Chorus Cool [JSJC95] CORBA-Implementierung, wobei ein System zur Kontrolle des Therapieerfolges in der Angiographie (Behandlung von Gefäßkrankheiten) [UHJ+96] mit einem auf Oracle [Bron89] basierenden Patientendatenverwaltungssystem gekoppelt wird. Das Angiographiesystem wurde in Zusammenarbeit mit dem Klinikum Wuppertal mit der objektorientierten Datenbank O_2 [BaDK92] entwickelt.

3 Die Schema-Architektur

Die Abhängigkeiten und Übereinstimmungen zwischen den Daten, die in den lokalen Datenbanken gespeichert sind, werden in föderierten Systemen entsprechend der 5-Ebenen-Referenzarchitektur nach [ShLa90] verwaltet. Diese Referenzarchitektur hat sich insbesondere für die Integration heterogener Schemata bewährt. Sie wurde primär für einen globalen (lesenden) Zugriff auf mehrere Datenbanken entworfen.

Um eine konsistente Replikation von Daten zu unterstützen, haben wir diese Referenzarchitektur durch eine explizite Unterscheidung zwischen Import- und Export-Schemata auf der dritten Ebene erweitert [Hass97] (Siehe Abb. 2). Die Spezifikation eines Import-Schemas entspricht einem *Abonnement* der entsprechenden Daten aus einer anderen Datenbank. Export-Schemata spezifizieren Daten, die an andere Datenbanken geliefert werden sollen. Die lokalen Datenmodelle sind die *konzeptionellen* Datenmodelle der lokalen Datenbanken entsprechend dem klassischen 3-Ebenen-Modell für Datenbanksysteme [Date95]. In Abb. 2 sind keine externen Schemata aus dem föderierten Schema für globale Applikationen abgeleitet worden, was aber möglich wäre (die fünfte Ebene fehlt also im Bespiel). Hier dient das Föderierungssystem nur der Konsistenzsicherung. Auf Basis der Abhängigkeiten, die in der Schema-Architektur spezifiziert sind, werden die Operationen geeignet transformiert und den lokalen Datenbanken zugeordnet.

Abb. 2 stellt ein Beispiel-Szenario für die Änderung replizierter Information anhand der Schema-Architektur dar. Die lokale Datenbank 1 exportiert eine Datum über das zugeordnete Komponenten-Schema und das daraus abgeleitete Export-Schema an andere Datenbanken. Die Datenbanken 2, 3 und 4 importieren diese Information über die Import-Schemata 2 und 3 und die ihnen zugeordneten Komponenten-Schemata. Die semantische Übereinstimmung dieser Daten wird über das föderierte Schema spezifiziert.

Die Komponenten-, Export-, Import- und föderierten Schemata werden einheitlich in der *kanonischen* Datenmodellierungssprache spezifiziert [ShLa90]. Wir verwenden den ODMG-93-Standard [Catt96] als Grundlage für das kanonische Datenmodell [SaHa96]. Bei den Abbildungen von und zu den lokalen Schemata muß evtl. zwischen verschiedenen Datenmodellierungssprachen transformiert

