Westfälische Wilhelms-Universität Münster
Institut für Geoinformatik
Robert-Koch-Str. 26-28
D-48149 Münster

Entwicklung eines Frameworks zur Visualisierung standardisierter Geodaten im Rahmen der SWE-Initiative

Diplomarbeit

vorgelegt von

Arne Henrik Bröring

März 2007

1. Gutachter: Prof. Dr. U. Streit
2. Gutachter: Prof. Dr. A. Krüger

Danksagung

An dieser Stelle möchte ich allen danken, die durch ihre fachliche oder persönliche Unterstützung zum Gelingen dieser Diplomarbeit beigetragen haben.

Dank gilt Dr. Ingo Simonis, meinem Betreuer, der mich oft mit Hinweisen und Anregungen unterstützt hat.

Herrn Prof. Dr. U. Streit möchte ich dafür danken, dass er mir hinsichtlich der Gestaltung des Themas und der Umsetzung der Arbeit freie Hand gelassen hat, mir bei Fragen aber hilfreiche Ratschläge geben konnte.

Herrn Prof. Dr. A. Krüger danke ich für die Bereitschaft das Zweitgutachten zu übernehmen.

Weiterhin möchte ich mich bei den Kollegen am Institut für Geoinformatik, meiner Familie und meinen Freunden bedanken.

Dank gilt insbesondere auch allen Korrekturlesern, die mich mit konstruktiver Kritik unterstützt haben. Danke Theo, Christoph, Johannes, Walli und Tonio!

Hinweis

Da diese Arbeit zahlreiche englische Begriffe verwendet, werden diese aus Gründen der Übersichtlichkeit und Lesbarkeit der deutschen Schreibweise angepasst.

Aus selbigen Gründen wird auf eine geschlechtsspezifische Differenzierung in der Schreibweise verzichtet. Bei Benutzung der männlichen Form ist daher stets die weibliche Form impliziert.

Abstract

The technologies of the SWE-initiative pave the road to access huge amounts of sensor data captured by sensor networks in an interoperable way. The visualization of the sensor data is a key task. It facilitates the user to extract information. Therefore this thesis describes a software-framework that offers a basis for the implementation of various visualization methods not only for sensor data but also for supporting spatio-temporal data provided by OGC web services. The proposed architecture allows the integration of different visualization methods for spatial, temporal and thematic aspects of sensor data. The framework supplies a reusable design which is applicable for client and server applications. These capabilities of the framework will be demonstrated by the implementation of several visualization methods and two prototypical applications.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Listings

Abkürzungsverzeichnis

CPS	Coverage Portrayal Service
DCP	Distributed Computing Platform
FPS	Feature Portrayal Service
GDI	Geodateninfrastruktur
GI-Dienst	Geoinformationsdienst
GIF	Graphics Interchange Format
GIS	Geoinformationssystem
GML	Geography Markup Language
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transport Protocol
ID	Identifikationsbezeichnung
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
ISO	International Organization for Standardization
JAI	Java Advanced Imaging
JPEG	Joint Photographic Expert Group
JVM	Java Virtual Machine
O&M	Observations and Measurements
OGC	Open Geospatial Consortium
OWS	OGC Web Service
OX-Framework	OWS Access Framework
PNG	Portable Network Graphics
SAS	Sensor Alert Service
SE	Symbology Encoding
SensorML	Sensor Model Language
SLD	Styled Layer Descriptor
SLD-WMS	Styled Layer Descriptor profile of the Web Map Service
SOA	Service Oriented Architecure
SOS	Sensor Observation Service

SPS	Sensor Planning Service
SWE	Sensor Web Enablement
TML	Transducer Markup Language
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WCS	Web Coverage Service
WCTS	Web Coordinate Transformation Service
WFS	Web Feature Service
WMS	Web Map Service
WPS	Web Processing Service
WWW	World Wide Web
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

1 Einleitung

Die Exploration, Kognition und Explanation raumzeitlicher Phänomene ist für viele Domänen, wie z.B. das Umweltmonitoring oder das Katastrophenmanagement, von großer Bedeutung (siehe z.B. BERNARD et al. 2002). Heutzutage sind Geodaten, die Phänomene wie atmosphärische Prozesse oder Schadstoffausbreitungen beschreiben, in großen Mengen verfügbar (MENG 2003). Um diese Daten auswerten zu können, sind spezielle Techniken notwendig. Ein Ansatz, der sich die menschliche Fähigkeit zu Nutze macht, komplexe Informationen mit Hilfe der visuellen Wahrnehmung erfassen zu können, ist die Visualisierung der Geodaten (BLOK 2005). Die Visualisierung erlaubt es dem Anwender Erkenntnisse über mögliche Zusammenhänge oder Anomalien innerhalb der Phänomene zu gewinnen. Für die Informationsextraktion und Entscheidungsfindung ist daher eine effektive, und aussagekräftige Visualisierung der Daten ein entscheidendes Hilfsmittel.

Die Erfassung der raumzeitlichen Phänomene ist insbesondere durch die seit einigen Jahren aufkommende Technologie der Sensornetzwerke realisierbar (siehe z.B. DELIN 2005). Diese Sensornetzwerke sind aus einer großen Anzahl räumlich verteilter Sensoren aufgebaut, welche Messdaten über die beobachteten Phänomene liefern. Bislang war allerdings die Heterogenität der verwendeten Sensortypen, Datenformate und Kommunikationsprotokolle ein Hindernis für einen einfachen, interoperablen Zugriff auf die erfassten Daten. Die Anstrengungen der *Sensor Web Enablement* (SWE)-Initiative des *Open Geospatial Consortium* (OGC) haben daher das Ziel, den gesamten Prozess von der Beschreibung der Sensoren durch Metadaten, dem Auffinden der Sensoren, die Steuerung der Sensoren und den Zugriff auf die von den Sensoren aufgenommenen Geodaten[1] zu standardisieren (SIMONIS 2004). Ein *Sensor Web* bezeichnet dabei ein über das Web zugängliches Sensornetzwerk, welches über standardisierte Web Service-Schnittstellen und Kommunikationsprotokolle zugreifbar ist (BOTTS et al. 2006a). Die Bestrebungen der SWE-Initiative ebnen den Weg für eine interoperable Integration von realzeitlichen oder archivierten Geosensordaten in Geodateninfrastrukturen (GDIs) (SLIWINSKI et al. 2005).

1.1 Zielsetzung

Bislang sind Applikationen, die in der Lage sind, Visualisierungen standardisierter Geosensordaten zu erzeugen, noch Gegenstand der Forschung (siehe z.B. TILLMAN & GARNETT 2006; LIANG & TAO 2005). Aufgrund der Wichtigkeit der Visualisierung von Geodaten, ist das Ziel dieser Arbeit daher die Konzeption und prototypische Implementierung eines Systems, auf dessen Basis Komponenten entwickelt werden können, welche

[1] Im Folgenden werden die über Sensoren erfassten Geodaten auch als *Geosensordaten* bezeichnet. Handelt es sich um Geosensordaten, die mit den Technologien der SWE-Initiative abgefragt und kodiert werden, so wird auch von *standardisierten Geosensordaten* gesprochen.

die Integration und anschließende Visualisierung standardisierter Geosensordaten[2] ermöglichen.

Das System soll eine Grundlage bieten, auf der vielfältige Visualisierungsmethoden[3] implementierbar sind, welche die verschiedenen Aspekte der Geosensordaten zum Ausdruck bringen können. Hierzu gehören neben den fachlichen Eigenschaften, also den Messwerten der erfassten Phänomene, der räumliche und zeitliche Bezug der Daten.

Ebenso ist die Integration mit unterschiedlichen ergänzenden Geodaten notwendig, um die Interpretation der Geosensordatenvisualisierungen zu erleichtern. Genau wie die Geosensordaten werden auch diese unterstützenden Geodaten über Geoinformationsdienste (GI-Dienste) einer GDI bezogen, deren Schnittstellen vom OGC definiert werden. Das System muss daher die Anbindung unterschiedlicher Typen dieser *OGC Web Services* (OWS) erlauben.

Die Architektur des Systems wird als (Software-)Framework konzipiert. Dies bedeutet, dass es sich um ein durch den Entwickler anpassbares und erweiterbares System kooperierender Klassen handelt, die einen wiederverwendbaren Entwurf implementieren (BALZERT 2001). Das Framework soll die Aufgaben der Geschäftslogik zur Integration und Visualisierung der Geosensordaten bereitstellen und in unterschiedlichen Applikationen eingesetzt werden können, wodurch es zur Wiederverwendung der Geschäftslogik sowie der implementierten Visualisierungsmethoden kommt. Beispielsweise soll die Entwicklung von Rich- und Thin-Clients[4], aber auch von Server-Applikationen möglich sein, die Visualisierungen von Geosensordaten anzeigen bzw. anbieten können.

Im Anschluss an den konzeptionellen Entwurf des Frameworks werden exemplarische Implementierungen zum Zugriff auf verschiedene OGC Web Services und zur Visualisierung der integrierten Geodaten entwickelt. Im Vordergrund stehen die Implementierungen zur Integration und Visualisierung der standardisierten Geosensordaten.

Um die Variabilität des Frameworks bezüglich aufsetzender Frontends zu demonstrieren, werden zwei Applikationen prototypisch auf dem Framework implementiert. Zum einen ist dies eine Rich-Client Applikation, die dem Nutzer individuelle Visualisierungen der integrierten Daten präsentieren kann. Zum anderen wird eine Server-Applikation implementiert, die als standardisierter Kartendienst eingesetzt werden kann.

Die Implementierung des Frameworks wird quelloffen unter dem Namen *OGC Web Service Access Framework* (OX-Framework) (BRÖRING et al. 2006) im Rahmen der Open Source Initiative 52°North (KRAAK et al. 2005) publiziert.

Aus der Beschreibung der Zielsetzung ergibt sich die folgende Fragestellung:

Wie kann der softwaretechnische Architekturentwurf eines Software-Frameworks gestaltet sein, welches den Zugriff auf verschiedene OGC Web Services unterstützt und das

[2] Aufgrund der vielfältigen Ausprägungsformen von Geosensoren und den von ihnen erfassten Daten wird in Abschnitt 3.1.1 eine Einschränkung auf bestimmte Ausprägungsformen von Geosensordaten festgelegt, die auf Basis des zu entwickelnden Systems visualisiert werden sollen.

[3] Die mit Hilfe des Systems umsetzbaren Visualisierungsmethoden sollen sich in dieser Arbeit auf die Visualisierung von zwei Raumdimensionen beschränken.

[4] Eine Definition zu den Begriffen *Rich-* und *Thin-Client* findet sich in Abschnitt 3.2.2.

flexible Einbinden vielfältiger Visualisierungsansätze für integrierte Geodaten und insbesondere Geosensordaten erlaubt?

1.2 Kapitelübersicht

Im folgenden Kapitel werden die Grundlagen vorgestellt, die für das Verständnis der weiteren Arbeit notwendig sind. Bevor mit der Sensor Web Enablement-Initiative jenes Tätigkeitsfeld des OGC vorgestellt wird, dessen Technologien im Rahmen dieser Arbeit den standardisierten Zugang zu den Geosensordaten bereitstellen, werden zunächst das OGC selbst und die am OGC entwickelten und für diese Arbeit relevanten Spezifikationen beschrieben.

In Kapitel 3 findet eine Analyse der Anforderungen statt, die sich für das zu entwickelnde Framework ergeben. Zunächst wird eine Analyse der *funktionalen* Anforderungen durchgeführt, die aufzeigt, welche Ansätze zur Visualisierung von Geosensordaten bestehen und welche Charakteristiken sich daraus für das Framework ergeben, um vielfältige Visualisierungsansätze zu unterstützen. Anschließend werden die *technischen* Anforderungen an den Entwurf der Framework-Architektur analysiert.

Kapitel 4 beschreibt die Konzeption der Architektur des Frameworks. Es werden die zur Integration der Geodaten notwendigen Datenmodelle vorgestellt und das Konzept der Anbindungskomponenten erarbeitet, welche die Aufgaben der Anbindung unterschiedlicher OGC Web Services übernehmen und unter anderem die Funktionalität zur Visualisierung der Geodaten kapseln.

Die prototypischen Implementierungen der auf dem Framework aufsetzenden Anwendungen werden in Kapitel 5 beschrieben. Außerdem werden die realisierten Anbindungskomponenten und insbesondere die implementierten Visualisierungsmethoden der Geosensordaten vorgestellt.

Bevor Kapitel 7 eine Zusammenfassung der Arbeit liefert, erfolgt in Kapitel 6 eine kritische Diskussion über das Erreichen der Anforderungen an das entwickelte System und die sich daraus ergebenden möglichen Weiterentwicklungen für aufbauende Arbeiten.

2 Grundlagen

2.1 Open Geospatial Consortium

Als Antwort auf das Problem der fehlenden Interoperabilität in der Informationsgesellschaft der Geoinformation wurde 1994 das *OpenGIS Consortium* gegründet. Diese nonprofit Organisation ging damals aus der *Open Grass Foundation* hervor und wurde im August 2004 in *Open Geospatial Consortium* umbenannt. Diese Umbenennung sollte eine breitere Ausrichtung nicht nur auf Geoinformationssysteme (GIS), sondern auch auf Bereiche wie das Sensor Web unterstreichen. Heute ist das OGC ein internationales Industriekonsortium an dessen Gremien mehr als 300 Mitglieder aus Industrie, öffentlicher Verwaltung sowie Wissenschaft und Forschung beteiligt sind. Das übergeordnete Ziel des OGC wird von BUEHLER & MCKEE (1998, S. III) wie folgt formuliert:

> *„OGC envisions the full integration of geospatial data and geoprocessing resources into mainstream computing and the widespread use of interoperable, commercial geoprocessing software throughout the global information infrastructure."*

Es soll die Realisierung interoperabler Komponenten ermöglicht werden, die in der Lage sind, Geodaten bereitzustellen und zu prozessieren. Das soll durch Spezifikation von Service-Schnittstellen für GI-Dienste und Datenschemata zur Kommunikation sowie zum Datenaustausch mit und zwischen diesen GI-Diensten erreicht werden. Geoinformation, die von heterogenen Quellen stammt, soll über Systemgrenzen hinweg in Geodateninfrastrukturen zur Verfügung gestellt werden.

Diese Spezifikationen werden im Rahmen des OGC als *offene Standards* entwickelt. Das bedeutet, dass ihre Entwicklung in einem offenen Konsensprozess verläuft und die resultierenden Spezifikationen ein hohes Maß an Technologieneutralität aufweisen sowie für jedermann frei verfügbar sind (OGC 2006). Die Spezifikationen basieren auf anerkannten Mainstream IT-Standards, die z.B. vom IETF[5], dem IEEE[6] oder dem W3C[7] festgelegt werden.

---

[5] Die *Internet Engineering Task Force* (IETF) (siehe IETF 2004) ist eine offene und internationale Standardisierungsorganisation, welche sich mit der Weiterentwicklung der dem Internet zugrundeliegenden Technologien beschäftigt. Vom IETF entwickelte Standards sind z.B. das *Hypertext Transport Protocol* (HTTP) oder der *Unified Resource Locator* (URL).

[6] Das *Institute of Electrical and Electronics Engineers* (IEEE) (siehe IEEE 2007) ist ein internationaler Verband von Ingenieuren der Elektrotechnik und Informatik. Das IEEE entwickelt Standards für ein breites Spektrum von Technologien. Für das OGC und die Entwicklung des Sensor Webs, ist dabei der IEEE 1451 Standard von besonderer Relevanz. Dieser definiert eine Schnittstelle für Sensoren, welche die Integration in Sensornetzwerke erleichtern soll.

[7] Das *World Wide Web Consortium* (W3C) (siehe W3C 2007) befasst sich mit der Standardisierung von Internettechnologien. Hierzu gehören z.B. die *Extensible Markup Language* und die *Hypertext Markup Language* (HTML).

Eine zentrale Position innerhalb der Spezifikationen des OGC nimmt die *Abstract Specification* ein. Sie ist nach funktionalen und technologischen Aspekten in 18 sogenannte *Topics* untergliedert. Jedes Topic beinhaltet und dokumentiert ein bestimmtes konzeptionelles Modell. Die Abstract Specification dient als Grundlage für die Entwicklung der *Implementation Specifications*.

Während die Abstract Specification von einem konkreten System abstrahiert, werden die Implementation Specifications für bestimmte verteilte Systemumgebungen (*Distributed Computing Platform*, DCP) entwickelt. Sie definieren interoperable Schnittstellen von Softwarekomponenten oder detaillierte Datenmodelle und zugehörige Datenkodierungen. Früher wurden diese Schnittstellen für unterschiedliche DCPs entwickelt. Heute beziehen sie sich nahezu ausschließlich auf das World Wide Web (WWW) (SIMONIS 2006b). Die Komponenten werden gemäß des Konzepts der Service Oriented Architecure (SOA) in Form von Web Services realisiert und als OGC Web Services bezeichnet. Die vom OGC spezifizierten Daten-Encodings basieren heute ausschließlich auf der *Extensible Markup Language* (XML)[8] und werden in entsprechenden Implementation Specifications durch die Definition von XML-Schemata[9] beschrieben.

Bevor in Abschnitt 2.1.1 das Verhältnis des OGC zur *International Organisation for Standardization* (ISO) beschrieben wird, erläutert Abschnitt 2.1.2 das im *OGC Reference Model* (ORM)[10] vorgestellte grundlegende Konzept zur Modellierung geografischer Information, das *Feature*. Dies ist für das zu entwickelnde Framework im Kontext der Integration der Geodaten von Bedeutung.

Ein für die vorliegende Arbeit zentrales Element der OGC Spezifikationen bildet die *OGC Web Services Common Specification*, welche gemeinsame Aspekte der Schnittstellenbeschreibung von GI-Diensten definiert und in Abschnitt 2.1.3 beschrieben wird.

Neben den Service Spezifikationen der SWE-Inititative, die in Abschnitt 2.2 behandelt werden, hat der *Web Map Service* (WMS) Relevanz für diese Arbeit und wird daher in Abschnitt 2.1.4 skizziert.

Mit der Serialisierung des Status einer Client Applikation befasst sich die *Web Map Context Documents* Spezifikation. Sie spielt im Rahmen dieser Arbeit ebenfalls eine Rolle und wird in Abschnitt 2.1.5 vorgestellt.

2.1.1 ISO und OGC

Ebenfalls 1994 wurde innerhalb der ISO das *Technical Commitee 211* (*ISO/TC211 Geographic Information/Geomatics*) gegründet. Diese Institution hat ähnliche Ziele und Visionen wie das OGC und befasst sich ebenso mit der Standardisierung im Bereich der

[8] Auf eine Beschreibung der Extensible Markup Language (XML) soll in dieser Arbeit verzichtet werden. Ausführlich werden XML und verwandte Technologien beispielsweise bei SKONNARD & GUDGIN (2001) behandelt.

[9] Ein XML-Schema bildet eine formale Beschreibung eines XML-Typsystems, das die Definition von neuen XML-Elementen und deren Attribute einschließt. Diese Thematik wird z.B. bei VAN DER VLIST (2002) detailliert behandelt.

[10] Das ORM (PERCIVALL 2003) stellt ein konzeptionelles Rahmenwerk für die Entwicklung der Implementation Specifications bereit und kann als Leitfaden für den Aufbau einer verteilten Systemarchitektur genutzt werden.

Geoinformatik[11]. Im Unterschied zum OGC handelt es sich bei der ISO jedoch um eine geschlossene Organisation, die *de jure* Standards, also international rechtsverbindliche Normen, festlegt. Das OGC entwickelt die Spezifikationen hingegen in dem bereits erwähnten offenen Konsensprozess und versucht mit der Unterstützung seiner Mitglieder die Spezifikationen als *de facto* Standards zu etablieren. Aufgrund dieser ergänzenden Kompetenzen und um parallele oder konkurrierende Entwicklungen zu vermeiden, wurde 1998 eine sogenannte *Class A Liaison* zwischen beiden Organisationen geschlossen (REED 2005), die eine gemeinschaftliche Zusammenarbeit vertraglich vereinbart. Im Zuge dessen kommt es seitdem zum Austausch und zur Harmonisierung zwischen den Standardisierungsprozessen der beiden Institutionen. Bestimmte Spezifikationen des OGC werden in den Standardisierungsprozess der ISO eingebracht, um somit zu de jure Standards zu avancieren. Das OGC adaptierte ebenfalls bestimmte ISO/TC211 Dokumente, die in die Abstract Specification mit eingingen.

2.1.2 Das Feature Konzept

Im *Information Viewpoint* des ORM werden konzeptionelle Schemata zur Beschreibung von Geoinformation vorgestellt. Diese konzeptionellen Schemata werden als Grundlage für die Entwicklung von Applikationsschemata verwendet. Ein Applikationsschema ist ein Informationsmodell einer bestimmten Informationsgemeinschaft. Das Basiskonzept zur Modellierung der Geoinformation ist dabei das *Feature*.

Ein Feature ist die Abstraktion eines Phänomens der realen Welt. Jedes Feature kann als Instanz eines *Feature-Typs* betrachtet werden. Ein Feature-Typ gruppiert Feature, die gleiche Charakteristiken aufweisen. Das ORM definiert folgende grundlegende Eigenschaften eines Features[12]:

- Innerhalb einer Informationsgemeinschaft wird *eine* Entität der realen Welt auf *ein* Feature abgebildet. Somit ist nur ein Objekt verantwortlich für Operationen, die die Aktualisierung, Erzeugung oder das Löschen von Attributen oder Metadaten einer Realwelt-Entität betreffen.
- Ein Feature hat eine eindeutige, persistente Identifikationsbezeichnung (ID). Diese kann zur Referenzierung des Features verwendet werden.
- Features beinhalten ein oder mehrere Attribute. Ein Attribut weist einen Namen, einen Datentyp sowie eine optionale Beschreibung auf. Außerdem ist es mit einem Wert assoziiert.
- Eine spezielle Form eines Features ist eine Feature-Sammlung. Sie fasst eine Menge von gleichartigen Features zusammen und besitzt Referenzen zu den in ihr enthaltenen Features.

Geometrische und zeitliche Aspekte der Features lassen sich ebenfalls über die Feature-Attribute abbilden.

---

[11] Die vom ISO/TC211 erarbeiteten Standards werden im Rahmen der *ISO-191xx* Standardserie veröffentlicht.

[12] Diese Charakterisierung folgt dem im *Topic 5: Features* (KOTTMAN 1999) ausgearbeiteten abstrakten Feature Modell.

