Realisierung verteilter Geodatenserver mit der OpenGIS SFS für CORBA

Alexander Zipf und Hidir Aras
European Media Laboratory – EML, Villa Bosch, Heidelberg
<vorname>.<nachname>@eml.villa-bosch.de

Zusammenfassung:
Im Rahmen dieses Artikels werden wir auf Details der Implementierung der Simple Feature Specification (SFS) für CORBA eingehen, die Ergebnisse diskutieren und Einsatzszenarien im Rahmen des Projektes „Deep Map“ des EML kurz vorstellen. Konkret wurden die SFS für CORBA auf Basis der Spatial Database Engine (SDE) von ESRI für Oracle unter Nutzung der Java-API von SDE entwickelt.

Summary:
Within this article we will introduce details of the implementation of the Simple Feature Specification (SFS) for CORBA and briefly discuss results and use scenarios in the context of the project "Deep Map” at the EML. More specifically we implemented the SFS using the Java API of ESRI’s Spatial Data Engine (SDE) for Oracle.

1 Einleitung

Verteilte Objekte und Interoperabilität in GIS

Der Trend zum Einsatz zentraler Geodatenserver und die Unterstützung der offenen Schnittstellen des OpenGIS Consortiums (OGC) von Seiten der meisten Hersteller dieser Geodatenserver ist unverkennbar. Die meisten Datenbank- oder GIS-Hersteller unterstützen dabei entweder die OpenGIS Simple Feature (SFS) für SQL oder eingeschränkt auf die Microsoft-Welt die entsprechende Spezifikation für COM. Damit scheinen die meisten Anwender aus der betrieblichen Praxis zufrieden zu sein. Oder sie werden von den Herstellern gar nicht erst auf die Existenz einer dritten Spezifikation der OGC Simple Features hingewiesen, da die Unterstützung mehrere Alternativen für die Hersteller natürlich zusätzlichen Aufwand bedeutet. Bei dieser bisher kaum verbreiteten Spezifikation handelt es sich um die SFS für CORBA. Wenn das Ziel einer möglichst plattformneutralen und verteilten, d.h. flexiblen und offenen Geodatenserverarchitektur verfolgt werden soll, bietet sich CORBA vom Prinzip her an, wurde es doch gerade von der OMG (Object Management Group) als Standard für programmiersprachenneutrale, verteilte und objektorientierte Anwendungen definiert. Gerade wegen dieser Eigenschaften wurde es schließlich auch vom OpenGIS Consortium als Basis für eine dritte Implementierungsspezifikation der Simple Features ausgewählt. Das Ziel ist dabei der transparente Zugriff auf heterogene Geodaten in einer verteilten Umgebung. Doch erst im April 2000 wurde die Verfügbarkeit eines ersten Produktes, das die SFS für CORBA unterstützt, nämlich GeoMation GeoAdapter/J for HiRDB von Hitachi auf COM-Basis, bekannt gegeben. Schon zuvor wurde im Rahmen des Forschungsprojektes „Deep Map“ des European Media Laboratory (EML), Heidelberg die Simple Feature Specification für CORBA in Java implementiert. Ausgangspunkt war einerseits der Wunsch nach maximaler Flexibilität bzgl. Plattformen und Programmiersprachen, da man gerade in Forschungsprojekten auf zahlreiche Veränderungen der Randbedingungen gewappnet sein sollte, sich aber auf eine stabile Geodatenserer-Architektur im Hintergrund verlassen muss. Zweitens war natürlich ein moderner d.h. durchgängig objektorientierter und potentiell verteilter Zugriff auf Geodaten eine Design-Richtline und andererseits bestand das wissenschaftliche Interesse die Potentiale und Nachteile einer Realisierung über CORBA zu evaluieren.

Im Rahmen dieses Artikels wollen wir auf Details der Implementierung der SFS für CORBA eingehen, die Ergebnisse diskutieren und Einsatzszenarien im Rahmen des Projektes „Deep Map“ kurz vorstellen. Im Rahmen von Deep Map bestand insbesondere die Anforderung mit in Java implementierten eigenen Clients auf den SFS lesend und schreibend zuzugreifen. Auch durch die Wahl von Java sollte einer maximalen Plattformunabhängigkeit Rechnung getragen werden.

Konkret wurden die SFS für CORBA am EML auf Basis der Spatial Database Engine (SDE) von ESRI unter Nutzung der Java-API von SDE entwickelt. Als Datenbanksystem kam Oracle 8i zum Einsatz, doch da SDE mit derselben Java-API auch für andere Datenbanken zur Verfügung steht, kann auch eine anderes DBMS genutzt werden.