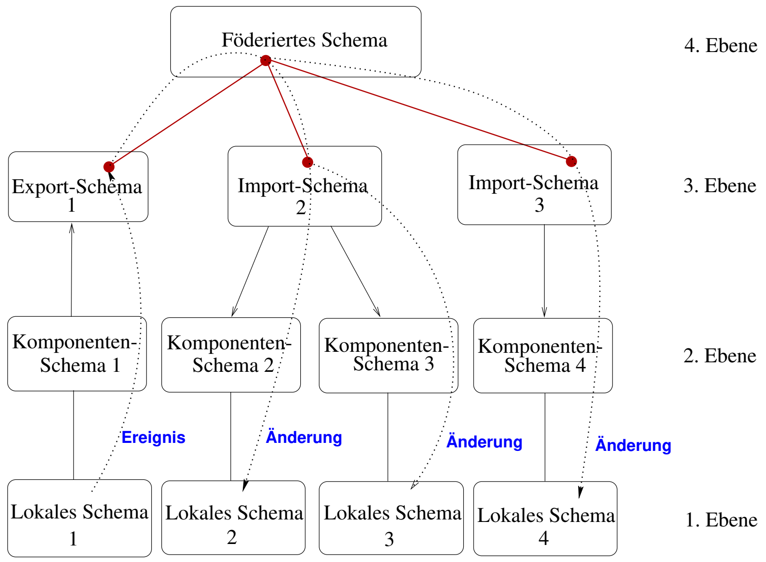

Abbildung 2: Ein Beispiel-Szenario für die Schema-Architektur. Die gepunkteten Pfeile illustrieren den Datenfluß für die Weiterleitung eines Ereignisses in einer lokalen Datenbank, das Änderungsoperationen in anderen lokalen Datenbanken zur Folge hat. Die fünfte Ebene ist hier leer.

werden.

Als ein Beispiel für den Krankenhausbereich wird in Abb. 3 der Fluß von Patientendaten in der Schema-Architektur zwischen einem Abrechnungssystem und einem medizinischen Informationssystem illustriert. Links unten in Abb. 3 ist ein kleiner Ausschnitt des lokalen Datenmodells des Abrechnungssystems, das mit einer relationalen Datenbank arbeitet, dargestellt. Dieses lokale Schema wird zunächst auf ein entsprechendes Schema im objektorientierten kanonischen Datenmodell des Föderierungssystems abgebildet, wobei die Fremdschlüsselbeziehung zwischen Patienten und Abrechnungen (Invoices) auf eine objektorientierte Beziehung [Catt96] abgebildet wird. Die Patientenstammdaten werden dann vom Abrechnungssystem durch ein Export-Schema über das föderierte Schema und ein Import-Schema in das medizinische Informationssystem exportiert wie durch die gestrichelten Pfeile angedeutet. Umgekehrt exportiert das medizinische Informationssystem Information über verbrauchte Materialien zusammen mit der Patientenidentifikation in das Abrechnungssystem (gepunktete Pfeile). Der Einfachheit halber gehen wir in diesem Szenario davon aus, daß Patienten zunächst in der Verwaltung aufgenommen werden, bevor die medizinische Behandlung beginnt. Notfälle müßten anders gehandhabt werden.

Die Schemata und deren Beziehungen zueinander werden als *Meta-Daten* des Föderierungssystems in der objektorientierten Datenbank O_2 [BaDK92] persistent gespeichert. Für eine genaue Beschreibung der Schema-Architektur und der darauf arbeitenden Algorithmen sei auf [Hass97] verwiesen. Ein zu lösendes Problem besteht nun darin, die Änderungsoperationen geeignet zu übertragen. Das ist die Aufgabe der Kommunikations-Schnittstelle, die im folgenden Abschnitt beschrieben wird.

4 Die Kommunikations-Schnittstelle

Die Kommunikations-Schnittstelle hat die Aufgabe, den Datenaustausch zwischen den föderierten, lokalen Datenbanken und dem Föderierungssystems abzuwickeln. Bei den zu übertragenden Daten

