Migrationsprojekt mit Corba

Andres Koch El.Ing.HTL, M.Math
Object Engineering GmbH, Zürich
Kontaktformular

Das beschriebene Projekt gewann den ersten Preis an der COMDEX Internet and Object World Frankfurt 1997

Vortrag anlässlich der OOP97 in München 1997 "Migration mit CORBA"

Zum Autor

Andres Koch promovierte 1976 als Elektro-Ingenieur an der Ingenieurschule des Interkantonalen Technikums in Rapperswil und 1981 mit dem Master Degree in Computer Science an der Universität von Waterloo, Kanada. Er weist mehr als 15 Jahre Praxiserfahrung sowohl in der industriellen wie auch kommerziellen Software-Entwicklung aus und befasst sich heute hauptsächlich mit objektorientierten, verteilten Systemen und der Migration von Legacy-Systemen. Er ist als Geschäftsführer der Object Engineering GmbH in Zürich und als lehrbeauftragter Dozent am Nachdiplomstudium für Software-Engineering der Ingenieursschule des Technikum Rapperswil tätig.

Zusammenfassung

Legacy Systeme sind in der Regel von keiner bestimmten Machart. Im Gegenteil, sie überraschen den Software-Ingenieur immer wieder mit neuen verborgenen "Schätzen", die es bei einer Migration zu erhalten oder zu ersetzen gilt. Im präsentierten Projekt wird gezeigt, wie neben dem Ersatz eines Rechners, aber ohne sofortigen Ersatz der darauf ablaufenden Programme (Assembler- PL/1) eine schrittweise Migration mit 'C' und 'C++' vollzogen wurde, bis die Applikation von anderen Applikationen über eine offene, CORBA-konforme Schnittstelle zugänglich wurde.

Was ist Migration?

Unter Migration versteht man im Ursprung des Wortes (lat. migratio) die Auswanderung, den Wegzug. Obwohl man sich vielleicht gerne von älteren Systemen trennen möchte und vorzugweise "auswandern" würde, prägen folgende Faktoren die Migration von bestehenden Systemen:
Erhaltenswerte Komponenten bewahren, verbessern (renovieren, sanieren) dagegen wertlose Komponenten ablösen und durch neue ersetzen.
Bei System-Migration wird selten der Ort gewechselt, sieht man einmal von einer Dezentralisation ab, hingegen findet eher eine Abwanderung in eine neue Technologie statt. Der Effekt ist in etwa der Gleiche, man geht einen neuen Weg und betritt neues Gebiet mit allen Vor- und Nachteilen.
Ist Migration das Thema, dann ist auch sofort von Legacy (Altlast, Erbschaft) die Rede, dies häufig auch im Zusammenhang mit Mainframe-Komponenten die 15 Jahre und älter sind.
Von jungen Informatikern zu gerne ignoriert, stellen diese Legacy-Komponenten für den Betrieb des täglichen Geschäftsablaufs den lebenswichtigen Kern dar. Weitgehend fehlerfrei und den grössten Teil der Kernanforderungen abgedeckt ist über die Zeit gesehen das positive Fazit solcher Komponenten. Auf der negativen Seite stehen ungenügende Flexibilität, veraltete Technologie, monolythischer Aufbau und ebenfalls "verflossenes" Know-How über deren Aufbau.
Wo und wie solche Komponenten erhalten, ersetzt oder abgeändert werden sollen, lässt sich nicht als einfache Kochbuchregel definieren. Man kommt nicht darum herum eine genaue Analyse (mining process) über das abzulösende System zu machen. Mit grosser Sicherheit werden Legacy-Daten-Komponenten sich nicht einfach ersetzen lassen und müssen mit in den Migrations-Prozess eingeschlossen werden. Auch nur unbedeutend einfacher lassen sich dazugehörende, in langer Erfahrung erstellte und erprobte Funktionen ersetzen. Eine individuelle Auswertung aller Faktoren ist angesagt, welche aber immer noch einen guten Teil an Überraschungs-Potential beinhaltet.

