Designentscheidungen für ein flexibles, CORBA-basiertes Workflow-Management-System

Mathias Weske, Martin Hacker, Jens Hündling, Dominik Kuropka,
Hilmar Schuschel, Thomas Serries
Lehrstuhl für Informatik, Universität Münster
Steinfurter Straße 107, D-48149 Münster
{weske,hackerm,hundlin,kuropka,schusch,serries}@helios.uni-muenster.de

1 Einleitung

Computerbasierte Systeme zur Unterstützung von Gruppenarbeit (Groupware) finden immer größere Verbreitung. Diese Systeme unterstützen in unterschiedlichem Maße die Elemente der Gruppenarbeit, also Kommunikation, Kooperation und Koordination. Workflow-Management-Systeme (WFMS) können als eine Ausprägung von Groupware verstanden werden. Das Workflow-Konzept entwickelte sich aus der zunehmenden Prozeßorientierung in der Produktion und insbesondere in Büroumgebungen [GHS95, LA94]. Derzeit kommerziell verfügbare Workflow-Management-Systeme werden bereits in vielen Organisationen eingesetzt; die zur kontrollierten Ausführung von Workflows notwendigen Workflow-Modelle werden dabei meist von identifizierten und ggf. optimierten Geschäftsprozessen einer Unternehmung abgeleitet [JBS97, VB96, Jab96].

Aufgrund ihres integrativen Charakters sollten Workflow-Management-Systeme generell unternehmensweit einsetzbar sein. Da Organisationen allerdings oftmals heterogene Software- und Hardware-Landschaften besitzen, sollte es darüber hinaus verteilt und plattformunabhängig implementiert werden [WHKS97, Wes98]. Daher bietet es sich an, auf aktuelle Entwicklungen im Bereich "Verteiltes Objekt-Management" zurückzugreifen. Wir haben uns bei unseren Entwicklungen für den CORBA-Ansatz (Common Request Broker Architecture) der Object Management Group (OMG) entschieden, da dieser ein hohes Maß an Plattform- und Herstellerunabhängigkeit zur Verfügung stellt und derzeit einen wichtigen Standard darstellt [OMG95]. Insbesondere im Workflow-Kontext kann CORBA sinnvoll als Middleware eingesetzt werden, da verteiltes Objekt-Management in heterogenen Umgebungen in Workflow-Anwendungen eine besonders wichtige Rolle spielt; eine CORBA *Workflow-Facility* befindet sich derzeit im Spezifikationsprozeß [OMG97c].

In den folgenden Abschnitten werden die wesentlichen Grundlagen für die Implementierung eines flexiblen Workflow-Management-Systems auf der Basis von CORBA vorgestellt. Die Entwicklungen finden im Rahmen des WASA-Projekts an der Universität Münster statt. Die aktuelle Version des Systems wurde im Rahmen von Diplomarbeiten rea-

lisiert [Hac98, Hün98, Kur98, Sch98, Ser98]; einen Überblick über das System und dem zugrunde liegenden Workflow-Meta-Modell findet sich in [WHKS97].

Dieser Bericht gliedert sich wie folgt: In Abschnitt 2 wird zunächst die verwendete Programmiersprache Java kurz vorgestellt. In Abschnitt 3 wird die CORBA-Architektur kurz beschrieben; Abschnitt 4 dokumentiert die Wahl einer konkreten CORBA-Implementierung für unser Projekt. Anschließend werden das Referenz-Modell eines Workflow-Management-Systems der Workflow Management Coalition und eine Einreichung zur Workflow-Management-Facility beschrieben. Zum Abschluß liefern wir eine Zusammenfassung und einen Überblick über verwandte Arbeiten.

2 Programmiersprache Java

Die Spezifikation der Programmiersprache Java wird von der Firma JavaSoft im Auftrag der Firma SUN entwickelt und ist derzeit in der Version 1.1 unter `http://java.sun.com` frei verfügbar. Java verfügt über zentrale Eigenschaften einer objektorientierten Programmiersprache; darüber hinaus besitzt sie Eigenschaften, die ihre Verwendung im Internet-Kontext besonders attraktiv machen.

2.1 Philosophie

Eine sehr wichtige Eigenschaft von Java ist die Plattformunabhängigkeit, die durch einen Mittelweg zwischen einer Compiler- und einer Interpretersprache realisiert wird. Der Java-Quellcode wird dabei von einem Java-Compiler in Bytecode umgewandelt, der von einer *Java-Virtual-Machine* interpretiert, d.h. ausgeführt werden kann. Sowohl der Quellcode als auch der generierte Bytecode sind plattformunabhängig; zum Ausführen von Java-Bytecode ist lediglich eine Java-Virtual-Machine erforderlich, die derzeit für praktisch alle gängigen Plattformen verfügbar ist. Der Nachteil dieses Konzeptes ist die relativ geringe Performanz der bisherigen Implementierung der Virtual-Machines im Hinblick auf rechenintensive Anwendungen.

Java ist eine objektorientierte Sprache mit einer Reihe von Eigenschaften wie Multithreading und strukturierter Ausnahmebehandlung [Fla97]. Allerdings existiert die multiple Vererbung in Java nur in einer abgeschwächten Form und ist lediglich für Schnittstellen (sog. *Interfaces*) erlaubt. Interfaces können, wie im objektorientierten Paradigma von Klassen gewohnt, von beliebig vielen anderen Interfaces erben. Im Unterschied zu Klassen implementieren Interfaces jedoch keine Funktionalität, sondern sie spezifizieren lediglich Methoden. Eine Klasse hingegen kann spezifizierte Methoden implementieren, darf von genau einer Klasse sowie beliebig vielen Interfaces erben. Desweiteren unterscheidet sich Java von klassischen Programmiersprachen wie C oder C++ dadurch, daß es keine expliziten Zeiger gibt, sondern lediglich Referenzen auf Objekte. Außerdem braucht Speicher nicht mehr explizit vom Programmierer verwaltet zu werden; er wird bei der Erzeugung von Objekten allokiert und dann von dem Speichermanager automatisch freigegeben, wenn keine Referenz auf das betreffende Objekt mehr existiert. Auf diese Weise wird eine Vielzahl von Fehlerquellen durch Speicherzugriffe und Zeigerarithmetik ausgeschlossen. Diese und weitere Eigenschaften von Java werden in [Krü97] sehr detailliert beschrieben.

