Realisierung einer Client/Server-Anwendung mit CORBA und Java unter Berücksichtigung bestehender C++-Komponenten[1]

Klaus Beschorner[1], Wolfgang Rosenstiel[2]

[1, 2] Universität Tübingen, Arbeitsbereich Technische Informatik, Sand 13,
D-72076 Tübingen
{beschorn, rosenstiel}@informatik.uni-tuebingen.de

[1] iT media Consult GmbH, Mörikestraße 11,
D-70178 Stuttgart

Kurzfassung. Dieses Papier stellt einige Vorgehensweisen zur Realisierung einer Client/Server-Anwendung mit CORBA und Java vor. Dazu gehört die Programmierung robuster und benutzerfreundlicher Java-Anwendungen in Form von Applets und Applikationen sowie die Wiederverwendung von bestehendem Code in Java- und CORBA-Objekten. Alle vorgestellten Ansätze sind erfolgreich im Rahmen von [1] in einer prototypisch realisierten Internet/Intranet-Anwendung umgesetzt worden, die ebenfalls vorgestellt wird. Die gefundenen Vorgehensweisen sollen den Entwicklungsprozeß von anderen Anwendungen ähnlicher Natur vereinfachen.

1 Einleitung

Eines der wohl größten Probleme in der Informationsverarbeitung ist der fehlende Konsens darüber, welche Hardwareplattformen, Netzwerkprotokolle, Betriebssysteme, Programmiersprachen und Anwendungen innerhalb eines Unternehmens[2] verwendet werden sollen.
Angesichts der Tatsache, daß bestehende Investitionen in Hardware und Software (Legacy Systeme) bis heute im Einsatz sind und dies auch in Zukunft sein werden, ist ein Konsens auch in Zukunft nicht zu erwarten.
Die geschilderte Situation führt innerhalb von Unternehmen zu heterogenen Systemen, die hohe Kosten verursachen. Die Herstellung einer einheitlichen Kommunikationsinfrastruktur stellt eine Herausforderung dar, da z.B. hostbasierte Anwendun-

[1] Die in diesem Papier dargestellten Verfahren wurden im Rahmen eines Projekts bei der iT media Consult GmbH entwickelt. Wir bedanken uns für die gute Zusammenarbeit.

[2] Die Ausführungen gelten auch für Behörden, Ämter und andere Institutionen, die Informationstechnologie einsetzen.

gen von Desktop-PCs aus genutzt werden müssen und zusätzlich Programme, die mit verschiedenen Programmiersprachen implementiert wurden, kommunizieren müssen.
Die Popularität des World Wide Web (WWW) erfordert nun die Präsenz im Internet, was o.g. Probleme zusätzlich verschärft. Die zugehörigen Schlagworte sind Electronic Banking, Electronic Commerce und Telearbeit. Die Diskussion ist aber nicht ausschließlich auf betriebswirtschaftliche Anwendungen beschränkt. So können technische Geräte und Anlagen ebenso mittels eines WWW-Browsers gesteuert und überwacht werden.
Für die Realisierung von WWW-Anwendungen ist die Programmiersprache Java durch ihre Portabilität und Integration in WWW-Browser unverzichtbar geworden. Um Anwendungen für das Internet/Intranet zu realisieren, müssen bestehende Systeme (C++, COBOL, usw.) mit neuen Systemen (Java) integriert werden und sowohl unternehmensintern, als auch unternehmensextern verfügbar gemacht werden.
In diesem Zusammenhang verspricht die Common Object Request Broker Architecture (CORBA) [2] Lösungen. Der sprachunabhängige Standard ist darauf ausgelegt, die Kommunikation zwischen unterschiedlichen Hardwareplattformen, Betriebssystemen und Programmiersprachen auf elegante Weise zu ermöglichen.
Dieses Papier beschreibt grundlegende Probleme, die zu lösen sind, wenn eine bestehende Anwendung mit einer auf CORBA und Java basierenden Architektur ausgestattet werden soll. Hierzu wird in Kapitel 2 eine bestehende Client/Server-Anwendung vorgestellt, die zur Identifikation von Problemen dient. Ausgehend von den identifizierten Problemen werden in Kapitel 3 Vorgehensweisen beschrieben, um diese zu überwinden. In Kapitel 4 wird ein Prototyp vorgestellt, der die in Kapitel 2 dargestellte Anwendung unter Verwendung der gefundenen Lösungsansätze implementiert.

2 Problemstellung

Das in Abbildung 1 dargestellte Agentur-Informationssystem für Versicherungen wurde im Rahmen von [3] mit „IBM Visual Age for C++“ auf Basis einer Kommunikation mit Sockets realisiert.