Die Geometrie eines Features kann durch die Assoziation mit einem geometrischen Objekt ausgedrückt werden. Dieses besteht aus einer geometrischen Beschreibung mit Hilfe von Koordinaten und der Angabe eines räumlichen Referenzsystems. Eine ausführliche Beschreibung des Geometrie-Modells des OGC findet sich im *Topic 1: Feature Geometry*[13] der Abstract Specification.

Durch Assoziation mit einem zeitlichen Objekt, kann der zeitliche Aspekt eines Features beschrieben werden. Zeitliche Objekte, wie Zeitpunkte oder Zeitintervalle, werden mit einem zeitlichen Referenzsystem verknüpft. Ein konzeptionelles, zeitliches Schema wurde vom OGC bislang nicht verabschiedet. Potentielle Quellen für ein solches Modell sind ISO 19108 (ISO 2002) sowie Teile der *Geography Markup Language* (GML)[14]. Beide Standards verwenden in diesem Bereich ISO 8601 (ISO 2004).

Weiterhin kann die Topologie von Features beschrieben werden, um mit Hilfe von Abfrageoperatoren topologische Beziehungen zwischen verschiedenen Features abzuleiten. Das ORM verweist auch hier auf das *Topic 1: Feature Geometry* der Abstract Specification.

Features können sowohl diskrete Phänomene, deren Geometrie mit Hilfe mehrerer primitiver Objekte (z.B. *point*, *line*, *polygon*) abgebildet werden kann, als auch kontinuierliche Phänomene beschreiben. Kontinuierliche, also feldbasierte Phänomene werden als *Coverages* modelliert. Ein Coverage ist ein spezialisiertes Feature. Es kann als Funktion aufgefasst werden, welche einen raumzeitlichen Definitionsbereich auf den Wertebereich eines Attributs abbildet. Der Coverage Typ sowie zahlreiche davon abgeleitete spezielle Untertypen werden im *Topic 6: The Coverage Type and its Subtypes* (KOTTMAN & ROSWELL 2000) der Abstract Specification beschrieben.

Eine Web Service Schnittstelle, welche die Abfrage von Coverages erlaubt, wird in der *Web Coverage Service* Spezifikation (WHITESIDE & EVANS 2006) definiert. Der *Web Feature Service* (WFS) (VRETANOS 2005b) bietet eine Service Schnittstelle, die lesenden und optional auch schreibenden Zugriff auf Features beliebigen Typs ermöglicht. Zur Kodierung der Feature-Daten wird dabei die GML verwendet.

2.1.3 OGC Web Services Common

Viele Aspekte bei der Spezifikation von OGC Web Service Schnittstellen können einheitlich festgelegt werden. Hierzu zählen beispielsweise der Aufbau von Operations-Anfragen (engl. „request“) oder Struktur, Inhalt und Kodierung von Operations-Antworten (engl. „response“). Mit der Intention einer Vereinheitlichung dieser Aspekte wurde die *OGC Web Services Common* (OWS Common) Spezifikation (WHITESIDE 2006) veröffentlicht. Sie soll von sämtlichen OWS Spezifikationen referenziert werden, um so die Interoperabilität durch konsistentere und übersichtlichere Spezifikationen zu verbessern.

[13] Das *Topic 1: Feature Geometry* der Abstract Specification ist äquivalent zum ISO Standard 19107 (ISO 2003).

[14] Die Spezifikation der Geography Markup Langugage (COX et al. 2004) definiert eine allgemeine, XML-basierte Beschreibungssprache für Features. Die Spezifikation wurde von der ISO als Standard ISO-19136 übernommen.

Jeder OGC Web Service muss verpflichtend die *GetCapabilities*-Operation anbieten. Der Aufbau von Request und Response dieser Operation wird daher in weiten Teilen einheitlich durch die OWS Common Spezifikation definiert. Die GetCapabilities-Operation erlaubt es Clients, Metadaten über die Service-Instanz abzufragen. Das Antwortdokument[15] ist eine Selbstbeschreibung des Service. Zudem enthält es Informationen über die vom Service bereitgestellten Datenressourcen. Es ermöglicht auf der Client-Seite das späte Einbinden des Dienstes.

Die OWS Common Spezifikation definiert ein gemeinsames XML-Schema, welches die Strukturierung des Capabilities-Dokuments in vier Abschnitte vorsieht: `ServiceIdentification`, `ServiceProvider`, `OperationsMetadata` und `Contents`.

Das `ServiceProvider`-Element enthält Informationen über die betreibende Organisation des Dienstes. Hierzu zählen Kontaktinformationen oder auch Referenzen zu Webseiten des Anbieters.

Die `ServiceIdentification` enthält grundlegende Metadaten über die vorliegende Service-Instanz. Zu diesen Metadaten gehören beispielsweise die Angabe von Titel, Typ sowie eine textuelle Beschreibung des Dienstes.

Im `Contents`-Element werden die vom Dienst angebotenen Datenressourcen aufgelistet und beschrieben. Da die Struktur der hier abzubildenden Informationen stark vom jeweiligen Service-Typ abhängt, kann dieser Abschnitt des Capabilities-Dokuments nicht vollständig durch die OWS Common Spezifikation vorgegeben werden. Von der OWS Common Spezifikation wird lediglich eine minimale Struktur definiert. Jede OWS Spezifikation kann diese bei Bedarf erweitern.

Der `OperationsMetadata`-Abschnitt besitzt für jede von der Service-Instanz unterstützte Operation ein Unterelement, welches eine Metadaten-Beschreibung der Operation beinhaltet. Diese umfasst z.B. Angaben zum Zugang der Operation über die verteilte Systemumgebung, das WWW. Außerdem wird für jeden Parameter der Operation der zulässige Wertebereich beschrieben.

2.1.4 Web Map Service

Die *Web Map Service Specification*[16], aktuell in der Version 1.3.0 (DE LA BEAUJARDIERE 2006), definiert eine Service Schnittstelle, die auf Anfrage räumlich referenzierte Karten als visuelle Repräsentation geografischer Daten generiert. Diese Karten werden in Form digitaler Bilder[17] zurückgegeben und können direkt im Browser oder in beliebigen, zur Spezifikation konformen WMS-Clients angezeigt werden. Die WMS Spezifikation definiert verbindlich die beiden Operationen *GetCapabilities* und *GetMap* sowie optional die Operation *GetFeatureInfo*.

[15] Das Antwortdokument der GetCapabilities-Operation ist in XML kodiert und wird auch als Service Metadaten- oder Capabilities-Dokument bezeichnet.

[16] Die WMS Spezifikation wurde von der ISO als Standard ISO-19128 übernommen.

[17] Bildformate, die ein WMS anbieten kann, sind z.B.: das *Portable Network Graphics* (PNG) Format, das *Graphics Interchange Format* (GIF), oder das Format der *Joint Photographic Expert Group* (JPEG).

Die GetCapabilities-Operation gibt, wie in Abschnitt 2.1.3 bereits beschrieben, eine Selbstbeschreibung der Service-Instanz in standardisierter Form zurück. Dazu gehört im Falle des WMS eine Beschreibung der angebotenen *Layer*. Der WMS nutzt das Konzept des Layers, um die von ihm visualisierbaren Geodaten zu strukturieren bzw. zu klassifizieren. Dem Capabilities-Dokument können für jeden Layer die zulässigen Wertebereiche für die Parameter der GetMap-Anfrage entnommen werden.

Über die GetMap-Operation können, mit Hilfe des Parameters `LAYERS`, einzelne oder mehrere überlagerte Layer angefragt werden. Für jeden Layer kann einer der vordefinierten Darstellungsstile (`STYLES`) angegeben werden. Fragt ein WMS-Client Kartendarstellungen unterschiedlicher WMS für den gleichen geografischen Raumausschnitt (`BBOX`), das gleiche räumliche Referenzsystem (`CRS`) und die gleiche Ausgabegröße (`WIDTH` und `HEIGHT`) an, so können diese clientseitig überlagert werden. Der Parameter `TIME` erlaubt die Anfrage von Karten für eine bestimmte Zeit. Dies können ein oder mehrere Zeitpunkte oder Zeitintervalle sein. Der Zeitparameter wird im Format des ISO 8601 Standards (ISO 2004) spezifiziert.

Die GetFeatureInfo-Operation ermöglicht die Anfrage von Informationen über in der Karte abgebildete Features. Dazu werden die zur Anfrage der Karte spezifizierten GetMap-Parameter erneut angegeben und zusätzlich die Bildschirmkoordinaten des für den Client interessanten Features definiert. Die zurückgegebenen Informationen können z.B. in Form einer HTML-Seite, als Bild, oder im GML-Format kodiert sein.

2.1.4.1 SLD-WMS

Die oben beschriebene Spezifikation des WMS unterstützt lediglich die Möglichkeit aus einer vordefinierten Auswahl einen Darstellungsstil für einen Layer festzulegen. Der Nutzer ist nicht in der Lage eigene Regeln für die Darstellung eines Layers zu definieren. Um dem Abhilfe zu verschaffen, beschreibt die Spezifikation *Styled Layer Descriptor profile of the Web Map Service* (SLD-WMS) (MÜLLER 2006a) in Verbindung mit der Spezifikation *Symbology Encoding Implementation Specification* (SE) (MÜLLER 2006b)[18] eine Erweiterung der Basis-WMS Schnittstelle sowie eine Beschreibungssprache für Darstellungsstile.

Die erweiterte Schnittstelle kann benutzt werden, um Feature- oder Coverage-Daten mit nutzerdefinierten Darstellungsstilen zu visualisieren. Die Darstellungsregeln werden in einem XML-kodierten *Styled Layer Descriptor* (SLD) definiert. Das Format dieser SLDs wird in der SE Spezifikation in Form von XML-Schemata festgelegt. Für Feature-Daten können über einen SLD beispielsweise Eigenschaften wie Farbe, Strichstärke oder Symbolik für einzelne Feature-Typen ausgewählt werden. Für Coverages, in Form von zweidimensionalen Rastermatrizen (z.B. Satellitenbilder), kann beispielsweise eine Kanalauswahl, die Zuweisung einer Farbpalette oder eine Kontrasterhöhung spezifiziert werden.

Um den vom Nutzer definierten SLD mit einer zu generierenden Karte verknüpfen zu können, erweitert die SLD-WMS Spezifikation die Signatur der GetMap-Operation um einen entsprechenden Parameter. Weiterhin wird die optionale *GetLegendGraphic-*

[18] Die beiden Spezifikationen ersetzen die *Styled Layer Descriptor Implementation Specification* (LALONDE 2002).

Operation spezifiziert, die zu den generierten Karten korrespondierende Legenden erzeugen kann.

Bezüglich der Art des Zugriffs auf die zu visualisierenden Daten unterscheidet die SLD-WMS Spezifikation zwei Typen von Server Implementierungen: *Integrated Server* und *Component Server*.

Beim erstgenannten Typ kommt es zu einer sogenannten *closely-coupled* (auch *tightly-coupled*) Beziehung zwischen der SLD-WMS Implementierung und der Datenquelle. Bei der Datenquelle kann es sich z.B. um einen WFS, WCS oder auch eine proprietäre Datenbank handeln. Welche Art von Datenquelle verwendet wird, bleibt dem Nutzer verborgen. Um Informationen über die mit einem angebotenen Layer assoziierten Coverages oder Feature-Typen abfragen zu können, definiert die SLD-WMS Spezifikation die optionale *DescribeLayer*-Operation.

Der Component Server geht hingegen eine *loosely-coupled* Beziehung mit der Datenquelle ein. Bei der Datenquelle handelt es sich entweder um WCS- oder um WFS-Instanzen. Der Nutzer kann zur Laufzeit Coverage- bzw. Feature-Daten einer Datenquelle auswählen, die dann nach den spezifizierten Regeln visualisiert werden. Dazu ist die GetMap-Operation der SLD-WMS Spezifikation um einen Parameter zur Angabe der URL des aggregierten Dienstes erweitert. Component Server, welche die Visualisierung von Feature-Daten unterstützen, werden auch als *Feature Portrayal Services* (FPS) bezeichnet; diejenigen, die in der Lage sind Coverages zu visualisieren, werden als *Coverage Portrayal Services* (CPS) bezeichnet[19].

2.1.5 Web Map Context Documents

Die *Web Map Context Documents* (WMC) Spezifikation des OGC (SONNET 2005) legt fest, wie die Konfiguration einer Karte, die aus beliebig vielen Layern verschiedener WMS-Instanzen besteht, in einem plattformunabhängigen Format beschrieben werden kann. Diese Beschreibung wird auch als Status, oder *Kontext*, eines WMS-Clients bezeichnet. Die Serialisierung dieser Kontexte geschieht in Form von Dokumenten deren Kodierung basierend auf XML von der WMC Spezifikation definiert wird.

Die primäre Verwendung der Kontext-Dokumente adressiert die Persistenzierung des Zustands einer Client-Sitzung, um diesen zu einem späteren Zeitpunkt wiederherstellen oder mit anderen Client-Applikationen austauschen zu können. Kontext-Dokumente können aber z.B. auch dazu eingesetzt werden, um beim Start einer Client-Applikation eine vordefinierte und auf bestimmte Nutzerklassen angepasste Kartenansicht zu präsentieren.

Ein serialisierter Kontext beinhaltet daher ausreichend operationelle Metadaten über die WMS-Layer, damit der Client die Kartenansicht reproduzieren kann. Zu diesen Informationen gehören beispielsweise der geografische Raumausschnitt, das räumliche Referenzsystem oder die Bildschirm-Dimensionen der Kartenansicht.

[19] Es existieren ebenfalls eigene Spezifikationen für FPS (WOODWARD & WHITESIDE 2005) und CPS (LANSING 2002). Diese haben jedoch nicht den Status der Implementation Specification erreicht und werden von der neueren SLD-WMS Spezifikation überholt.

Die aktuelle Version der WMC Spezifikation unterstützt lediglich die Serialisierung von Karten, die aus WMS-Layern bestehen. Im Hinblick auf die Entwicklung von Applikationen, die die Visualisierung von Geodaten verschiedener OWS Typen erlauben, ist diese Einschränkung suboptimal. Daher wurde im Juli 2004 mit der Gründung des *OpenGIS Web Services Context Document Schema Interoperability Experiment* (DALY & KRALIDIS 2005) eine Initiative ins Leben gerufen, welche die Definition eines XML-Schemas zur Kodierung von sogenannten *OWS Context*-Dokumenten zum Ziel hatte. Diese OWS Context Dokumente sollten nicht mehr auf die Referenzierung von WMS-Layern beschränkt sein, sondern verschiedene OWS Typen unterstützten. Im Rahmen dieses Interoperability Experiments wurden die Schemata der WMC Spezifikation in der Form erweitert, dass nun Datenanfragen an WFS- oder WCS-Instanzen aus den Dokumenten rekonstruiert werden können.

2.2 Sensor Web Enablement

Die *Sensor Web Enablement* (SWE)-Initiative des OGC hat die Realisierung des Sensor Webs zum Ziel. Sensornetzwerke waren bislang durch die Heterogenität der verwendeten Sensortypen, Datenformate und Kommunikationsprotokolle durch mangelnde Interoperabilität gekennzeichnet. Sie sollen durch die Schaffung eines Frameworks von Web Service Spezifikationen und assoziierten Beschreibungssprachen über das WWW einheitlich nutzbar werden. Durch die Definition einer Sammlung aufeinander abgestimmter Standards soll die Integration raumzeitlicher Sensordaten in Geodateninfrastrukturen ermöglicht werden. Reale oder virtuelle[20] Sensoren und Sensorsysteme sollen interoperabel über das Web auffindbar, zugreifbar, und wenn möglich auch kontrollierbar werden (BOTTS et al. 2006a).

Das durch die SWE-Initiative geschaffene Framework muss dabei die folgenden Aufgaben erfüllen (BOTTS et al. 2006b):

- Standardisierte Beschreibung der Sensor-Eigenschaften und der damit zusammenhängenden Verlässlichkeit der erfassten Sensordaten.
- Wohldefinierte Kodierung von Sensordaten, die eine weiterführende Prozessierung erlaubt.
- Standardisierung des Prozesses der Sensor- und Sensordatenrecherche.
- Abfrage von aufgenommenen Sensordaten über eine interoperable Schnittstelle.
- Standardisierte Beauftragung und Steuerung von Sensoren zur Akquirierung von Messdaten.
- Registrierung für den Erhalt von Nachrichten beim Auftreten bestimmter Ereignisse innerhalb des Sensor Webs.

Um diese Anforderungen zu erfüllen, besteht das SWE-Framework aus einem *Information Model*, welches Beschreibungssprachen für Messinstrumente und Sensordaten um-

[20] Simulationsmodelle werden auch als „virtuelle Sensoren“ aufgefasst (siehe z.B. SIMONIS et al. 2003).

fasst, sowie einem *Service Model* in dem Web Service Schnittstellen für die zu erfüllenden Aufgaben spezifiziert werden.

Das Information Model setzt sich aus folgenden Spezifikation zusammen: die *Observations & Measurements* Spezifikation (Abschnitt 2.2.1) definiert ein Datenmodell und ein XML-Encoding für Beobachtungen und Messungen. Die Metadaten von Sensoren können mit Hilfe der *Sensor Model Language* (Abschnitt 2.2.2) beschrieben werden. Außerdem enthält das Information Model die *Transducer Markup Language* (Abschnitt 2.2.3), die sowohl zur Kodierung von Metadaten als auch der Sensordaten selbst eingesetzt werden kann. Gemeinsam genutzte Datentypen der SWE-Komponenten werden als *SWE Common* Elemente definiert[21].

Das Service Model beinhaltet diese Web Service Spezifikationen: Eine standardisierte Schnittstelle zur Abfrage der Sensordaten stellt der *Sensor Observation Service* (Abschnitt 2.2.4) bereit. Über den *Sensor Alert Service* (Abschnitt 2.2.5) kann sich ein Nutzer für die automatische Alarmierung registrieren. Der *Sensor Planning Service* (Abschnitt 2.2.6) ermöglicht die Kontrolle und Steuerung assoziierter Sensoren. Der *Web Notification Service* (Abschnitt 2.2.7) bietet die notwendig gewordenen Funktionalitäten zur asynchronen Kommunikation. Um Sensoren und Messdaten auffinden zu können, wird im SWE-Framework der *Catalogue Service*[22] benutzt.

Im Folgenden werden die Spezifikationen skizziert, die im Rahmen der SWE-Initiative entwickelt werden. Zum Zeitpunkt des Schreibens dieser Arbeit ist der Standardisierungsprozess dieser Spezifikationen noch nicht abgeschlossen und der Status der Implementation Specification noch nicht erreicht. Infolgedessen wird hier mit vorläufigen Entwürfen der Spezifikationen gearbeitet. Die Architektur des Frameworks sowie die im Rahmen dieser Arbeit implementierten Komponenten und Applikationen müssen daher bei zukünftigen Änderungen der Spezifikationen gegebenenfalls angepasst werden.

2.2.1 Observations and Measurements

Die Spezifikation *Observations and Measurements* (O&M) (COX 2006) definiert ein domänenunabhängiges, konzeptionelles Modell zur Repräsentation der Resultate von Beobachtungen und Messungen. Außerdem stellt die O&M Spezifikation eine Implementierung dieses Modells in Form eines Applikationsschemas basierend auf der GML bereit. Dies umfasst eine Reihe von XML-Schemata, die die GML Spezifikation erweitern und eine XML-basierte Kodierung raumzeitvarianter Messdaten ermöglichen.

Modell und Implementierung der O&M Spezifikation werden insbesondere für den Aufbau der Antwortdokumente der GetObservation-Operation des Sensor Observation Service (Abschnitt 2.2.4) benötigt, können aber auch als generelles Hilfsmittel für den standardisierten Umgang mit technischen Messungen benutzt werden. Für die vorliegen-

[21] Bei den *SWE Common* Elementen handelt es sich um fundamentale Datentypen, die Typen der GML erweitern. Diese können z.B. als Eingabeparameter von Service-Operationen oder als Grundlage für komplexere Typen zur Kodierung von Sensor- oder Metadaten benutzt werden. Die Schemata der SWE Common Datentypen werden aktuell allerdings nicht in einer eigenen Spezifikation veröffentlicht, sondern sind Teil der *Sensor Model Language* und *Observations & Measurements* Spezifikationen.

[22] Die *Catalogue Service* Spezifikation wird nicht innerhalb der SWE-Initiative entwickelt. Für eine Beschreibung sei daher auf NEBERT & WHITESIDE (2005) verwiesen.

de Arbeit hat die O&M Spezifikation große Bedeutung, da die zu visualisierenden Geosensordaten gemäß dem definierten Datenschema kodiert sind.

2.2.1.1 Das Basic Observation Model

Das konzeptionelle Modell der O&M Spezifikation definiert Typen und deren Beziehungen zueinander, die aus einer nutzerorientierten Perspektive die Beschreibung von *Observations* ermöglichen. Mit dem Begriff Observation ist allgemein ein Vorgang zur Schätzung eines Wertes eines bestimmten Phänomens gemeint. Der Schätzvorgang einer Observation kann eine Messung, eine Berechnung, eine Simulation oder ähnliches sein.

Die Grundlage des Modells der O&M Spezifikation bildet das *Basic Observation Model*. Es ist in UML-Notation[23] in Abbildung 2-1 dargestellt. Im Zentrum steht dabei die abstrakte Observation (`AbstractObservation`). Dieser Typ wird als Ereignis (`Event`) in der Zeit (`time`) aufgefasst und kann einen gewissen Raumbezug (`location`) aufweisen. Der Typ `Event` ist eine Spezialisierung des Typs `FeatureType` des Featuremodells der GML Spezifikation, wodurch eine Observation ebenfalls ein spezielles Feature darstellt.

Das zentrale Element einer Observation ist das Resultat (`result`). Der abstrakte Typ `AbstractObservation` weist hierzu noch kein Attribut auf, definiert aber, dass ein konkreter Untertyp dieses Attribut bereitstellen muss. Der Wert des Resultats beschreibt eine bestimmte Beobachtungsvariable (`observedProperty`) eines im Raum identifizierbaren Geoobjekts (`featureOfInterest`). Um den Wert des Resultats zu bestimmen, wird eine bestimmte Prozedur (`procedure`) verwendet.