2.2 Internetfähigkeit

Durch seine Plattformunabhängigkeit eignet sich Java besonders für die Anwendung im Internet; sie wird daher bereits heute oft als Internet-Programmiersprache bezeichnet. Zu diesem Zweck bietet Java mit *Applets* eine neue Art von Programmen. Ein Applet kann direkt in eine HTML-Seite eingebunden werden und bietet somit die Möglichkeit, Java-Bytecode in normale Web-Seiten einzubinden. Ein Java-fähiger Browser, wie zum Beispiel Netscape Navigator in der Version 4, ist in der Lage, solche Applets über eine eigene Java-Virtual-Maschine auszuführen. Dieses Konzept wird dadurch unterstützt, daß der Bytecode von Java insbesondere bei der Benutzung von graphischen Paketen im Vergleich zu anderen Programmiersprachen extrem kompakt ist und daher nur geringe Datenmengen übertragen werden müssen. Die Sicherheit war eines der wichtigsten Designziele von Java. Daher gibt es eine Reihe von Sicherheitsmechanismen, die unmittelbar in der Java-Virtual-Machine implementiert sind. Somit sind zum Beispiel Dateioperationen für Applets auf dem lokalen Rechner ausgeschlossen. Auch Verbindungen zum Internet (TCP/IP) sind nur eingeschränkt verwendbar.

2.3 Pakete

Die Laufzeitbibliothek von Java, wie im Java Development Kit (JDK) verfügbar, bietet sehr viele Zusatzfunktionen. Dazu gehören sowohl Klassen wie zum Beispiel `Vector`, `Stack` und `Hashtable` als auch eine Vielzahl von komplexen und mächtigen Paketen, von denen die für unser Projekt zentralen hier kurz vorgestellt werden; eine ausführliche Beschreibung befindet sich unter anderem in [Fla97].

Das *Abstract Windowing Toolkit* (AWT) bietet umfassende graphische Klassen an. Mit dem AWT kann man graphische Benutzeroberflächen gestalten, die beispielsweise unter Unix, Windows und OS/2 ohne Portierungsaufwand benutzt werden können und im *Look-and-Feel* [Bal94, Mye96] der jeweiligen Oberfläche erscheinen. Unterstüzt wird dies insbesondere dadurch, daß ein Minimum an Zeichensätzen auf den unterschiedlichen Plattformen von Java selbst zur Verfügung gestellt wird. Graphische Oberflächen können leicht derart gestaltet werden, daß sich diese den unterschiedlichen Gegebenheiten auf den verschiedenen Plattformen automatisch anpassen (die Fenstergröße kann beispielsweise optimal zu der jeweiligen Graphikauflösung, Schriftgröße etc. angepaßt werden).

Eine wichtige Rolle spielt die Möglichkeit, von Java-Programmen aus leicht auf eine Datenbank zugreifen zu können. Dazu wurde ein spezieller Treiber entwickelt, der als JDBC (Java Data Base Connectivity) bezeichnet wird. JDBC ist mittlerweile unter Java ein Standard für den Zugriff auf relationale Datenbanken. Über einfache Klassenmethoden kann auf jede beliebige SQL-Datenbank zugegriffen werden, sofern diese JDBC oder zumindest ODBC (Open Data Base Connectivity) unterstützt.

Reflection ist ein Java-Paket, mit dem dynamisch (also zur Laufzeit eines Programms) Meta-Informationen zu einem Objekt erfragt werden können. Beispielsweise können neue Objekte erzeugt werden, wobei ihre Klassenzugehörigkeit aber erst zur Laufzeit festgelegt wird. Somit kann man in Java eine Vielzahl solcher Probleme leicht lösen, die ansonsten entweder selbstverändernden Code benötigen oder einen erheblichen zusätzlichen Programmieraufwand erforderlich machen würden.

3 Die Object Management Group und CORBA

In diesem Abschnitt wird die bereits in der Einleitung erwähnte CORBA-Architektur vorgestellt, auf der unser Workflow-Management-System basiert. Begonnen wird mit der Vorstellung der Object Management Group (OMG), der bei der Spezifikationen federführenden Organisation. Anschließend wird die Object Management Architecture (OMA) erläutert, die eine Form des Distributed Object Management (DOM) darstellt. In diesen architektonischen Rahmen ist die Common Object Request Broker Architecture (CORBA) eingebettet, deren Beschreibung diesen Abschnitt beendet.

3.1 Die Object Management Group

Die Object Management Group ist ein internationales Konsortium, das mit dem Ziel gegründet wurde, durch die Entwicklung herstellerunabhängiger Spezifikationen einen Markt für objektorientiert programmierte, verteilte Software-Komponenten zu schaffen. Neben den acht Firmen, die 1989 an der Gründung beteiligt waren, sind inzwischen über 800 weitere Software-Anbieter, -Hersteller und Endbenutzer Mitglied der OMG. Hierzu zählen bedeutende Firmen wie IBM, Sun Microsystems, Hewlett-Packard, NEC, Microsoft und Oracle.

Die Object Management Group definiert “Object Management” als eine Software-Entwicklung, bei der Entitäten der realen Welt durch Objekte modelliert werden, die Daten und Funktionalität kapseln. Diese Betrachtungsweise findet bei der Herstellung von Software immer größere Verbreitung, da sie die Entwicklung und Wartung der Software vereinfacht. Ein zentraler Vorteil bei dieser Entwicklungsart besteht in der Möglichkeit, die Funktionalität von objektorientiert entwickelten Anwendungen zu erweitern, indem neue Objekte hinzugefügt werden. Halten sich Software-Hersteller bei der Implementierung von Software-Komponenten an eine gemeinsame Spezifikation, ist es möglich, ein System aus Komponenten unterschiedlicher Hersteller zusammenzusetzen.