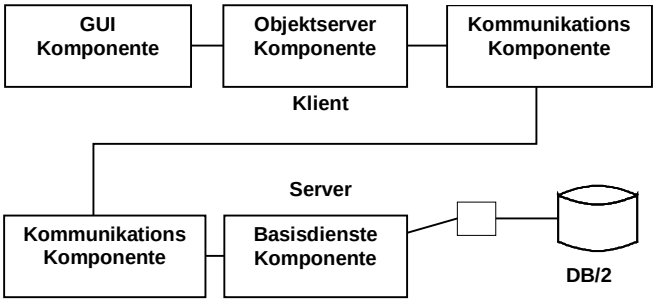

Abbildung 1. Agentur-Informationssystem

Nachfolgend sind die Aufgaben der einzelnen Komponenten dargestellt:

- *Grafische Benutzeroberfläche (GUI)* aus mehreren Masken, die zur Präsentation und Verarbeitung von Informationen erforderlich sind.
- *Objektserver* zur Bereitstellung der Anwendungslogik des jeweiligen Anwendungsbereichs.
- *Datenbasen* zur Bereitstellung von Informationen für das Gesamtsystem, hier eine IBM DB/2-Datenbank mit ca. 10000 Kundendatensätzen gefüllt.
- *Basisdienste* zur Bereitstellung weiterer notwendiger Dienste, wie z.B. Events.

Die vorliegende Anwendung soll unter Wiederverwendung der Funktionalität des Objektservers mit CORBA und Java realisiert werden. Dabei treten neben der Integration eines Object Request Brokers (ORB) insbesondere die folgenden Probleme auf:

- die Programmierung robuster und benutzerfreundlicher Java-Anwendungen, insbesondere Java-Applets mit *mehreren* Masken,
- die Kapselung von bestehendem Code in Java- und CORBA-Objekte.

3 Problemlösungen

Die Realisierung, der in Abbildung 1 dargestellten Anwendung mit CORBA und Java erfordert eine Lösung der folgenden Probleme:

1. Die System-Komponenten müssen durch CORBA-Objekte ersetzt werden. Diese Objekte müssen teilweise bestehenden Code wiederverwenden und als Gesamtheit die selbe Funktionalität, wie vor der Umsetzung, bereitstellen. Eine mögliche Lösung wird in Abschnitt 3.1 vorgestellt und beruht auf zustandslosen CORBA-Objekten, die bestehenden Code kapseln.
2. Die vorhandene C++-GUI muß in Java nachgebildet werden, da sie im Internet und Intranet verfügbar sein soll. Dabei soll ein Applet und eine Applikation zur Verfügung stehen. Das Hauptproblem besteht dabei in der angemessenen Präsentation einer komplexen, aus mehreren Ein-/Ausgabemasken bestehenden Anwendung, innerhalb eines WWW-Browsers. Die in Abschnitt 3.2 vorgestellte Lösung schaltet die benötigten Masken innerhalb des Browsers um und berücksichtigt außerdem die zusätzlich notwendige Erstellung einer Java-Applikation.
3. Der bestehende C++-Code des Objektservers muß wiederverwendet werden. Aufgrund der Sprachunabhängigkeit von CORBA sollte dies problemlos möglich sein. Es zeigt sich aber, daß die unterschiedlichen C++-Compiler u.a. unterschiedliche Objekt- und Bibliotheksformate besitzen und damit die Wiederverwendung von bestehendem C++-Code mit einem C++-ORB verhindern, weil beide Bestandteile jeweils für unterschiedliche Compiler ausgelegt sind. Bei der Erstellung des hier vorgestellten Prototypen trat dieses Problem auf. Aufgrund der Möglichkeit, C++-

Programme in Java einzubinden, wurde der wiederzuverwendende C++-Code hier über einen Java-ORB in das Gesamtsystem eingebunden. Die Vorgehensweise in Abschnitt 3.3 beschreibt hierzu einige Ansätze zur eleganten C++-Code-Integration in Java und kann deshalb auch unabhängig von der Problemstellung in diesem Papier für Projekte, die ein solches Vorgehen erfordern, herangezogen werden. Das allgemeingültige Problem, bestehenden Code in CORBA-Objekte zu kapseln wird in Abschnitt 3.4 behandelt. Darüber hinaus zeigt die Integration des Monitor/Protokoll-Servers in das System (vgl. Abschnitt 3.1), daß ein reines C++-CORBA-Objekt mit Java-CORBA-Objekten problemlos kommunizieren kann.

4. Der Datenbankserver soll neu mit Java implementiert werden. Die hier betrachtete Problematik betrifft allerdings nicht den Datenbankzugriff mit Java selbst, sondern die Integration von diesen Routinen in ein CORBA-Objekt. Diese Frage stellt sich vor allem dann, wenn wie in diesem Fall, die Routinen von einem anderen Entwickler implementiert werden, der sich selbst nicht mit CORBA beschäftigt. Diese Problematik wird in Abschnitt 3.4 diskutiert.