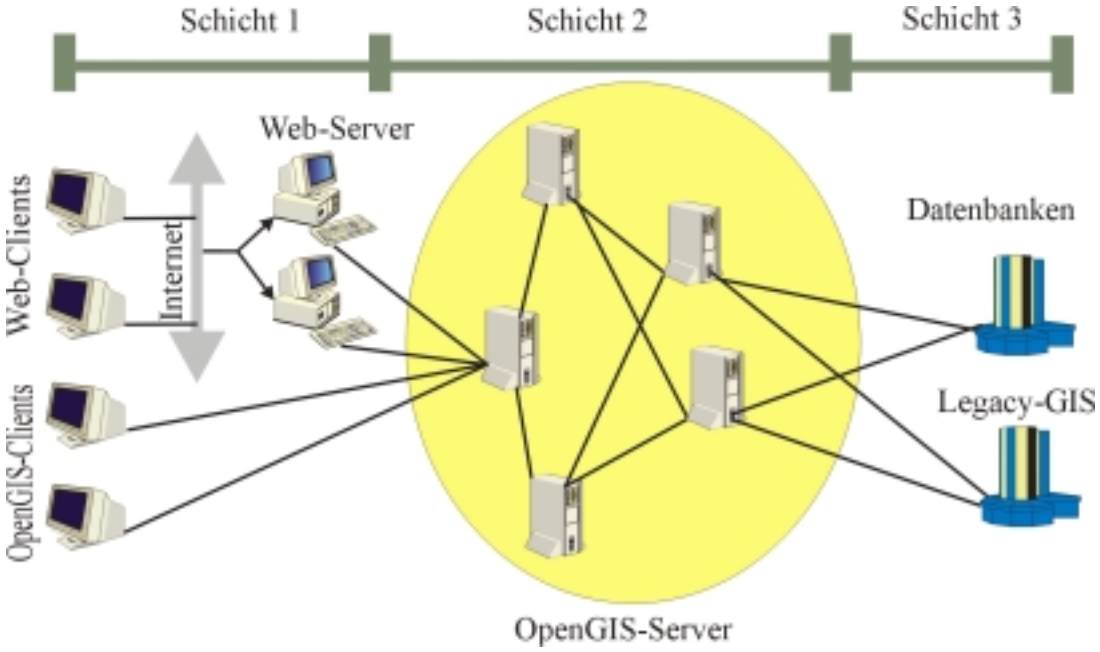

Fig. 1: 3-Tier-Architektur eines verteilten GIS

2 OMG und CORBA

CORBA ist eine verteilte Software-Objekte-Architektur [1], die Objekten und aus Objekten zusammengesetzten Applikationen erlaubt, unabhängig von Ort und Plattform miteinander zu kommunizieren. Der ORB (*Object Request Broker)* ist zentraler Kern von CORBA. Dieser dient als eine Art Middleware, die Client/Server-Beziehungen zwischen unterschiedlichen Objekten ermöglicht. Es handelt sich um einen Software-Bus, über den Objekte miteinander kommunizieren. Jedes Objekt, das eine Operation eines anderen (entfernten oder lokalen) Objektes aufrufen möchte, wird als Client bezeichnet. Die Menge von Programmcode, die festlegt, welche Aktionen beim Methodenaufruf ausgeführt werden sollen, stellt die Objektimplementierung dar. Die Zusammenfassung einer oder mehrerer Objektimplementationen zu einem ausführbaren Programm fungiert als Server des aufrufenden Objektes. Ein großer Kritikpunkt an der Programmierung auf der Basis von CORBA war, daß Objekte nur als Referenz übermittelt werden dürfen. Jedoch soll das aus C++ bekannte „pass-by-value“ für Objekte in den CORBA-Standard integriert werden. Dies ist gerade für GIS von großer Bedeutung, da elementare Konstrukte solcher Systeme nicht „structs“ und „sequences“ sind, sondern geometrische Objekte wie Point, Line, Polygon etc. Objektschnittstellen können in CORBA über eine sprach- und betriebssystemunabhängige *Interface Definition Language* (IDL) for-

muliert werden. Die CORBA-IDL ist beschränkt auf Deklarationen und enthält keinerlei prozedurale Strukturen oder Variablen. Die IDL-Schnittstellendefinitionen werden im *Interface Repository* (IR) gespeichert, das zur Lokalisierung der aufrufbaren Objekte dient. Daneben kann ein Objekt auch über den Corba-Naming-Service erreicht werden.

Aus den Schnittstellen werden bei der Übersetzung durch einen IDL-Compiler *Stubs- und Skeletons* (auf Client oder Serverseite) in der jeweiligen Implementationssprache generiert. Weitere Bestandteile von CORBA (siehe Abbildung) sind CORBA-*Services* (*system level services*) und CORBA-*Facilities* (horizontal:*widely used services*, vertical*: services specific to patricular industries*). Auf diese soll hier nicht näher eingegangen werden, stattdessen sei auf die entsprechende Litertaur zu CORBA verwiesen [2, 3, 4].