Die Erstellung einer neuen Spezifikation durch die OMG findet wie folgt statt: Jedes Mitglied hat die Möglichkeit, bei einem der regelmäßigen Treffen der Gruppe einen sogenannten *Request for Information* (RFI) einzureichen, in dem ein Problembereich dargestellt und Anforderungen an eine Lösung aufgezeigt werden. Für jede RFI wird eine sog. Task Force gebildet, die daraus einen entsprechenden *Request for Proposals* entwickelt. Es handelt sich hierbei um eine Aufforderung an die Mitglieder, Lösungsvorschläge einzureichen. Damit ein Lösungsvorschlag zur gültigen Spezifikation wird, muß er von einer Mehrheit der Mitglieder unterstützt werden. Um diesen Einigungsprozeß zu vereinfachen, erarbeiten in der Regel mehrere Mitglieder einen gemeinsamen Vorschlag. Vorschläge, die zur gültigen Spezifikation erhoben werden, müssen innerhalb eines Jahres implementiert werden. Hierfür sind diejenigen Mitglieder verantwortlich, die den Vorschlag erstellt haben. Dies soll dazu führen dazu, daß stets Produkte am Markt verfügbar sind, die den aktuellen Spezifikationen entsprechen. (In der Vergangenheit hat sich gezeigt, daß diese Forderung nicht immer eingehalten werden konnte.)

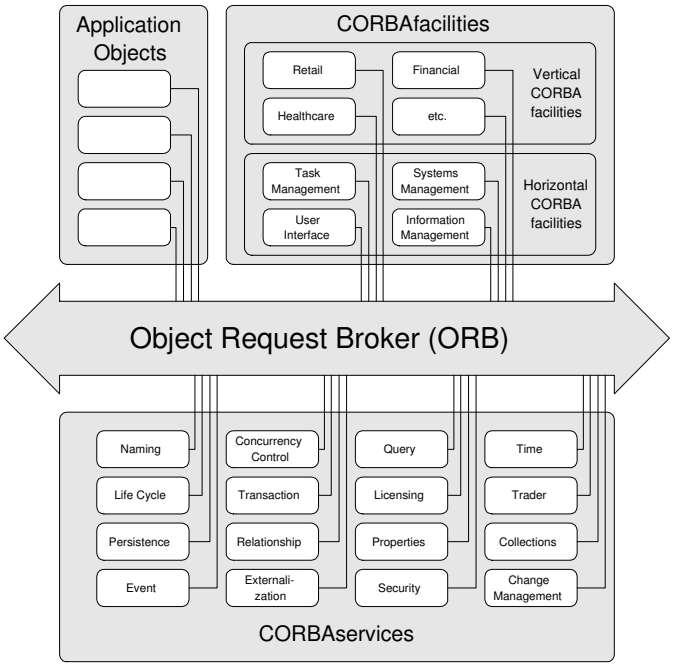

Abbildung 1: Die Object Management Architecture. Quelle: OMG

3.2 Object Management Architecture

Die Object Management Architecture (OMA) bildet einen architektonischen Rahmen für die Zusammmenarbeit objektorientierter Anwendungen, die auf einer gemeinsamen Spezifikation basieren (siehe Abbildung 1). Zentrales Element der Object Management Architecture ist ein *Object Request Broker* (ORB), der entsprechend der CORBA-Architektur aufgebaut ist (siehe Abschnitt 3.3). Der Object Request Broker ermöglicht die Kommunikation zwischen Objekten in heterogen verteilten Umgebungen. Es handelt sich um eine Client/Server-Middleware, die Anfragen von Client-Objekten an die entsprechenden Server-Objekte weiterleitet und gegebenenfalls Antworten an die aufrufenden Objekte zurückliefert.

Unter *Application Objects* versteht man Objekte, die vom Anwender in das System integriert werden. Sie sind die Nutzer der CORBA-Infrastruktur. Die vom Object Request Broker zur Verfügung gestellte Kernfunktionalität wird durch weitere Systemdienste (CORBA-Services [OMG97a]) erweitert. Sie unterstützen die Kommunikation zwischen den Objekten und vereinfachen deren Handhabung durch den ORB. Daneben existieren höherwertige Dienste, sog. CORBA-Facilities, die weitere Funktionalität zum Umgang mit Anwendungsobjekten zur Verfügung stellen. CORBA-Services müssen voneinander unabhängig sein, wohingegen CORBA-Facilities sowohl auf CORBA-Services als auch auf anderen CORBA-Facilities aufbauen dürfen. Man unterscheidet horizontale und vertikale CORBA-Facilities. Während horizontale CORBA-Facilities die Arbeit mit Objekten unabhängig von einer Branche unterstützen, definieren vertikale CORBA-Facilities branchenspezifische Funktionen, die durch *Special Interest Groups* der OMG spezifiziert werden. Bisher wurden Special Interest Groups unter anderem für die Bereiche Gesundheit, Finanzen, Telekommunikation und Handwerk gebildet.

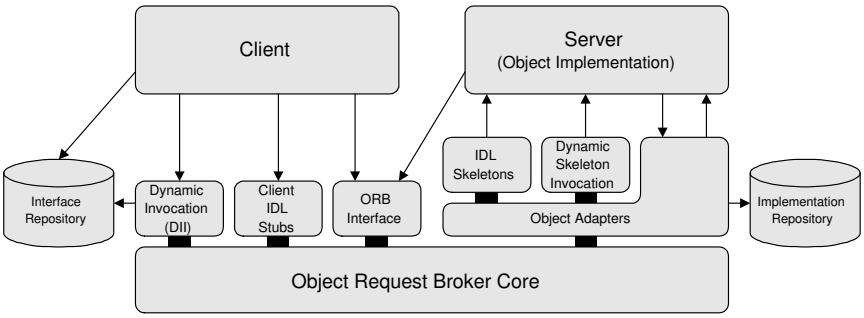

Abbildung 2: Die Common Object Request Broker Architecture

3.3 Common Object Request Broker Architecture (CORBA)

Die Anzahl der verschiedenen Hardware- und Software-Produkte stieg in den letzten Jahren immens. Vielfach besteht die Notwendigkeit einer Interoperabilität zwischen einzelnen Komponenten. Die OMG trägt dieser Notwendigkeit mit der CORBA-Spezifikation Rechnung und zielt darauf ab, daß Anwendungen unterschiedlicher Hersteller durch CORBA in verteilten, heterogenen Umgebungen miteinander kommunizieren können. Im folgenden soll die CORBA-Architektur (Abbildung 2) und ihre einzelnen Komponenten vorgestellt werden.