3.1 Integration eines Object Request Brokers

Die in Abbildung 1 dargestellte Anwendung ist bereits in Komponenten aufgeteilt, die weitgehend erhalten bleiben sollen. Es ist daher naheliegend die Komponenten in CORBA-Objekte, wie in Abbildung 2 dargestellt zu überführen.

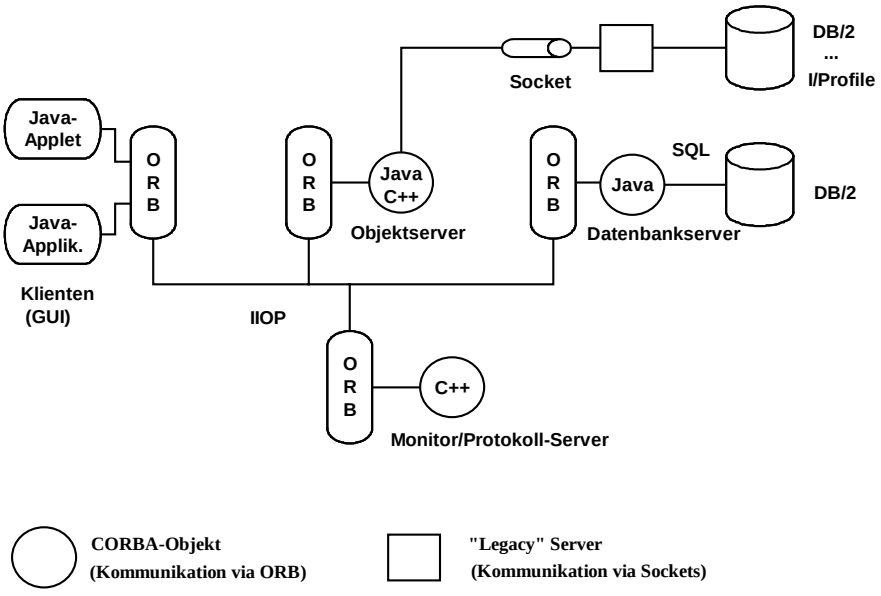

Abbildung 2. ORB-Architektur

Beschreibung der Komponenten:

- *Klienten (GUI)*: Java-Implementierung mit dem AWT 1.1.
- *Objektserver*: Java-Implementierung, die wiederzuverwendende C++-Funktionalität mittels JNI 1.1 [4] kapselt. Realisierung als Factory, d.h. jeder Klient erhält sein eigenes CORBA-Objekt und seine privaten Daten.
- *Monitor/Protokoll-Server*: C++-Implementierung, protokolliert und zeigt Anmeldevorgänge im System an.
- *Datenbankserver*: Java-Implementierung, kapselt getrennt entwickelte JDBC [5] Datenbankzugriffsroutinen.
- *Legacy-Server*: Deutet die prinzipiell mögliche Wiederverwendung bestehender Kommunikationsmechanismen an.
- *ORB*: Für Java-Komponenten: VisiBroker 3.0 für Java, für die C++-Komponente: VisiBroker für C++.

Nachfolgend werden einige der wichtigsten zu leistenden Lösungsansätze vorgestellt.

3.2 Programmierung von Java-Applets und –Applikationen

Bei Anwendungen, die über mehrere Bildschirmmasken verfügen, stellt sich die Frage, wie diese in einem Applet bzw. in einem Browser zu realisieren sind. Es ist zwar denkbar, mehrere Fenster durch ein Applet zu öffnen, Dialoge zu verwenden oder weitere Browser-Fenster zu öffnen, dies kann jedoch vom Benutzer als störend empfunden werden oder ihn dazu verleiten, den Browser zu schließen, was zum Beenden der Anwendung und zur Freigabe der durch das Applet belegten Ressourcen führt.
An eine Lösung für ein Applet-Design, das die o.g. Probleme löst, ist außerdem die Forderung nach Wiederverwendung des Codes in Java-Applikationen zu stellen.
Das hier vorgestellte Verfahren stellt eine Verallgemeinerung und Erweiterung des in [6] vorgestellten Designs von Applets dar. Die Lösung der Applet-Problematik erfolgt durch die Darstellung der Masken innerhalb *eines* Browser-Fensters, wobei die Masken entsprechend der Benutzerarbeitsschritte umgeschaltet werden. Bei der Applikation wird jede Maske innerhalb eines eigenen Fensters dargestellt, wodurch sich die Anwendung dem Benutzer wie eine herkömmliche Windows-Applikation präsentiert. Abbildung 3 gibt einen Überblick über das Verfahren.
Die Entwicklung eines Applets und einer Applikation vollzieht sich in den folgenden Aktivitäten:

- Erstellen der benötigten Bildschirmmasken, jeweils als eigene Panelklasse.
- Erstellen eines Fenster-Managers, der für die Darstellung der benötigten Maske verantwortlich ist (einmal als Applet, einmal als Applikation).
- Implementierung der durch die Bildschirmmaske erreichbaren Funktionalität.