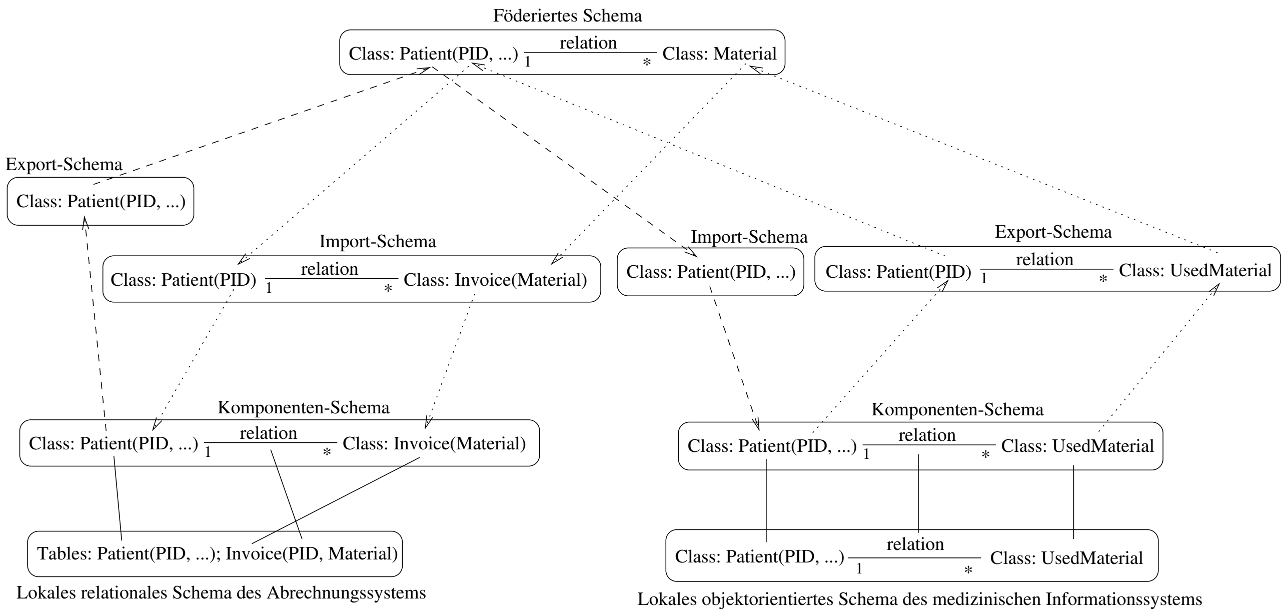

Abbildung 3: Ein Beispiel für den Import und Export von Daten zwischen einem Abrechnungssystem und einem medizinischen Informationssystem.

handelt es sich um Spezifikationen von Änderungsoperationen (z.B. Insert, Update oder Delete). Werden solche Operationen auf einer Datenbank durchgeführt, erfolgt über die Kommunikations-Schnittstelle die Weitergabe an das Föderierungssystem und von dort (wieder über die Kommunikations-Schnittstelle) die Verteilung an die anderen angeschlossenen Datenbanken entsprechend den in der Schema-Architektur spezifizierten Abhängigkeiten. Die Kommunikations-Schnittstelle stellt für die einzelnen Komponenten des föderierten Datenbanksystems eine Schnittstelle bereit, die den Datenaustausch vereinfacht und den Kontrollfluß weitgehend übernimmt. Die Architektur der Kommunikations-Schnittstelle ist in Abb. 4 und Abb. 5 dargestellt, in denen die UML-Notation für Klassendiagramme verwendet wird [Oest97].

Um den Datenaustausch für die Komponenten des föderierten Systems so einfach wie möglich zu gestalten, gleichzeitig aber auch möglichst flexibel in Bezug auf die Struktur der zu übertragenden Daten zu sein, verwendet die Kommunikations-Schnittstelle zwei grundsätzliche Mechanismen:

Operations-Objekte: Die Spezifikationen von Datenbank-Operationen werden in C++-Objekten gekapselt. Für jeden Operationstyp (z.B. Insert, Update oder Delete) wird eine eigene Klasse definiert, deren Struktur auf die zu übertragenden Daten abgestimmt ist. Alle Operations-Klassen erben von der abstrakten Klasse `Operation`, die eine einheitliche Schnittstelle für alle Operationen vorgibt. Abb. 4 stellt das Prinzip der abstrakten Operationen dar. Die Kommunikations-Schnittstelle benutzt nur die Schnittstelle von `Operation`, kann daher polymorph mit beliebigen Operationsklassen arbeiten.

Damit die Kommunikations-Schnittstelle Objekte von beliebigen Operations-Klassen erzeugen kann, muß ein Operations-Objekt in der Lage sein, sich selbst zu duplizieren. Dazu dient die `Clone`-Methode. Das Verfahren wird in Abschnitt 5.2 beschrieben. Jeder Operationstyp wird über eine eindeutige Identifikation (ID) identifiziert.