Im Mittelpunkt der Architektur steht der *Object Request Broker Core (ORB)*, durch den Objekte eine Client/Server-Verbindung eingehen können. Wenn ein Client den ORB benutzt, kann er transparent Methoden eines Server-Objekts aufrufen. Der ORB ist also für die Weiterleitung der Methodenaufrufe und die Rückgabe der Ergebnisse verantwortlich. Das Server-Objekt kann sich auf derselben Maschine wie der Client oder an einem beliebigen Ort des Netzwerkes befinden. Der Client braucht bei seinem Aufruf weder die Programmiersprache noch das Betriebssystem oder andere systemspezifische Aspekte des Servers zu beachten; die entsprechenden Umsetzungen finden durch den ORB statt.

Die Methoden des Clients werden entweder von den statischen IDL Stubs (*Client IDL Stubs*) oder dem *Dynamic Invocation Interface* (DII) aufgerufen. Auf der Server-Seite ist der "Ansprechpartner" des ORB-Kerns der *Object Adapter*, von dem der Client anschließend auch mögliche Ergebnisse entgegen nimmt. Die Ergebnisse werden über den IDL Stub bzw. das DII an den Klienten zurückgegeben. Im Object Adapter ist definiert, wie Objekte aktiviert werden. Die OMG schreibt vor, daß eine CORBA-Implementierung mindestens einen *Basic Object Adapter* (BOA) mit festgelegter Funktionalität unterstützt.

Die Schnittstellen der Objekte werden in der *Interface Definition Language (IDL)* definiert. Es werden hier lediglich Angaben über die Methoden und Attribute des Objektes gemacht. Dies beinhaltet die Methodennamen sowie die Datentypen der Ein- und Ausgabe-Parameter. Darüber hinaus werden auch Ausnahmen (Exceptions) definiert und den Methoden zugeordnet. Die IDL ist eine rein deklarative Sprache, womit eine strikte Trennung von Schnittstelle und Objekt-Implementierung erreicht wird. Für die Implementierung der Methoden kann eine beliebige Programmiersprache benutzt werden, die der IDL-Compiler der jeweiligen CORBA-Implementierung unterstützt (siehe Abbildung 3). Der IDL-Compiler ist ein Pre-Compiler, der zu den Schnittstellen alle notwendigen Operationen zur Bereitstellung der Kommunikation in der jeweiligen Programmiersprache generiert.

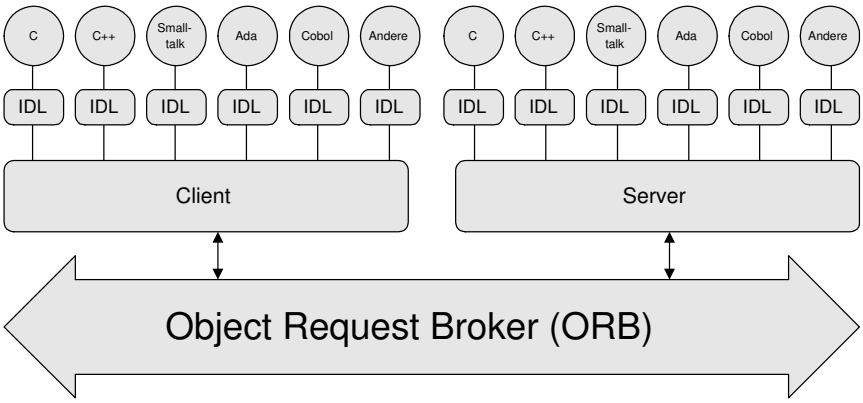

Abbildung 3: IDL-Kapselung der Implementierung

Die Grundlage für die Methodenaufrufe bildet das *Interface Repository (IR)*, in dem sich alle compilierten IDL-Interfaces befinden. Es stellt die Funktionalität zur Verfügung, mit der Objekte Beschreibungen von allen registrierten Interfaces bekommen. Insbesondere werden Methoden und deren Parameter zurückgeliefert. Ebenfalls unterstützt werden dynamische (erst zur Laufzeit bestimmbare) Aufrufe. Das IR wird gefüllt, wenn die Schnittstellen übersetzt werden oder durch eine eigene Registrier-Funktion, mit der sich Objekte aus anderen ORBs registrieren können.

Das *Implementation Repository* ist eine Datenbank, die zur Laufzeit Informationen über unterstützte Objektklassen, instanziierte Objekte und deren IDs sowie ergänzende Daten (z.B. Zugriffsrechte oder sonstige Verwaltungsdaten) enthält. Mit dem Implementation Repository wird ermittelt, welches Objekt bei einem Aufruf angesprochen wird. Somit entspricht es dem Interface-Repository der Client-Seite.

Eine weitere Komponente der CORBA-Architektur ist das *Object Request Broker Interface (ORB-Interface)*, welches grundlegende Funktionen zur Verfügung stellt, die auf jede Objektreferenz anwendbar sind. Eine Objektreferenz ist eine eindeutige ID, die sich auf genau ein Objekt bezieht. Mit einer der Funktionen des ORB-Interface ist es beispielsweise möglich, Objektreferenzen in Strings umzuwandeln und umgekehrt. Objektreferenzen können daher systemunabhängig gespeichert werden. Außerdem liefert das ORB-Interface Methoden, mit denen sich die Interface- bzw. Implementation-Repository-Einträge auslesen lassen.

Die statischen Methodenaufrufe werden durch *IDL-Stubs* auf der Client-Seite und *IDL-Skeletons* auf der Server-Seite realisiert. Sie entstehen bei der Übersetzung einer Schnittstelle, wobei die Server-Objekte bereits zur Übersetzungszeit bekannt sind. In den IDL-Stubs wird festgelegt, wie die Methoden aufgerufen werden. Für jedes Server-Objekt des Interfaces wird ein eigener Stub generiert. In den Stubs werden Methodenaufrufe in ein Nachrichtenformat übersetzt, welches an den Server gesendet werden kann. Auf der Server-Seite nehmen die *IDL-Skeletons* einen Methodenaufruf entgegen. Dabei wird für jede Methode ein eigener Skeleton benötigt. Eine Antwort des Servers wird über das Skeleton gesendet und im Stub aufbereitet.

Neben statischen Aufrufen sind auch dynamische Aufrufe möglich. Diese werden durch das *Dynamic Invocation Interface* (DII) und das *Dynamic Skeleton Interface* (DSI)

ermöglicht. Ein Client kann so während der Laufzeit festlegen, welches Server-Objekt er wie ansprechen möchte. Dazu erhält der Client vom Interface Repository Informationen über die Methoden von Objekten. Der Server nimmt entsprechende Aufrufe über das DSI entgegen. Es existieren drei Arten von Aufrufen. Zum einen kann der Client eine Methode aufrufen und auf das Ergebnis warten, zum anderen kann der Client das Ergebnis zu einem späteren Zeitpunkt abrufen. Die dritte Art ist ein Aufruf, bei dem kein Ergebnis erwartet wird.