- Verwendung der Panelklassen innerhalb eines Applets direkt.
- Erstellen von Frameklassen für die Applikation, die die Panelklassen enthalten.
- Implementieren spezieller Unterschiede zwischen Applet und Applikation.

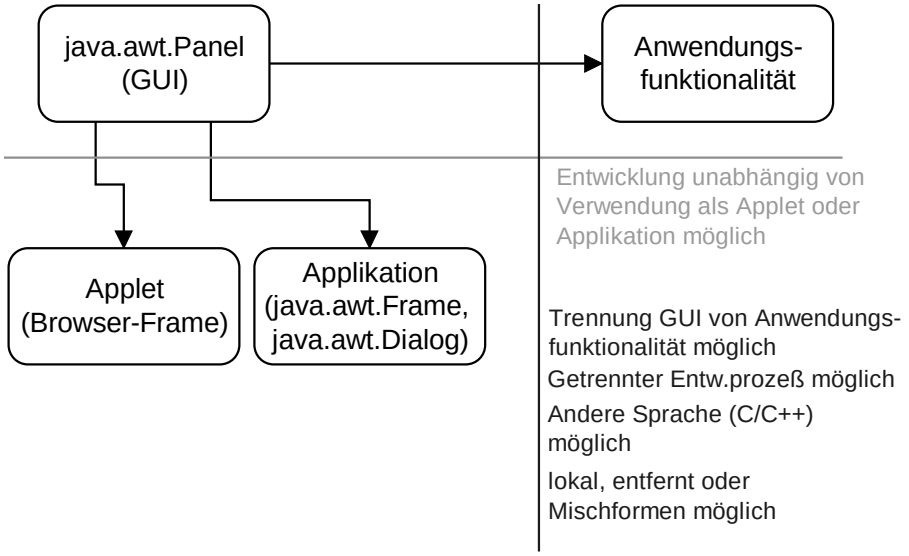

Abbildung 3. Java-Anwendungserstellung

Wird eine einzige Anwendung erstellt, die sowohl als Applet, als auch Applikation verwendet werden kann, sind Abfragen nötig, die feststellen, ob innerhalb eines Browsers oder von der Kommandozeile gestartet wurde. Je nachdem um welche Umgebung es sich handelt muß dann der zugehörige Code ausgeführt werden. Dies führt zu langsameren und größeren Anwendungen und wirkt sich vor allem bei Applets negativ aus, die auf den Rechner des Benutzers heruntergeladen werden müssen. Die hier gezeigte Vorgehensweise ermöglicht die Trennung von Applet und Applikation. Durch ein Anwendungsdesign können die Browser- und Applikationsspezifika in eigenen Klassen gekapselt werden. Durch den Austausch einer Klasse kann dann wahlweise eine Applikation oder ein Applet entstehen.

3.3 Einbindung von C++-Code in Java-Objekte

3.3.1 Gründe für die Einbindung von C++-Code

Die Einbindung von C++-Code in Java-Programme mittels Java Native Interface 1.1 (JNI 1.1) [4] kann aus Gründen, wie Performance, Wiederverwendung, hardwarenahe Programmierung und Einbindung von Tools und Bibliotheken, die für Java nicht exi-

stieren, sinnvoll sein. Für die Problemstellung in diesem Papier war die mangelnde Verfügbarkeit eines C++-ORBs für „IBM Visual Age C++“ ausschlaggebend.
Nachfolgend wird eine grundlegende Vorgehensweise zur Integration von bestehendem C++-Code in Java-Objekte vorgestellt, die den Vorgang der Integration möglichst einfach gestalten soll.

3.3.2 Vorgehensweise

Es ist zunächst festzulegen, welche Methoden im C++-Code zur Verwendung kommen sollen. Für die ausgewählten Methoden müssen korrespondierende Methoden in Java erstellt werden. Dabei sind die im C++-Code verwendeten Datentypen und Datenstrukturen auf Java-Datentypen und Java-Datenstrukturen abzubilden oder umgekehrt. Ist eine direkte Abbildung nicht möglich muß ein Wrapper erstellt werden, in dem die Datenstrukturen konvertiert werden.
Die Code-Integration soll möglichst einfach und übersichtlich gelöst werden. Als sinnvolle Lösung erweist sich die in Abbildung 4 dargestellte Methode, bei der die für das JNI erforderlichen Routinen und notwendige Umsetzungen zwischen Java und C++ in einer eigenen Wrapper-DLL enthalten sind. Der eigentlich wiederzuverwendende C++-Code befindet sich in einer eigenen Bibliothek.