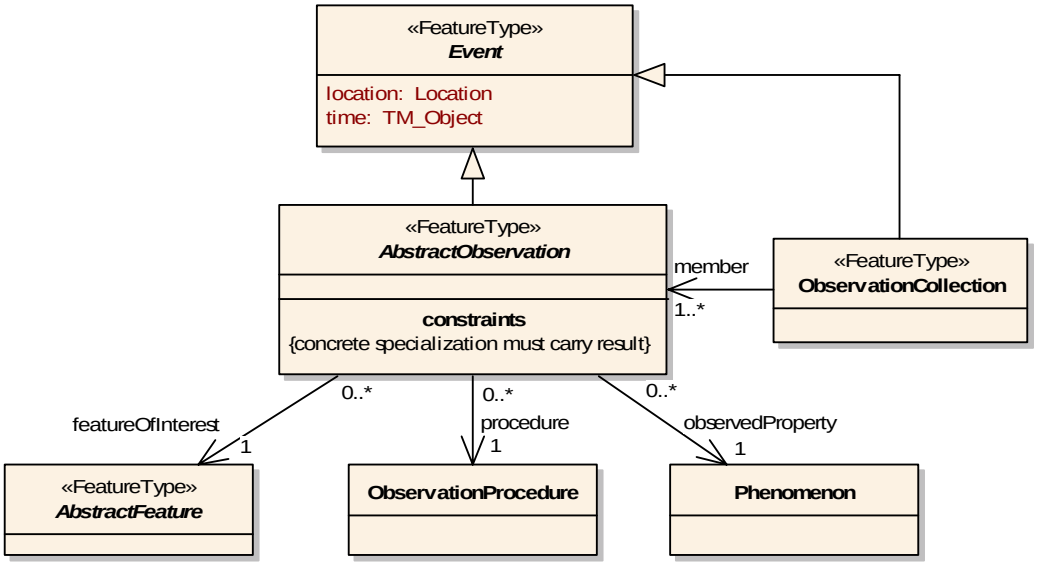

Abbildung 2-1: Das Basic Observation Model in vereinfachter Darstellung (nach COX 2006)

[23] Eine Beschreibung der *Unified Modeling Language* (UML) und der von ihr unterstützten Diagrammtypen findet sich z.B. bei BOOCH et al. (2005).

Das `featureOfInterest`, die `procedure` und die `observedProperty` werden in separaten Typen gekapselt, die mit der Observation assoziiert sind. Diese werden im Folgenden näher betrachtet.

Die Beobachtungsvariable (`observedProperty`) ist eine Charakteristik des Geoobjekts. Sie ist eine Instanz des Phänomens (`Phenomenon`), welches beim Schätzvorgang der Observation betrachtet wird. Das Phänomen kann eine beliebige mess- oder beobachtbare Größe sein, wie zum Beispiel Strahlung, Temperatur, Konzentration bestimmter chemischer Stoffe oder auch Alter und Anzahl auftretender Individuen einer Spezies. Da beabsichtigt ist, dass die O&M Spezifikation über viele verschiedene Applikationsdomänen hinweg eingesetzt wird, ist es nicht möglich ein umfassendes und maßgebendes Modell für alle denkbaren Phänomene zu konstruieren. Die O&M Spezifikation definiert hingegen eine vom Typ `Phenomenon` ausgehende Menge von Basis-Phänomenen, welche die Beschreibung davon abgeleiteter Phänomene ermöglichen. Damit einmal definierte Phänomene über mehrere Applikationen hinweg eingesetzt und identifiziert werden können, bietet sich die Möglichkeit sie als Einträge in einem GML *Dictionary* zugreifbar und über *Uniform Resource Identifier* (URI) identifizierbar zu machen. Solche Dictionaries könnten von bekannten Organisationen wie dem OGC oder auch von Drittanbietern bereitgestellt werden (NA & PRIEST 2006).

Das `featureOfInterest` ist die Instanz eines Feature-Typs, der in einem separaten Applikationsschema definiert wird. Es repräsentiert das Geoobjekt, bezüglich dessen die Observation stattfindet. Das `featureOfInterest` kann z.B. Informationen zur räumlichen und zeitlichen Ausdehnung enthalten. Die O&M Spezifikation enthält beispielhaft die Definition des Feature-Typs `Station`, welcher eine stationäre Sensorplattform repräsentiert die mit einer beliebigen Anzahl von Sensoren bestückt und durch eine Position im Raum gekennzeichnet ist. Zu beachten ist die Unterscheidung der spezifizierten räumlichen Ausdehnung des `featureOfInterest` und der `location` der Observation, welche die Lokalisierung der Prozedur zur Zeit des Schätzvorgangs darstellt. Während bei einer *in-situ* Observation die `location` übereinstimmt mit der Lokalisierung des `featuresOfInterest`, ist dies bei einer *remote* Observation nicht der Fall[24].

Die `procedure` beschreibt die für den Schätzvorgang verwendete Prozedur. Dies kann neben dem typischen Fall des Sensors oder Messinstruments auch eine Simulation, ein menschlicher Beobachter oder ähnliches sein. Das in der O&M enthaltene Modell zur Beschreibung der Prozedur ist auf allgemeine Parameter, wie z.B. die Seriennummer eines Messinstruments, beschränkt. Eine ausführlichere Beschreibung der Prozedur kann mit Hilfe der SensorML (Abschnitt 2.2.2) oder der TML (Abschnitt 2.2.3) erfolgen.

2.2.1.2 Skalare Observation-Typen

Im Modell der O&M Spezifikation werden verschiedene Typen konkreter Observations unterschieden. Diese werden vom abstrakten Typ `AbstractObservation` abgeleitet. Die

[24] Im Falle einer *in-situ* Observation findet die Erfassung eines Wertes für ein Phänomen in unmittelbarer Umgebung des Sensors bzw. der Prozedur statt. Bei einer *remote* Observation befindet sich das Geoobjekt bezüglich dessen der Schätzvorgang stattfindet in einiger Entfernung. Das zu beobachtende Phänomen wird dann für gewöhnlich mit der vom Objekt ausgehenden Strahlung gemessen (BOTTS 2006). Stationäre Sensoren, die eine in-situ Messung durchführen, sind beispielsweise Windgeschwindigkeitsmessgeräte an Wetterstationen. Ein typisches Beispiel für mobile, remote Sensoren ist die Satellitenfernerkundung.

einfachen, skalaren Observation-Typen besitzen jeweils einen festen Resultattyp. Es werden die Observation-Typen `Measurement`, `CategoryObservation`, `TruthObservation` sowie `CountObservation` unterschieden.

Der Typ `Measurement` eignet sich zur Kodierung von allgemeinen Messungen. Im Resultat wird neben dem gemessenen Wert eine zugehörige Maßeinheit angegeben. Listing 2-1 zeigt beispielhaft die Kodierung einer `Measurement`-Instanz bei der es sich um eine Temperaturmessung handelt.

Eine `CategoryObservation` kann zur Kodierung textueller Werte verwendet werden. Beispielsweise kann das Ergebnis der Erfassung des Phänomens `Windrichtung` (in der Form `Nord`, `Ost`, …) mit diesem Typ kodiert werden.

Das Resultat der `CountObservation` ist eine Zahl, welche der Anzahl einer Beobachtungsvariablen entspricht. Hiermit kann z.B. die Anzahl von Individuen einer Spezies pro Flächeneinheit kodiert werden.

Die `TruthObservation` ermöglicht die Kodierung boolscher Werte. Sie kann z.B. genutzt werden, um eine Aussage über die Existenz eines beobachteten Phänomens zu treffen.

```
<om:Measurement gml:id="ID" xsi:type="om:MeasurementType">
  <om:time>
    <gml:TimeInstant xsi:type="gml:TimeInstantType">
      <gml:timePosition>2006-11-01T11:30:00</gml:timePosition>
    </gml:TimeInstant>
  </om:time>
  <om:location/>
  <om:procedure xlink:href="urn:ogc:sensor:TempSensor_TMP75"/>
  <om:observedProperty xlink:href="urn:ogc:phenomenon:temperature"/>
  <om:featureOfInterest>
    <om:Station gml:id="Pretoria">
      <gml:name>Weatherstation_Pretoria</gml:name>
      <gml:location>
        <gml:Point srsName="EPSG:4326">
          <gml:coordinates>27.23 -27.63</gml:coordinates>
        </gml:Point>
      </gml:location>
    </om:Station>
  </om:featureOfInterest>
  <om:result uom="units.xml#Celsius">13.4</om:result>
</om:Measurement>
```

Listing 2-1: Temperaturmessung kodiert in O&M

2.2.1.3 Komplexe Observation-Typen

Die skalaren Observation-Typen sind beschränkt auf Resultate, die lediglich einen einzelnen Wert repräsentieren. Viele Messvorgänge produzieren jedoch komplexe Ergebnisse die in mehreren Komponenten ausgedrückt werden müssen. Einer oder mehrere der folgenden Gründe können dafür ursächlich sein:

- Das beobachtete Phänomen setzt sich aus mehreren Teilen zusammen. Ein Beispiel hierfür wäre das Phänomen `Wetter`, welches aus den Phänomenen `Temperatur` und `Niederschlag` zusammengesetzt ist.
- Das `featureOfInterest` besteht aus mehreren Elementen, die jeweils separate Ergebnisse liefern, z.B. ein Netz von Messstationen.

- In einer Observation sollen mehrere Messzeiten und dazu korrespondierende Ergebnisse ausgedrückt werden.

Das Modell der O&M Spezifikation bietet verschiedene Möglichkeiten diese komplexen Observations zu kodieren.

Ein Weg komplexe Observations auszudrücken ist die Verwendung einer `ObservationCollection` (Abbildung 2-1). Die `ObservationCollection` ist ein spezialisiertes `Event`, welches eine Sammlung von `member`-Observations aggregiert. Die `member`-Observations können einen beliebigen Untertyp der `AbstractObservation` annehmen.

Ebenso kann der generische Typ `Observation` angewendet werden, um komplexe Observations zu repräsentieren. Er legt als Resultattyp `Any` fest, wodurch impliziert wird, dass jeder beliebige Typ erlaubt ist. Damit ein Client ein solches Resultat interpretieren kann, muss ein Verweis auf das Schema des Resultattyps in der Instanz der Observation spezifiziert werden.

Eine weitere Alternative zur Repräsentation komplexer Observations bietet die `CommonObservation`. Das Resultat stellt hierbei eine Liste von Tupeln dar, die Werte der beobachteten Phänomene enthalten und gemäß der Aufnahmezeit sortiert sind.

2.2.1.4 Resultate als externe Referenz

Bei der Erläuterung der oben aufgeführten konkreten Observation-Typen, findet die Kodierung der Sensordaten *in-line* statt, d.h. direkt in der serialisierten Instanz der Observation. Nutzt man die generische `Observation`, so kann der Resultattyp auch als externe Referenz festgelegt werden. In diesem Fall werden in der Observation lediglich die Metadaten des Schätzvorgangs enkodiert, während die erfassten Daten von einer externen Ressource (*out-of-band*) bereitgestellt werden. Die im O&M-Format kodierte Observation referenziert diese Ressource durch die Angabe einer URI. Dies kann beispielsweise die URL einer Anfrage eines zweiten OGC Web Service sein. Der out-of-band Mechanismus ist insbesondere nützlich für Sensordaten, die im Binärformat (z.B. digitale Bilder oder Videos) oder als Datenstrom (z.B. TML, Abschnitt 2.2.3) zur Verfügung stehen sollen.

2.2.2 Sensor Model Language

Die Spezifikation der *Sensor Model Language* (SensorML) (BOTTS 2006) definiert ein Modell sowie dessen XML-Kodierung zur Beschreibung von *Prozessen*. Unter einem Prozess kann dabei ein vom Sensor durchgeführter Messprozess oder eine Weiterverarbeitung erfasster Daten, im Sinne einer Post-Prozessierung, verstanden werden. Aber auch Sensoren und Sensorsysteme selbst werden als Prozesse modelliert und können mit Hilfe der SensorML beschrieben werden.

Für einen Prozess können die Eingaben und Ausgaben, die Parametrisierung sowie die Beschreibung der Prozessmethode spezifiziert werden. Außerdem können für einen Prozess zahlreiche Metadaten definiert werden. Hierzu zählen Angaben zur Identifikation und Klassifikation eines Prozesses oder auch Charakteristiken wie z.B. die zeitliche Verfügbarkeit oder die räumliche Beschreibung eines Prozesses.

Die SensorML Spezifikation kann im Kontext der SWE-Initiative unterschiedliche Aufgaben erfüllen; zu diesen gehören:

- Das Prozess-Konzept ist so generisch, dass unterschiedlichste Typen von Sensoren und Sensorsystemen funktional beschrieben werden können. Diese Beschreibungen können in Katalogdiensten publiziert werden, um somit sämtliche Sensoren im Sensor Web auffindbar und nutzbar zu machen.
- Sowohl die Akquirierung der Rohdaten durch ein oder mehrere Sensoren, als auch die einzelnen Schritte der weiteren Prozessierung können durch verknüpfte Prozesse beschrieben werden. Wird dem Client die Beschreibung dieser Prozesskette bereitgestellt, so können ihn die darin enthaltenen detaillierten Informationen bei der Interpretation und Analyse der Daten unterstützten.
- Weiterhin kann die SensorML Spezifikation dazu verwendet werden Prozessketten zu definieren, die von speziellen Softwaresystemen zur Ausführung gebracht werden können. In eine derartige *on-demand* Prozessierung können die Resultatwerte erfasster Geosensordaten eingegeben werden, um daraus neue Daten zu generieren.

2.2.3 Transducer Markup Language

Die *Transducer Markup Language* (TML) Spezifikation (HAVENS 2006) definiert eine XML-basierte Sprache, die benutzt werden kann, um zum einen aus hardwarenaher Sicht die Charakteristika von Sensoren und Sensorsystemen zu beschreiben und zum anderen erfasste Daten auszutauschen oder zu archivieren.

TML ermöglicht die Übertragung der Sensordaten zusammen mit den zu ihrer Interpretation notwendigen Metadaten. Diese Metadaten betreffen die Systemeigenschaften, räumliche und zeitliche Beschreibungen sowie Beziehungen zu anderen Sensoren. Außerdem wird die Struktur der übertragenen Daten beschrieben. Diese kann flexibel und individuell für einen Sensor definiert werden. Somit können Daten unterschiedlichster Sensortypen interoperabel ausgetauscht werden. Die Komplexität der Daten kann stark variieren. Beispielsweise können einfache Temperaturmessungen wie auch komplexe Satellitenbilder mit Hilfe der TML übertragen werden.

TML adressiert insbesondere die Übertragung von Sensordaten in Form eines Echtzeit-Datenstroms. Um mit einer möglichst geringen Bandbreite auszukommen, wurde bei der Entwicklung von TML auf ein schlankes Format Wert gelegt, welches sich möglichst nahe an den originalen, „rohen“ Daten orientiert. Im Gegensatz dazu ist das in der O&M Spezifikation definierte Format semantisch reicher und beinhaltet „höherwertige“ Informationen, wodurch die Interpretation durch den Client erleichtert wird. TML eignet sich als Vorstufe, um eine spätere Übersetzung der Daten in das O&M-Format vorzunehmen. Im Rahmen dieser Arbeit werden daher Sensordaten betrachtet, die gemäß der O&M Spezifikation kodiert sind.

2.2.4 Sensor Observation Service

Aufgabe der *Sensor Observation Service* (SOS) Spezifikation[25] (NA & PRIEST 2006) ist die Beschreibung einer Web Service Schnittstelle, die den standardisierten Zugang zu den von Sensoren erfassten Daten möglich macht. Der Service dient dabei als Mittler zwischen einem Client und einem Sensordatenarchiv bzw. einem Echtzeit-Sensorsystem. Die unterschiedlichen Kommunikationsprotokolle, Datenformate und Standards der verschiedenen Sensorsysteme verbirgt der SOS hinter seiner standardisierten Schnittstelle. Vom Client angefragte Sensordaten werden gemäß der O&M Spezifikation in Form von Observations enkodiert. Der generische Charakter der Service Schnittstelle und der verwendeten Spezifikationen, ermöglicht die Unterstützung nicht nur stationärer sondern auch mobiler Sensoren, die ihre Messungen sowohl in-situ oder ebenso remote erfassen können.

Da die Observation-Typen als Spezialisierungen des GML Feature-Typs implementiert sind, könnten die O&M-enkodierten Sensordaten auch über einen WFS bereitgestellt werden. Die WFS Spezifikation macht keine Einschränkungen hinsichtlich der zu unterstützenden GML Applikationsschemata. Über die Schnittstelle des WFS können Features beliebiger Feature-Typen abgefragt werden. Die Schnittstelle des SOS ist hingegen explizit auf die Abfrage von Sensordaten angepasst. Der SOS kann daher als domänenspezifische Spezialisierung des WFS verstanden werden (WYTZISK 2003).

Die von der Schnittstelle des SOS definierten Operationen sind aufgeteilt in drei Profile: das *Core Profile*, das *Transactional Profile* sowie das *Enhanced Profile*. Während eine SOS-Instanz das Core Profile verbindlich anbieten muss, sind die Operationen der beiden letztgenannten Profile optional. Bietet eine SOS Implementierung alle Operationen an, so erfüllt sie das *Entire Profile*.

2.2.4.1 Das Core Profile

Das Core Profile spezifiziert die *GetCapabilities*-, die *GetObservation*- sowie die *DescribeSensor*-Operation. Die *GetCapabilities*-Operation gibt ein XML-kodiertes Dokument zurück, welches den Client mit Metadaten über die Service-Instanz sowie über die bereitgestellten Sensordaten versorgt. Die SOS Spezifikation hält sich bei Syntax und Struktur des Request und Response der GetCapabilities-Operation an die Festlegungen der OWS Common Spezifikation (Abschnitt 2.1.3). Neben den von der OWS Common Spezifikation bereits vorgegebenen Schemata für die Abschnitte `ServiceIdentification`, `ServiceProvider` und `OperationsMetadata` definiert die SOS Spezifikation ein eigenes Schema für das `Contents`-Element[26] und fügt den neuen `FilterCapabilities`-Abschnitt hinzu.

Der `FilterCapabilities`-Abschnitt legt fest welche Filterausdrücke bei der Ausführung der GetObservation-Operation (siehe unten) benutzt werden können. Die SOS Spezifikation bezieht sich auf die *Filter Encoding Specification* (VRETANOS 2005a) und verwendet eine Untermenge der dort definierten Schemata, um die Formulierung komplexer Filter-

[25] Der Sensor Observation Service ist Nachfolger des *Sensor Collection Service* (SCS) (MCCARTHY 2003). Die Arbeiten an der SCS Spezifikation begannen während der OGC Web Services 1.1 Initiative.

[26] Die SOS Spezifikation verzichtet auf eine Erweiterung des von der OWS Common Spezifikation definierten minimalen Schemas für den Contents-Abschnitt und definiert eine unabhängige Struktur.

ausdrücke zu ermöglichen. Neben räumlichen und zeitlichen Filtern können über die Parameter der GetObservation-Operation auch Filter definiert werden, die das Resultat der zurückzugebenden Observations betreffen.

Der `Contents`-Abschnitt enthält eine Liste der vom SOS bereitgestellten *Observation Offerings*. Ein Observation Offering bildet eine logische Gruppierung von verwandten Observations, die über gleiche oder ähnliche Charakteristiken beschrieben werden können. Im Allgemeinen werden so Observations zusammengefasst, die vom gleichen Sensorsystem aufgenommen werden. Für jeden Parameter der GetObservation-Operation besitzt das Observation Offering ein Element, welches dessen zulässigen Wertebereich beschreibt.

Mit Hilfe der *DescribeSensor*-Operation können detaillierte Metadaten über die mit dem SOS assoziierten Sensoren abgefragt werden. Die Antwort dieser Operation wird gemäß den in der SensorML (Abschnitt 2.2.2) oder der TML (Abschnitt 2.2.3) Spezifikation definierten Formaten enkodiert.

Über die *GetObservation*-Operation lassen sich die von den Sensoren erfassten Daten abfragen. Verbindlich festzulegende Parameter dieser Operation sind die ID des ausgewählten Observation Offerings (`offering`), ein oder mehrere Beobachtungsvariablen, für die Sensordaten abgefragt werden sollen (`observedProperty`), sowie das Datenformat in dem die Daten zurückgegeben werden sollen (`resultFormat`)[27]. Weiterhin stehen zur Einschränkung der Ausgabemenge der Observations einige optionale Parameter zur Verfügung. Es können die Messzeit (`eventTime`), verwendete Prozeduren (`procedure`), oder eine Auswahl von Geoobjekten (`featureOfInterest`) definiert werden. Der `Result`-Parameter erlaubt die Spezifikation eines Filterausdrucks, so dass lediglich Observations zurückgegeben werden deren Resultatwerte diesem Filter entsprechen. Über die Parameter `resultModel` und `responseMode` können die Kodierung und das Format des Antwortdokuments konfiguriert werden.

In Listing 2-2 ist das Beispiel einer GetObservation-Anfrage dargestellt, die für das Observation Offering *Weather_Southafrica* und einen spezifizierten Zeitpunkt Temperaturmessungen anfordert. Die Messungen sollen bezüglich dem `featureOfInterest` (bzw. der Wetterstation) mit der ID *Pretoria* stattgefunden haben und einen Wert größer als 10° Celsius aufweisen.

[27] Der übliche Fall ist, dass die Resultate in-line zurückgegeben werden, und damit ebenfalls im O&M-Format enkodiert sind. Für den Fall, dass die Resultate von einer externen Ressource, also out-of-band, bereitgestellt werden, sind hier verschiedene Datenformate denkbar (Abschnitt 2.2.1.4).

```
<sos:GetObservation service="SOS" version="0.0.31">
  <sos:offering>Weather_Southafrica</sos:offering>
  <sos:eventTime>
    <ogc:TEquals>
      <gml:TimeInstant>
        <gml:timePosition>2006-11-01T11:30:00</gml:timePosition>
      </gml:TimeInstant>
    </ogc:TEquals>
  </sos:eventTime>
  <sos:observedProperty>urn:ogc:phenomenon:temperature</sos:observedProperty>
  <sos:procedure>urn:ogc:sensor:TempSensor_TMP75</sos:procedure>
  <sos:featureOfInterest>
    <sos:ObjectID>Pretoria</sos:ObjectID>
  </sos:featureOfInterest>
  <sos:Result>
    <PropertyIsGreaterThan>
      <Literal>10</Literal>
    </PropertyIsGreaterThan>
  </sos:Result>
  <sos:resultFormat>text/xml;subtype="OM"</sos:resultFormat>
  <sos:resultModel>Measurement</sos:resultModel>
</sos:GetObservation>
```

Listing 2-2: GetObservation Request zur Abfrage von Temperaturmessungen

Listing 2-1 zeigt eine im O&M-Format enkodierte Temperaturmessung, die als mögliche Antwort auf diese Anfrage betrachtet werden kann.