Es existieren verschiedene Protokolle, durch die sich ORBs verschiedener Hersteller verständigen können. Eines dieser Protokolle ist das *General Inter-ORB Protocol* (GIOP). Es spezifiziert Nachrichtenformate und Datendarstellungen für die Kommunikation zwischen ORBs und arbeitet unter allen verbindungsorientierten Transportprotokollen. Ein spezielles GIOP-Protokoll ist das *Internet Inter-ORB Protocol* (IIOP), welches festlegt, wie GIOP-Nachrichten über ein TCP/IP-Netzwerk (z. B. das Internet) transportiert werden. Weiterhin sind noch diverse *Environment-Specific Inter-ORB Protocols (ESIOPs)* spezifiziert worden, die netzwerkspezifische Protokolle für die Kommunikation zwischen ORBs darstellen.

4 Auswahl der ORB-Implementierung

Im vorhergehenden Abschnitt wurde die CORBA-Architektur überblickartig dargestellt; nun wird die Entscheidung für eine konkrete CORBA-Implementierung motiviert, die im Rahmen unseres Projektes eingesetzt wird. Hierzu werden zunächst die grundlegenden Unterscheidungsmerkmale verschiedener CORBA-Implementierungen vorgestellt und anschließend die konkrete Entscheidung begründet.

4.1 Grundlegende Unterscheidungsmerkmale

Zum Vergleich unterschiedlicher CORBA-Implementierungen müssen mehrere Aspekte beachtet werden. Neben der verwendeten Systemarchitektur (z.B. Hardware, Betriebssystem, Netzwerk) sind noch weitere Punkte von Bedeutung. Wichtig für die Interoperabilität zwischen unterschiedlichen CORBA-Implementierungen und den Umfang der von ihnen zur Verfügung gestellten (bzw. stellbaren) Dienste ist die Version der CORBA-Spezifikation, welche die Implementierung realisiert. Da zur Zeit keine CORBA-Implementierung alle spezifizierten Dienste (CORBA Common Object Services) umfaßt, sind die für das angestrebte Projekt benötigten Dienste zu beachten. Sollten notwendige Dienste fehlen, muß es möglich sein, die Implementierung um diese Funktionalität zu erweitern.

In die Auswahl einer CORBA-Implementierung für die Entwicklung eines Systems fließen außerdem folgende Aspekte ein:

- Die gewählte Implementierung sollte für unterschiedliche Plattformen verfügbar sein, um Interoperabilität in heterogenen Hardware- und Software-Landschaften leicht realisieren zu können.

- Die für eine Plattform erzeugten Programme und Dienste sollten mit möglichst geringem Aufwand auf andere Plattformen portierbar sein.

Je besser eine CORBA-Implementierung diese Aspekte erfüllt, desto einfacher sind sowohl spätere Umstiege auf andere Plattformen und die Integration weiterer Anwendungssysteme.

Ein anderer Entscheidungsaspekt ist die Unterstützung von Programmiersprachen. Die Spezifikation der Schnittstellen von CORBA-Objekten in IDL ist unabhängig von der Programmiersprache, in der die Implementierung erfolgen wird (siehe Abschnitt 3.3). Um die Freiheit der Programmiersprachenwahl auch in der Realität nutzen zu können, sollte eine CORBA-Implementierung zentrale Programmiersprachen unterstützen. Wünschenswert wäre auch, daß unterschiedliche Paradigmen unterstützt werden. Auf diese Weise können unterschiedliche Aufgabenbereiche, in denen sich bestimmte Paradigmen als vorteilhaft herausgestellt haben, mit Sprachen des jeweiligen Paradigmas realisiert werden. So sollten für rechenintensive Aufgaben beispielsweise Programmiersprachen wie C und Fortran unterstützt werden, da diese auf die ausführende Plattformen optimiert sind und gute Performanz bieten. Für Aufgaben, bei denen Beziehungen zwischen einzelnen Entitäten dargestellt und verwaltet werden müssen, sollten objektorientierte Programmiersprachen wie Smalltalk, C++ oder Java unterstützt werden.

Häufig sind die Erweiterbarkeit der CORBA-Implementierung um Dienste und die Übertragbarkeit der entwickelten Anwendungen auf andere Plattformen entscheidende Kriterien zur Bestimmung der Einsetzbarkeit einer CORBA-Implementierung. Setzt ein ORB-Kern direkt auf das Betriebssystem auf, ist damit zu rechnen, daß weder entwickelte Anwendungen noch die selbstprogrammierten Dienste auf andere Plattformen übertragbar sind. Erfolgversprechender ist ein Ansatz, der die gesamte Funktionalität über Bibliotheken zur Verfügung stellt. Dabei wird auf jedem Rechner, der einen ORB bereitstellen soll, ein Hintergrundprozeß gestartet, der die Kommunikation mit den anderen Rechnern im System bereitstellt. Sowohl die Client-Applikationen als auch die Implementierung der Schnittstellen erhalten die notwendige Funktionalität zur Kommunikation durch plattformabhängige Bibliotheken. Da die Bibliotheken die Besonderheiten der einzelnen Systeme verdecken, wird die Portierung erleichert.

4.2 OrbixWeb

Im Rahmen des Projektes haben wir die Umsetzung des Workflow-Management-Systems unter Verwendung der CORBA-Implementierung *OrbixWeb* von Iona Technologies vorgenommen [Ion96b, Ion96a]. OrbixWeb besitzt eine Java-Schnittstelle und ist aus *Orbix* hervorgegangen, Iona's ORB für die Programmiersprache C/C++. Die Eigenschaften von OrbixWeb werden im folgenden dargestellt. Dabei wird an einigen Stellen ein Vergleich zu einer anderen CORBA-Implementierungen (*NEO* von SunSoft [NEO95]) durchgeführt.

Orbix und OrbixWeb sind zur Zeit (April 1998) unter anderem für die Plattformen Sun Solaris, Windows-NT und MVS verfügbar. NEO existiert nur für Sun Solaris und basiert auf betriebssystemspezifischer Funktionalität. Auch bei den Programmiersprachen bietet NEO mit C++ und C eine eingeschränktere Unterstützung als Orbix und OrbixWeb.