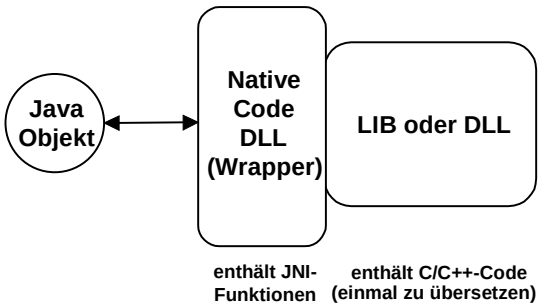

Abbildung 4. Vorschlag zur Einbindung von C++-Code in Java

Es ergeben sich die folgenden Vorteile:

- wenige Veränderungen am C++-Code und einmalige Übersetzung in eine Bibliothek,
- bei vorhandener Spezifikation kann der C++-Code als Black-Box betrachtet werden,
- verwendbar für Bibliotheken mit oder ohne Quellcode,
- JNI-Methoden und notwendige Konvertierungsfunktionen befinden sich in einem eigenen Programm.

Falls das in Java einzubettende C++-Programm verschiedene Funktionen anbietet, die immer mit der selben Datenstruktur arbeiten empfiehlt sich der in Abbildung 5 dargestellte Aufbau der Wrapper-DLL. Dabei existiert *eine* Funktion zur Konvertierung

einer Datenrepräsentation in Java nach C++ (`javaToCpp()`) und *eine* Funktion zum umgekehrten Konvertieren (`cppToJava()`).

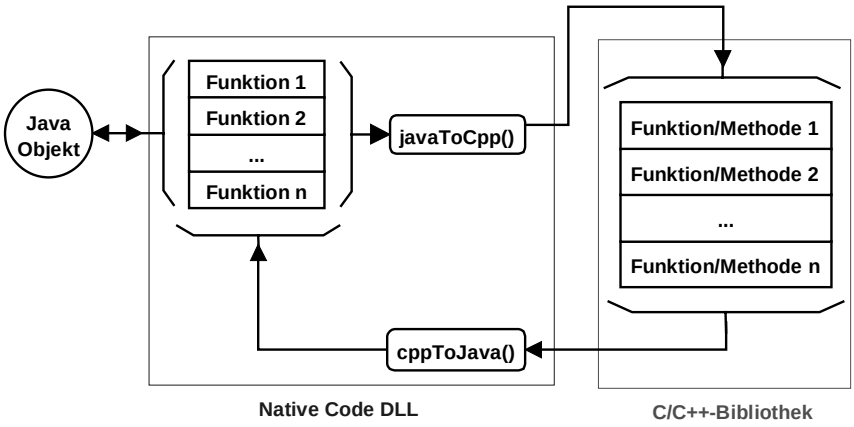

Abbildung 5. Konvertierung von Daten zwischen Java und C++

3.4 Kapselung von bestehendem Code in CORBA-Objekte

Unter dem Begriff „bestehender Code“ können Altanwendungen (Legacy Code), Bibliotheken von Fremdherstellern und Code aus anderen Projekten verstanden werden. Zusätzlich soll hier auch Code, der aufgrund einer Arbeitsteilung entsteht, um anschließend in ein CORBA-Objekt integriert zu werden, als bestehender Code aufgefaßt werden.
CORBA ist aufgrund seines Programmiermodells (u.a. Trennung Schnittstelle-Implementierung) und seiner Sprachunabhängigkeit besonders geeignet, bestehenden Code in CORBA-Objekte zu kapseln. Ein besonders wichtiger Aspekt ist in diesem Zusammenhang die mögliche Verfügbarkeit von bestehenden Anwendungen im WWW, ohne umständliche Maßnahmen, wie z.B. HTTP/CGI einplanen zu müssen.

3.4.1 Aufgaben

Hier muß eine Abbildung der Programmierschnittstelle des bestehenden Codes (Funktionen, Methoden, Parameter, Datenstrukturen, usw.) auf die IDL-Schnittstelle (Interface, Operationen, Datenstrukturen, usw.) des CORBA-Objekts erfolgen. Abbildung 6 verdeutlicht dies.
Die Umsetzung von Datenstrukturen entspricht im einfachsten Fall dem Language-Mapping (IDL → Zielsprache) der jeweils verwendeten Zielsprache. Bei komplexeren Datenstrukturen, die nicht direkt mit IDL beschrieben werden können und damit nicht dem Language-Mapping entsprechen, müssen Konvertierungen in einem Wrapper durchgeführt werden.