Da die IDL-Schnittstelle für CORBA unabhängig von den Operationstypen bleiben soll, muß ein Operationsobjekt für den Versand an eine andere Komponente sequentialisiert werden. Zur

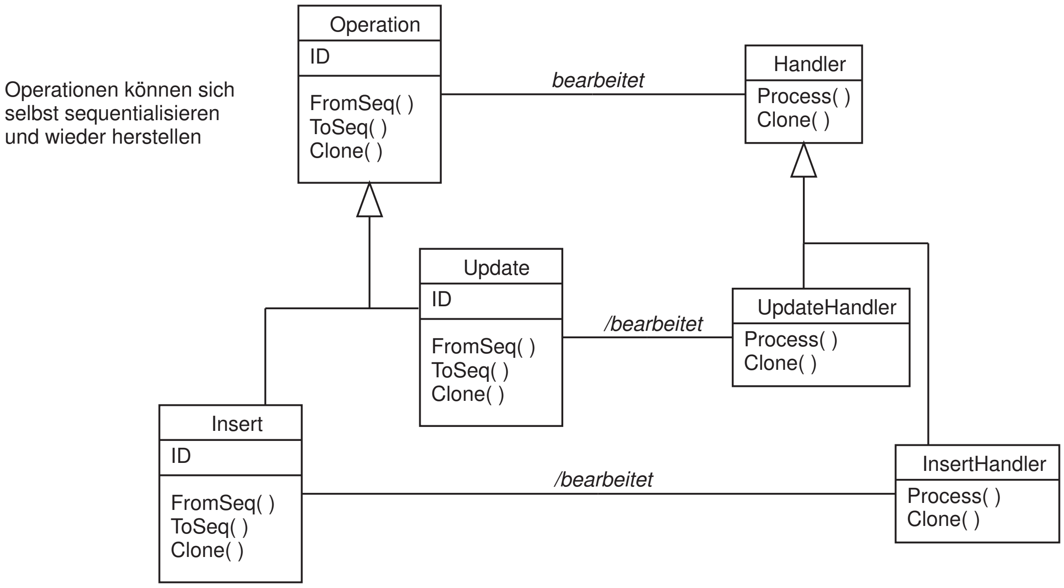

Abbildung 4: Das Prinzip der abstrakten Operationen. Das '/' vor den unteren zwei 'bearbeitet'-Assoziationen zeigt die Vererbungsbeziehung zu der entsprechenden oberen Assoziation an.

Übertragung bieten sich CORBA-Sequenzen an, weil sonst die genaue Struktur der zu übertragenden Operationen fest in der IDL spezifiziert werden müßte, was sehr inflexibel wäre. Eine konkrete Operations-Klasse implementiert dazu die virtuellen Methoden `FromSeq` und `ToSeq` (Abb. 4). Die Kommunikations-Schnittstelle wandelt eine sequentialisierte Operation in ein Operationsobjekt um, indem sie zunächst die `Clone`-Methode des Prototypen des entsprechenden Operationstyps aufruft. Anschließend wird die sequentialisierte Operation an die `FromSeq`-Methode des neuen Objekts übergeben, wodurch das Objekt sich selbst mit Daten füllt.

Operations-Handler: Der Versand einer Operation durch den Sender geschieht durch Erzeugung eines entsprechenden Operations-Objektes, dessen korrekte Initialisierung und die Übergabe an die Kommunikations-Schnittstelle. Zur Verarbeitung einer Operation sind je nach Komponente und Art der Operation unterschiedliche Aufgaben zu erledigen. Eine Komponente, die Operationen eines bestimmten Typs empfangen will, muß daher für jeden Typ einen *Handler* bereitstellen (Abb. 4), der als C++-Klasse implementiert wird. Wenn die Kommunikations-Schnittstelle eine Operation empfängt, benutzt sie Kopien der Prototypen, um die Bearbeitung durchzuführen. Dieser Mechanismus entspricht dem Entwurfsmuster *Prototype*, das in Abschnitt 5.2 beschrieben wird.