Wie in [WHKS97] dargestellt wird, ist für die Entwicklung eines Workflow-Management-Systems der Einsatz einer objektorientierten Programmiersprache vorteilhaft, da sich so die Beziehungen zwischen den Workflow-Objekten und die Vererbung von Funktionalität leicht realisieren lassen.

Beide Implementierungen halten die CORBA-Spezifikation in der Version 2.0 ein [OMG95]. Allerdings besitzt NEO beim Vergleich der implementierten Services Vorteile. Hier sind unter anderem der Naming-, Event-, Property-, Lifecycle- und Persistency-Service mit Einschränkungen implementiert. Für OrbixWeb stehen ein Event- und ein Naming-Service zur Verfügung.

Die Vorteile von OrbixWeb gegenüber NEO liegen in der Erweiterbarkeit. Die Architektur von NEO erschwert durch die Basierung auf Betriebssystem-Funktionalität eine Erweiterung durch den Programmierer. OrbixWeb hingegen unterstützt die Erweiterung durch das Konzept der Bibliotheken.

Die Plattformunabhängigkeit ist bei der Entwicklung von Workflow-Management-Systemen ein zentrales Kriterium. Die Forderung nach möglichst umfassender Unterstützung organisationsweit zu bearbeitender Aufgaben erfordert, daß Workflow-Management-Systeme möglichst viele Aufgaben auf (in der Regel) heterogenen Plattformen koordinieren können. Orbix und OrbixWeb gewährleisten Plattformunabhängigkeit durch ihre Verfügbarkeit für unterschiedliche Betriebssysteme und die unterstützten Programmiersprachen. Die Programmiersprachen C++ und Java sind durch ihre genaue Spezifikation durch ANSI bzw. Sun in hohem Maße portabel.

Insgesamt ist OrbixWeb eine für unser Projekt geeignete CORBA-Implementierung. Das System wird zunächst auf dem Betriebssystem Sun OS Version 5.5 und Sun Solaris 3.5 mit Orbix in der Version 2.2 bzw. OrbixWeb in der Version 2.0.1 für Sun Solaris realisiert. Unter Verwendung von OrbixWeb kann somit die Implementierung des Systems komplett in der Programmiersprache Java vorgenommen werden.

5 Die Workflow Management Coalition

Die *Workflow Management Coalition* (WfMC) ist eine internationale Organisation von Herstellern und Anwendern von Workflow-Management-Systemen mit mehr als einhundert Mitgliedern. Die Ziele der WfMC sind unter anderem das Erstellen und Etablieren von Standards in Terminologie und Workflow-Produkten, um eine Interoperabilität zwischen verschiedenen WFMS zu gewährleisten [WfM96]. Zu diesem Zweck ist ein Workflow-Referenz-Modell erstellt worden, welches in [WfM94] spezifiziert wird. Dieses Referenz-Modell wird im folgenden kurz vorgestellt, und der Vorschlag einer darauf basierenden Workflow-Management-Facility wird erörtert. Es wird herausgestellt, warum dieser Ansatz für die Realisierung unseres Projektes nicht ohne Modifikationen geeignet erscheint.

5.1 Das Workflow-Referenz-Modell

Das Workflow-Referenz-Modell der WfMC stellt die einzelnen Komponenten eines WFMS und ihre Beziehung zueinander dar. Das Referenz-Modell soll durch die Standardisierung

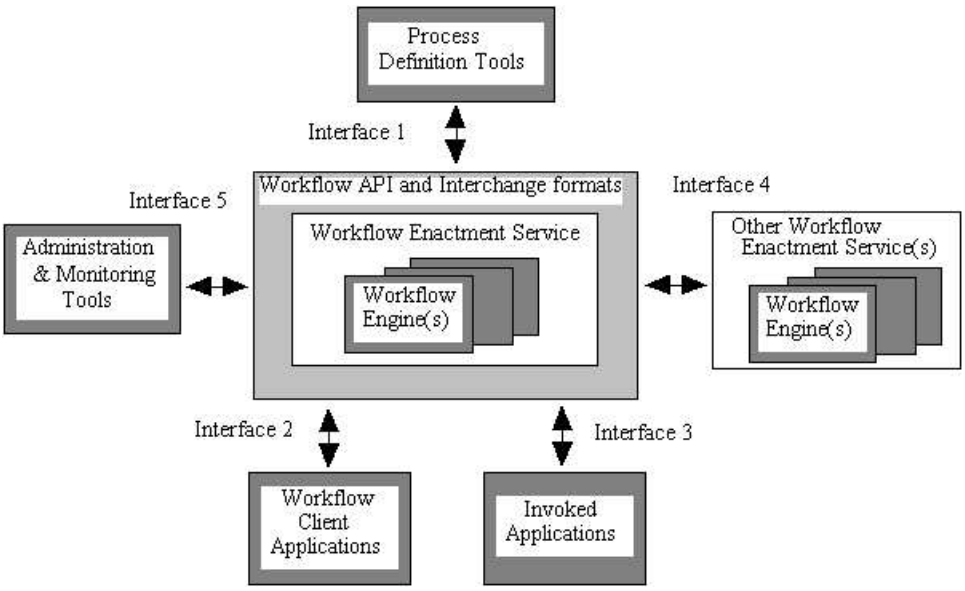

Abbildung 4: Das Workflow-Referenz-Modell der WfMC [Quelle: WfMC [WfM94]]

von Schnittstellen die Interoperabilität zwischen verschiedenen Workflow-Produkten gewährleisten. Anwendungen der verschiedenen funktionalen Bereiche, die in der Referenz-Architektur beschrieben werden, können durch die Schaffung einheitlicher Schnittstellen mit wenig Aufwand ausgetauscht oder ergänzt werden. Es wird daher über den internen Aufbau eines WFMS oder einer Anwendung der spezifizierten funktionaler Bereiche von der WfMC keine Aussage getroffen [WfM94]. Das Workflow-Referenz-Modell definiert fünf Schnittstellen für jeweils einen funktional abgegrenzten Bereich (vgl. Abbildung 4). Die Schnittstellenspezifikationen sind jedoch noch in Bearbeitung und existieren teilweise nur als unvollständige Vorversionen.