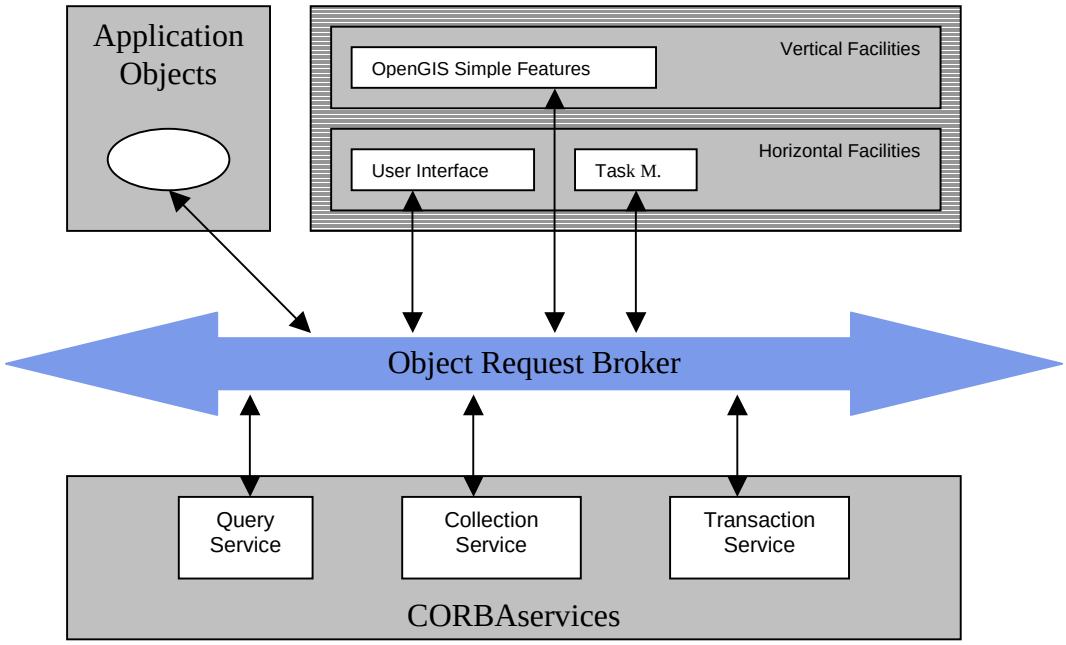

Fig. 2: Die Rolle der SFS innerhalb von CORBA

Der Zweck der OpenGIS Simple Features-Spezifikation für CORBA ist die Bereitstellung von Schnittstellen für den Zugriff auf und die Manipulation von Geodaten unter Benutzung der CORBA-Technologie. Somit ist diese Simple Features-Spezifikation als „Kandidat“ für eine vertikale *CORBA - Facility* zu sehen, welche die Verwaltung von Geodaten übernimmt. Die Beschreibung der abstrakten Architektur erfolgt mittels der Definition von Schnittstellen in CORBA-IDL, welche die unterschiedlichen OpenGIS – Konstrukte repräsentieren.

3 Geographische Softwarekomponenten und SFS

Im Gegensatz zu einem monolithischen System funktioniert die Kommunikation zwischen den einzelnen Schichten in einem verteilten Softwaresystem über offene (standardisierte) Schnittstellen. Die in der Architektur oben angesiedelte Anwendungsschicht nutzt diverse Dienste der darunterliegenden Schichten über sogenannte Applikationsserver, welche grundsätzlich für den Zugriff und die Bereitstellung der Geodaten verantwortlich sind, und Dienste für alle Client-Systeme bereitstellen. In der Zugriffsschicht auf Geodaten (*Spatial Data Access Provider*) ist das *Geodatenmodell* der Anwendung bekannt. Diese ist in der Lage, Anfra-

gen in eine für die Datenbanksoftware verständliche Form zu übertragen. Die OpenGIS-Schnittstellen der jeweiligen Schichten sind prinzipiell für OpenGIS-konforme Softwareprodukte anderer Hersteller zugänglich.

Features und Featuremengen

Grundlage der SFS ist das „**Feature**". Es handelt sich bei Features um digital kodierte Abstraktionen von Objekten oder Phänomen der Realwelt, welche eine geometrische Repräsentation in Raum und Zeit und andere mit ihnen assoziierte Attribute besitzen. Features werden durch OpenGIS-Dienste erzeugt, verwaltet, ausgetauscht oder manipuliert.

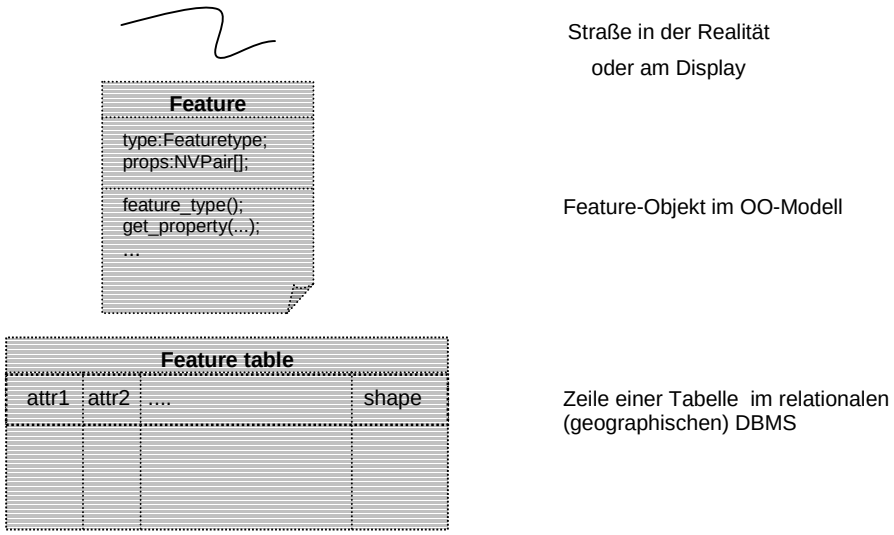

Fig. 3: Feature-Repräsentation in der realen Welt, im OO- und relationalen Modell

Das OGC bietet wie erwähnt drei verschiedene Implementationsspezifikationen an, die sich in der Umsetzung des Feature-Konzepts unterscheiden:

SQL: Features werden als Zeilen (*rows*) in Tabellen (*tables*) oder Sichten (*views*) dargestellt. Geometrien werden als explizite Koordinaten in zusätzlichen Geometrietabellen, als BLOB oder ADT verwaltet [5].