Eine Migration hat häufig folgende Anforderungen:

- Eine "Big Bang"-Ersatzstrategie ist unerwünscht.
- Der Betrieb darf nicht gestört werden
- Möglichst keine Anpassung von bestehenden, zentralen Legacy-Komponenten sollte erfolgen.
- Ein Parallel-Betrieb von Alt- und Neu-System sollte für eine beschränkte Zeit möglich sein.
- Der Benutzer - an alte Funktionen des Systems gewöhnt - sollte nur positive Veränderungen bemerken.
- Eine Migration wird oft zu spät begonnen und muss daher möglichst schnell abgeschlossen werden.
- Performance-Einbussen müssen vermieden werden
- Die Komplexität des neuen Systems sollte die Betreibbarkeit nicht erschweren.
- Verbesserte Qualität und gewährleistete Wartbarkeit.

Eine Disziplin, welche diese Anforderungen vollständig erfüllt, darf sich uneingeschränkt Software-Engineering nennen, und muss von mit allen Wassern gewaschenen Fachkräften beherrscht werden.
Migration ist keine Disziplin, welche auf den Umstand "Start auf der grünen Wiese" zählen darf, sondern beinhaltet oft viele Kompromisse und angemessene Pragmatik, fordert aber auch Erfindergeist und Durchhalte-Wille, um das Ziel zu erreichen. Unerwartete Ereignisse, die an einen Wassereinbruch im Tunnelbau erinnern, fordern dann noch jeden Software-Ingenieur aus der Reserve. Das ist Migration.

Projektszenario

Nachfolgend wird ein über mehrere Jahre dauernder Migrationsprozess in einem Projekt bei der Schweizer Telecom beschrieben. Es handelt sich um ein zentrales Informationssystem mit allen Abonnenten-Daten (Telefon-Teilnehmer-Anschlüsse), welche von siebzehn regional verteilten Telefondirektionen verwaltet werden. Die Mutationen werden von ca. 2500 Endgeräten aus durchgeführt.
Das System hat sich über die letzten 15 Jahre entwickelt und natürlich einige Technologie-Epochen durchgemacht. Dank den eher technisch aber systematischen Denkweise der Entwickler wurde das System und modular entwickelt. Man kann durchaus sagen, dass die Systemarchitektur schon einige wesentliche Eigenschaften eines ausgewachsenen Client/Server-Systems hatte, wenn die Implementation auch auf propriäteren Plattformen erfolgte. Die Migration wurde dadurch einiges einfacher.

Ausgangslage

Als zentrales Informationssystem steht ein auf MVS/IMS basierter Mainframe im Einsatz. Die Transaktionen wurden früher über Standleitungen (heute über das Telecom-Netz) zu einem dezentralen, in den regionalen Direktionen plazierten Minirechner (IBM Series/1 mit RPS-Betriebssystem) geleitet. In diesem dezentralen Rechner wurden die vom Host eintreffenden Meldungen mit den lokalen Bildschirm-Masken-Vorlagen gemischt und an die an diesem Rechner angeschlossenen Bildschirme (Borroughs-Bildschirme mit Seitenspeicher) ausgegeben. Die im Bildschirm bearbeiteten Informationen wurden vom dezentralen Prozessor wieder extrahiert, komprimiert und an die Host-Applikation als Meldung zurückgesandt. Die Programmierung auf dem Mainframe erfolgte in PL/1 auf den dezentralen Rechnern in PL/1 und S/1-Assembler.
Weiter hatte der Vorprozessor, auf dem ein Teil der Applikation lief, zur Aufgabe die Zugriffsrechte des Benutzers zu kontrollieren und die Plausibilität der Eingabedaten zu sichern. Ebenfalls wurden einige Funktionen für den Benutzerkomfort dezentral implementiert. Zu einem späteren Zeitpunkt wurden die Bildschirme durch proprietäre Microcomputersysteme ersetzt und die Büro-Funktionalität für das Erstellen von Dokumenten ergänzt. Das Protokoll für die Bildschirme wurde aber beibehalten.
Dann kam der Zeitpunkt, zu welchem der Lieferant des dezentralen Minirechners die Herstellung einstellte und schon bald darauf auch die Wartung so stark einschränkte, dass ein kontinuierlicher Betrieb nicht mehr 100% sichergestellt werden konnte. Das Faktum war klar: Die Hardware musste ersetzt werden und die Kosten-Folgen für den gleichzeitigen Ersatz der Applikations-Software wurden zeitlich und kostenmässig als sehr umfangreich beurteilt.