Die Kernkomponente des Modells stellt der *Workflow Enactment Service* dar. Er bietet eine Umgebung an, in der zur Laufzeit Workflows instanziiert und ausgeführt werden können. Der Workflow Enactment Service kann dabei aus einer oder mehreren Workflow-Engines bestehen. Anwendungen außerhalb des Workflow Enactment Service kommunizieren mit diesem über die Schnittstelle *Workflow Application Programming Interface & Interchange Formats* (WAPI). Eine *Workflow Engine* trägt die Verantwortung für die kontrollierte Ausführung eines (Teil-) Workflows. Sind mehrere Workflow Engines beteiligt, so kann die Ausführung eines Workflows unter ihnen aufgeteilt werden. Die WAPI ist eine Menge von Schnittstellen, die es Ressourcen und Anwendungen erlaubt, mit dem Workflow Enactment Service zu kommunizieren. Die WAPI besteht aus fünf funktionalen Bereichen, die von der WfMC durch die *Interfaces 1–5* beschrieben werden.

- **Process Definition Tools** Die erste Schnittstelle stellt die Funktionalität für *Process Definition Tools* zur Verfügung. Dabei handelt es sich um Anwendungen, die Workflows analysieren, modellieren, beschreiben und dokumentieren. Der Output eines Process Definition Tool ist ein Workflow-Modell, das zur Laufzeit durch die Workflow Engine innerhalb des Workflow Enactment Service interpretiert und ausgeführt werden kann. Die Process Definition Tools kommunizieren über das Interface 1 der

WAPI mit dem Workflow Enactment Service.

- **Workflow Client Applications** Unter Workflow Client Applications versteht die WfMC einen Benachrichtigungsdienst für den Endbenutzer. Durch das Interface 2 wird eine Interaktion zwischen den Workflow Client Applications und dem Workflow Enactment Service gewährleistet, mit der sich beispielsweise das Konzept der Worklist als Workflow Client Application realisieren läßt.

- **Invoked Applications** Das Interface 3 stellt eine Schnittstelle zur Verfügung, die Aufrufe von Applikationen ermöglicht. In heterogenen WFMS soll auch ein plattformübergreifender Applikationsaufruf ermöglicht werden.

- **Workflow Interoperability** Mit dem Interface 4 wird eine Schnittstelle geschaffen, die Interoperabilität zwischen verschiedenen WFMS ermöglicht. Dabei beschreibt die WfMC vier verschiedene Stufen der Interoperabilität zwischen WFMS: Eine verkettete Ausführung von Workflows, eine hierarchische Ausführung, eine Peer-to-Peer- und eine parallel synchronisierte Ausführung. Zur weiteren Vertiefung bzgl. der verschieden Stufen der Interoperabilität sei hier auf [WfM94] verwiesen.

- **Administration & Monitoring Tools** Anwendungen erhalten über das Interface 5 den Ausführungszustand (Status) eines Workflows. Dies ist insbesondere für ein Monitoring und Auditing von Workflows bedeutsam. Weiterhin soll das Interface 5 in Zukunft erweitert werden, um eine *Management Information Base* zu ermöglichen, die statistische Informationen zur Ausführung von Workflows speichert.

5.2 Die Spezifikation der Workflow Management Facility jFlow

In einem *Request for Proposals* [OMG97b] forderte die OMG zum Einreichen von Vorschlägen für eine Workflow Facility in CORBA auf. Die OMG versucht damit, Workflow-Funktionalität durch spezielle Dienste in der CORBA-Architektur zu etablieren. Die Workflow Facility soll dabei möglichst auf den bereits spezifizierten CORBA-Services aufbauen. Vorschläge zur Spezifikation sollen die Definition von Schnittstellen enthalten, die das Ausführen und Manipulieren von Workflows und ihrer Meta-Daten ermöglicht. Diese Schnittstellen sind in CORBA IDL zu erstellen. Im einzelnen ist eine Schnittstelle zu spezifizieren, die eine komplette semantische Definition eines Workflow-Modells ermöglicht. Dies soll in einer bekannten Notation erfolgen. Weiterhin ist eine Schnittstelle zu definieren, um Workflows ausführen zu können. Eine weitere Schnittstelle erlaubt ein Workflow-Monitoring. Über diese Schnittstelle können Anwendungen für ein Workflow-Monitoring Informationen über den Status laufender Workflow-Instanzen erhalten. Außerdem ist eine Schnittstelle zu erstellen, die ein Workflow-Auditing ermöglicht. Dadurch sind Informationen über die Historie von Workflow-Instanzen möglich.

Die Mitglieder der WfMC haben mit *jFlow* einen Vorschlag über die Architektur und Spezifikation einer Workflow Facility in CORBA als Antwort zu dem oben vorgestellten Request for Proposals eingereicht [jFl97]. Dieser Vorschlag soll im folgenden kurz diskutiert werden. In jFlow wird ein WFMS in drei funktionale Ebenen aufgeteilt.

Zuerst wird zwischen einer Build-Time Funktionalität, die das Erstellen von Workflow-Modellen ermöglicht, und einer Run-Time Funktionalität, die das Ausführen von Instanzen der Workflow-Modelle ermöglicht, unterschieden. Schließlich ermöglichen Run-Time-Interactions die Funktionalität, daß Ausführungen von Workflow-Instanzen mit menschlichen oder informationstechnologischen Ressourcen interagieren können. Ein wesentlicher Kritikpunkt an jFlow ist die strikte Trennung zwischen Build-Time und Run-Time. Dies erschwert eine Flexibilisierung von Workflows, deren Notwendigkeit in [WV98] und technische Realisierung in [WHKS97] beschrieben wird. Die Probleme der Spezifikation von jFlow resultieren zum Großteil aus der Basierung auf den von der WfMC definierten, nicht objektorientierten Standards [Sch97].