In der Klasse `Pool` werden Operations/Handler-Paare verwaltet (Abb. 5). Zu einer Operations-ID liefert die Klasse ein zuvor registriertes Operations- und ein Handler-Objekt. Empfängt die Kommunikations-Schnittstelle eine (sequentialisierte) Operation, so werden mit Hilfe der ID die zugehörigen Operations- und Handler-Objekte aus dem Pool abgerufen. Die Operation wird desequentialisiert und an die `Process`-Methode des Handlers übergeben, der dann für die Bearbeitung verantwortlich ist. Das Abrufen von Operations- und Handler-Objekten wird ebenfalls in Abschnitt 5.2 genauer beschrieben.

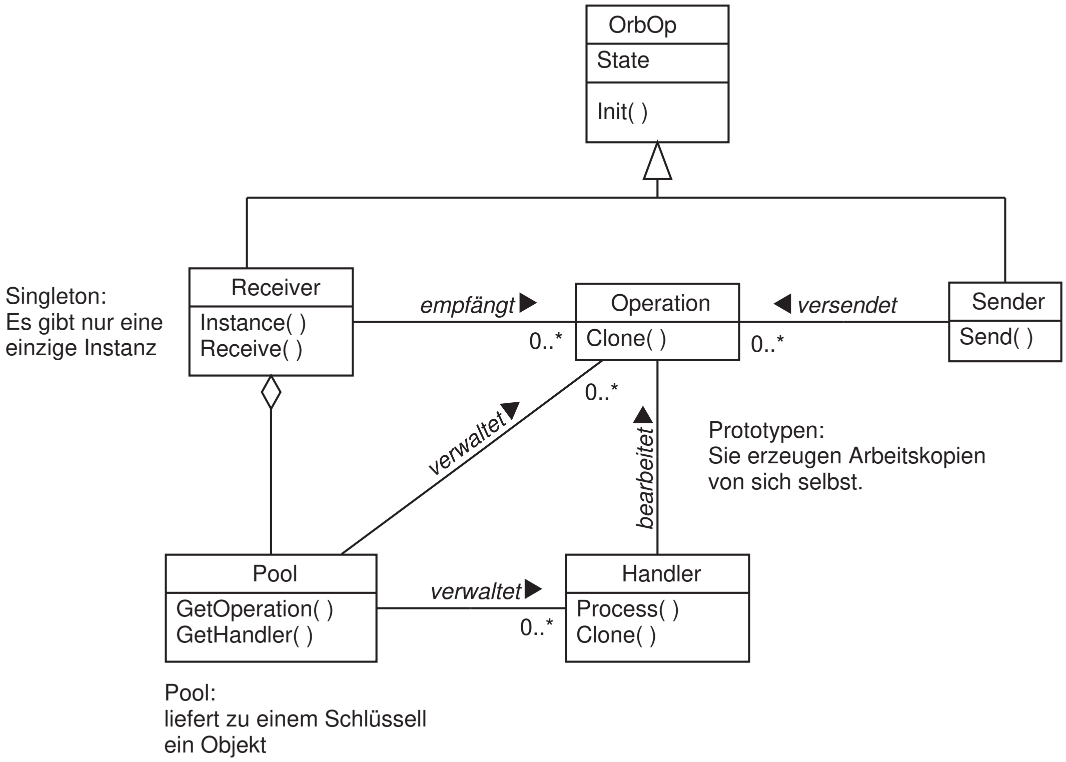

Abbildung 5: Die Architektur der Kommunikations-Schnittstelle. Die verwendeten Entwurfsmuster sind durch Annotationen gekennzeichnet.