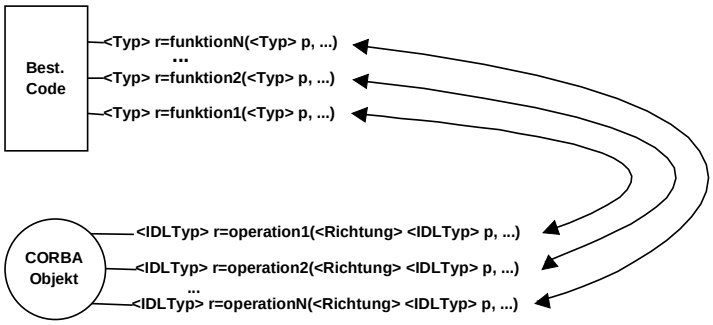

Abbildung 6. Abbildungsvorgang zwischen IDL und bestehendem Code

3.4.2 Vorgehensweise

Grundvoraussetzung ist, daß für den bestehenden Code Spezifikationen existieren sollten, die seine Funktionalität offenlegen.
Der evtl. erforderliche Wrapper-Code kann bei einem CORBA-Objekt in der Objektimplementierung, dem bestehenden Code oder in einem eigenen Programm angesiedelt werden. Als gute Lösung stellt sich die isolierte Programmierung des Wrappers gemäß Abbildung 7 heraus.

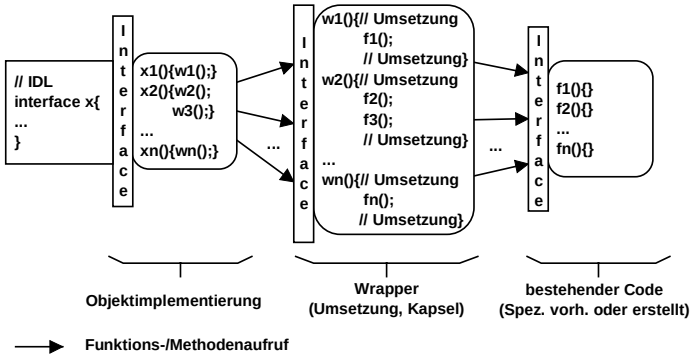

Abbildung 7. Isolierter Wrapper-Code

Dies führt zu einem klar nach Aufgaben getrennten Design. Die Aufgaben können z.B. auf mehrere Entwickler verteilt werden. Außerdem sind die einzelnen Komponenten weitgehend austauschbar.
Wird aus Sicht des CORBA-Programmierers Code für eine Objektimplementierung parallel von anderen Entwicklern erstellt, kann die Notwendigkeit eines Wrappers auf ein Minimum reduziert bzw. eliminiert werden. Die Code-Schnittstelle muß dabei IDL-konforme Datentypen und –strukturen entgegennehmen. Durch weitere Absprachen zwischen den Entwicklern kann die Übernahme von Code in CORBA-Objekte optimiert werden, womit die Erstellung von komplexen, aus vielen verteilten Objekten bestehenden Anwendungen unter Aspekten der Arbeitsteilung erfolgen kann.

4 Prototyp

4.1 GUI

Mit den in Kapitel 3 dargestellten Ansätzen wurde das Agentur-Informationssystem als Internet/Intranet-Anwendung realisiert. In Abbildung 8 sind drei der acht implementierten Anwendungsmasken in der Browser-Anwendung dargestellt. Zu jedem Zeitpunkt ist die dem Arbeitsschritt des Benutzers zugehörige Maske sichtbar. Die Knopfleiste bietet dem Benutzer zusätzlich eine Navigationsmöglichkeit durch die Masken.

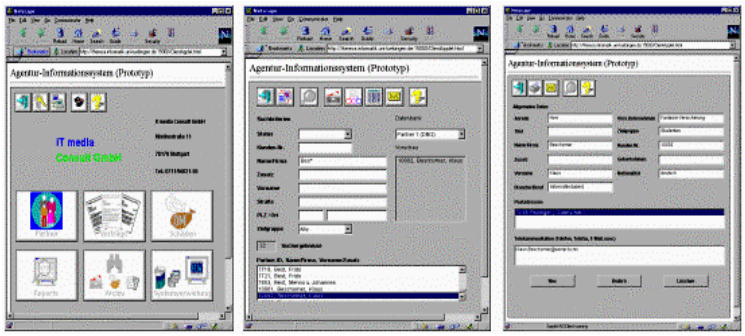

Abbildung 8. Applet: Hauptauswahl - Suchen - Bearbeiten

Im Gegensatz dazu stellt sich in Abbildung 9 die Java-Applikation wie eine herkömmliche Windows-Anwendung dar. Der Benutzer kann frei entscheiden, welche Fenster geöffnet bleiben. Zusätzlich ist eine Menü-Leiste verfügbar.