Migriertes System

Das migrierte System zeigt sich heute wie folgt:

Das dezentrale System ist durch einen RISC-Rechner (RS6000) mit UNIX (AIX) ersetzt worden.

Die dezentrale Software läuft heute in identischer Form auf einer Emulation der alten Hardware auf dem UNIX-System (Geschwidigkeits-Steigerung Faktor 5).

Die Applikationssoftware erfuhr keine Aenderung

Die Kommunikation vom dezentralen System zum Mainframe erfolgt heute über TCP/IP statt SNA.

Die Mikrorechner-Systeme für die Bildschirme sind heute über TCP/IP und BSD-Sockets angeschlossen.

Benutzer mit Standard-PCs können heute über eine VT320-Terminal-Emulation und eine Emulations-Software über die dezentralen Systeme auf die Applikation zugreifen.

Eine CORBA-basierte Schnittstelle unter Orbix (ORB von IONA) auf dem dezentralen System erlaubt beliebigen Applikationen den Zugriff auf die Host-Applikation mit einer Datenstruktur-Abstraktion statt einer Bildschirm-Struktur.

Eine Applikation mit GUI auf den Standard-PCs (WIN/95) löst die Mikrosysteme ab und greift über die CORBA-Schnittstelle auf die Applikations-Daten zu. Diese Applikation löst auch die bisherigen 24x80 Bildschirm-Masken ab.

Sobald alle Mikrosysteme und Bildschirm-Emulationen auf den PCs abgelöst sind, kann auch die S/1-Emulation und dezentrale Software abgelöst werden, was wiederum eine Durchsatzsteigerung und eine Komplexitätsminderung ergeben wird.

Migrationsschritte

Von der unter Ausgangslage beschriebenen System bis zur heutigen Systemkonstellation mussten mehrere Schritte durchgeführt werden.

Ersatz der Hardware

Der Ersatz der Hardware wurde möglich, indem man in der USA eine in 'C' geschriebene Emualtions-Software fand, welche unter UNIX lauffähig war. Diese virtuelle S/1-Maschine emulierte nicht nur die Hardware sondern sogar das darauf ablaufende Multitasking-Betriebssystem RPS. Die dezentrale, für S/1-Rechner übersetzte Software konnte unverändert auf der Emulation ausgeführt werden. Auch wenn der Rechner natürlich von einer anderen Leistungsklasse war als eine Series/1 und durch die Emulation prozessormässig voll ausgelastet wurde, konnte trotzdem eine 5-fache Geschwindigkeitssteigerung erreicht werden. Das Hardware-Problem war vorerst einmal, wenn auch nicht für alle Zeiten gelöst.