COM: Die Spezifikation basiert auf OLE/DB. Es werden COM-Schnittstellen zu gemeinsamen relationalen Elementen bereitgestellt. Auch hier werden die Zeilen (*rows*) als Feature interpretiert. Die Geometrien werden über COM-Objekte dargestellt [6, 7, 8].

CORBA: Objektorientiertes Modell für geographische Features. Sowohl Features als auch Geometrien werden als Objekte betrachtet, die über CORBA-Schnittstellen angesprochen werden können.

In der Tat fehlt bei der Spezifikation für SQL und COM ein klares Feature-Konzept. Zwar werden Zeilen in Tabellen und Sichten als Features interpretiert, diese aber nicht als Objekte im darunterliegenden System interpretiert [9].

Die CORBA-basierte Spezifikation sieht ein durchgängiges Objektmodell und ein klares Feature-Konzept vor. Die Simple Features-Spezifikation besteht im wesentlichen aus zwei Tei-

len: Den Schnittstellenbeschreibungen der beiden Module Feature und Geometry. Das Feature-Modul enthält ein Objektmodell mit Klassen und Methoden zum lesenden und schreibenden Zugriff auf Features und ihre Attribute. Das Geometry-Modul enthält dagegen Klassen und Methoden für den Zugriff auf eine bestimmte Teilmenge der Feature-Attribute, welche dem Client zur weiteren Verarbeitung bereitgestellt werden.

Feature-Modell
Das Feature-Modul enthält Schnittstellen zur Erstellung, Verwaltung und Abfrage einfacher geographischer Features und Featuremengen. Jedes Feature beziehungsweise jede Feature-Instanz besitzt in der Regel einen Feature-Typ, der zur Kategorisierung von Gegenständen der realen Welt dient. Ein Feature-Typ definiert eine Menge von Eigenschaften, *Properties*, und ein bestimmtes Verhalten, das allen Features eines bestimmten Feature-Typs gemein ist. Eine *Property* besitzt einen Namen und einen Typ. Es kann sich hierbei entweder um einfache Typen (short, long, float, string etc.), zusammengesetzte Typen (struct, union, sequence), also prinzipiell jeden IDL-Typ oder um Objektreferenzen (auch Referenzen auf andere Features) handeln. Der Zugriff auf einzelne Features einer Featuremenge kann über die Objektklasse FeatureIterator erfolgen. In einem Client/Server-System wird auf der Client-Seite eine Anfrage generiert, die durch die Vermittlungskomponente an den entsprechenden Geodatenserver weitergeleitet wird. Dieser wertet die Anfrage aus und löst entsprechende Datenbankaufrufe aus, um die angeforderten Geoobjekte aus der Datenbank zu extrahieren.

Geometrie-Modell
Die interoperable Verarbeitung raumbezogener Daten benötigt die eindeutige Bereitstellung von geometrischen Instanzen. Alle geometrischen Instanzen gehören einer abstrakten Kategorie Geometrien an, welche durch die Schnittstelle `Geometry` repräsentiert wird. Alle haben eine Anzahl allgemeiner (räumlicher) Eigenschaften und benutzen irgendeine Form des räumlichen Bezugssystems. Durch ihre Dimension werden sie als Nulldimensionale-Geometrien (Punkte), Eindimensionale-Geometrien (Kurven) und Zweidimensionale-Geometrien (Oberflächen) kategorisiert.

Im nächsten Abschnitt wird das Konzept der Geodatenmodellierung bei SDE näher vorgestellt, da es für die Implementierung des Database-Layers verwendet wird.

4 Spatial Database Engine

Das Datenzugriffsmodell von SDE (siehe Abbildung) basiert auf dem Standard-Cursor-Modell in SQL. Dabei können in den Abfragen auch geographische Nebenbedingungen formuliert werden. Damit definiert eine Abfrage (*query*) eine bestimmte Menge von *Features*, welche von der Datenbank gelesen werden sollen. Es werden diejenigen Elemente zurückgeliefert, die den Nebenbedingungen (*constraints*) der Abfrage genügen. Abfragen können sowohl einfach als auch komplex sein. Komplexe Abfragen schließen dabei mehrere Datenbanktabellen, die über Primär-/Fremdschlüssel gekoppelt sind, ein. Die Abfrage kann aus einer nicht-geographischen und/oder geographischen Komponente aufgebaut sein. Man unterscheidet in diesem Fall zwischen *attribute query* und *spatial(layer) query*. Das Resultat der Abfrage ist ein Cursor, der die Iterierung über die Menge der „Treffer“ bestehend aus einer Menge von *Features* erlaubt. Mit jedem Cursor ist zugleich ein Zustand assoziiert, das dem aktuellen Feature in der Kollektion entspricht. Die Werte der Feature-Attribute können ebenfalls einfach oder komplex (z.B. *geometric shape*) sein und über den Cursor abgefragt werden. SDE implementiert das Cursor-Model mit Hilfe von *Streams*, welche die Kommunikati-

on zwischen der Server- und Client-Applikation ermöglichen. Die Datenbankverbindung ist eine Verbindung mit einer Geodatenquelle, die durch den SDE-Server repräsentiert wird.