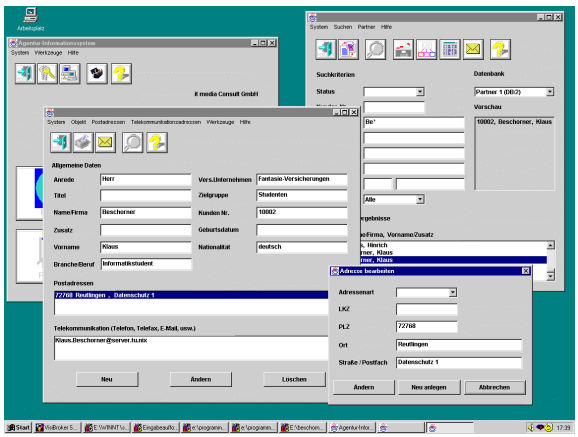

Abbildung 9. Java-Applikation unter Windows NT

4.2 Leistungsverhalten

Bei der Implementierung des Prototypen stand die Performance nicht im Mittelpunkt. Dennoch wurden Performance-Messungen vorgenommen, um einen Eindruck über das Leistungsverhalten der Anwendung zu gewinnen. Nachfolgend sind die Meßbedingungen dargestellt:

- Server: Pentium II/266 MHz, 64 MB Hauptspeicher.
- Klient: Pentium 60 MHz, 64 MB Hauptspeicher.
- Datenbank: DB/2 2.1 für Windows NT mit ca. 10000 Datensätzen.
- Netzwerk: 10 MBit/s Ethernet.
- Alle Java-Komponenten wurden mit einer Early Access Version eines Just In Time Compilers von Sun gestartet.

Die Ergebnisse für Suchanfragen nach Kunden mit unterschiedlicher Ergebnisanzahl sind in Abbildung 10 dargestellt. Übertragen wurde jeweils Datensatznummer, Name und Vorname.

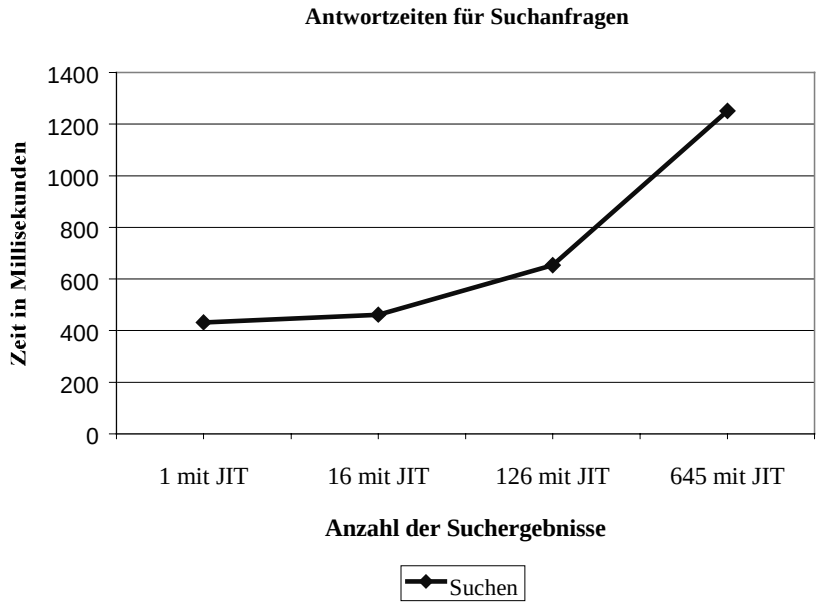

Abbildung 10. Zeitverhalten des Prototypen (1)

In Abbildung 11 sind die Zeiten für das Anlegen, Ändern und Löschen kompletter Kundendatensätze dargestellt. Die geringfügig höhere Zeit beim Anlegen ergibt sich aus dem Aufruf des mittels JNI wiederverwendeten C++-Codes und den dafür notwendigen Konvertierungsroutinen.

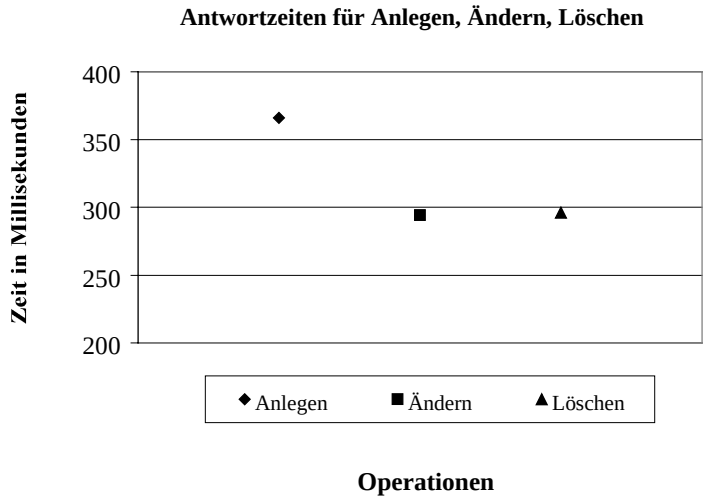

Abbildung 11. Zeitverhalten des Prototypen (2)

Die Übertragungszeit des Applets zum Klienten beträgt bei Nutzung des in Netscape Navigator ab Version 4 integrierten VisiBroker ORBs ca. 15 Sekunden.