Integration in die bestehende Umgebung durch IPC

Wie oben erwähnt handelt es sich beim RPS-Betriebssystem um ein Multitasking-System, welches von der Applikation auch voll genutzt wurde. Die verschiedenen Tasks kommunizierten untereinander auf der Original-Maschine via Meldungen, welche von einem Communication-Manager verwaltet wurden. Dieses Stück Software war nicht mehr verfügbar und musste mittels einem individuell erstellten Message Handler (MSH) in die Emulations-Software VS1 (Virtual Series/1) eingebaut werden. Dieser Message Handler wurde so entworfen, dass nicht nur RPS-Tasks miteinander kommunizieren konnten, sondern auch externe UNIX-Prozesse über UNIX IPC Queues via MSH mit den RPS-Tasks transparent verbunden waren. Dieser Mechanismus erlaubte nun die Anbindung an die Terminal-Emulationen und als Terminal-Server dienenden Mikrocomputer sowie die SNA-Verbindung zum Host über UNIX-Tasks zu lösen. Damit erkaufte man sich einen weiteren Vorteil: man konnte damit einen sprachenmässigen Technologie-Schritt machen, indem 'C++' für die externen UNIX-Tasks als Sprache eingesetzt werden und gleichzeitig das Design (inkl. Code-Generierung) auf einem CASE-Tool durchführen konnte. Zudem konnte man mit dieser komponentenorientierten Architektur eine höhere Flexibilität erreichen. Nach diesem Schritt konnten die Systeme (es waren 54 Systeme) in Produktion gehen und die alten Systeme ablösen. Das grösste Risiko, der Hardware-Ausfall war damit abgewendet, obwohl man sich im Klaren war, dass dies nur ein Zwischenschritt sein durfte, da ja auch die Wartung der dezentralen, zum Teil in Assembler geschriebenen Software wegen schwindendem Know-How immer schwieriger wurde. Ebenfalls kam ein zwar weniger akutes, doch potentielles Problem des Ausfalles der Terminal-Emualtions-Hardware auf den Mikrorechner dazu. Dieses konnte teilweise durch einen weiteren UNIX-Task (Terminalemulations-Frontend) zum VS1 und einer VT320-Terminal-Emulation (über telnet) auf dem Standard-PC erfolgen. Das alte "Look and Feel" im Zeitalter der glimmernden PC-Oberflächen aber blieb.

Oeffnung des Systems durch CORBA und GUI-basierte Client-Applikation

Wie bereits erwähnt, konnte die gerettete Legacy-Komponente nicht weiterhin so betrieben werden, und man musste sich den nächsten Schritt für die Zukunft überlegen. Als Restriktion, die nach wie vor aus Kapazitätsmangel bestehen blieb, war es weiterhin ein Tabu die Mainframe-Applikation in der Grundstruktur zu ändern. Trotzdem wollte man ein moderneres Erscheinungsbild auf den Benutzer-Rechnerstationen erhalten. Gleichzeitig galt es als Anforderung sich von der relativ starren Bildschirm-Transaktions-Struktur zu lösen und ein logischere Abstraktions-Ebene einzubauen. Dabei kam der Vorschlag CORBA einzusetzen, da man sich in

der IT-Organisation kurz davor entschlossen hatte auf die CORBA- statt auf die DCE-Schiene für die zukünftige Applikations-Entwicklung zu setzen.
Als erstes wurde ein technischer Prototyp erstellt, welcher durchgängig von der Client-Software über die CORBA-Schnittstelle, die virtuelle Maschine zur Mainframe-Applikation einen vorbestimmten Geschäftsfall-Ablauf ermöglichte. Dieser Prototyp konnte in 2 Monaten erstellt werden, und es konnte damit bewiesen werden, dass man konzeptionell wie technisch auf einem richtigen Pfad war. Im Applikationsserver setzte man direkt auf ein virtuelles Bildschirm-Image auf, welches dann als eine Datenstruktur dem Client gegenüber abstrahiert wurde. Mit Konversionsroutinen wurden die Host-Masken auf Tabellen konvertiert, welche der Client-Software die Feldzugriffe der jeweiligen Datenstruktur erlaubte. Dabei wurde weder auf Durchsatz noch auf Netzbelastung geachtet, welche durch relativ einfache aber häufige Schnittstellen-Aufrufe (getField/putField) recht hoch wurde. Gleichzeitig machte man Erfahrungen wie sich Inkonsistenzen der Client- und Serverseite bei den Zugriffstabellen auswirken. Alle diese Aspekte konnten bei der Architektur und beim Design der produktiven Datenschnittstelle berücksichtigt werden und waren entsprechend wertvoll.
Für die Entwicklung der Client-Software setzte man auf Visual C++ und MFC. Um die für den vorgegebenen Geschäftsfall benötigten Host-Masken schnell und genau realisieren zu können, konvertierte man die Original-Masken in das Ressourcen-Format von VC++ und konnte so im Masken-Editor (Visual C++) darauf aufbauen.