2.2.4.2 Das Transactional Profile

Über die im Transactional Profile definierten Operationen kann eine SOS-Instanz angewiesen werden neue Sensoren (*RegisterSensor*) zu registrieren oder von registrierten Sensoren erfasste Observations (*InsertObservation*) einzufügen. Die Idee dabei ist, die Entwicklung von kommerziellen SOS-Implementierungen zu unterstützen, die in verschiedenen Sensornetzwerken zum Einsatz kommen können. Die Betreiber eines Sensornetzwerks können eine existierende SOS-Implementierung nutzen und müssen lediglich einfache Skripte entwickeln, welche die RegisterSensor- bzw. InsertObservation-Operation aufrufen, um erfasste Daten zu publizieren (NA & PRIEST 2006).

2.2.4.3 Das Enhanced Profile

Das Enhanced Profile spezifiziert sechs Operationen. Die *GetResult*-Operation erlaubt dem Client eine wiederholte Anfrage von Observations für unterschiedliche Zeiten. Hierzu muss der Client eigentlich wiederholt die GetObservation-Operation verwenden, wobei sich die einzelnen Requests lediglich in der spezifizierten Zeit unterscheiden. Diese Redundanz kann durch Verwendung der GetResult-Operation vermieden werden, die somit den Vorteil mit sich bringt, eine geringere Bandbreite zu benötigen. Über die *GetFeatureOfInterest*-Operation kann der Client die Instanz eines `featureOfInterest` abfragen. Die *GetFeatureOfInterestTime*-Operation gibt diejenigen Zeitintervalle zurück, für die eine SOS-Instanz für ein spezifiziertes `featureOfInterest` Daten zurückliefern kann. Das XML-Schema des Typs eines `featureOfInterest` kann über die *DescribeFeatureOfInterest*-Operation bezogen werden. Die *DescribeObservationType*-Operation gibt das Schema des Observation-Typs zurück, welcher zur Kodierung von Messungen

eines spezifizierten Phänomens benutzt wird. Über die *DescribeResultModel*-Operation kann die Beschreibung des `result`-Elements eines Observation-Typs in Form eines XML-Schemas abgefragt werden.

2.2.5 Sensor Alert Service

Die in der *Sensor Alert Service* (SAS) Spezifikation (SIMONIS 2006a) definierte Web Service Schnittstelle kann mit einem *Event Notification System* verglichen werden. Sensoren können über eine SAS-Instanz eintretende Ereignisse publizieren, so dass registrierte und am Ereignis interessierte Clients darüber benachrichtigt, also „alarmiert“ werden.

Ein Sensor oder anderer *Ereignis-Produzent* registriert dazu die durch ihn erzeugten Typen von Ereignissen über die *Advertise*-Operation beim SAS. Ereignisse können dabei unterschiedlichster Art sein. Die Überschreitung eines Grenzwertes bei einer durchgeführten Messung oder die Änderung des Zustands eines Sensors (z.B.: niedriger Batterieladezustand) seien hier beispielhaft genannt.

Über die *Subscribe*-Operation kann sich der Client als *Ereignis-Konsument* für die registrierten Ereignisse subskribieren. Die Subskription kann dabei mit verschiedenen Bedingungen verknüpft werden. Beispielsweise könnte ein Client die Bedingung definieren, lediglich über Ereignisse informiert zu werden, die in einem bestimmten geografischen Raumausschnitt auftreten.

Die Publizierung von Ereignissen durch den Ereignis-Produzenten geschieht nicht direkt über den SAS sondern über einen assoziierten Messaging Server, dessen Implementierung nicht Teil der SAS Spezifikation ist. Der Ereignis-Produzent veröffentlicht ein auftretendes Ereignis beim Messaging Server. Dies geschieht über das *Extensible Messaging and Presence Protocol* (XMPP). Der Client kann durch Anmeldung am Messaging Server ebenfalls via XMPP notifiziert werden. Es ist allerdings auch möglich, dass der SAS im sogenannten *Last-Mile-Mode* andere Protokolle, wie E-Mail oder Telefon, zur Benachrichtigung anbietet. In diesem Fall kann der SAS auf einen Web Notification Service (Abschnitt 2.2.7) zurückgreifen, um die Notifizierung des Clients an diesen Dienst zu delegieren.

2.2.6 Sensor Planning Service

Mit der *Sensor Planning Service* (SPS) Spezifikation (SIMONIS 2005) wurde eine interoperable Web Service Schnittstelle entwickelt, die Clients erlaubt assoziierten Sensoren (Mess-)Aufträge zu erteilen. Dazu aggregiert die Schnittstelle des SPS Operationen, die den gesamten Prozess der Steuerung und Planung von Aufträgen abdecken. Dies beinhaltet die Prüfung eines Auftrags auf Durchführbarkeit (*GetFeasibility*) sowie das Absetzen (*Submit*) und die Statusverfolgung (*GetStatus*) eines Auftrags. Damit dem Client ausreichend Informationen zur Formulierung eines Messauftrages zur Verfügung stehen, kann die Beschreibung der Syntax und Parametrisierung zur Definition von Aufträgen abgefragt werden (*DescribeTasking*). Weiterhin besteht die Möglichkeit zur Modifikation (*Update*) und zum Abbruch (*Cancel*) eines abgesetzten Auftrags.

Der SPS reicht die abgesetzten Messaufträge an die assoziierten Sensoren weiter. Die von den Sensoren erfassten Daten werden allerdings nicht vom SPS gesammelt. Stattdessen

bietet die Schnittstelle des SPS die Möglichkeit, in Erfahrung zu bringen, über welchen Service die aufgenommenen Daten bereitgestellt werden (*DescribeResultAccess*).

Die Durchführung definierter Aufträge kann eventuell längere Zeitspannen in Anspruch nehmen. Daher verwendet der SPS ein asynchrones Interaktionsmuster, um mit dem Client zu kommunizieren und nutzt hierfür die Funktionalitäten eines Web Notification Service (Abschnitt 2.2.7).

2.2.7 Web Notification Service

Die Spezifikation des *Web Notification Service* (WNS) (SIMONIS & ECHTERHOFF 2006) definiert eine Web Service Schnittstelle, die Mechanismen zum Versand XML-kodierter Benachrichtigungen zur asynchronen Kommunikation bereitstellt.

Bestimmte Service-Operationen können längere Ausführungszeiten benötigen. Eine synchrone Kommunikation, also eine direkte Beantwortung der über das HTTP-Protokoll gestellten Anfrage ist eventuell nicht möglich bzw. sinnvoll. In so einem Fall können die vom WNS bereitgestellten Funktionalitäten zur asynchronen Kommunikation zwischen einem Sender und einem Empfänger genutzt werden. Bei dem Sender handelt es sich typischerweise um einen Web Service. Der Empfänger kann ein menschlicher Nutzer oder wiederum ein Web Service sein.

Der Empfänger registriert sich bei einem WNS (*Register*) und spezifiziert dazu Kommunikationsprotokoll und Adresse, über die Nachrichten empfangen werden sollen. Protokolle, die ein WNS hierzu anbieten kann, sind z.B. HTTP, E-Mail, Telefon oder Fax.

Dem registrierten Empfänger können nun Nachrichten zugestellt werden (*DoNotification*). Der WNS dient dabei als Protokollbrücke und leitet die Nachricht lediglich weiter. Von der Spezifikation werden zwei Kommunikationsmuster unterschieden, die durch die Nutzung spezieller Nachrichtentypen verwendet werden können. Bei der *One-Way-Communication* wird eine Benachrichtigung an den Client versandt, ohne darauf eine Antwort zu erwarten. Initiiert der Sender eine *Two-Way-Communication*, so sollte der Empfänger mit dem Versand einer Antwortnachricht reagieren, die vom WNS an den Sender weitergeleitet wird.

3 Anforderungsanalyse

In diesem Kapitel wird eine Analyse der Anforderungen durchgeführt, die an das zu entwickelnde System, das *OX-Framework*, gestellt werden. ‚OX-Framework' steht für *OWS Access Framework*, wodurch bereits angedeutet wird, dass eine Kerneigenschaft des Systems darin besteht, den Zugang zu unterschiedlichen OGC Web Services bereitzustellen.

Es wird zwischen funktionalen und technischen Anforderungen unterschieden. Erstere werden in Abschnitt 3.1 analysiert. Hier gilt es festzustellen welche Möglichkeiten zur Visualisierung von Geosensordaten bestehen, um daraus zu fordernde Eigenschaften für das zu entwickelnde Framework abzuleiten. Dazu wird zunächst eine Einschränkung vorgenommen, für welche Formen von Geosensordaten das System die Umsetzung von Visualisierungsmethoden ermöglichen soll.

Im Anschluss werden in Abschnitt 3.2 die Anforderungen aus technischer Sicht analysiert. Die zentrale Anforderung wird dabei der Entwurf der Systemarchitektur als (Software-)Framework sein. Welche Eigenschaften dazu von der Architektur zu erfüllen sind, wird in diesem Abschnitt festgestellt.

3.1 Funktionale Anforderungen

3.1.1 Restriktive Vorgaben an die Eigenschaften der Geosensordaten

Die Ausprägungsformen von Geosensoren und den von ihnen erzeugten Geosensordaten sind sehr vielfältig. Aufgrund der zeitlichen Begrenzung dieser Diplomarbeit können nicht sämtliche Formen in die nun folgenden Überlegungen einbezogen werden. Stattdessen wird im Vorfeld eine Einschränkung auf bestimmte Typen von Geosensordaten vorgenommen, für die Visualisierungsmethoden auf Basis des zu entwickelnden Frameworks realisierbar sein sollen.

Dies seien Geosensordaten, die von Sensoren eines Sensornetzwerks erfasst werden, welche zu einer diskreten Messzeit einen Messvorgang durchführen und dessen Messresultate in skalare Werte aufgespalten werden können. Als Beispiel sei hier ein Temperatursensor aufgeführt. Dieser Sensor könnte z.B. zu der diskreten Messzeit „23.01.2007 15:24 Uhr" einen Messvorgang durchführen, dessen Messergebnis sich durch den skalaren Wert „7" (° Celsius) beschreiben lässt. Ausgeschlossen werden somit beispielsweise binäre Daten, die von einem Videosensor erfasst werden und kontinuierlich in Echtzeit über einen Datenstrom abrufbar sind.

Die im Rahmen dieser Arbeit zu implementierenden Visualisierungskomponenten sollen auf O&M-enkodierte Geosensordaten angewandt werden können. Bezieht man die genannte Restriktion auf das O&M-Datenformat, so sind diese Geosensordaten mit Hilfe der skalaren Observation-Typen (Abschnitt 2.2.1.2) darstellbar. In Abschnitt 5.3.9 wird

diskutiert, inwieweit das Framework auch die Implementierung von Visualisierungsmethoden komplexerer Formen von Geosensordaten zulässt.

3.1.2 Visualisierung zur Exploration der Geosensordaten

Bei der Erfassung raumzeitlicher Phänomene mit Hilfe von Geosensornetzwerken können große Mengen an Daten erzeugt werden. Die Interpretation dieser Sensordaten erfordert geeignete Techniken und Methoden. In der Einleitung wurde bereits darauf hingewiesen, dass die visuelle Darstellung der Daten ein wichtiges Hilfsmittel zur Informationsextraktion sein kann.

Bereits DIBIASE (1990) ermutigte in seiner Beschreibung des visuellen Forschungsprozesses die Werkzeuge der geografischen Visualisierung (auch *Geovisualisierung*, MACEACHREN & KRAAK 2001) zur *Exploration* großer Datenmengen zu benutzen. Die Techniken und Methoden der visuellen Exploration sind durch eine hohe Interaktivität gekennzeichnet, und helfen dem Nutzer, die unbekannten Datenräume und deren Charakteristiken zu erkunden. Die Visualisierung der Daten ermöglicht dem Nutzer einen tieferen Einblick in die Daten, wodurch er nach Zusammenhängen suchen und neue Hypothesen formulieren kann (MACEACHREN & KRAAK 1997). Wesentlich bei diesen Techniken ist die Einbeziehung des Menschen. Mit Hilfe der Visualisierung kann der Mensch Informationen aus den Daten ableiten und anschließend durch Verstehen in Wissen überführen (MACEACHREN & KRAAK 2001). Dabei wird die Leistungsfähigkeit des menschlichen visuellen Systems mit der immer weiter fortschreitenden Leistungsfähigkeit moderner Computertechnologie verbunden und für die Interpretation der Daten genutzt. Die visuelle Exploration kann somit z.B. auch eine geeignete Alternative gegenüber vollautomatisierten Verfahren aus der Statistik sein (KEIM 2002).

In der vorliegenden Arbeit soll die Visualisierung daher zur *Exploration* der Sensordaten eingesetzt werden. Um den mentalen Erkenntnisprozess zu stimulieren und die Interpretation der Daten zu erleichtern, ist es wichtig, dass mehrere alternative Repräsentationen darstellbar sind (KRAAK 2002). Von der Architektur des OX-Frameworks ist also gefordert, eine softwaretechnische Basis zu bieten, auf der verschiedenartige Visualisierungsmethoden flexibel realisiert werden können.

3.1.3 Möglichkeiten zur Visualisierung von Geosensordaten und daraus resultierende Anforderungen

Die aus Geosensordaten ableitbaren Informationen weisen drei für die Visualisierung wesentliche Aspekte auf: den *Sachbezug*, den *Raumbezug* sowie den *Zeitbezug*. Dies entspricht der traditionellen, triadischen Komposition von Geoinformation (WYTZISK 2003). Da in dieser Arbeit die zu visualisierenden Geosensordaten gemäß dem Schema der O&M Spezifikation kodiert sein sollen, nutzen die zu implementierenden Visualisierungsmethoden folglich die in den Observations bereitgestellten Informationen. Auch in diesen Informationen finden sich die drei genannten Aspekte wieder.

Mit dem *Sachbezug* der Sensordaten seien hier die von den Sensoren erzeugten Messwerte gemeint. Im Sinne der O&M ist der Sachbezug das Resultat einer Observation, also die Schätzung des Wertes einer fachlichen Eigenschaft des Geoobjekts bezüglich dessen die

Observation stattfindet (COX 2006). Die Gesamtheit all dieser fachlichen Eigenschaften kann auch als *Thematik* des Geoobjekts bezeichnet werden (STREIT 2004).

Der *Zeitbezug* ist das wesentliche Charakteristikum eines Observation-Ereignisses. Diese Messzeit ist typischerweise ein Zeitpunkt oder ein Zeitintervall. Aus dem zeitlichen Bezug der Geosensordaten lassen sich die zeitlichen Veränderungen, die *Dynamik*, der Eigenschaften eines Geoobjekts ableiten.

Ein für die Visualisierung relevanter *Raumbezug* kann zum einen über die Beschreibung der *Geometrie* des „observierten" Geoobjekts bereitgestellt werden und zum anderen kann in der Observation die Position des Sensors zur Zeit der Messung gespeichert werden.

Vom OX-Framework wird gefordert, die softwaretechnischen Rahmenbedingungen zu schaffen, welche die Implementierung von Visualisierungsmethoden ermöglichen, die sowohl den *Sachbezug*, den *Raumbezug*, den *Zeitbezug* als auch Kombinationen dieser drei Aspekte der Geosensordaten abbilden können. Im Folgenden wird eine Einordnung möglicher Visualisierungstechniken für die unterschiedlichen Aspekte der Geosensordaten durchgeführt. Daraus sollen die Anforderungen an das Framework abgeleitet werden, die notwendig sind, um die Implementierung solcher Visualisierungsmethoden zu ermöglichen. Die Benennung möglicher Techniken zur Visualisierung von Geosensordaten erhebt dabei keinen Anspruch auf Vollständigkeit. Vielmehr sollen die daraus abgeleiteten Funktionalitäten des Frameworks ein hohes Maß an Flexibilität bieten und so generisch sein, dass möglichst vielfältige Visualisierungsmethoden realisiert werden können.

3.1.3.1 Visualisierung des Sachbezugs

Um den Sachbezug, also die thematischen Aspekte der Sensordaten bzw. der sie betreffenden Geoobjekte zu visualisieren, kann man sich statistischer Grafiken bedienen. Hierzu zählen beispielsweise Box-Plots, Liniendiagramme, Streudiagramme oder Histogramme. Die theoretischen Grundlagen dieser Visualisierungstechniken sind in den grafischen Methoden der Deskriptiven Statistik und der Explorativen Daten Analyse[28] zu sehen. Diese Techniken können zur Suche nach Strukturen, Auffälligkeiten oder Anomalien in den Daten verwendet werden und Anstöße zur Bildung von Hypothesen und Modellen liefern (HEILER & MICHELS 1994).

Der Raumbezug wird bei diesen Visualisierungstechniken vernachlässigt. Der zeitliche Bezug kann hingegen bei einigen der Techniken mit einbezogen werden. Beispielsweise wird bei einem Zeitreihendiagramm, welches eine spezielle Form des Liniendiagramms darstellt, die Zeit auf der Abszisse abgetragen, so dass z.B. zeitliche Trends oder Muster in den Daten ersichtlich werden können.

Für das zu entwickelnde System ergibt sich die Anforderung, vielfältige Formen von Grafiken bzw. Diagrammansichten visualisieren zu können, die den Sachbezug der Geosensordaten abbilden.

[28] Eine Einführung in die Deskriptive Statistik, die Explorative Datenanalyse und die dort verwendeten grafischen Darstellungstechniken findet sich beispielsweise bei POLASEK (1994) oder bei HEILER & MICHELS (1994).

3.1.3.2 Visualisierung des Raumbezugs

Visualisierungen, die den Raumbezug der Geosensordaten abbilden, sollen in einzelnen georeferenzierten Kartenlayern dargestellt werden können. Verschiedene Kartenlayer sollen in einer gemeinsamen Kartenansicht grafisch überlagert werden können. Für den Nutzer sollen Interaktionen mit dieser Kartenansicht möglich sein. Ein Hinein- bzw. Herauszoomen sowie ein Verschieben des dargestellten Raumausschnitts sind notwendige Funktionalitäten. ANDRIENKO & ANDRIENKO (1999) unterstreichen die Wichtigkeit der Interaktion mit der Karte für die visuelle Exploration der Daten.

Um die Visualisierung des Raumbezugs der Geosensordaten möglichst aussagekräftig zu gestalten, soll außerdem die Möglichkeit zur visuellen Überlagerung mit unterstützenden Geodaten bestehen, die nicht zwingend von Geosensoren erfasst wurden. Diese Geodaten sollen über verschiedene OGC Web Services bezogen werden. Der Architekturentwurf für die Anbindung der Dienste soll ein so hohes Maß an Flexibilität bieten, dass eine vordefinierte Eingrenzung auf bestimmte Typen von OGC Web Services ausbleiben kann.

Von Interesse ist ebenfalls die Kombination des Raumbezugs mit den anderen Aspekten der Geosensordaten. Zur kombinierten Darstellung mit dem Sachbezug gibt es zwei grundsätzliche Möglichkeiten: zum einen die *direkte* und zum anderen die *indirekte* Darstellung (SCHUMANN & MÜLLER 2000).

Direkte Darstellung des Raumbezugs

Bei der direkten Darstellung wird Raum- und Sachbezug in derselben visuellen Repräsentation veranschaulicht. Hierzu sollen die Methoden der thematischen, zweidimensionalen Kartografie verwendet werden. Formen thematischer Karten sind beispielsweise Choroplethenkarten, Isolinienkarten, Symbolkarten oder Diagrammkarten. Eine ausführliche Klassifizierung der Ausprägungsformen thematischer Karten findet sich beispielsweise bei KRAAK & ORMELING (1997).

Indirekte Darstellung des Raumbezugs

Häufig ist es schwierig, alle fachlichen Eigenschaften in einer kartografischen Darstellung abzubilden. Ebenso kann es sein, dass eventuell nicht alle Werte der ausgewählten Attribute gleichzeitig in einer direkten Darstellung des Raumbezugs abgebildet werden können. Aus diesem Grund kann es sinnvoll sein, eine *indirekte* Darstellung des Raumbezugs zu benutzen. Diese veranschaulicht Raum- und Sachbezug über getrennte Repräsentationen, die aber in einem gemeinsamen Kontext stehen.

In dem zu entwickelnden System soll sowohl die direkte Darstellung des Raumbezugs als auch die indirekte Darstellung ermöglicht werden.

3.1.3.3 Visualisierung des Zeitbezugs

Um die von den Geosensoren erfassten raumzeitlichen Prozesse verstehen zu können, ist die Einbeziehung des zeitlichen Bezugs der Daten essentiell (BLOK 2005). Um Sach-, Raum- und Zeitbezug kombiniert zu visualisieren und somit die Dynamik der „beobachteten“ Geoobjekte darzustellen, existieren eine Vielzahl von Methoden. SCHUMANN & MÜLLER (2000) unterscheiden grundsätzlich zwei Visualisierungstechniken für den

Zeitbezug: *statische* und *dynamische* Repräsentationen. Bezogen auf die Visualisierung von geografischen Daten ergeben sich daraus die folgenden beiden Techniken.

Temporale statische Karten

Kartografische Darstellungen sind immer mit einem bestimmten Zeitpunkt oder Zeitintervall assoziiert. Auch wenn dies oft nicht explizit dokumentiert ist, so repräsentieren kartografische Darstellungen die Geoobjekte immer in einem bestimmten *Zustand*, der einen gewissen Zeitbezug besitzt (MACEACHREN 1995). Statische temporale Karten versuchen hingegen explizit raumzeitliche Informationsinhalte zu visualisieren. Dazu wurden in der Kartografie verschiedene Methoden entwickelt, die versuchen, die Prozesse und Veränderungen der Realität wiederzugeben. Eine dieser Methoden ist die Darstellung von Zeitreihendiagrammen in einer Karte. Mit dieser Methode kann die Dynamik der thematischen Aspekte dargestellt werden. Eine Übersicht über existierende Methoden zur Repräsentation raumzeitlicher Informationen in statischen Karten findet sich beispielsweise bei VASILIEV (1997).