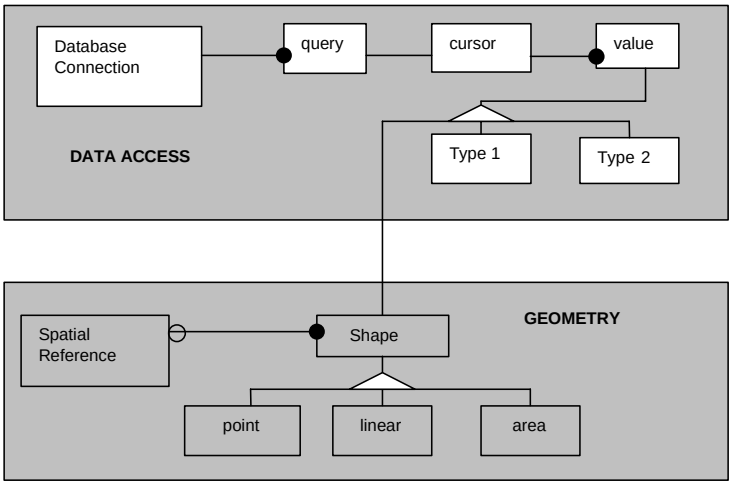

Fig. 5: SDE-Datenmodell auf Basis von SQL-Cursormodell

Mit Hilfe der mitgelieferten API (`com.esri.sde.client`) in Java lassen sich Datenbankverbindungen zu SDE aufbauen, Streams generieren und sowohl geographische als auch nicht geographische Daten aus der Datenbank extrahieren oder verwalten.

Die Zugriff auf SDE-Daten erfolgt prinzipiell folgendermaßen:

1. Aufbau der Verbindung zum Server (*connect*) und Initialisierung eines *Streams*
2. Definition einer Abfrage
3. Ausführung einer Abfrage
4. Iterativer Zugriff auf die erhaltenen *rows* (Zeilen einer Datenbanktabelle)
5. Für jede row: lese selektierte *column*-werte (Spaltenwerte)
6. Verbindungsabbau von der SDE

Eine geographische Nebenbedingung wird über das Konzept eines geometrischen Filters (*spatial filte*r) definiert. Ein Filter wird definiert durch die Angabe der Suchtabelle, der geometrischen Spalte, eine Suchgeometrie und eine zu überprüfende geometrische Beziehung (z.B. schneidet, berührt, etc.).

Aufsetzpunkt für einen Client zur Kommunikation mit dem OpenGIS-Server
Agenten, die in der ersten Schicht angesiedelt sind, sind in der Lage auf Dienste des OpenGIS-Servers über seine Schnittstellen zuzugreifen. Primär erfolgt der Zugriff auf die durch einen OpenGIS-Server repräsentierte GIS-Datenbank über eine *ContainerFeatureCollection*, deren Elemente über die Feature-Schnittstelle erreichbar sind. Somit stellt die *ContainerFeatureCollection*-Schnittstelle den Kommunikationsaufsetzpunkt zum OpenGIS-Server dar. Über die *Feature*-Schnittstelle werden die *Properties* eines Features (einschließlich Geometrie) über entsprechende Methoden iterierbar beziehungsweise abfragbar. Clients können Instanzen einer *ContainerFeatureCollection* durch die Benutzung *der ContainerFeatureCollectio-Factory*-Schnittstelle auch neu erzeugen. Referenzen auf Instanzen dieser Klassen können über einen Namensdienst erhalten werden.

5 Schrittweise Implementierung der OpenGIS-Schnittstellen

Schritt 1: Konvertierung der Schnittstellen nach Java
Die Umwandlung der in CORBA spezifizierten Schnittstellen und Konstrukte nach Java erfolgt durch einen *IDL-to-Java* – Compiler [10].

Schritt 2: Umsetzung des Feature-Moduls
Der Kern des Feature-Moduls besteht aus den Schnittstellen *Feature*, *FeatureType* und *FeatureCollection*. In einer objektorientierten Umgebung stellt die Feature-Schnittstelle eine Art leichtgewichtigen Wrapper eines GIS-Feature-Objekt (siehe Abstrakte Spezifikation) dar. Die internen Attribute dieser Objekte können über *set*- und *get*-Methoden gelesen bzw. modifiziert werden. Die Identität eines Features (*feature-id*) wird in die Objektreferenz dieser Objekte gekapselt. In der Regel werden keine Unterklassen von Feature mehr benötigt, jedoch kann ein OpenGIS-Server beispielsweise eine Schnittstelle Road definieren, das von Feature erbt. Durch diese speziellere Schnittstelle könnte man zusätzliche Dienste implementieren (Spezialiserung der Feature-Definition).