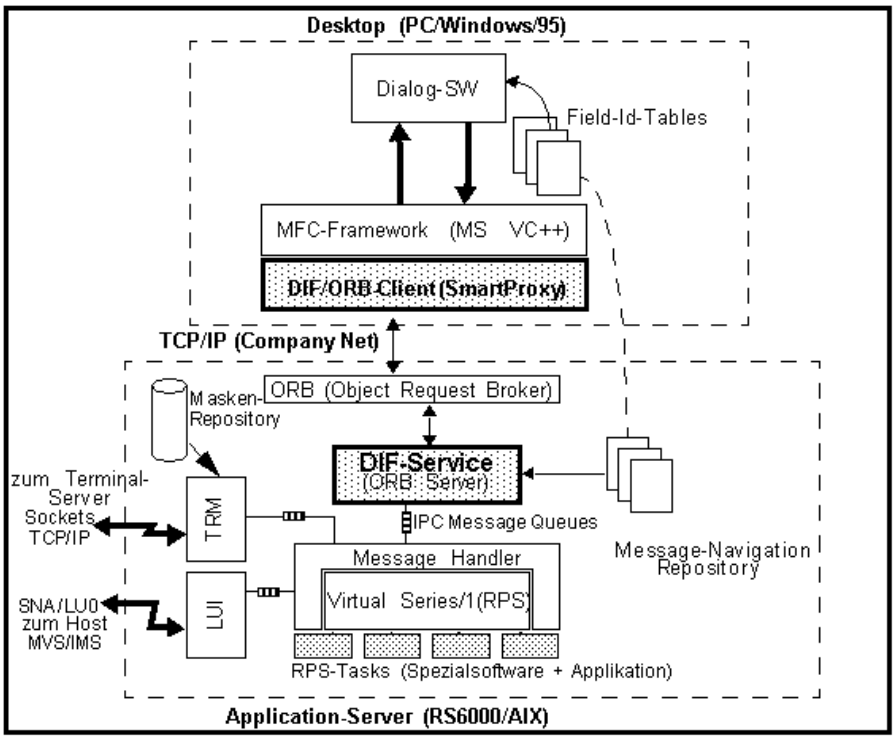
Ersatz der virtuellen Maschine und Spezialsoftware

Als nächster, geplanter Schritt, nach der Realisation der Datenschnittstelle und dem Ersatz aller proprietären Endgeräte durch PCs und der Terminal-Emulationen durch die GUI-basierte Applikation, kann mit dem Redesign resp. dem Ersatz der virtuellen Maschine begonnen werden. In diesem Schritt muss man sich klar werden, ob die Host-Applikation weiterhin unangetastet bleiben soll oder ob auch hier die Maskenorientiertheit schrittweise in Datenorientiertheit umgewandelt werden soll. Weiter wird zu diesem Zeitpunkt CORBA auch auf der MVS-Welt verfügbar sein (z.B. Orbix von IONA), was dann das Endziel der Migration darstellen könnte.