Temporale Animation

Die Visualisierung der Dynamik raumzeitlicher Phänomene mittels der obengenannten statischen Verfahren ist allerdings starken Einschränkungen unterworfen. Dies resultiert daraus, dass versucht wird, den vierdimensionalen Georaum – also die drei Raumdimensionen und die Dimension der Zeit – in die statische, zweidimensionale Kartenebene abzubilden. Eine andere Möglichkeit der Visualisierung raumzeitlicher Veränderungen bieten temporale Animationen. Nach DRANSCH (1997) ist die temporale Animation eine Sequenz von kartografischen Darstellungen. Sie zeigen die Veränderungen räumlicher Daten, die innerhalb eines Zeitintervalls der *Realzeit* stattgefunden haben. Diese Veränderungen können somit zur *Präsentationszeit* der Animation beobachtet werden. Mit der Präsentationszeit steht der Kartografie neben den zwei Dimensionen der Darstellungsebene (und den grafischen Variablen) ein weiteres Ausdrucksmittel zur Verfügung. Die Animation erweitert damit die Darstellungsmöglichkeiten der traditionellen Kartografie um eine zusätzliche Dimension (DRANSCH 1997).

Aufgrund dieser Vorteile der dynamischen Techniken und aufgrund der Wichtigkeit der Berücksichtigung des Zeitbezugs bei der Visualisierung von Geosensordaten, ergibt sich die Anforderung für das zu entwickelnde System, die Erzeugung temporaler Animationen zu ermöglichen.

3.2 Technische Anforderungen

3.2.1 Entwicklung der Systemarchitektur als Framework

Wiederverwendung von Software-Architekturen ist einer der wichtigsten Faktoren im Software-Entwicklungsprozess, um Entwicklungsaufwand und -kosten zu reduzieren und gleichzeitig die Zuverlässigkeit und die Qualität von Software zu steigern (SCHERP & BOLL 2006). Auch für das hier zu entwickelnde System soll die Wiederverwendung einmal implementierter Funktionalität ein zentrales Ziel darstellen.

FAYAD et al. (1999) stellen fest, dass objektorientierte *Software-Frameworks* eine vielversprechende Technologie sind, um bewährte Software-Architekturen und Implementierungen wiederzuverwenden. Software-Frameworks senken durch ihre Wiederverwendbarkeit in verschiedenen Applikationen den Wartungsaufwand, da nur noch eine statt einer Vielzahl von Varianten getestet und gewartet werden muss. Der mehrfache Einsatz von Frameworks führt außerdem zur frühzeitigeren Aufdeckung von Fehlern, wodurch Frameworks im Allgemeinen stabiler sind als spezifische Lösungen (BIRRER et al. 1995). Somit ist die zentrale Anforderung an die Architektur des Systems die Entwicklung als Framework.

Der Aufwand zur Entwicklung eines Frameworks kann im Vergleich zur konventionellen Anwendungsentwicklung signifikant höher sein. Dies wird durch den generischen Charakter der Architektur hervorgerufen. Daher rentiert sich die Entwicklung eines Frameworks erst, wenn es in mehreren Anwendungen eingesetzt wird (SCHERP & BOLL 2006). Das im Rahmen dieser Arbeit entwickelte OX-Framework wird als Open Source Projekt innerhalb der 52°North Initiative (KRAAK et al. 2005) publiziert. Dadurch sind auch die organisatorischen Rahmenbedingungen für eine häufige Wiederverwendung in verschiedenen Applikationen und Projekten geschaffen.

BALZERT (2001, S. 843) definiert den Begriff „Framework“ wie folgt:

> *„Ein Rahmenwerk (Framework) ist ein durch den Software-Entwickler anpassbares oder erweiterbares System kooperierender Klassen, die einen wiederverwendbaren Entwurf für einen bestimmten Anwendungsbereich implementieren.“*

Der Autor unterstreicht weiterhin den Unterschied zu einfachen Klassenbibliotheken, welche die Code-Wiederverwendung anstreben. Die Eigenschaften eines Frameworks gehen über die einer Klassenbibliothek hinaus, indem es die *Entwurfs-Wiederverwendung* zum Ziel hat. Zwei zentrale Charakteristika die ein Framework dazu aufweist und die ebenfalls im OX-Framework umgesetzt werden sollen, sind die Anpassbarkeit der Framework-Architektur durch vordefinierte *Variationspunkte* (engl. „variation points“) sowie die *Umkehrung des Kontrollflusses* (engl. „inversion of control“) (SCHERP & BOLL 2006).

3.2.1.1 Variationspunkte

Damit Frameworks in verschiedenen Anwendungen eingesetzt werden können, stellen sie flexible Variationspunkte in Form von Interfaces oder abstrakten Klassen zur Verfügung, an denen konkrete Anwendungen anknüpfen und die Funktionalität entsprechend erweitern können. Die Typisierung der Funktionalität ist an der Stelle der Variationspunkte zwar bereits festgelegt, die tatsächliche Ausprägung der Funktionalität wird allerdings erst durch die konkreten Unterklassen bestimmt (SCHERP & BOLL 2006).

Im OX-Framework sollen die verschiedenen Aufgaben zur Anbindung unterschiedlicher OWS Typen gekapselt werden. Folgende Aufgaben sind zu unterscheiden:

- der Zugriff auf einen bestimmten OWS Typ und die Ausführung der von der Service-Schnittstelle angebotenen Operationen.
- die Integration der vom Service angebotenen Geodaten, so dass diese zur weiteren Verarbeitung zur Verfügung stehen.
- die Visualisierung der empfangenen Geodaten.

Diese Aufgaben sollen von sogenannten *Anbindungskomponenten* realisiert werden. Der Begriff „Komponente“ wird in dieser Arbeit wie folgt verstanden:

> *„…eine Komponente (component) ist ein abgeschlossener, binärer Software-Baustein, der eine anwendungsorientierte, semantisch zusammengehörende Funktionalität besitzt, die nach außen über Schnittstellen zur Verfügung gestellt wird.“* (BALZERT 2001, S.856)

Die Schnittstellen der Anbindungskomponenten sollen als Variationspunkte des Frameworks entworfen werden. Somit ist das System nicht von vornherein auf bestimmte Typen von OGC Web Services eingeschränkt. Stattdessen werden die architektonischen Voraussetzungen geschaffen, um das Framework je nach Anwendung flexibel um neue Anbindungskomponenten für bestimmte Service-Typen zu erweitern.

3.2.1.2 Umkehrung des Kontrollflusses

Konventionell entwickelte Anwendungen nutzen einfache Klassenbibliotheken und steuern die Reihenfolge und das Zusammenspiel der aufzurufenden Operationen selbst. D.h. der Kontrollfluss folgt dem *Call-Down-Prinzip*. Der Entwickler entwirft die Architektur der Anwendung selbst. Der Aufrufmechanismus von Frameworks folgt hingegen dem *Call-Back*-Prinzip. Die vom Anwendungsentwickler definierten Klassen werden vom Framework aufgerufen. Damit wird der Hauptkontrollfluss der auf dem Framework aufbauenden Anwendung invertiert und vom Framework gesteuert (SCHERP & BOLL 2006).

Der vom OX-Framework gesteuerte Hauptkontrollfluss soll im Wesentlichen die Aufgaben der Integration sowie der anschließenden Visualisierung der Geodaten realisieren. Ein Entwickler, der eine Anwendung auf dem Framework aufbaut, kann mit Hilfe des Call-Back-Mechanismus diese Funktionalitäten nutzen und sich um die Details seiner Anwendung und um die Entwicklung von anwendungsspezifischen Zusatzfunktionalitäten kümmern.

3.2.2 Einordnung des OX-Frameworks in der Drei-Schichten-Architektur

Bezieht man sich auf die Terminologie einer Schichten-Architektur, die nach FOWLER et al. (2003) aus den drei Hauptschichten *Präsentation*, *Geschäftslogik* und der *Datenverwaltung* besteht, so soll das OX-Framework auf der Ebene der Geschäftslogik in verschiedenen Applikationen eingesetzt werden. Die Schicht der Geschäftslogik (auch Domänen- oder Anwendungslogik) (engl. „business logic“) steht, wie in Abbildung 3-1 dargestellt, zwischen Präsentations- und Datenverwaltungs-Schicht.

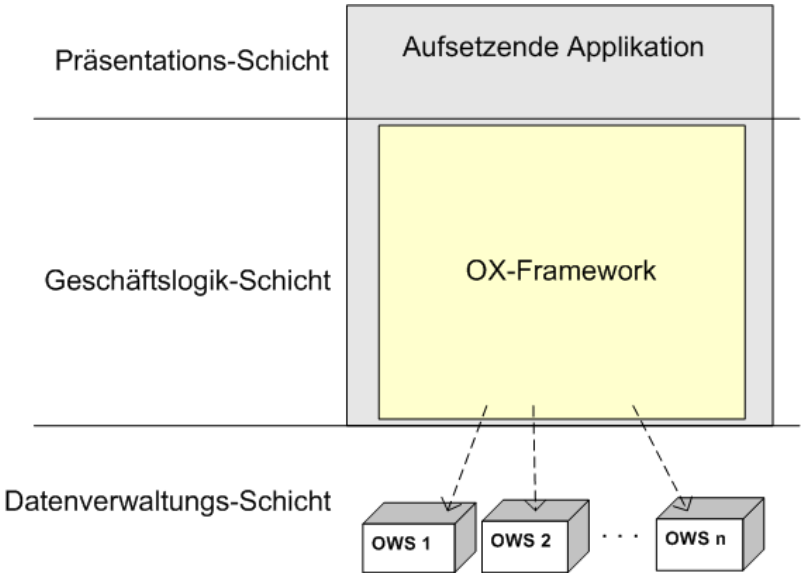

Abbildung 3-1: Das OX-Framework in der Drei-Schichten-Architektur

Die *Datenverwaltungs-Schicht* ist im Wesentlichen für die Speicherung persistenter Daten verantwortlich. Im Falle des OX-Frameworks wird die Datenverwaltungs-Schicht durch die einzelnen OWS-Instanzen repräsentiert, auf die das System zugreift.

Die *Geschäftslogik-Schicht* führt die Kernaufgaben aus, die ein System in seiner bestimmten Domäne erfüllt. Das OX-Framework soll für verschiedene Applikationen die in Abschnitt 3.1 beschriebenen Aufgaben der Integration und Visualisierung von Geosensor- bzw. allgemeinen Geodaten bereitstellen. Funktionalitäten einer aufsetzenden Anwendung, die über die bereitgestellte Geschäftslogik hinausgehen, müssen von dieser Applikation selbst implementiert werden.

Die *Präsentations-Schicht* behandelt die Interaktion des Nutzers mit dem Software-System. Es werden Nutzeranfragen interpretiert, verarbeitet und in Befehle der Geschäftslogik umgesetzt. Außerdem werden Informationen an den Nutzer zurückgegeben. Im Falle des OX-Frameworks soll die Präsentation des Systems durch die aufsetzende Anwendung realisiert werden.

Bei der Entwicklung der Framework-Architektur ist es wichtig, die Abhängigkeiten zwischen den einzelnen Schichten zu minimieren. Insbesondere dürfen die Geschäftslogik und die Datenverwaltung nicht von der Präsentation abhängen. Ist diese Anforderung erfüllt, so ist es möglich unterschiedliche Präsentationen mit derselben Basis zu realisieren, wodurch es zur Wiederverwendung der Geschäftlogik in verschiedenen Applikationen kommt. Außerdem haben Änderungen in der Präsentation keinerlei Konsequenzen für die tieferen Schichten.

In der klassischen Drei-Schichten-Architektur wird die Präsentations-Schicht durch eine grafische Benutzerschnittstelle (engl. „graphical user interface“, GUI) realisiert. Dies kann z.B. ein Rich- oder ein Thin-Client sein[29]. Die Applikationen der Präsentations-

[29] Ein Rich-Client (auch als Fat- oder Thick-Client bezeichnet) führt die komplette Präsentations-Schicht, und eventuell auch noch tiefere Schichten, auf dem Client-Rechner aus. Bei einem Thin-Client wird hingegen die Präsentation auf Server und Client aufgeteilt. Häufig wird ein solcher Thin-Client durch einen Browser

Schicht, die auf dem OX-Framework aufbauen, sollen allerdings nicht auf grafische Benutzerschnittstellen beschränkt werden. Es soll ebenso möglich sein, die vom Framework bereitgestellten Funktionalitäten z.B. für die Entwicklung eines neuen Webdienstes zu benutzen.

Für die Visualisierung von Sensordaten bieten Rich- oder Thin-Clients oft Vorteile wie gute Interaktionsmöglichkeiten mit den Daten oder sie erlauben individuell angepasste Darstellungen der Daten. Es ist aber ebenso interessant, Sensordaten, die von einem Sensor Observation Service bereitgestellt werden, über einen Web Service anzubieten, der beispielsweise konform zur Spezifikation des Web Map Service (Abschnitt 2.1.4) ist und Visualisierungen von Kartenlayern zurückgibt. Dies hat den Vorteil, dass die Visualisierungen der Sensordaten von einer größeren Nutzergemeinde betrachtet werden können, da WMS-Clients im Gegensatz zu SOS-Clients bereits heute weit verbreitet sind.

Im Rahmen dieser Arbeit soll daher basierend auf dem OX-Framework zum einen ein Frontend im Sinne eines Rich-Clients (Abschnitt 5.2.1) und zum anderen eine Server-Applikation, welche ein WMS-Frontend darstellt (Abschnitt 5.2.2), als Proof-of-Concept implementiert werden.

repräsentiert, der HTML-Code interpretiert und diesen über die Web Schnittstelle des Server-Teils der Präsentation bezieht (FOWLER et al. 2003). Andere Autoren sprechen bei einer Architektur, die die Präsentations-Schicht auf Client und Server verteilt, auch von einer Vier-Schichten-Architektur (siehe z.B. THOMAS 2006).

4 Architektur des Frameworks

In diesem Kapitel wird die Architektur des Frameworks beschrieben, welche die Grundlage zur Implementierung aufsetzender Anwendungen sowie die Basis für die Implementierung der in Abschnitt 3.2.1.1 eingeführten Anbindungskomponenten bildet. Beispielhafte Implementierungen von Anbindungskomponenten und insbesondere von Visualisierungskomponenten für Geosensordaten werden, ebenso wie die beiden prototypischen Applikationen, im Kapitel 5 vorgestellt.

4.1 Einteilung der Architektur in Subsysteme

Um den beschriebenen Anforderungen (Abschnitt 3.2) an die Architektur des Frameworks gerecht zu werden, ist diese in zwei Subsysteme unterteilt. Abbildung 4-1 zeigt als Erweiterung der Abbildung 3-1 die Subsystem-Aufteilung und wie diese im Kontext der Drei-Schichten-Architektur zu sehen ist.

Das *Core Subsystem* stellt das Zentrum des Frameworks dar. Es steuert den Hauptkontrollfluss und beinhaltet die Kernbereiche der Geschäftslogik. Außerdem stellt dieses Subsystem Mechanismen bereit, welche die Kommunikation mit der aufsetzenden Applikation ermöglichen.

Das *Service-Connector Subsystem* enthält die Implementierungen der Komponenten, welche die Anbindung an die Datenverwaltungs-Schicht, also die OGC Dienste, realisieren und deren Funktionalität im OX-Framework sowie den aufsetzenden Anwendungen nutzbar machen. Wie die gestrichelten Pfeile in Abbildung 4-1 deutlich machen, ist der Core *vollständig unabhängig* von diesem Subsystem und somit auch von der Datenverwaltungs-Schicht.

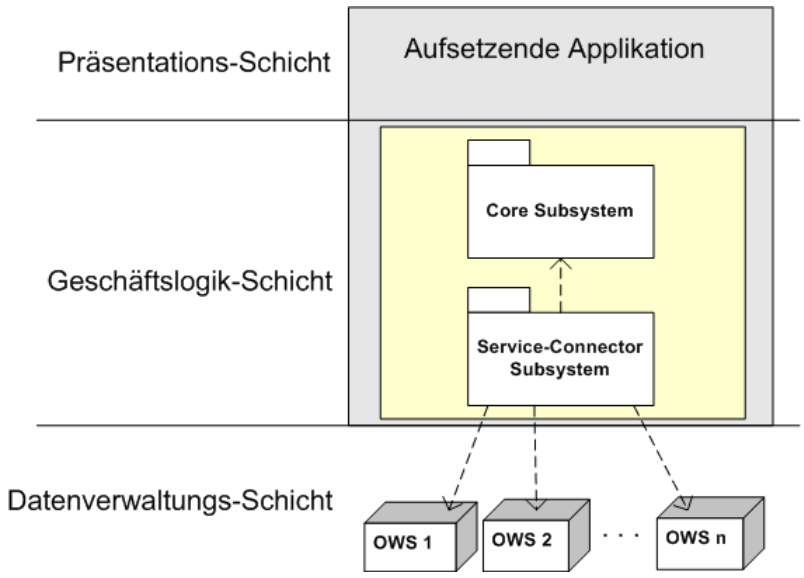

Abbildung 4-1: Die Subsysteme des Frameworks in der Drei-Schichten-Architektur

4.2 Datenmodelle des Core Subsystems

Um die Aufgaben der Geschäftslogik realisieren zu können, beinhaltet der Core drei auf OGC Spezifikationen basierende Datenmodelle: das *Common Capabilities Modell*, das *Feature Modell* sowie das *Context Modell*. Die beiden ersten Modelle werden vom Service-Connector Subsystem zur Integration der Datenverwaltungs-Schicht verwendet. Sie ermöglichen die Anbindung der OGC Web Services sowie die einheitliche Verwendung der angebotenen Geodaten. Das Context Modell hat die primäre Aufgabe, den Status einer auf dem Framework basierenden Applikation zu verwalten.

4.2.1 Das Common Capabilities Modell

Im Abschnitt 3.1.3.2 wurde die Anforderung formuliert, nicht nur Sensor Observation Services sondern unterschiedliche OWS Typen ansprechen zu können, um somit auch unterstützende Geodaten in die Visualisierung der Sensordaten einbeziehen zu können. Um dies zu ermöglichen, müssen die Metadaten der OWS Typen im Framework einheitlich zugreifbar sein. Dazu wird ein Datenmodell benötigt, dessen Klassenstruktur die Abbildung der Capabilities der unterschiedlichen OWS Typen ermöglicht.

Wie bereits in Abschnitt 2.1.3 beschrieben, legt die OWS Common Spezifikation für bestehende und zukünftige OWS Typen einen Großteil des Inhalts und der Struktur der Service Metadaten fest. Daher kann diese Spezifikation hier als Grundlage für die Entwicklung des Common Capabilities Modells benutzt werden.

Mit Hilfe dieses Modells ist es möglich, die Capabilities der anzusprechenden Dienste in Objekte zu überführen[30]. Über diese Objekte kann dann durch einfache Methodenaufrufe auf die Service Metadaten zugegriffen werden. Von besonderem Interesse für aufsetzende Applikationen sind dabei die zulässigen Wertebereiche der Parameter der vom Dienst bereitgestellten Operationen.

Im Folgenden wird das im OX-Framework verwendete Common Capabilities Modell in vereinfachter Form beschrieben.

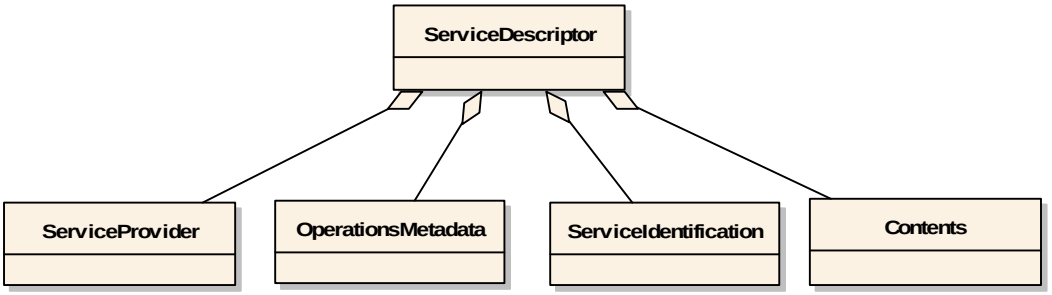

Abbildung 4-2: Übersicht über das Common Capabilities Modells in UML-Notation

Ein Objekt vom Typ `ServiceDescriptor` stellt den Zugang zum Common Capabilities Modell bereit. Die vier Abschnitte, die nach der OWS Common Spezifikation für die

[30] Diese Überführung eines XML-Dokuments in Objekte wird auch als *Unmarshalling* bezeichnet (Abschnitt 5.1.5)

Capabilities-Dokumente vorgesehen sind, sind in Form der Klassen `ServiceProvider`, `ServiceIdentification`, `Contents` und `OperationsMetadata` mit der `ServiceDescriptor`-Klasse assoziiert (Abbildung 4-2).

Die `ServiceProvider`- und `ServiceIdentification`-Klasse bilden genau die Informationen ab, die im Schema der OWS Common Spezifikation definiert wurden. Dies sind auf der einen Seite Informationen zur Betreiberorganisation des Dienstes und auf der anderen Seite ist dies eine grundlegende Beschreibung des Dienstes mit Angaben wie Typ und Titel. Diese Informationen können z.B. von aufsetzenden Client-Applikationen benutzt werden, um sie dem Nutzer über die grafische Benutzerschnittstelle zu präsentieren. Aufgrund des einheitlichen Modells würde die Implementierung *einer* grafischen Nutzerschnittstelle ausreichen, um die Metadaten unterschiedlicher OWS Typen anzeigen zu können.

Der Contents-Abschnitt eines Capabilities-Dokuments beschreibt die vom Service angebotenen Datenressourcen. Diese Datenressourcen sind je nach Service-Typ unterschiedlich geartet. Beispielsweise ist dies bei einem WMS ein Layer, bei einem WCS ein Coverage und bei einem SOS ein Observation Offering. Die OWS Common Spezifikation definiert daher für die Beschreibung der Datenressourcen lediglich ein minimales Schema, welches von den OWS Spezifikationen erweitert werden soll. Im Rahmen dieser Arbeit wurde allerdings festgestellt, dass die OWS Spezifikationen im Allgemeinen auf die Erweiterung dieses minimalen Schemas verzichten, und stattdessen eigene, unabhängige Schemata definieren (Beispiel SOS, Abschnitt 2.2.4.1). Daher enthält ein Objekt der hier definierten `Contents`-Klasse eine Liste von `Dataset`-Objekten, die lediglich die ID, den Titel und die textuelle Beschreibung (Abbildung 4-3) der Datenressource beinhalten.