Für den Versand und den Empfang von Operationen steht je eine Klasse zur Verfügung (Abb. 5). `Sender` versendet eine Operation an eine bestimmte Komponente des föderierten Systems. Dies geschieht durch Übergabe eines Operationsobjektes an die `Send`-Methode. `Receiver` empfängt Operationen und ruft die entsprechenden Handler auf. Der Nutzer muß sich daher keine Gedanken über den Kontrollfluß innerhalb der Kommunikations-Schnittstelle machen. Von der Klasse `Receiver` gibt es nur eine einzige C++-Instanz je CORBA-Objekt (siehe Abschnitt 5.1). Sender-Objekte kann es mehrere geben.

Die Klassen `Sender` und `Receiver` erben von der abstrakten Klasse `OrbOp`, die CORBA-spezifische Funktionaltät zur Verfügung stellt, so daß die eigentliche Benutzung des ORB nicht in den konkreten Klassen programmiert werden muß.

5 Entwurfsmuster zur Strukturierung der Kommunikations-Schnittstelle

5.1 Das Entwurfsmuster *Singleton*

Die Kommunikations-Schnittstelle stellt sich anderen CORBA-Objekten im System als ein CORBA-Objekt dar. Die Klasse `Receiver` implementiert diese Server-Schnittstelle, die nur aus der Methode `Receive` besteht. Deshalb darf innerhalb einer Kommunikations-Schnittstelle stets nur eine einzige Instanz (C++-Objekt) dieser Klasse existieren. Das Entwurfsmuster *Singleton* [GHJV94, Seite 127ff] bietet eine gute Lösung für dieses Problem (Abb. 5).

Programmiertechnisch wird der Konstruktor der Klasse in C++ als `private` deklariert [Stro86].

Eine direkte Instanziierung von außen mit Hilfe des `new`-Operators ist somit ausgeschlossen. Stattdessen stellt `Receiver` die `static`-Methode `Instance` zur Verfügung. Beim ersten Aufruf wird eine neue Instanz von `Receiver` erzeugt (als Methode der Klasse hat `Instance` Zugriff auf den Konstruktor). Die Adresse dieses *Singleton-Objektes* wird in einem `static`-Attribut der Klasse gespeichert und zurückgegeben. Bei wiederholten Aufrufen von `Instance` wird einfach die gespeicherte Adresse zurückgegeben. Auf diese Weise kann an jeder Stelle im Programm, an der die Klasse `Receiver` bekannt ist, auf diese einzige Instanz (den *Singleton*) zugegriffen werden.

5.2 Das Entwurfsmuster *Prototype*

Die Kommunikations-Schnittstelle soll unabhängig von den zu bearbeitenden Operationen sein. Beim Empfang einer Operation liegt diese jedoch als CORBA-Sequenz in sequentialiserter Form vor. Die Kommunikations-Schnittstelle muß intern Instanzen der richtigen Klassen für *Operation* und *Handler* erzeugen, ohne den Klassen-Namen fest, d.h. namentlich zu codieren. Dazu wird das Entwurfsmuster *Prototype* eingesetzt [GHJV94, Seite 117ff].

Die Schnittstellen sind durch die abstrakten Basisklassen `Operation` und `Handler` beschrieben, von denen konkrete Operations- bzw. Handlerklassen erben (Abb. 4). Ein Nutzer der Kommunikations-Schnittstelle erzeugt bei der Initialisierung seines Programms für jedes konkrete Operations/Handler-Paar C++-Instanzen (*Prototypen*) und registriert diese im Pool der Kommunikations-Schnittstelle. Beim Empfang einer (sequentialisierten) Operation ermittelt `Receiver` über die Operations-ID (siehe Abschnitt 4) das zuvor registrierte Operations/Handler-Paar, desequentialisiert die Operation und übergibt sie dem Handler zur Bearbeitung. Innerhalb der Kommunikations-Schnittstelle selbst wird keine konkrete Klasse namentlich angesprochen.