Architektur und Implementation der Datenschnittstelle

Die Architektur der Datenschnittstelle beinhaltet folgende Komponenten (Fig. 1):

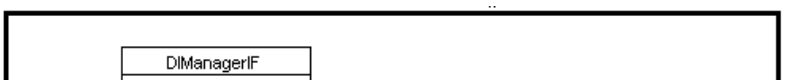
Fig. 1: System-Architektur der Datenschnittstelle

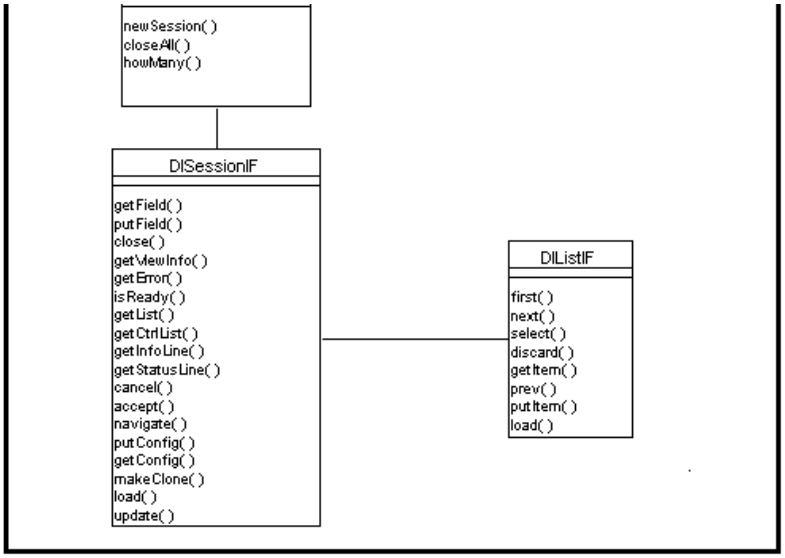


CORBA basierte Daten-Schnittstelle

Die Datenschnittstelle wurde aufgeteilt in drei Interfaces - DIManagerIF (Session-Factory), DISessionIF (virtuelle Verbindung zum Host) und DIListIF (Listenabstraktion für über mehrere Transaktionen verknüpfte Tabellen) entworfen und realisiert. Administrations-Methoden für Caching wurden darin ebenfalls definiert. (Fig. 2)

Fig. 2: Vereinfachtes Datenschnittstellen-Modell.





Einige Operationen mit applikatorischer Bedeutung (z.B. Konfiguration, SignOn, Funktionstasten u.s.w.) wurden abstrahiert, damit die frühere Bildschirm-Orientiertheit abgelöst werden konnte. Die Client-Vorstellung dieser Schnittstelle sollte eher diejenige von vernetzten Datenstrukturen sein (vergleichbar mit Netzwerkdatenbank) worin navigiert und zugegriffen wird. Dies ist noch nicht die ideale Lösung. Man darf dabei aber nicht vergessen, dass hinter dem Interface immer noch eine Mainframe-Applikation steht, welche den Ablauf und die Reihenfolge weitgehend bestimmt. Zumindest bietet es die Möglichkeit eine logische Abkapselung zu schaffen, mit welcher nachher ein Redesign überhaupt ermöglicht werden kann.

Client-Seite Caching des Dateninterfaces

Auf der Client-Seite wurden die notwendigen Funktionen mittels "call back", asynchrone Aufrufe und die Integration zum lokalen Security-System ins Proxy (respektive Smart Proxy) so integriert, dass es dem Client-Programmierer einzig das Dateninterface erscheint und einfach zu benutzen ist. Für die Implementation des zugehörigen Clients war Visual C++ 4.0 und der MFC-Klassenbibliothek bestimmt.
Alle aktuellen View-Daten werden beim ersten Zugriff auf ein Feld zum Client transportiert, und alle weiteren Zugriffe und Mutationen werden dann lokal auf dem Client gemacht. Für die Synchronisation mit dem Server werden mit Client-"Call Back"-Schnittstellen, die beim Server registriert sind, implementiert. Dadurch kann unnötiger Netzverkehr verhindert werden.