Um trotzdem die Metadaten eines `Datasets` abbilden zu können, die zur Ausführung der datenofferierenden Service-Operation benötigt werden, wird das generische Konzept der `ValueDomains` (Abschnitt 4.2.1.1) verwendet. Mit Hilfe dieses Konzepts können die Wertebereiche der Operations-Parameter in den Beschreibungen der Service-Operationen aufgelistet werden. Die Informationen des Contents-Abschnitts eines Capabilities-Dokuments werden somit bei der Instanziierung des `OperationsMetadata`-Objekts (siehe unten) benutzt. Auf die Definition spezieller `Dataset`-Typen für die einzelnen OWS Spezifikationen kann durch diese Vorgehensweise verzichtet werden.

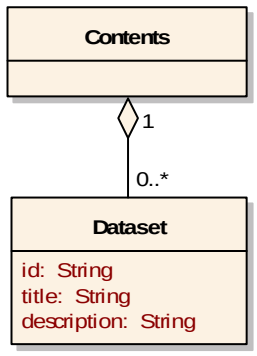

Abbildung 4-3: Die Contents-Klasse in UML-Notation

Die `OperationsMetadata`-Klasse (Abbildung 4-4) bildet die Grundlage für Applikationen, um die von der Service-Schnittstelle angebotenen Operationen inspizieren zu können. Die assoziierten `Operations` sind durch einen Namen gekennzeichnet. Weiterhin ist

eine Operation mit einem oder mehreren DCP-Objekten assoziiert. Diese besitzen Attribute, die den Zugang der Operation über das Internet beschreiben.

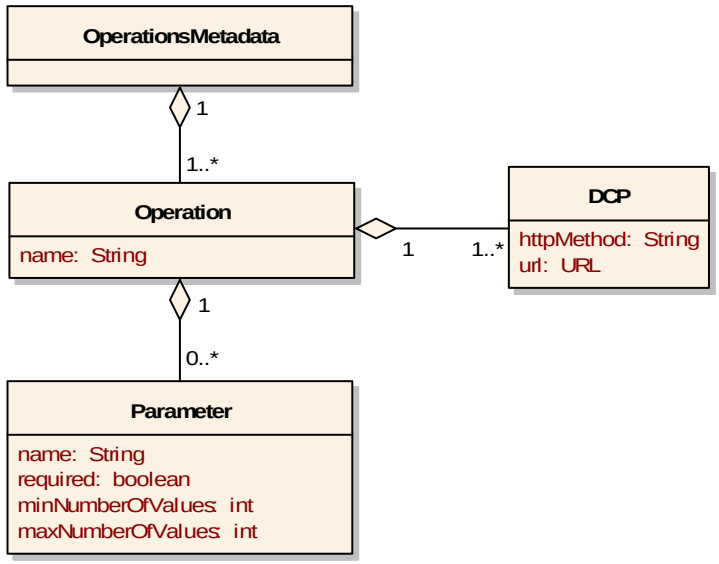

Abbildung 4-4: Die OperationsMetadata-Klasse in UML-Notation

Von besonderer Bedeutung sind die mit der `Operation` verknüpften `Parameter` (Abbildung 4-4). Diese `Parameter`-Objekte stellen die zur Ausführung der Service-Operation verwendbaren Parameter dar. Beispiele solcher Parameter sind `offering` für die GetObservation-Operation (Abschnitt 2.2.4.1) des SOS oder der `BBOX`-Parameter der GetMap-Operation (Abschnitt 2.1.4) des WMS. Für die spätere Ausführung einer Service-Operation muss eine Verknüpfung der Operations-Parameter mit einem oder gegebenenfalls mehreren Werten stattfinden. Diese Verknüpfung erfolgt mit Hilfe von Objekten der Klasse `ParameterShell`, die in Abschnitt 4.2.3 erläutert wird.

Das `required`-Attribut gibt darüber Auskunft, ob der Parameter notwendig ist für die Ausführung der Service-Operation. Wie viele Werte maximal oder minimal mit dem Parameter verknüpft werden müssen bzw. können, wird in den Attributen `maxNumberOfValues` und `minNumberOfValues` festgelegt.

Das Common Capabilities Modell muss eine Beschreibung der für einen Parameter zulässigen Wertemenge ermöglichen. Dies ist z.B. nötig, um in einer grafischen Benutzerschnittstelle eine Auswahl gültiger Werte für die Ausführung einer Operation präsentieren zu können (Abschnitt 5.2.1.1). Außerdem kann durch eine solche Wertemengen-Beschreibung bereits vor der Ausführung der Service-Operation eine clientseitige Kontrolle über die Zulässigkeit der mit einem Parameter verknüpften Werte erfolgen. Daher wird in dem vorliegenden Common Capabilities Modell das Konzept der `ValueDomains` eingeführt, welches die Beschreibung des gültigen Wertebereichs ermöglicht.

4.2.1.1 Das ValueDomain Konzept

Jeder `Parameter` ist mit mindestens einem `ValueDomain`-Objekt assoziiert (Abbildung 4-5). Der Wertetyp, der von einer `ValueDomain` repräsentiert wird, kann mit Hilfe der

`getValueType()`-Methode abgefragt werden. Die Überprüfung, ob sich ein spezifischer Wert in der Wertemenge befindet, erfolgt mittels der `containsValue()`-Methode.

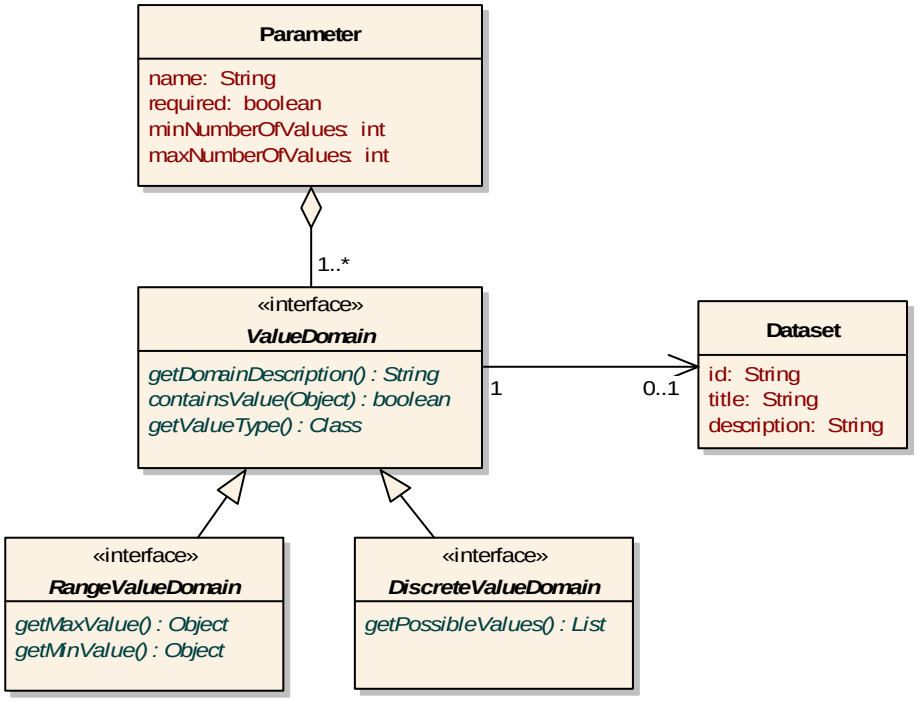

Abbildung 4-5: Das ValueDomain Konzept in UML-Notation

Es werden zwei Spezialisierungen des `ValueDomain`-Interfaces unterschieden: `DiscreteValueDomain` und `RangeValueDomain`. Diese stellen Methoden bereit, um Informationen über die von der `ValueDomain` repräsentierten Werte beziehen zu können. Während die `DiscreteValueDomain` die Beschreibung von diskreten Wertemengen ermöglicht, repräsentiert die `RangeValueDomain` einen kontinuierlichen Wertebereich. Von diesen abstrakten Schnittstellen existieren im OX-Framework eine Reihe konkreter Unterklassen.

Beispielsweise existiert eine konkrete `DiscreteValueDomain` für Zeichenketten. Mit Hilfe dieser Klasse lassen sich z.B. die zulässigen Werte des `offering`-Parameters der GetObservation-Operation (Abschnitt 2.2.4.1) abbilden. Die Methode `getPossibleValues()` listet dann die IDs sämtlicher von der SOS-Instanz bereitgestellten Observation Offerings auf.

Wohingegen der Parameter `eventTime` der GetObservation-Operation mit einer speziellen `RangeValueDomain` für Zeit-Objekte assoziiert sein kann. Die Methoden `getMinValue()` und `getMaxValue()` geben den Anfangs- bzw. Endzeitpunkt einer Zeitperiode zurück, innerhalb derer `Observations` vom SOS abgerufen werden können.

Am Beispiel `eventTime` ist zu erkennen, dass es Parameter gibt, die spezifische Wertebereiche für jede vom Service angebotene Datenressource besitzen. Der Wertebereich des `eventTime`-Parameters ist im Allgemeinen nicht für alle Observation Offerings gleich, sondern kann zwischen diesen differieren. Ein anderes Beispiel ist der `BBOX`-Parameter der GetMap-Operation des WMS, welcher ebenfalls zwischen den vom WMS angebotenen Layern differieren kann. Um diese Situationen im Modell abbilden zu können, ist es möglich, eine `ValueDomain` mit einem `Dataset` des `Contents`-Objekts zu assoziieren

(Abbildung 4-5). Außerdem kann ein `Parameter` mit mehreren `ValueDomains` für unterschiedliche `Datasets` verknüpft werden.

Über die Verknüpfung eines Parameters mit einer `ValueDomain` und der `ValueDomain` mit einem `Dataset`, ist nun indirekt die Beschreibung der Metadaten eines `Datasets` möglich, die für die Ausführung der datenofferierenden Service-Operation relevant sind. Auf die direkte Speicherung der Metadaten wurde in der `Dataset`-Klasse noch verzichtet (siehe oben).

Mit dem `ValueDomain` Konzept können Wertebereiche unterschiedlicher Parametertypen beschrieben werden. Falls die bereitgestellten konkreten Typen von `ValueDomains` für die Abbildung der Capabilities bestimmter OWS Typen nicht ausreichen, so kann das Modell für einzelne Anwendungen um spezifische `ValueDomain`-Typen erweitert werden.

4.2.2 Das Feature Modell

Das in dieser Arbeit realisierte Feature Modell bildet die Basis für den Zugang, die Visualisierung und eine mögliche Prozessierung von Feature-Daten. Es basiert auf dem Feature Konzept des OGC (Abschnitt 2.1.2) und dem im *Topic 5: Features* (KOTTMAN 1999) und der *GO-1 Application Objects* Spezifikation (REYNOLDS 2005) definierten Modell für Feature-Daten.

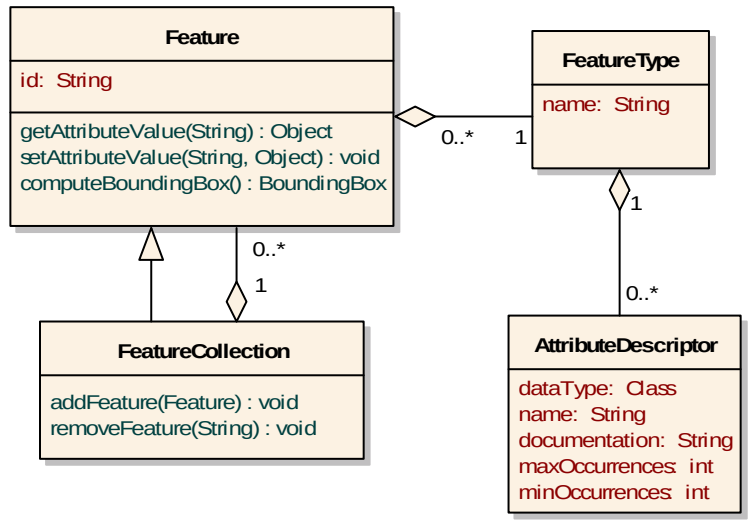

Abbildung 4-6: Vereinfachte Darstellung des Feature Modells in UML-Notation

Ein `Feature` (Abbildung 4-6) ist durch eine eindeutige ID gekennzeichnet. Jedes `Feature`-Objekt besitzt eine Referenz zu einem Feature-Typ (`FeatureType`). Dieser stellt die Beschreibung der Attribute eines `Features` bereit und ist nach der Erzeugung eines `Features` nicht mehr änderbar. Ein `FeatureType` beinhaltet beliebig viele `AttributeDescriptoren`, welche die Metadaten der mit dem `Feature` assoziierten Attribute kapseln. Für Komponenten die das Feature Modell verwenden, bietet der `AttributeDescriptor` eine einheitliche Schnittstelle, um die Eigenschaften der Feature-Attribute abfragen zu können. Neben dem Namen und einer textuellen Beschreibung kann hier der Datentyp des Attributwertes abgefragt werden. Die Attribute `maxOccurences` und `minOc-`

currences geben darüber Auskunft, wie viele Feature-Attribute diesen Typs maximal bzw. minimal mit dem Feature assoziiert werden können.

Der Wert eines Attributs wird über die Methoden getAttributeValue() und setAttributeValue() der Feature-Klasse abgefragt bzw. gesetzt. Beim Setzen des Attributwerts findet eine Überprüfung statt, ob der Wert den vom zugehörigen AttributeDescriptor definierten Eigenschaften entspricht.

Eine FeatureCollection ist ein spezialisiertes Feature, weshalb eine FeatureCollection mit einem eigenen FeatureType assoziiert ist. Die entscheidende Eigenschaft dieser Klasse ist jedoch, dass ihre Instanzen in der Art eines Kompositums[31] andere Feature-Objekte enthalten können.

Die Geometrie eines Features wird wie alle anderen Attribute durch einen AttributeDescriptor beschrieben und über die setAttributeValue()-Methode mit dem Feature assoziiert. Es ist allerdings möglich, mittels der computeBoundingBox()-Methode direkt die räumliche Ausdehnung des Features abzufragen. Im Falle der FeatureCollection-Klasse wird diese Methode überschrieben und gibt dort die Ausdehnung der Vereinigung der Geometrien der in ihr enthaltenen Features zurück.

Die dargestellte Struktur des Feature Modells bietet Anwendungsentwicklern die Möglichkeit darauf aufbauend eigene Applikationsschemata beliebiger Komplexität zu entwerfen. Somit ist eine flexible Erweiterbarkeit des Feature Modells gegeben, wobei die Framework-Komponenten trotzdem auf einer einheitlichen Struktur arbeiten können. In dieser Arbeit wird auf diese Weise das Schema der O&M Spezifikation implementiert (Abschnitt 5.3.3).

4.2.3 Das Context Modell

Die Klassenstruktur des hier entwickelten Context Modells orientiert sich an der in Abschnitt 2.1.5 bereits vorgestellten *Web Map Context Documents* (WMC) Spezifikation. Im OX-Framework wird dieses Modell benutzt, um den aktuellen Status, oder auch *Kontext*, einer Anwendung zu verwalten. Die Klassen dieses Modells spielen eine zentrale Rolle bei der Steuerung des Kontrollflusses der auf dem Framework aufsetzenden Applikationen. Dazu stellen die Methoden dieser Klassen wesentliche Funktionalitäten der Geschäftslogik bereit (Abschnitt 4.2.3.1).

[31] Eine Beschreibung des Entwurfsmusters „Kompositum“ findet sich z.B. bei GAMMA et al. (1995).

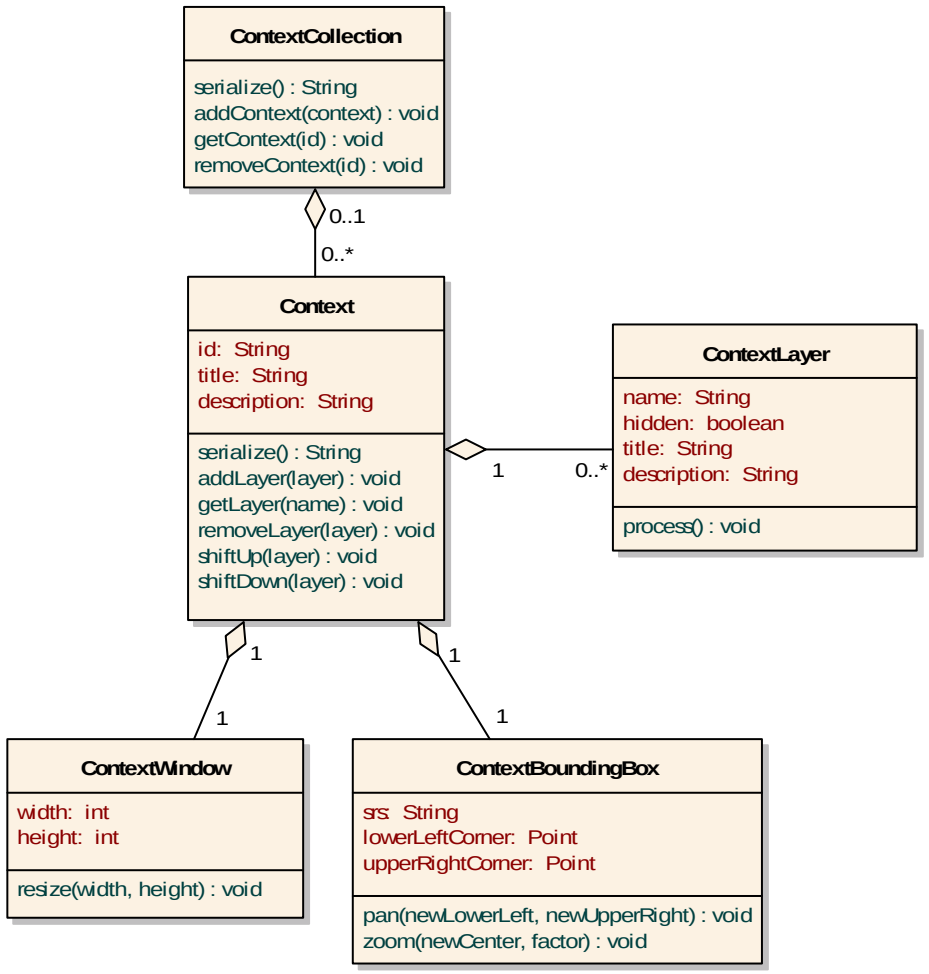

Abbildung 4-7: Vereinfachte Darstellung des Context Modells in UML-Notation

Abbildung 4-7 zeigt das in dieser Arbeit realisierte Context Modell in vereinfachter Darstellung. Ein Objekt der `Context`-Klasse, welche als Attribute eine ID, einen Titel und eine Beschreibung besitzt, repräsentiert den Zustand einer Kartenansicht. Die konkrete Realisierung einer solchen Kartenansicht ist jedoch nicht Teil des OX-Frameworks, sondern wird von der jeweiligen aufsetzenden Applikation implementiert. Um mehrere Sitzungen innerhalb einer Applikation verwalten zu können, kann das `Context`-Objekt Bestandteil einer `ContextCollection` sein, die zu diesem Zweck Methoden zur Abfrage, zum Hinzufügen und zum Entfernen besitzt.

Um die Dimensionen des aktuellen Bildschirmausschnitts zu speichern ist ein `Context` mit einem `ContextWindow` assoziiert. Die Verwaltung des geografischen Raumausschnitts geschieht über eine `ContextBoundingBox`, welche die Koordinaten des unteren linken und des oberen rechten Eckpunkts sowie das räumliche Referenzsystem speichert.

Jeder `Context` verwaltet eine geordnete Liste von assoziierten `ContextLayern`. Ein `ContextLayer` steht für eine Informationsschicht innerhalb eines `Contexts`, die Datenressourcen einer OWS-Instanz beliebigen Typs referenziert. Anders als in der WMC Spezifikation, ist die `ContextLayer`-Klasse hier nicht auf die Repräsentation eines WMS-Layers beschränkt. Ein `ContextLayer` kann beispielsweise einen WMS-Layer, ein Cove-

rage einer WCS-Instanz oder ein Observation Offering einer SOS-Instanz repräsentieren. Das `ContextLayer`-Objekt speichert einen eindeutigen Namen sowie Titel und Beschreibung. Das boolsche Attribut `hidden` zeigt an, ob die grafische Repräsentation des `ContextLayers` im aktuellen `Context` sichtbar oder „versteckt“ ist.

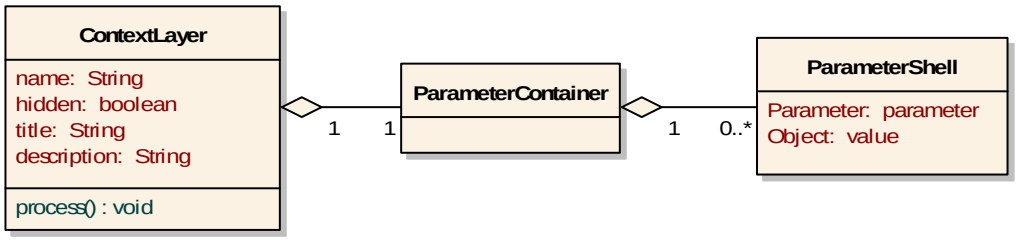

Abbildung 4-8: Assoziation der ContextLayer mit einer Parameter-Konfiguration in UML-Notation

Da ein ContextLayer beliebige Typen von Datenressourcen referenzieren soll, wird ein generischer Mechanismus zur Verwaltung der Parameterwerte benötigt, die zur Ausführung der datenofferierenden Service-Operation verwendet werden sollen. Dazu wird sich an dieser Stelle der bereits im Common Capabilities Modell vorgestellten Klasse `Parameter` (Abschnitt 4.2.1) bedient. Jeder Parameter wird durch ein Objekt der Klasse `Parameter` repräsentiert und mit Hilfe einer Instanz der Klasse `ParameterShell` mit den zur Ausführung der Service-Operation zu verwendenden Werten verknüpft (Abbildung 4-8). Die Gesamtheit der Parameter wird durch ein mit dem `ContextLayer` assoziiertes `ParameterContainer`-Objekt verwaltet.

4.2.3.1 Geschäftsmethoden des Context Modells

`Context`, `ContextWindow` sowie `ContextBoundingBox` bieten Geschäftsmethoden an, die den aktuellen Status der Anwendung verändern können und im Anschluss die notwendigen Prozesse anstoßen. Hierzu gehören z.B. die `shiftUp()`- und `shiftDown()`-Methode der `Context`-Klasse, die die Position eines `ContextLayers` in der Liste verschieben. Die Methode `resize()` der `ContextWindow`-Klasse wird nach einer Veränderung der Bildschirmdimensionen aufgerufen. Sowie die `zoom()`- und `pan()`-Methode der `ContextBoundingBox`-Klasse. Diese Methoden erlauben ein Zoomen und Verschieben des geografischen Raumausschnitts.