Schritt 3: Umsetzung des Geometry-Moduls

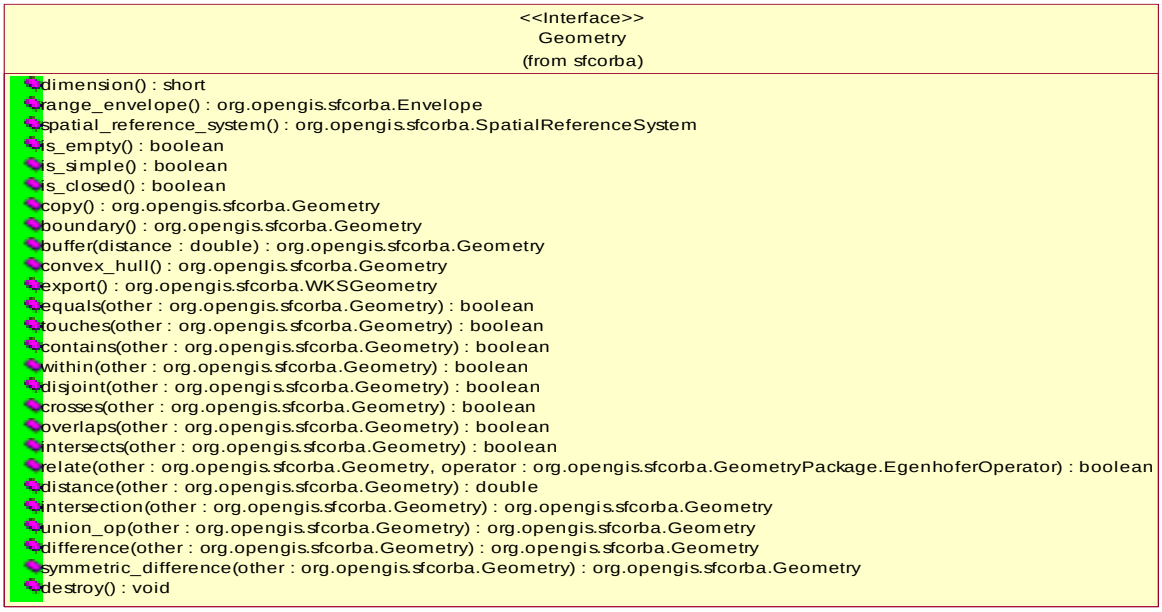

Fig. 6:Geometry-Schnittstelle mit den wichtigsten Methodendefinitionen

Neben der Basisklasse Geometry werden die Klassen GeometryCollection, GeometrieIterator und einfache von Geometry abgeleitete Geometrien behandelt. Zur Modellierung der Abhängigkeiten (Vererbungshierarchie der Geometrien) zwischen den Klassen des Geometry-Moduls dienen UML-Klassendiagramme. Die Implementierung eines Iterators auf Geometrieobjekte und die Generierung neuer Geometrieobjekte über Factories erfolgt analog dem Feature-Modul. Alle Geometrien implementieren die grundlegende Geometry-Schnittstelle, wie sie in Fig. 6 als UML-Klassendiagramm abgebildet ist. Die (abstrakte) Basisklasse Geometry hat die Subklassen Point, Curve, Surface und GeometryCollection. Jedes Geometrieobjekt steht mit einem SpatialReferenceSystem in Beziehung. Daraus werden spezialisierte nulldimensionale, eindimensionale und zweidimensionale Collection-Klassen mit den Bezeichnungen MultiPoint, MultiLineString und MultiPolygon für die Modellierung von Mengen von Point, LineString und Polygon-Objekten abgeleitet.

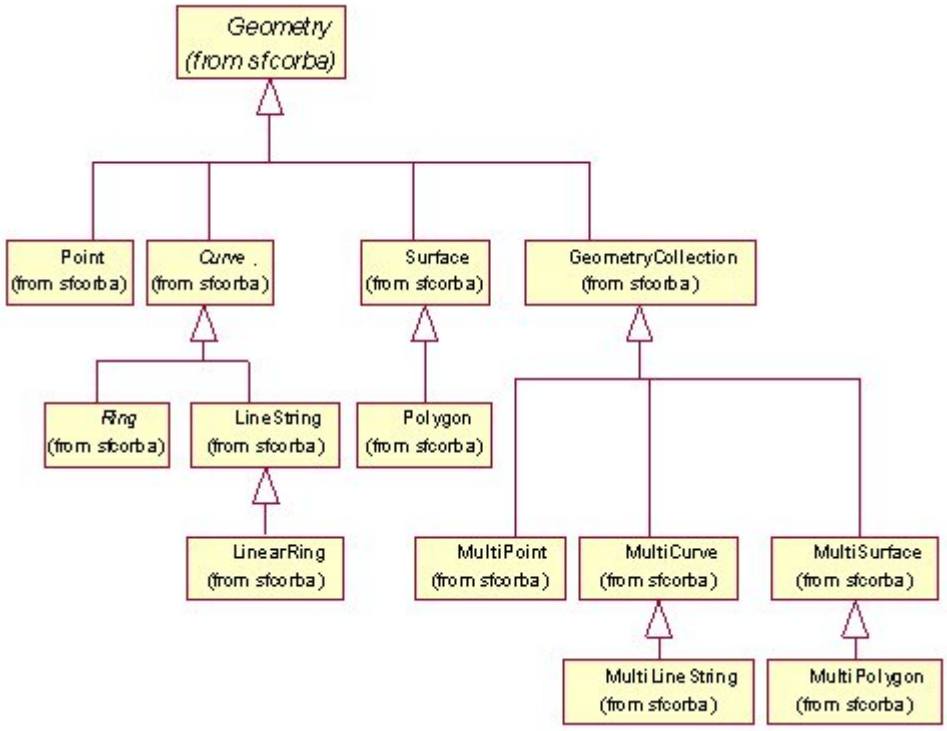

Fig. 7: Klassendiagramm zum Geometry-Modul

6 Einbettung in Deep Map

Fig. 8: Mobiler Prototyp von Deep Map

Im Rahmen von DeepMap wird der Prototyp eines digitalen mobilen Touristikinformationssystems [11,12] entwickelt, in das verschiedenste Forschungsergebnisse aus unterschiedlichen Bereichen der Computerwissenschaft einfließen. Das System soll auf einfache und für den Menschen natürliche Weise bedienbar sein. Der intelligente Touristenführer soll nicht nur als tragbarer, individueller Tourenplaner und- führer der Stadt Heidelberg dienen, sondern außerdem virtuelle Zeitreisen in das historische Heidelberg ermöglichen. Das System soll sich dem Benutzer anpassen (Adaptivität), indem es sowohl den sozialen und kulturellen Hintergrund seines Benutzers als auch dessen Interessen, sein Zeitbudget und vieles andere berücksichtigt. Wissen in Form von geographischer, historischer und weiterer Daten soll multimedial präsentiert werden, wie z.B. virtuelle 3D-Szenarien oder virtuelle Zeitreisen mit VR-Techniken.