Server-Seite und Implementation des Dateninterfaces

Auf der Serverseite wurde C++ unterstützt durch Orbix und RWTools.h++ verwendet. Die Protokollabhandlung zum VS1 konnte als Klassenframework von früheren Komponenten übernommen werden. Gegenüber dem VS1 verhält sich das Dateninterface darum identisch wie der Terminalmanager (TRM) aus einem der vorangegangenen Migrationsschritte. Dadurch musste an der Spezialsoftware, die auf der VS1 abläuft, keine Aenderung vorgenommen werden. Es können mehrere Dateninterfaces pro VS1 im Einsatz sein, womit eine Lastverteilung oder organisatorische Aufteilung eingerichtet werden kann.
Multithreading wird so eingesetzt, dass es pro Session-Objekt (entspricht DISessionIF) einen Thread gibt, da pro Client-Programm mehr als eine Session aktiv sein kann. Das Anmelden auf dem Host (Sign On) erfolgt automatisch durch den Server, beim Eröffnen einer Session. Die Client-Seite hat durch den Einsatz des Smart-Proxys damit nichts zu tun. Dadurch konnte man auch der Anforderung gerecht werden, dass sich ein Benutzer an einem PC nicht für jede von ihm verwendete Host-Applikation mit Benutzer-Identifikation und Passwort zu identifizieren hat (integriert mit darunterliegendem Security-System).
Für die produktive Version des Servers wurde direkt auf den Host-Meldungen aufgesetzt, was eine weitere Performance-Verbesserung brachte und man ein aus 800 Dateien bestehendes Maskenrepository durch ein Meldungs-Repository aus zwei Dateien ersetzen konnte. Dies wurde erreicht, indem man die auf dem Mainframe verwalteten Bildschirm-Maskenlayouts und den dazugehörenden Meldungsstrukturen (PL/1-Records) mittels verschiedenen Konversionsprogrammen zu einer Konfigurations-Tabelle (Message-navigation-Repository) konvertierte. Bei dieser Konversion, wurden ebenfalls benötigte Informationen herausgefiltert, welche die Konfiguration für die Listen-Bearbeitung (Browsing).
Die in der ursprünglichen Version der Applikation auf mehreren Bildschirmseiten verteilten Listen, Tabellen und Verzeichnisse werden durch den Server mittels DIListIF zu einer abstrahierten Liste ohne Seitenbegrenzung umgewandelt. Damit sollte auch die endgültige Ablösung der Legacy-Komponenten vereinfacht werden, ohne dass auf der Client-Seite unnötig viel ändern sollte.
Der Dateninterface-Service ist weiter so ausgelegt, dass im Betrieb Zustandsinformationen abgefragt oder registriert werden können (Logging). Durch die laufend ändernden applikatorischen Anforderungen und demzufolge Aenderungen, ist der Service einfach konfigurierbar (z.B. einfacher Wechsel des Meldungs-Repository).

Virtueller Applikations-Server (migrierte Komponente VS1)

Für den Mainframe-Applikationszugriff wird weiterhin die VS1 mit Ihren proprietären Programmen eingesetzt. Dadurch kann die Ablösung der bestehenden Infrastruktur (Bildschirme, Mikrocomputer, Software) durch PCs, welche in 17 Lokationen und auf rund 3000 Geräten vorgenommen werden muss, in Etappen erfolgen.

Object Request Broker

Als CORBA-Umgebung wurde Orbix von IONA eingesetzt, dies sowohl auf der Client-Seite wie auf der Server-Seite. Durch die OLE-Bridge von Orbix, die zwar hier nicht eingesetzt wurde, könnte somit die Datenschnittstelle auch von VB- und anderen OLE-treuen Komponenten eingesetzt werden. Auf der Serverseite wird die multithreaded-fähige Version des Brokers eingesetzt. Die Erfahrungen mit dem Einsatz der CORBA-Middleware waren sehr gut und haben verglichen mit dem Aufwand, der z.B. bei Protokollen über Socket-Verbindungen aufgebracht werden muss, sowie an Zuverlässigkeit eine grosse Einsparung und Qualitätsverbesserung gebracht.