Nachdem eine dieser statusverändernden Methoden ausgeführt wurde, muss eine Prozessierung der betroffenen `ContextLayer`[32] stattfinden. Die genauen Abläufe dieser Prozessierung werden in Abschnitt 4.3.4 erläutert.

4.2.3.2 Kontext Serialisierung

Durch die in der WMC Spezifikation festgelegte Kodierung ist eine standardisierte Serialisierung des Zustands einer Client-Sitzung möglich. Diese Serialisierung ist allerdings auf eine Gruppierung von WMS-Layer beschränkt. Im *OpenGIS Web Services Context Document Schema Interoperability Experiment* wurde mit den *OWS Context* Dokumenten ein Nachfolger entwickelt. Die Kodierung dieser Dokumente ist im Moment allerdings

[32] Betroffen sind alle `ContextLayer`, die über das `hidden`-Attribut als sichtbar gekennzeichnet sind.

eingeschränkt auf Web Coverage Services und Web Feature Services (Abschnitt 2.1.5). Referenzierungen von Sensordaten, die über einen SOS bereitgestellt werden, unterstützen die OWS Context Dokumente bislang nicht. Hierzu müsste das Schema der OWS Context Dokumente so erweitert werden, dass die Kodierung von Parameter-Konfigurationen zum Aufbau einer GetObservation-Anfrage möglich ist.

Für das OX-Framework wäre allerdings noch interessanter, die Kodierung der Kontext Dokumente nicht von vornherein auf bestimmte OWS Typen einzuschränken, sondern einen generischen Mechanismus zu entwickeln. Das in dieser Arbeit vorgestellte Konzept der mit einem `ContextLayer` verknüpften `Parameter` und deren `ValueDomains` könnte als grundlegende Idee für die Entwicklung eines solchen Mechanismus verwendet werden. Dazu würde nicht mehr für jeden OWS Typ eine Schema Erweiterung entwickelt werden. Stattdessen würde ein allgemeines Schema die Serialisierung von Parameter-Konfigurationen beliebiger OWS Operationen unterstützen. Die Ausarbeitung eines derartigen Konzepts geht allerdings über den Rahmen dieser Arbeit hinaus und kann daher Thema zukünftiger Arbeiten sein.

4.3 Komponenten für die Service-Anbindung

Das Service-Connector Subsystem stellt die Anknüpfung an die Datenverwaltung – die OGC Web Services – dar und enthält dazu die Implementierungen von Komponenten für die Anbindung der Dienste an das Framework. Diese *Anbindungskomponenten* sind spezialisiert für einen bestimmten OWS Typ.

Die Schnittstellen der Anbindungskomponenten werden bereits im Core Subsystem definiert. Dies entspricht der Vorgehensweise des *Separated Interface*-Musters (FOWLER et al. 2003). Der Core kann auf diese Weise vollkommen ohne Kenntnis über die konkreten Implementierungen auskommen. Somit bleibt er vom Service-Connector Subsystem und von der Datenverwaltungs-Schicht unabhängig. Trotzdem kann er die von den Schnittstellen der Anbindungskomponenten definierte Funktionalität nutzen. Die Separation der Implementierungen von den Schnittstellen ermöglicht deren Rolle als *Variationspunkte* des Frameworks, die in den Anforderungen bereits beschrieben wurde.

Um die einzelnen Aufgaben der Anbindung von OGC Web Services (Abschnitt 3.2.1.1) zu kapseln, werden drei Schnittstellen für Anbindungskomponenten unterschieden: `ServiceAdapter`, `Renderer` und `FeatureStore`. Die Funktionen dieser Schnittstellen, die sämtlich das Interface `ServiceConnector` (siehe auch Abschnitt 4.3.5) erweitern, werden im Folgenden beschrieben. Das Zusammenspiel der Anbindungskomponenten wird in Abschnitt 4.3.4 erläutert.

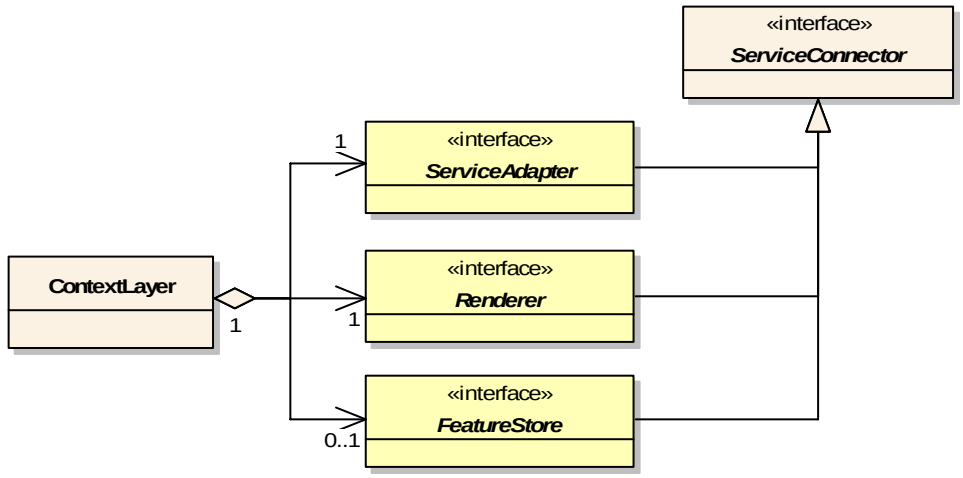

Abbildung 4-9: Die Schnittstellen zur Service-Anbindung (gelb) in UML-Notation (vereinfachte Darstellung)

Wie bereits oben erwähnt stellt ein `ContextLayer` die Referenz zu einer Datenressource eines OWS dar. Die `ContextLayer`-Klasse stellt daher die Anbindung an den Dienst bereit und ist zu diesem Zweck mit den Anbindungskomponenten assoziiert (Abbildung 4-9).

4.3.1 ServiceAdapter

Ein `ServiceAdapter` (Abbildung 4-10) ermöglicht den Zugriff auf einen Dienst. Dazu besitzt er zum einen die Aufgabe, das Common Capabilities Modell für eine spezifizierte Service-Instanz zu initialisieren. Die `initService()`-Methode bietet diese Funktion an. Sie erwartet die URL der Service-Instanz und gibt einen `ServiceDescriptor` (Abschnitt 4.2.1) zurück.

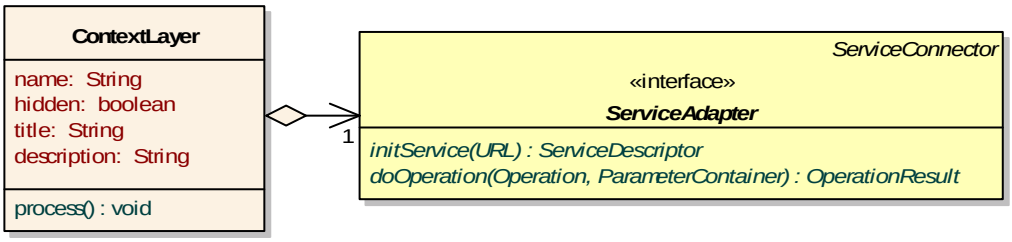

Abbildung 4-10: Die ServiceAdapter-Schnittstelle in UML-Notation

Eine weitere Aufgabe des `ServiceAdapters` ist, die vom Service angebotenen Operationen im Framework einheitlich ausführbar zu machen. Diese Funktion realisiert die `doOperation()`-Methode. Die von der Methode erwarteten Parameter sind zum einen das spezifische `Operation`-Objekt (Abbildung 4-4), welches Informationen zur Ausführung der Service-Operation enthält (z.B. die URL). Zum anderen wird der `doOperation()`-Methode der `ParameterContainer` (Abbildung 4-8) eines `ContextLayers` übergeben. Dieser enthält die Verknüpfungen der zur Ausführung der Service-Operation notwendigen Parameter mit entsprechenden Werten. Die `doOperation()`-Methode gibt nach der Ausführung der Service-Operation ein `OperationResult` (Abbildung 4-14) zurück, welches die vom Dienst empfangenen binären Daten in „roher“ Form kapselt.

4.3.2 FeatureStore

Die Implementierung eines `FeatureStores` für einen OWS Typ ist notwendig aber auch nur sinnvoll, wenn über die Schnittstelle des OWS Typs Feature-Daten bereitstellt werden können. Dann hat der mit dem `ContextLayer` assoziierte `FeatureStore` zur Aufgabe, die vom Dienst empfangenen Daten in das interne Modell zu integrieren.

Zu diesem Zweck steht die `produceFeatures()`-Methode bereit (Abbildung 4-11), welche die Daten auf Instanzen des oben beschriebenen Feature Modells, bzw. davon abgeleitete Applikationsschemata, abbildet und diese zusammengefasst in einer `FeatureCollection` zurückgibt. Dieser Methode wird ein `OperationResult`-Objekt übergeben, welches das Ergebnis der `doOperation()`-Methode eines `ServiceAdapters` ist und die vom Service zurückgegebenen Feature-Daten in „roher" Form enthält. Eine konkrete, für ein bestimmtes Datenformat bzw. einen bestimmten Service-Typ implementierte `FeatureStore`-Komponente kann diese „rohen" Daten interpretieren und ins Feature Modell überführen.

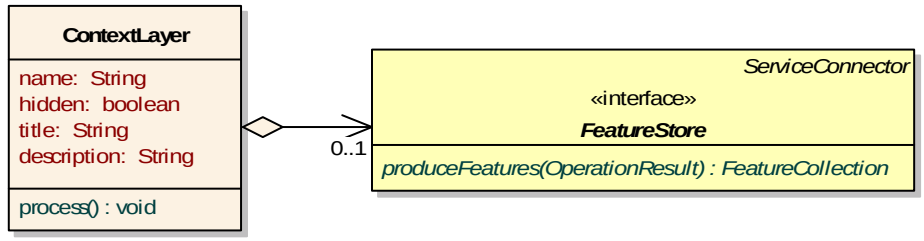

Abbildung 4-11: Die FeatureStore-Schnittstelle in UML-Notation

In Zukunft könnte die `FeatureStore`-Schnittstelle dahingehend erweitert werden, als dass sie auch die Möglichkeiten zum editieren und modifizieren von Features unterstützt. Dies ist beispielsweise im Hinblick auf die Anbindung von transaktionalen Web Feature Services (VRETANOS 2005b) sinnvoll.

4.3.3 Renderer

Ein implementierter `Renderer` kann als Visualisierungskomponente aufgefasst werden, die eine bestimmte Visualisierungsmethode realisiert. Die vom Service empfangenen Daten werden über die `render()`-Methode (Abbildung 4-12) in eine grafisch darstellbare Form konvertiert. Es wird eine `Visualization` produziert, welche den `ContextLayer` grafisch repräsentiert[33].

Um der Anforderung nachzukommen, sowohl statisch als auch animierte Visualisierungen zu unterstützten (Abschnitt 3.1.3.3), besitzt das Interface `Visualization` zwei Realisierungen: `StaticVisualization` und `AnimatedVisualization`. Beide überschreiben die `getRendering()`-Methode, welche das Resultat des Rendervorgangs zurückgibt.

[33] Diese Repräsentation muss, wie unten beschrieben, nicht zwingend als Layer in einer Karte darstellbar sein.

Während dies im ersten Fall, der `StaticVisualization`, ein einfaches Bild-Objekt[34] ist, wird im zweiten Fall, der `AnimatedVisualization`, eine geordnete Liste von Bild-Objekten zurückgegeben, welche die Einzelbilder (engl. „frame“) der Animation darstellen. Über die `getLegend()`-Methode kann eine Legende in Form eines Bildes abgefragt werden, welche die Visualisierung beschreibt.

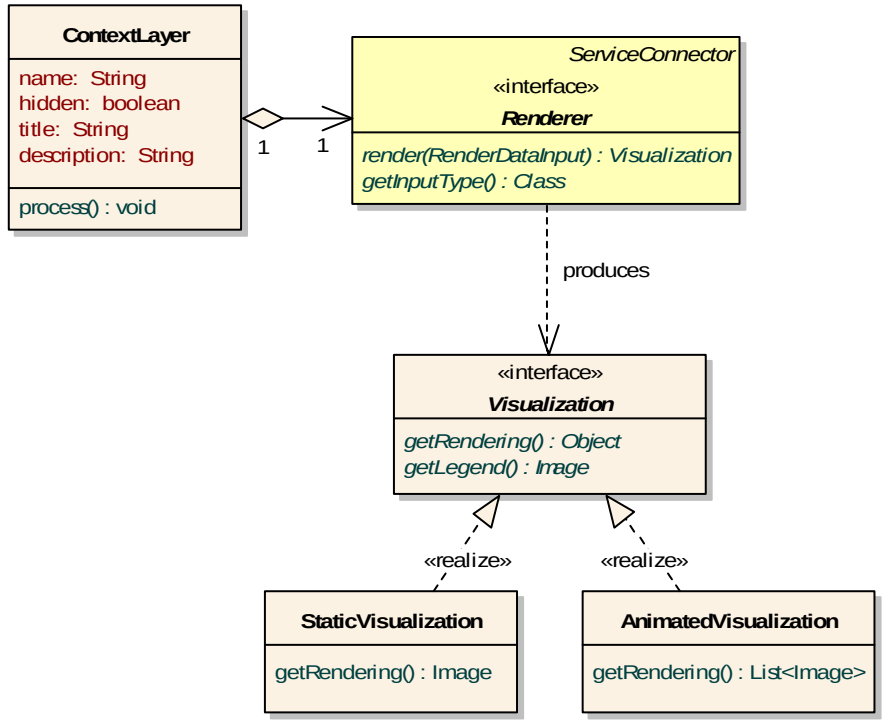

Abbildung 4-12: Die Renderer-Schnittstelle in UML-Notation

Die Anforderung, sowohl eine Visualisierung des Raumbezugs als auch eine separate Visualisierung des Sachbezugs von Geosensor- bzw. allgemeinen Geodaten erzeugen zu können (Abschnitt 3.1.3.1), verlangt die Unterscheidung zweier Arten von `Renderern`. Dies sind zum einen `Renderer`, deren produzierte `Visualization` als Kartenlayer interpretiert werden kann und zum anderen `Renderer`, die eine bestimmte Form von Diagramm oder Grafik visualisieren.

[34] Ein Bild wird in der Implementierung des OX-Frameworks durch die Klasse `java.awt.Image` des *Java Development Kits* (Abschnitt 5.1.1) realisiert. Ein Objekt dieser Klasse kann mit Hilfe der Bildverarbeitungsbibliothek *Java Advanced Imaging* (Abschnitt 5.1.3) in ein digitales Bild unterschiedlichen Formats (z.B. GIF oder JPEG) konvertiert werden.

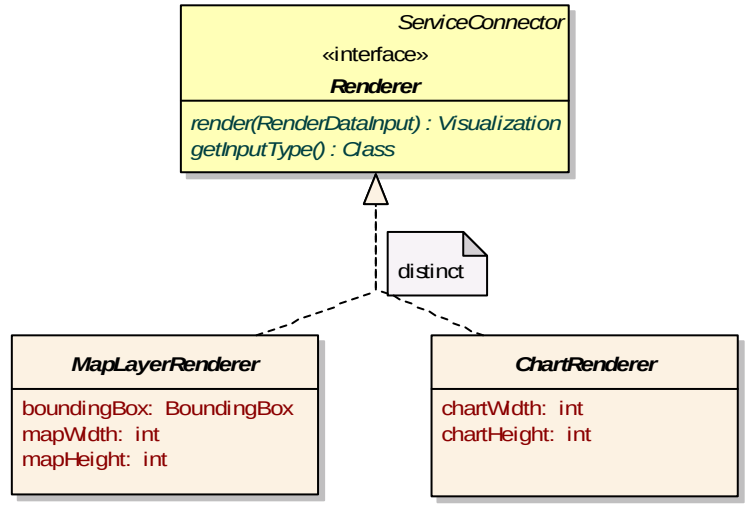

Abbildung 4-13: MapLayerRender und ChartRenderer in UML-Notation

Dazu wird eine `Renderer`-Implementierung von einer der beiden abstrakten Klassen `MapLayerRenderer` oder `ChartRenderer` (Abbildung 4-13) abgeleitet. Beide besitzen Attribute zur Spezifizierung der Dimensionen der von ihnen zu erzeugenden Visualisierung – also Karte oder Diagramm. Der `MapLayerRenderer` benötigt allerdings zudem das Attribut `boundingBox`, um beispielsweise die Geometrien von Geoobjekten in korrekter Position und Form in der Karte abbilden zu können. Diese Attribute werden, bevor ein Rendervorgang angestoßen wird, mit aktuellen Werten initialisiert.

Eine weitere Unterscheidung die getroffen werden muss, betrifft die Art der Daten, die in den Rendervorgang eingegeben werden. Hier wird zwischen Feature- und Raster-Daten unterschieden. Erstere werden z.B. von einem SOS oder einem WFS bereitgestellt. Unter Raster-Daten werden hier Coverage- und binäre Bilddaten verstanden, die z.B. von einem WCS bzw. einem WMS empfangen werden. Die `render()`-Methode der `Renderer`-Schnittstelle erwartet dazu als Eingabeparameter eine Instanz vom Typ `RenderDataInput` (Abbildung 4-14). Das `RenderDataInput`-Interface wird durch die bereits erwähnte Klasse `OperationResult` (Abschnitt 4.3.1) sowie die Klasse `FeatureDataInput` realisiert. Welcher dieser beiden Typen von einer Visualisierungskomponente als Eingabe verlangt wird, kann über die `getInputType()`-Methode abgefragt werden.

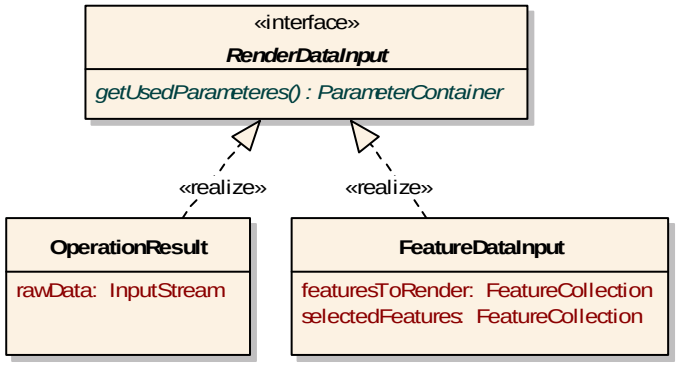

Abbildung 4-14: Eingabetypen für den Rendervorgang in UML-Notation

Ein Objekt vom Typ `OperationResult` enthält die „rohen“ Daten und wird für die Übergabe von Raster-Daten benutzt. Ein `FeatureDataInput` beinhaltet die Feature-Daten, die von einem `FeatureStore` in das frameworkinterne Feature Modell überführt wurden. Das `selectedFeatures`-Attribut kann genutzt werden, um eine Visualisierungskomponente darüber zu informieren, welche `Feature`-Objekte als selektiert betrachtet werden sollen. Dies kann in der Visualisierung z.B. zu einer besonderen Farbgebung führen. Damit eine Visualisierungskomponente auf die Operations-Parameter schließen kann, die zur Ausführung der Service-Operation verwendet wurden, können diese von einem `RenderDataInput`-Objekt abgefragt werden.

4.3.4 Prozessierung der ContextLayer

Wie oben (Abschnitt 4.2.3.1) beschrieben, stellen die Klassen des Context Modells Geschäftsmethoden bereit, die Statusänderungen hervorrufen. Hierzu zählen beispielsweise Änderungen des geografischen Raumausschnitts oder der Dimensionen des `ContextWindows`. Eine auf dem Framework aufsetzende Applikation ruft die Geschäftsmethoden nach Eintritt entsprechender Ereignisse auf, um die Darstellung einer Kartenansicht[35] zu aktualisieren.

Die Kartenansicht soll zu jeder Zeit die korrekte grafische Überlagerung der Visualisierungen sämtlicher sichtbarer `ContextLayer` abbilden. Die am Context Modell ausgeführten Änderungen erfordern daher die Anpassung der Kartenansicht. Hierzu müssen zunächst die vom `ContextLayer` repräsentierten Daten aktualisiert und entsprechend der sich geänderten Parameter-Konfiguration vom Dienst angefragt werden. Im Anschluss müssen die grafischen Repräsentationen der einzelnen `ContextLayer` aktualisiert werden. Dieser Prozess, der das Zusammenspiel der Anbindungskomponenten eines `ContextLayers` erfordert, ist in Abbildung 4-15 dargestellt und wird im Folgenden in vereinfachter Form geschildert.

[35] Der Fall, dass eine Applikation eine Diagrammansicht realisiert, wird unten erläutert.