Das Szenario von DeepMap ist für eine verteilte, mobile, heterogene und sehr dynamische Umgebung konzipiert. Um das Ziel einer hierfür geeigneten intelligenten Infrastruktur zu er-

reichen, wird eine Agentenplattform eingesetzt, deren Arbeitsweise von einem zentralen Modul unabhängig ist. Agenten sind Software-Komponenten, die selbstständig agieren, miteinander über eine an der Sprechakttheorie angelehnten Agentensprache (Agent Communication Language, ACL) kommunizieren und kooperieren. Diese Sprache und die zugrundeligende Plattform wurde von der Foundation for Intelligent Physical Agents (FIPA) standardisiert. Die asynchrone Kommunikation dieser Agenten im System basiert auf dem Paradigma von verteilten Tupelräumen (Message Bus). Agenten können sich zur Laufzeit dynamisch an den Bus anschließen oder diesen wieder verlassen. Beispielsweise kann eine GIS-Anfrage entweder auf den GIS-Server oder aber an unterschiedliche mobile GIS, die sich im Netzwerk befinden, weiterdirigiert werden.

Die Architektur des DeepMap-Systems basiert also auf unabhängigen Agenten. So besitzt das System Agenten für den Zugriff auf Geodaten, einen zur Berechnung von Routen, zur Kartenvisualisierung etc. Im folgenden wird anhand eines Geo-Agenten (Spatial Agent) beschrieben, wie der Zugriff auf die Dienste des OpenGIS-Servers innerhalb der Kommunikationsinfrastruktur des Systems bewerkstelligt wird. Die Kommunikation verläuft dabei über einen Nachrichtenbus (Message oriented Middleware, MOM) auf Basis des am EML entwickelten asynchronen Agenten-Kommunikationsframeworks (JAMFrame)[13]. Das folgende Beispielszenario erläutert und veranschaulicht das Zusammenspiel von Agent, OpenGIS-Client, Message-Bus und OpenGIS-Server.

Beispielszenario
Im DeepMap-System entsteht aufgrund einer User-Anfrage auf ein Geoobjekt eine entsprechende FIPA-Nachricht an Message-Bus. Da der `SpatialAgent` sich für Objekte dieses Typs interessiert, werden diese an ihn weitergeleitet. Der interne Ablauf wird über einen Filtermechanismus realisert. Aus den Anfragedaten extrahiert der SpatialAgent die notwendigen Informationen (name, oid, etc.) und startet über CORBA-Methodenaufrufe eine Feature-Anfrage an den OpenGIS-Server. Aus den zurückgelieferten Feature-Daten erzeugt der SpatialAgent anschließend ein passendes Antwortobjekt (FIPA-konform) und schickt es zurück zum Sender(-Agenten) der Anfrage.

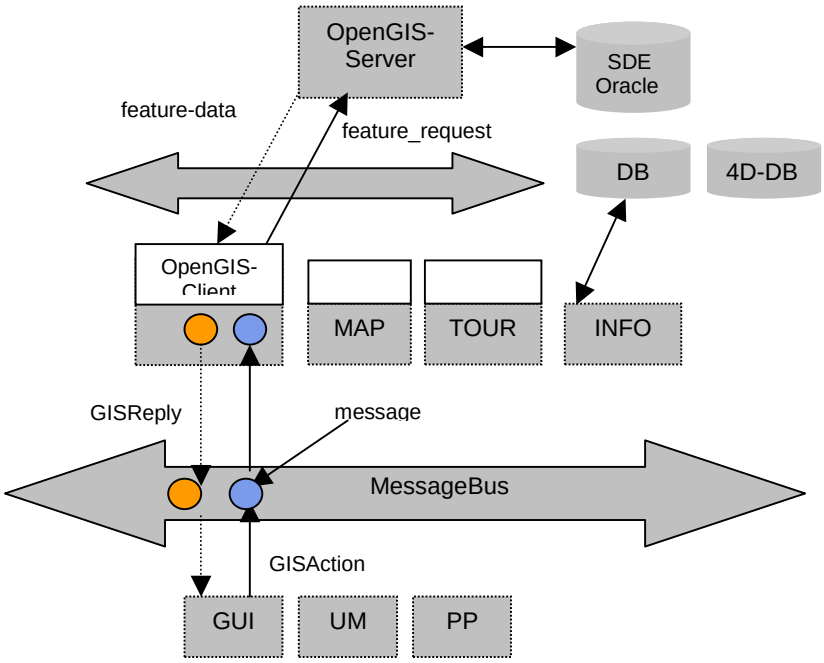

Fig. 9: Agentenkommunikation über Nachrichten (messages)

7 Zusammenfassung und Ausblick

Zwar stehen eine Evaluierung und detaillierte Performanzuntersuchungen noch aus, doch kann festgestellt werden, dass mit der Implementierung der OGIS SFS für CORBA für Deep Map Vorteile zu verzeichnen sind. So sind die verschiedenen als Agenten realisierte Geo-Dienste unabhängig von der API eines bestimmten Herstellers, sondern verwenden eine standardisierte objektorientierte Schnittstelle.
Bezüglich der Performanz werden die Indexierungsmechanismen von SDE und Oracle als zugrundeliegendes System verwendet und müssen nicht mehr neu programmiert werden. Da SDE auch für mehrere Datenbanken zur Verfügung steht, bietet der Ansatz die Schnittstellen auf eine existierende Middleware aufzusetzen den weiteren Vorteil, dass unsere Realisierung damit sofort auf mehrere Datenbanken übertragbar ist.