`Receiver` verwaltet die registrierten Paare in der Klasse `Pool` (Abb. 5). Der Pool liefert zu einer Operations-ID Arbeitskopien der registrierten Prototypen. Dazu benutzt er die `Clone`-Methoden der Operations- bzw. Handler-Prototypen, die jeweils eine Kopie von sich selbst liefern. Die Prototypen selbst bleiben dadurch unverändert und können beliebig oft als Kopiervorlage dienen. In [GHJV94] befinden sich dazu auch Hinweise zur Programmierung in C++.

In einem früheren Stadium des Projekts verwendeten wir für die Kommunikations-Schnittstelle statt der Prototypen das Entwurfsmuster *Abstract Factory* [GHJV94, Seite 87ff]. Es war vorgesehen, in den einzelnen CORBA-Objekten verschiedene *Produkt-Familien* für die Handler zu verwenden, um so unterschiedliches Verhalten der Handler in unterschiedlichen Komponenten des Systems zu realisieren. Eine *Abstract Factory* sieht jedoch die feste Codierung der Produkt-Arten vor. Damit war die Kommunikations-Schnittstelle nicht mehr unabhängig von der Art und Anzahl der unterstützten Operationen. Die *Abstract Factory* ließ sich auf dieses Problem daher nicht optimal anwenden. Diese Erfahrung deckt sich grundsätzlich mit den Aussagen von Erich Gamma zum Einsatz von Entwurfsmustern: es ist nicht immer sofort klar, welche Muster für ein Problem geeignet sind [Gamm96].

6 Abschlußbemerkungen

Ziel der vorgestellten Arbeit ist, aus den isolierten rechnerunterstützten Informationssystemen einzelner Krankenhausabteilungen ein verteiltes, kooperierendes Informationssystem zu schaffen. Solche *Systeme von Systemen* benötigen eine flexible Software-Architektur. Die Konsistenzsicherung von Daten, die in verschiedenen Teilsystemen redundant gespeichert sind, ist ein zentrales zu lösendes Problem. Die Architektur eines föderierten KIS wurde als Lösungsansatz präsentiert und der Einsatz von Entwurfsmustern für die Strukturierung der CORBA-basierten Kommunikations-Schnittstelle für dieses System wurde beschrieben.

Der Einsatz von Entwurfsmustern unterstützte dabei den Entwurf einer *flexibel* strukturierten Kommunikations-Schnittstelle. Die CORBA-spezifischen Aspekte konnten insbesondere durch das Entwurfmuster *Prototype* sehr einfach verkapselt werden, so daß es jetzt relativ einfach wäre, die Kommunikation auf eine andere Middleware-Plattform mit CORBA-ähnlicher Funktionalität zu portieren, ohne die Programmteile, die die Kommunikations-Schnittstelle nutzen, ändern zu müssen.

Die von uns benutzten Entwurfsmuster hatten neben der guten Strukturierung des Entwurfs noch einen zweiten großen Vorteil: Die Beschreibungen der Muster in [GHJV94] liefern auch Hinweise für die Implementierung, sowie Beispiel-Quelltexte. Diese Beispiele konnten mit geringfügigen Modifikationen von uns übernommen werden. Der Implementierungs- und Testaufwand wurde dadurch erheblich reduziert.

Der Versand und die Bearbeitung der Operationen erfolgt in der Kommunikations-Schnittstelle zur Zeit synchron. Nachdem ein Sender eine Operation verschickt hat, erhält er die Kontrolle erst nach deren Bearbeitung durch den Empfänger zurück. Es ist daher geplant, die Kommunikations-Schnittstelle so zu erweitern, daß das Warten auf Nachrichten in einem eigenen Programmzweig (Thread) stattfindet, um so die Parallelität innerhalb der Komponenten zu erhöhen. Gegebenenfalls soll auch die Bearbeitung der Operationen selbst parallelisiert werden, d.h. jedes Handler-Objekt arbeitet dann in einem eigenen Programmzweig.

Danksagung

Unser Dank gilt Klaus Alfert, Andreas Dinsch, Mario Ellebrecht, Xi Gao, Sven Gerding, Betül Ilikli, Djamel Kheldoun, Patrick Koehne, Mischa Lohweber, Ulf Radmacher, Karl-Heinz Schulte und Dilber Yavuz für die gute Zusammenarbeit im Projekt FOKIS.