Schlussfolgerung

Migration bedeutet meistens einen Kompromiss einzugehen. Eine Beurteilung ist darum entsprechend subjektiv, ob eine Migration erfolgreich war oder ob es eine bessere Lösung gegeben hätte. Die Migration wird so erfolgreich sein, wie gut das migrierte System

weiter im Betrieb läuft und sich wiederum weiter migrieren lässt. Eine andere und vielleicht bessere Lösung lässt sich immer finden, wenn man einen Migrationspfad beschritten hat und Erfahrungen damit gesammelt hat. Wichtig scheint dem Autor, dass mit jedem Migrationsschritt eine allgemeinere, logischere Abstraktionsstufe geschaffen werden muss, damit eine wirkliche Verbesserung vorgewiesen werden kann.

Die Kompromisse im vorliegenden Projekt waren:

- dass sich Applikationsarchitektur der Mainframe-Seite (hier Bildschirm-Orientiertheit) sich nicht ganz verstecken liess.
- dass eine Client-Abhängigkeit (datenmässig) vom Server nach wie vor vorhanden ist (Feld-Identifikationen) und eine Laufzeit-Versionskontrolle notwendig machte.
- dass man eine erhöhte Komplexität in Kauf nahm, die bei einem Redesign dann wieder abgebaut werden muss.
- dass man auf der Client-Seite Visual C++ einsetzte und dadurch den Client relativ schwergewichtig machte. (Alternative JAVA + Browser?).
- Weitere Erkenntnisse aus dem Projekt waren:
- Der geeignete Migrations-Pfad muss für jedes System individuell angeschaut werden, wenn sich mit der Zeit sicher auch hier bestimmte Muster herauskristallisieren lassen.
- CORBA-Tools sind eine Hilfe und problemlos einsetzbar. Es wird in der Regel zuviel über die Werkzeuge diskutiert, statt dass man sich mit dem Design von verteilten Applikationen auseinandersetzen würde.
- Die Definition einer übergeordneten Architektur und der Interface-Frameworks für CORBA-basierte Applikationen innerhalb der IT-Organisation hat sich als sehr positiv erwiesen.
- Kommunikation zwischen Mainframe-, Server- und PC-Client-UI-Entwickler ist ein lebenswichtiger Punkt für den Erfolg des Projektes.
- Der Wartbarkeit und Betreibbarkeit von verteilten Applikationen muss während der Entwicklung Rechnung getragen werden.

Referenzen

[1]
Comer Douglas E., David L. Stevens, "Internetworking with TCP/IP Volume III", Client/Server-Programming and Applications BSD Socket Version, Prentice Hall International Editions, 1993, ISBN 0-13-020272-X.

[2]
Meyer Bertrand, Objektorientierte Softwareentwicklung, Carl Hanser München, Prentice Hall, ISBN 3-446-15773-5, 1990

[3]
Mowbray Th, Zahari R, "Essential Corba: System Integraton with Distributed Objects", John Wiley&Sons, ISBN 0-471-10611-9, 1995

[4]
Koch, A., "Interprozesskommunikation in verteilten Systemen", Offene Systeme (1995) Band 4/ Nr. 2, Mai 95, S. 70-75, Springer Verlag International Heidelberg.

[5]
Koch, A., "Objektorientierte Entwicklung verteilter Applikationen", Offene Systeme (1993) Band 2/ Nr. 4, November 93, S. 201-209, Springer Verlag International Heidelberg.

[6]
Object Management Group, "Object Management Architecture Guide", Rev. 2.0, Sept. 92, John Wiley&Sons, Inc, New York, OMG TC Document 92.11.1, ISBN: 0-471-58563-7.

[7]
Orbix 2, Programming Guide, Release 2.0, November 1995, IONA Technologies Ltd.

[8]
Orbix 2, Reference Guide, Release 2.0, November 1995, IONA Technologies Ltd.

[9]
W. Richard Stevens, "UNIX Network Programming", Prentice Hall Software Series, ISBN 0-13-949876-1.