6 Zusammenfassung

In diesem Bericht wurden Design-Entscheidungen bezüglich verwendeter Spezifikationen und Produkte zur Entwicklung eines Workflow-Management-Systems erörtert. Hierbei stellt sich heraus, daß die einzelnen Design-Entscheidungen starke Interdependenzen aufweisen. So hängt beispielsweise die Entscheidung über die zu verwendende CORBA-Implementierung von der Wahl der Programmiersprache ab und umgekehrt. Bei der Modellierung des Systems [WHKS97], der Auswahl der zu verwendenen Programmiersprache und der Middleware sowie der Implementierung selbst, also in allen Phasen der Entwicklung, wurde konsequent das objektorientierte Paradigma verfolgt. Würde man an einer Stelle der Entwicklung die Objektorientierung verlassen, wären zusätzliche Transformationen notwendig. Würde man beispielsweise ein objektorientiert modelliertes System in einer prozeduralen Porgrammiersprache implementieren, können beispielsweise Vererbungen nicht direkt dargestellt werden. Allgemein werden somit die Probleme, die durch Paradigmenwechsel auftreten können, verhindert. Die Plattformunabhängigkeit von Java und die Möglichkeit, heterogene Umgebungen durch eine CORBA-Architektur zu verbinden, ermöglicht weitreichende Verteilung in allen Komponenten des Workflow-Mangement-Systems.

Literatur

[Bal94] BALZERT, HELMUT: *Lehrbuch der Software-Technik*. Spektrum, Akademischer Verlag, Heidelberg, Berlin, Oxford, 1994.

[Fla97] FLANAGAN, DAVID: *Java in a Nutshell*. O'Reilley & Associates, Inc., Cambridge, Köln, Paris, Sebatopol, Tokyo, second edition, 1997.

[GHS95] GEORGAKOPOULOS, DIMITRIOS, MARK HORNICK, and AMIT SHETH: *An overview of workflow management: From process modeling to workflow automation infrastructure*. In *Distributed and Parallel Databases*, volume 3, chapter D, pages 119–153. Kluwer Academic Publishers, 1995.

[Hac98] HACKER, M.: *Prozeß-Controlling und -Auditing in CORBA-basierten Workflow-Management-Systemen*. Diplomarbeit Lehrstuhl für Informatik, Universität Münster, April 1998.

[Hün98] HÜNDLING, J.: *Entwicklung einer graphischen Benutzeroberfläche für ein CORBA-basiertes , flexibles Workflow-Management-System*. Diplomarbeit Lehrstuhl für Informatik, Universität Münster, April 1998.

[Ion96a] IONA TECHNOLOGIES: *OrbixWeb programming guide*, November 1996.

[Ion96b] IONA TECHNOLOGIES: *OrbixWeb reference guide*, November 1996.

[Jab96] JABLONSKI, STEFAN: *Anforderungen an die Modellierung von Workflows*. In: ÖSTERLE, HUBERT und PETRA VOGLER (Herausgeber): *Praxis des Workflow-Management*, Kapitel 4, Seiten 65–81. Vierweg, 1996.

[JBS97] JABLONSKI, STEFAN, MARKUS BÖHM und WOLFGANG SCHULZE (Herausgeber): *Workflow-Management: Entwicklung von Anwendungen und Systemen*. dpunkt-Verlag, Heidelberg, 1997.

[jFl97] OBJECT MANAGEMENT GROUP: *jFlow. Submission to Request for Proposals OMG Workflow Facility*, 1997. OMG Document bom/97-08-05. Available at http://www.omg.org.

[Krü97] KRÜGER, GUIDO: *Java 1.1 lernen*. Addison-Wesley Publishing Company, Bonn, Paris [u.a.], 1997.

[Kur98] KUROPKA, D.: *Spezifikation und Implementierung einer verteilten persistenten Workflow-Engine auf der Basis von CORBA*. Diplomarbeit Lehrstuhl für Informatik, Universität Münster, April 1998.

[LA94] LEYMAN, F. and W. ALTENHUBER: *Managing business processes as an information resource*. IBM Systems Journal, 33:326–347, 1994.

[Mye96] MYERS, BRAD A.: *User interface software technology*. ACM Computing Surveys, 28(1), March 1996.

[NEO95] SUN MICROSYSTEMS INC., Mountain View, CA: *Neo Programming Guide*, 1995.

[OMG95] OBJECT MANAGEMENT GROUP: *The Common Object Request Broker: Archtitecture and Specification*, July 1995. Revision 2.0, updated July 1996.

[OMG97a] OBJECT MANAGEMENT GROUP: *Common Object Services Specification*, July 1997.

[OMG97b] OBJECT MANAGEMENT GROUP: *Workflow Management Facility Request For Proposals*, May 1997. OMG Document CF/97-05-06. Available at http://www.omg.org.

[OMG97c] OMG BUSINESS OBJECT DOMAIN TASK FORCE: *Workflow Management Facility*, second edition, 1997.

[Sch97] SCHULZE, WOLFGANG: *Evaluation of the submissions to the workflow management facility RFP*. Technical Report Document bom/97-09-02, Object Management Group, 1997.

[Sch98] SCHUSCHEL, H.: *Flexibilisierung von Workflows in einem verteilten, CORBA-basierten Workflow-Management-System*. Diplomarbeit Lehrstuhl für Informatik, Universität Münster, April 1998.

[Ser98] SERRIES, T.: *Verteilung und Skalierbarkeit in CORBA-basierten Workflow-Management-Systemen*. Diplomarbeit Lehrstuhl für Informatik, Universität Münster, April 1998.

[VB96] VOSSEN, GOTTFRIED und JÖRG BECKER: *Geschäftsprozeßmodellierung und Workflow-Management*. International Thomson Publishing, Bonn, 1996.

[Wes98] WESKE, MATHIAS: *Flexible modeling and excecution of workflow activities*. In *Proceedings of the 31th Hawaii Conference on System Sciences*. Software Technology Track (Vol VII), 713-722. IEEE Computer Science Press, 1998.

[WfM94] WORKFLOW MANAGEMENT COALITION: *The Workflow Reference Model*, 1994. Available at http://www.wfmc.org.

[WfM96] WORKFLOW MANAGEMENT COALITION: *Coalition Overview*, 1996. Available at http://www.wfmc.org.

[WHKS97] WESKE, M., J. HÜNDLING, D. KUROPKA und H. SCHUSCHEL: *Konzeption eines flexiblen Workflow-Management-Systems für Corba-Architekturen*. Fachbericht Angewandte Mathematik und Informatik 18/97-I, Westfälische Wilhelms-Universität, 1997.

[WV98] WESKE, MATHIAS and GOTTFRIED VOSSEN: *Workflow languages*. To appear in: P. Bernus, K. Mertins, G. Schmidt (Editors): Handbook on Architectures of Information Systems, Springer, 1998