5 Zusammenfassung und Ausblick

Das vorliegende Papier hat sich mit der Realisierung einer Client/Server-Anwendung mit CORBA und Java beschäftigt. Typische Anforderungen sind dabei anspruchsvolle Klienten, die mehrere Masken besitzen und im Internet/Intranet verfügbar sind sowie die Wiederverwendung von bestehenden Komponenten. CORBA und Java bieten hierfür eine ausgezeichnete Basis. Bei der Entwicklung mit CORBA ergaben sich insbesondere die folgenden Vorteile:

- CORBA stellt eine wesentliche Erleichterung bei der Entwicklung von Client/Server-Anwendungen dar, da die Netzwerkprogrammierung entfällt.
- Aufgrund des Programmiermodells ist der Code leichter erweiterbar und wartbar.
- Die Standardschnittstelle von Objekten (IDL) und der Standardkommunikationsmechanismus der ORBs (IIOP) ermöglichen die projektübergreifende Auslegung von CORBA-Objekten und damit eine komponentenbasierte Softwareentwicklung.
- Die Sprachanbindung an Java kann die Funktionalität einer auf CORBA basierenden Anwendung unmittelbar durch Java-Applets und Java-Applikationen im WWW verfügbar machen. Dabei kann eine Kommunikation mit anderen CORBA-Objekten erfolgen, die in einer völlig anderen Programmiersprache erstellt wurden (z.B. COBOL, C++).

- Bestehender Code kann in CORBA-Objekte gekapselt werden und so in einem neuen System wiederverwendet werden.
- Die Java-Sprachanbindung an CORBA ist einfacher als andere Sprachanbindungen (z.B. C++).
- Ein zu 100% in Java implementierter ORB kann in jeder javafähigen Umgebung eingesetzt werden.

Um diese Vorteile sinnvoll nutzen zu können, wurden in diesem Papier Verfahren zur Java-Applet und -Applikationsprogrammierung sowie zur Code-Integration vorgestellt. Diese Verfahren sollen die technischen Probleme bei der Realisierung einer Anwendung minimieren, um die dadurch gewonnene Zeit dem Design und der Funktionalität der Gesamtanwendung zur Verfügung zu stellen.

Als besonders kritisch bei der Entwicklung mit CORBA haben sich die nachfolgend aufgeführten Punkte herausgestellt:

- Die Verfügbarkeit eines ORBs für die vorhandene Zielplattform ist maßgebend für den erfolgreichen Einsatz von CORBA. Insbesondere bei der Wiederverwendung von bestehendem (C++-) Code spielt der verwendete Compiler eine wesentliche Rolle, da der ORB mit diesem Compiler verwendbar sein muß.
- Ein weiterer Kritikpunkt bei der Entwicklung mit CORBA sind die zahlreich vorhandenen proprietären Features der ORBs. Durch die Verwendung solcher, nicht im CORBA-Standard enthaltener Features, wird die herstellerunabhängige Verbindung von CORBA-Objekten stark gefährdet.

In Zukunft sind weitere Fragen zu klären, die sich z.B. mit der Skalierbarkeit, Architektur (z.B. Business-Objects) und der Integration von den zahlreich spezifizierten CORBA-Services in auf CORBA basierenden Anwendungen beschäftigen. Gegebenenfalls sind auch hier Verfahren zu entwickeln, die den sinnvollen und projektübergreifenden Einsatz der vorhandenen Möglichkeiten bewerkstelligen.

Literatur

1. Beschorner, K.: Realisierung einer Client/Server-Anwendung mit CORBA und Java unter Berücksichtigung bestehender C++-Komponenten. Diplomarbeit, Universität Tübingen (1998)

2. Object Management Group: The Common Object Request Broker: Architecture and Specification, 2.1 ed. (1997)

3. Koch, T.: Objektorientierter Entwurf und Realisierung der Benutzerschnittstellen eines Agentur-Informationssystems. Diplomarbeit, Universität Tübingen (1997)

4. JavaSoft: Java Native Interface Specification, Release 1.1 (1997)

5. Hamilton G., Cattell, R.: JDBC: A Java SQL API, Version 1.20, JavaSoft (1997)

6. Hemrajani, A.: MPAD: A new Design and Development Methodology for Multi-Panel Applets. In: Java World, Vol. 2 Issue 5, http://www.javaworld.com/javaworld/jw-05-1997/jw-05-appdesign.html (1997)