Es wurden an anderer Stelle auch Anstrengungen unternommen, eine *Simple Features for Java*-Spezifikation [14] voranzutreiben, beziehungsweise einen Vorschlag beim OGC einzureichen. Ein rein auf Java basierender OpenGIS-Server könnte somit ähnliche Schnittstellen wie die CORBA-Spezifikation zur Verfügung stellen, wobei man sich bei der Umsetzung der OpenGIS-Konstrukte nicht auf die IDL-Konstrukte zu beschränken bräuchte. Doch da mit CORBA eine sprachneutrale Spezifikation vorliegt, unternimmt der OGC keine weiteren Bemühungen bezüglich einer weiteren sprachspezifischen Spezifikation. Interessanterweise wird vielmehr mit der GML (Geographic Markup Language) eine zusätzliche XML-Repräsentation spezifiziert, die als Austauschformat für verteilte Anwendungen dienen kann.
Für eine später anvisierte GML-Unterstützung unseres Geodatenservers bietet die objektorienterte Sicht auch schon sehr gute Vorraussetzungen, da die Features als JavaObjekt-Repräsentation vorliegen und aus einer solchen recht einfach automatisiert XML erzeugt werden kann.

Der in Deep Map entwickelte Geodatenserver wird zukünftig auch in einem neuen EU-Projekt namens CRUMPET (der Name steht für "**Cr**eation of **u**ser-friendly **m**obile services **pe**rsonalised for **t**ourism") eingesetzt werden. Hier dient er als Basis für den dort implementierten Web-Map-Server gemäß der Web-Map-Server Specifikation des OGC. Ziel des in CRUMPET geplanten mobilen touristischen Informationssystems ist es, multimediale Zusatzdienste für den Individualtouristen zu entwickeln. Sie sollen leicht bedienbar sein und über drahtlos vernetzte Geräte angeboten werden können – zum Beispiel über künftige UMTS-Mobiltelefone. Die anvisierte Integration der Standardisierungsbemühungen des Open GIS Consortiums mit denen aus dem Bereich intelligenter Agenten (Foundation for Intelligent Physical Agents, www.fipa.org), soll die Interoperabilität der entwickelten Dienste über Domänengrenzen hinweg fördern.

Referenzen

[1] Object Managment Group (OMG): *Common Object Request Broker Architecture*, Rev.2.3, TC DOC, http://www.omg.org, 1999.
[2] Orfali, R. und D. Harkey: *Client/Server Programming with Java and CORBA*, Wiley Computer Publishung, New York, 1998.
[3] Rosenberger, Jeremy: *CORBA in 14 Tagen*, SAMS – Markt & Technik, München, 1998.
[4] Lewis G.; Barber, S. und E. Siegel: *Programming with the java IDL – Developing Web Applications with Java and CORBA*, Wiley Computer Publishing, 1998.
[5] Open GIS Consortium: *OpenGIS Simple Features Specification For CORBA*, http://www.opengis.org/techno/specs.htm,1998.

[6] Hartman, Robert: *Focus on GIS Component Software*, Onword Press, Santa Fe, USA, 1997.
[7] Orfali, R. und D. Harkey: *The Essential Distributed Objects Survival Guide*, Wiley Computer Publishung, New York, 1996.
[8] OpenGIS Consortium: *OpenGIS Simple Features Specification For OLE/COM*, Internet: http://www.opengis.org/techno/specs.htm,1998.
[9] Cuthbert, A.: *OpenGIS: Tales from a Small Market Town* , in: Vckovski, Brassel und Schek (Hrsg.): Interoperating Geographic Information Systems, Proc. Interop'99, Zurich, March, S. 17–28, 1999.
[11] Malaka, R. and A. Zipf (2000): DEEP MAP - Challenging IT research in the framework of a tourist information system. In: Fesenmaier, D. R., Klein, S., Buhalis, D (eds.): Information and Communication Technologies in Tourism 2000. (Proceedings of the 7th. International Conference in Barcelona, Spain),(ENTER 2000)). Springer Computer Science, Wien, New York.
[12] Zipf A.,V. Chandrasekhara, J. Häußler und R. Malaka: *GIS hilft Touristen bei Navigation – ein erster Prototyp des mobilen DeepMap Systems für das Heidelberger Schloß.* In: HGG-Journal. Heft 14. 2000/1, Heidelberg.
[13] Chandrasekhara, V.: *JAMFrame – Java Agent Modules, DeepMap's Integration Fraumework For Communication*, Internal Paper, EML Heidelberg, 1999.
[14] Fitzke, J.; T. Friebe und M. Müller: *Simple Features from CORBA to JAVA*, Presentation at Department of Geography, University of Bonn, Germany, 2000.