Literatur

[BaDK92] F. Bancilhon, C. Delobel, und P. Kanellakis. *Building an Object-Oriented Database System: The Story of O_2.* Morgan Kaufman, 1992.

[Bron89] M. Bronzite. *Introduction to Oracle.* McGraw-Hill, London, 1989.

[Catt96] R. Cattell, Hrsg. *The Object Database Standard: ODMG-93, Release 1.2.* Morgan Kaufman, 1996.

[Date95] C.J. Date. *An introduction to database systems.* Addison-Wesley, 6th Auflage, 1995.

[Gamm96] E. Gamma. Entwurfsmuster — ein weiteres Allheilmittel für die Softwareentwicklung? In *Proc. GI-Fachtagung Softwaretechnik 96*, Softwaretechnik-Trends 16/3, Seiten 1–3, Koblenz, September 1996.

[GHJV94] E. Gamma, R. Helm, R. Johnson, und J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software.* Addison Wesley, 1994.

[Hass97] W. Hasselbring. Federated Integration of Replicated Information within Hospitals. International Journal on Digital Libraries, 1997 (im Druck).

[JSJC95] C. Jacquemot, P. Strarup Jensen, und S. Carrez. CHORUS/COOL: CHORUS Object Oriented Technology. In *Object-Based Parallel and Distributed Computation (OBPDC '95)*, Band 1107 der Reihe *Lecture Notes in Computer Science*, Seiten 187–204. Springer-Verlag, 1995.

[LOPH96] M. Lange, N. Osada, H.-U. Prokosch, und W. Hasselbring. Ein Ansatz zur Klassifikation und zum Vergleich von Kommunikationsservern. In *Abstracts zur 41. GMDS-Jahrestagung*, Bonn, September 1996.

[MoZa95] T.J. Mowbray und R. Zahavi. *The Essential CORBA: Systems Integration Using Distributed Objects*. Wiley, 1995.

[Oest97] B. Oestereich. *Objektorientierte Softwareentwicklung: Analyse und Design mit der Unified Modeling Language*. R. Oldenbourg Verlag, 1997.

[SaHa96] S. Sander und W. Hasselbring. Realisierung eines föderierten Datenbanksystems auf Basis der Standards CORBA und ODMG-93. In *Kurzfassungen zum 2. Workshop "Föderierte Datenbanken"*, Seiten 45–50, Dortmund, Dezember 1996.

[ShLa90] A. Sheth und J. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[SPFW90] E.H. Shortliffe, L.E. Perreault, L.M. Fagan, und G. Wiederhold, Hrsg. *Medical informatics: computer applications in health care*. Addison-Wesley, 1990.

[Stro86] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, 1986.

[UHJ+96] I. Ullrich, W. Hasselbring, T. Jahnke, A. Röser, und A. Christmann. Ein objektorientiertes System zur Kontrolle des Therapieerfolges in der Angiographie. In *Abstracts zur 41. GMDS-Jahrestagung*, Bonn, September 1996.

[WiZi95] A. Winter und R. Zimmerling. Die Bedeutung von Referenzmodellen für das Management von Krankenhausinformationssystemen. In F. Huber-Wäschle, H. Schauer, und P. Widmayer, Hrsg., *GISI 95 Herausforderungen eines globalen Informationsverbundes für die Informatik*, Seiten 703–710. Springer-Verlag, 1995.

[WZB+96] A. Winter, R. Zimmerling, O. Bott, S. Gräber, W. Hasselbring, R. Haux, A. Heinrich, R. Jaeger, I. Kock, D.P.F. Möller, O. Penger, J. Ritter, A. Terstappen, und A. Winter. Das Management von Krankenhausinformationssystemen: Eine Begriffsdefinition. In *Abstracts zur 41. GMDS-Jahrestagung*, Bonn, September 1996.