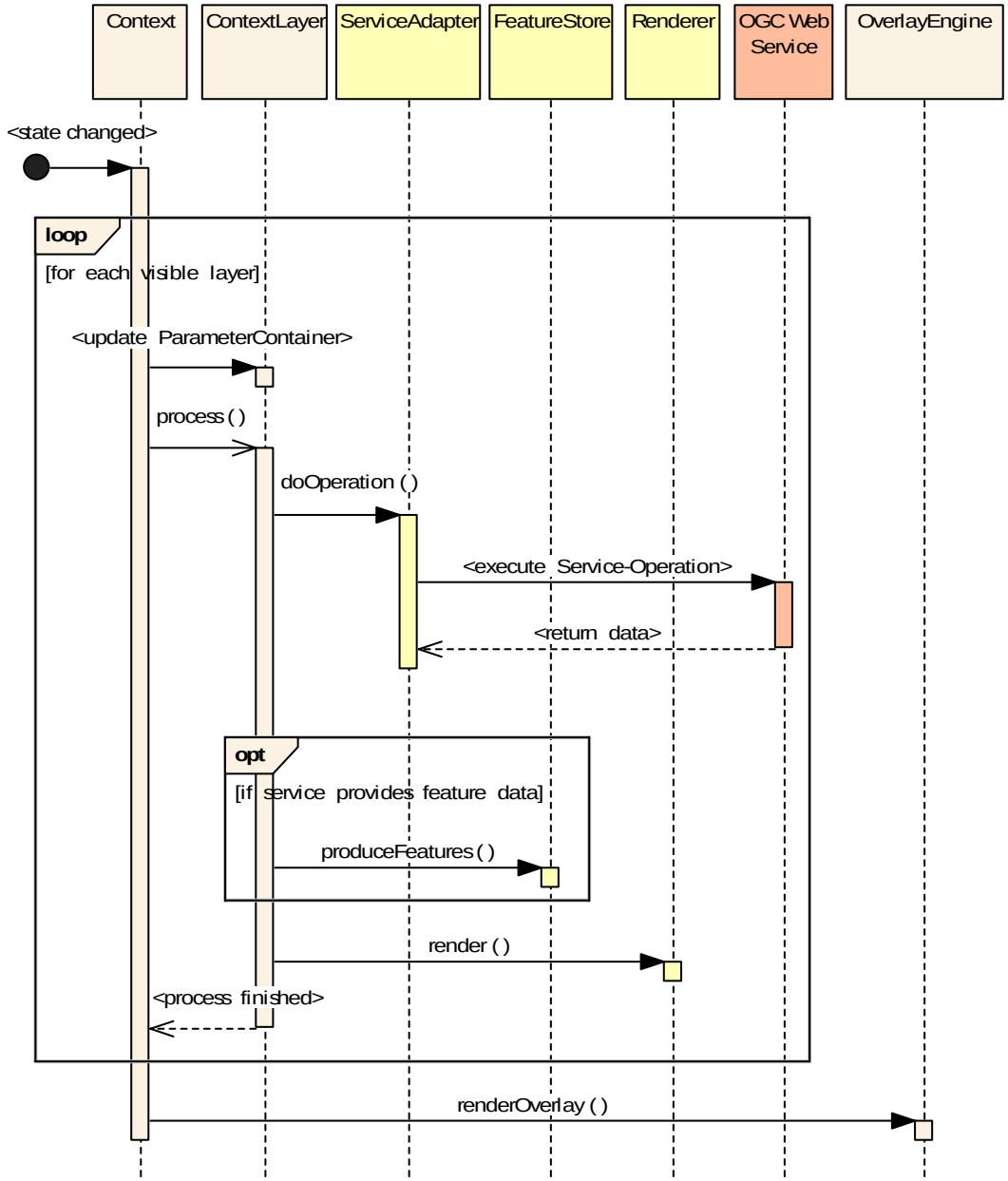

Abbildung 4-15: Zusammenspiel der Anbindungskomponenten in UML-Notation (vereinfachte Darstellung)

Für jeden dieser `ContextLayer` wird zunächst der mit ihm assoziierte `ParameterContainer` (Abbildung 4-8) aktualisiert. Die Werte der dort enthaltenen `Parameter` werden entsprechend der Konfiguration des aktuellen `Contexts` angepasst[36]. Der Anstoß der `process()`-Methode durch das `Context`-Objekt startet daraufhin die Prozessierung eines `ContextLayers`.

[36] Als Beispiel für einen solchen Parameter kann der `BBOX`-Parameter der GetMap-Operation des WMS genannt werden. Dieser wird gemäß den Werten der `ContextBoundingBox` aktualisiert, die mit dem `Context` assoziiert ist.

Im nächsten Schritt werden durch den Aufruf der `doOperation()`-Methode des `ServiceAdapters` die Daten beim Web Service angefordert. Die konkrete `ServiceAdapter`-Komponente baut hierzu intern die entsprechende Anfrage auf und schickt diese zur Service-Instanz, um die Service-Operation auszuführen.

Falls der anzusprechende Dienst Feature-Daten zurückgibt, werden die empfangenen Daten mit Hilfe des `FeatureStores` ins Feature Modell überführt, um sie für die Weiterverarbeitung nutzbar zu machen.

Daran anschließend wird die `render()`-Methode des `Renderers` aufgerufen, um die Daten zu visualisieren. Bei den vom `Renderer` zu bearbeitenden Daten handelt es sich dann entweder um Feature-Daten des frameworkinternen Modells oder um unbehandelte, „rohe“ Raster-Daten, deren Interpretation der `Renderer`-Implementierung obliegt.

Sind die Prozessierungen abgeschlossen und liegen für alle sichtbaren `ContextLayer` Visualisierungen vor, so werden diese an ein Objekt vom Typ `OverlayEngine` übergeben. Die Klasse `OverlayEngine` ist Teil des Core Subsystems. Die `renderOverlay()`-Methode dieser Klasse nimmt eine Liste von `Visualization`-Objekten entgegen und berechnet daraus die grafische Überlagerung. Diese wird wiederum als `Visualization`-Objekt zurückgegeben. Ist eines der entgegengenommenen `Visualization`-Objekte animiert, also vom Typ `AnimatedVisualization`, so gilt dies auch für die berechnete grafische Überlagerung. Auf dem Framework aufbauende Anwendungen empfangen das resultierende `Visualization`-Objekt und präsentieren das darin enthaltene Bild als Kartenansicht, bzw. stellen, für den Fall einer `AnimatedVisualization`, die Liste von Bildern als animierte Kartenansicht dar.

Die hier beschriebene Prozessierung findet in ähnlicher Weise auch für Diagrammansichten einer Applikation statt. Diagrammansichten stellen allerdings anders als Kartenansichten lediglich die Visualisierung *eines* `ContextLayers` dar. D.h. es kann auf eine iterative Prozessierung sämtlicher sichtbarer `ContextLayer` sowie eine abschließende visuelle Überlagerung verzichtet werden. Die Applikation ruft die `process()`-Methode des `ContextLayers` auf und kann die resultierende Visualisierung direkt darstellen.

4.3.4.1 Nebenläufigkeit bei der Prozessierung der ContextLayer

Um die Performance der beschriebenen Prozessierung zu verbessern, wird das Prinzip der Nebenläufigkeit eingesetzt. Dies ist insbesondere von Interesse, falls ein `Context` eine Vielzahl von `ContextLayern` beinhaltet.

Wie durch das offene Pfeilende in Abbildung 4-15 angedeutet, wird die `process()`-Methode eines `ContextLayers` asynchron aufgerufen. Die einzelnen Prozessierungen der verschiedenen `ContextLayer`, starten daher jeweils in einem separaten Ausführungsstrang (engl. „thread“). Dadurch wartet der `Context` nicht auf das Ergebnis der `process()`-Methode, und wird durch das Starten der `ContextLayer`-Prozessierung nicht blockiert. So kann direkt die Prozessierung des nächsten `ContextLayers` angestoßen werden und mehrere Prozessierungen können zueinander parallel ablaufen.

Da eine Prozessierung unterschiedliche Betriebssystemressourcen belastet, nämlich das Netzwerk (zum Empfang der Daten) sowie den Prozessor (z.B. zur Berechnung der Visualisierung), wird durch die beschriebene Parallelisierung auch auf Einprozessormaschinen eine Leistungssteigerung erreicht. Denn während eine `ContextLayer`-

Prozessierung noch mit dem Empfang der Daten beschäftigt ist und die Netzwerkressource beansprucht, kann eine andere Prozessierung, bei der dies bereits abgeschlossen ist, den Prozessor in Anspruch nehmen.

4.3.5 Anbindungskomponenten als Plugins

Die Separation der Schnittstellen von den Implementierungen der Anbindungskomponenten macht ihren Einsatz als *Plugins*[37] möglich. D.h. eine auf dem Framework basierende Anwendung kann eine konkrete Anbindungskomponente erst zur Konfigurations- bzw. Laufzeit des Systems einbinden.

Beispielsweise könnte eine Client-Applikation einen Plugin-Mechanismus anbieten, der es dem Nutzer erlaubt, zur Laufzeit neue Visualisierungskomponenten zum System hinzuzuladen. Hierzu könnte ein Dialog dem Nutzer die Auswahl kompilierter `Renderer` über das Dateisystem oder das Netzwerk erlauben.

Abbildung 4-16: Die ServiceConnector Schnittstelle in UML-Notation

Damit für einen solchen Plugin-Mechanismus die Anbindungskomponenten durch den Nutzer sinnvoll ausgewählt werden können, realisieren sie die von der gemeinsamen Schnittstelle `ServiceConnector` (Abbildung 4-16) geerbten Methoden. Diese Methoden informieren über den von der Anbindungskomponente unterstützten OWS Typ und dessen Versionen.

4.4 Realisierung der Umkehrung des Kontrollflusses

Wie in Abschnitt 3.2.1.2 gefordert, funktioniert der Aufrufmechanismus des OX-Frameworks nach dem Call-Back-Prinzip. Anstatt dass die Applikationen den Kontrollfluss steuern und das Framework im Sinne einer Klassenbibliothek nutzen, indem sie lediglich dessen Funktionen aufrufen, wird die Steuerung zentraler Abläufe an das Framework abgegeben. Dieses Prinzip wird hier mit Hilfe eines *Event-Listener Konzepts* realisiert. Dieses Konzept basiert auf dem Entwurfsmuster *Beobachter*[38]. Es erlaubt die Publikation von Ereignissen die während der Steuerung des Kontrollflusses auftreten. Ebenso können Änderungen des Zustands bestimmter Objekte veröffentlicht werden. Interessierte Objekte müssen dazu den Empfang dieser Ereignisse abonnieren.

Klassen, deren Instanzen Ereignisse publizieren, implementieren die `EventEmitter`-Schnittstelle (Abbildung 4-17). Die `EventListener`-Schnittstelle implementieren die Klassen, deren Objekte an Ereignissen interessiert sind. Sie können sich über entspre-

[37] Eine Beschreibung des Entwurfsmusters „Plugin“ findet sich zum Beispiel bei FOWLER et al. (2003).

[38] Das *Beobachter*-Muster (engl.: „observer pattern“ oder „publish/subscribe pattern“) wird beispielsweise von GAMMA et al. (1995) beschrieben.

chende Methoden des `EventEmitters` für den Ereignis-Empfang anmelden bzw. wieder abmelden. Zum Versenden von Ereignissen wird die `fireEvent()`-Methode des `EventEmitters` ausgelöst, die zum Aufruf der `eventCaught()`-Methode sämtlicher registrierter `EventListener` führt.

Primär sind die ereignisaussendenden Objekte innerhalb des Cores angesiedelt, da hier der Hauptkontrollfluss gesteuert wird, während die Objekte der aufsetzenden Anwendungen typischerweise die Rolle der `EventListener` einnehmen. Ein entscheidender Nutzeffekt des beschriebenen Konzepts ist, dass die `EventEmitter` die konkreten Klassen der `EventListener` nicht kennen müssen, um mit ihnen indirekt kommunizieren zu können. Somit kann das Core Subsystem vollständig unabhängig von der Präsentation des Systems, also der aufsetzenden Applikation, bleiben. Die Klassen des Cores müssen für das Hinzufügen neuer `EventListener` nicht modifiziert werden, wodurch der Einsatz des Frameworks in verschiedenen Applikationen erleichtert wird.

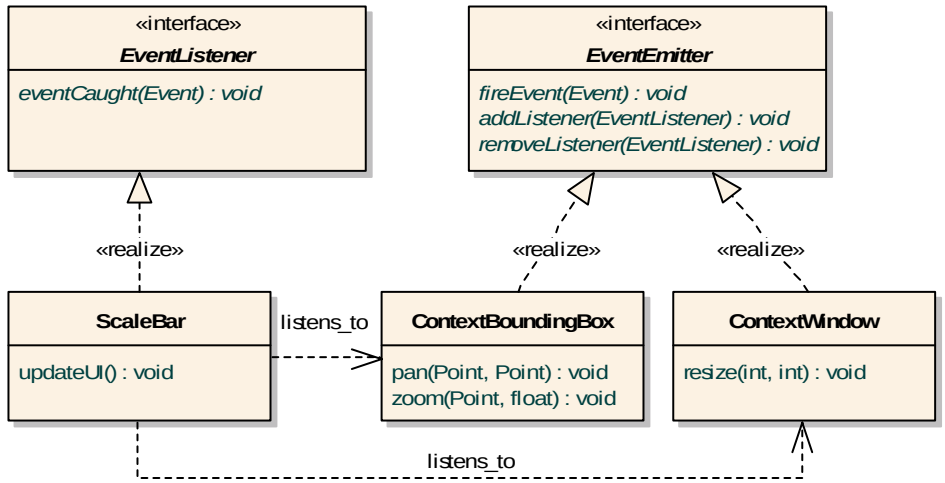

Abbildung 4-17: Das Event-Listener Konzept in UML-Notation

Abbildung 4-17 zeigt mit den Klassen `ContextBoundingBox` und `ContextWindow` (Abschnitt 4.2.3) Beispiele für `EventEmitter`. Diese Klassen sind Teil des Cores. Weiterhin ist die Klasse `ScaleBar` abgebildet, die hier als Beispiel für einen `EventListener` steht. Die `ScaleBar` ist als GUI-Komponente Teil einer aufsetzenden Client-Applikation. Sie hat die Aufgabe dem Nutzer zu jeder Zeit den Maßstab der Kartenansicht zu präsentieren. Dazu wird das Objekt dieser Klasse als Beobachter bei der `ContextBoundingBox` und dem `ContextWindow` registriert. Finden nun durch Nutzerinteraktionen mit der Kartenansicht Statusänderungen an den beiden Objekten statt, so werden diese als Ereignisse publiziert. Die `ScaleBar` empfängt die Ereignisse und kann sich automatisch entsprechend des neuen Zustands aktualisieren.

4.5 Auf dem Framework aufsetzende Applikationen

Wie in Abschnitt 3.2.2 gefordert, ist es möglich, das OX-Framework als Grundlage für die Entwicklung verschiedener Applikationen zu verwenden. Dabei kommt es zur Wiederverwendung der im Core enthaltenen Geschäftslogik sowie der einmal implementierten Anbindungskomponenten. Diese generische Konzeption des Frameworks wird in

Abschnitt 5.2 anhand der exemplarischen Implementierung einer Rich-Client- sowie einer Server-Applikation gezeigt. Diese Applikationen bilden die Präsentations-Schicht des jeweiligen Gesamtsystems.

Die Server-Applikation wird als Kartendienst konform zur Spezifikation des Web Map Service (Abschnitt 2.1.4) implementiert. Das Prinzip dieses WMS-Frontends zeigt Abbildung 4-18. Für die Visualisierung von Sensordaten aggregiert das WMS-Frontend einen oder gegebenenfalls mehrere SOS-Instanzen. Dem Nutzer wird nicht bewusst, dass der WMS aus einer Aggregation ein oder mehrerer Dienste besteht. Dieses Vorgehen entspricht einer opaken Verkettung von Diensten (PERCIVALL 2002).

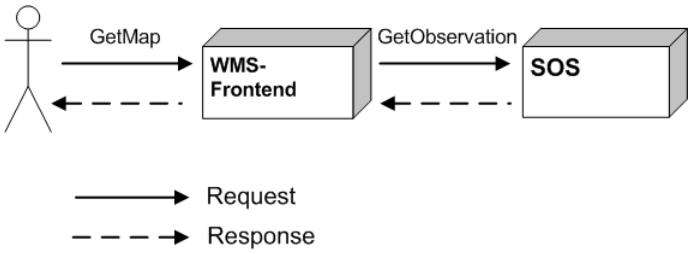

Abbildung 4-18: Prinzip des WMS-Frontends

Zur Beantwortung einer vom Nutzer spezifizierten GetMap-Anfrage, nutzt die WMS-Applikation zunächst einen für den SOS implementierten `ServiceAdapter`, der die GetObservation-Operation ausführen kann und die Sensordaten abfragt. Diese werden mit Hilfe eines `FeatureStores` in das Feature Modell überführt. Anschließend werden spezifische `Renderer` verwendet, um dem Nutzer die Visualisierung eines Kartenlayers zurückzugeben. Die prototypische Implementierung des WMS-Frontends wird in Abschnitt 5.2.2 erläutert. Nun folgend werden Konzepte für aufsetzende Applikationen skizziert, die in aufbauenden Arbeiten als Weiterentwicklung des WMS-Frontends umgesetzt werden können.

4.5.1 Aggregation unterschiedlicher OWS Typen

Da Anbindungskomponenten für unterschiedliche OWS Typen implementiert werden können, muss das beschriebene Konzept eines auf dem OX-Framework basierenden WMS-Frontends künftig nicht auf die Aggregation von Sensor Observation Services beschränkt sein. Stehen entsprechende Anbindungskomponenten zur Verfügung, so können z.B. auch OGC Web Services vom Typ WFS oder WCS aggregiert werden, um Visualisierungen von Feature- bzw. Coverage-Daten anzubieten. Das WMS-Frontend könnte dann konform zur SLD-WMS Spezifikation (Abschnitt 2.1.4.1) weiterentwickelt werden, so dass der Nutzer den Darstellungsstil der Feature- bzw. Coverage-Daten über die Definition von Styled Layer Descriptoren selbst festlegen kann.

4.5.2 Das Konzept des Workflow Service

Die für das WMS-Frontend beschriebene einfache, lineare Verkettung zweier Dienste könnte in aufbauenden Arbeiten zu komplexeren Verkettungen von Diensten ausgebaut werden. Wie in Abbildung 4-19 dargestellt, könnte ein auf dem OX-Framework aufgebauter Workflow Service `ServiceAdapter`-Komponenten nutzen, um Daten verschiede-

ner OGC Dienste abzufragen und diese mit Hilfe eines Processing Service miteinander zu kombinieren[39]. Im Anschluss wird das Ergebnis dieser Prozessierung von einer speziellen `Renderer`-Komponente in eine grafische Repräsentation überführt und an den Nutzer zurückgegeben.

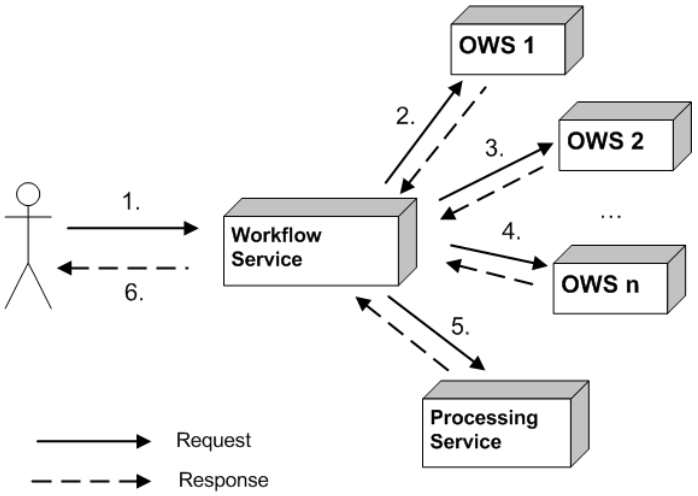

Abbildung 4-19: Beispielhafte Anwendung eines Workflow Service (nach SLIWINSKI et al. 2005)

Ein beispielhaftes Szenario für die Nutzung eines solchen Workflow Service im Kontext der Visualisierung von Geosensordaten könnte folgendermaßen aussehen: Der Workflow Service ruft punkthafte Messungen eines raumzeitlichen Phänomens von verschiedenen SOS-Instanzen ab, um sie im nächsten Schritt einem Processing Service zu übergeben. Dieser interpoliert auf Grundlage der Daten eine Oberflächenrepräsentation des Phänomens, die anschließend vom Workflow Service visualisiert und zurückgegeben wird.

4.5.3 Das Konzept des O&M Portrayal Service

Eine andere mögliche Weiterentwicklung des WMS-Frontends könnte die Art der Beziehung zum aggregierten Dienst betreffen. Aktuell ist hier eine *closely-coupled* Beziehung vorgesehen. D.h. es besteht eine Vorkonfiguration darüber, welche SOS-Instanzen vom WMS-Frontend angesprochen und welche Visualisierungmethoden eingesetzt werden. Flexibler wäre eine Erweiterung zu einem Kartendienst, welcher eine *loosely-coupled* Beziehung eingeht, so dass Geosensordaten beliebiger SOS-Instanzen visualisiert werden können. In Anlehnung an die Konzepte von Feature Portrayal Service und Coverage Portrayal Service (Abschnitt 2.1.4.1) könnte von einem *O&M Portrayal Service* gesprochen werden.

FPS und CPS nutzen SLDs, um die visuelle Darstellung von Feature- respektive Coverage-Daten festzulegen. Für den O&M Portrayal Service müsste diese Sprache erweitert

[39] Workflow und Processing Services sind Kategorien der *Geographic Services Taxonomy*, welche im *Topic 12: OpenGIS Service Architecture* (PERCIVALL 2002) der Abstract Specification beschrieben wird. *Workflow Services* führen bestimmte Aufgaben oder Aktivitäten durch, deren Lösung mehrere aufeinanderfolgende Arbeitsschritte und die Einbindung verschiedener Ressourcen erfordert. *Processing Services* operieren auf raumzeitlichen Daten. Sie transformieren oder kombinieren eingegebene Geodaten und erzeugen als Ergebnis neue Daten.

werden, um Visualisierungsmethoden auf O&M-enkodierte Sensordaten anwenden zu können. Ebenso müsste der Dienst für die angebotenen Visualisierungsmethoden Beschreibungen bereitstellen. Diese müssten z.B. darüber Auskunft geben können, auf welche Eingabedaten die Visualisierungsmethoden angewandt werden können und welche Semantik sie besitzen, damit die Interpretierbarkeit für den Nutzer gegeben ist.

Ein Konzept zur Lösung dieser Anforderungen könnte die SensorML Spezifikation (Abschnitt 2.2.2) einbeziehen, indem die Visualisierung als Post-Prozessierung der Geosensordaten verstanden wird und die vom O&M Portrayal Service angebotenen Visualisierungsmethoden mit Hilfe des dort spezifizierten Prozess-Modells beschrieben werden. Neben der Struktur und den Datentypen der Eingaben eines Prozesses, können zahlreiche Metadaten spezifiziert werden. Auch die Beschreibung des Algorithmus einer Visualisierungsmethode ist möglich. Es können also Informationen zur sinnvollen Nutzung und zur Ableitung der Semantik der Visualisierungsmethoden bereitgestellt werden.

5 Implementierung

Mit der Entwicklung der Architektur des OX-Frameworks wurden die Grundlagen für die Implementierung von verschiedenartigen, auf dem Framework aufsetzenden Applikationen gelegt sowie die Basis für die Implementierung von Anbindungskomponenten für unterschiedliche OWS Typen bereitet.

In Abschnitt 5.1 werden zunächst die zur Implementierung benutzte Programmiersprache sowie die verwendeten Bibliotheken vorgestellt. Abschnitt 5.2 stellt die prototypischen Realisierungen der Client- und Server-Applikation vor, die im Sinne eines Proof-of-Concept zeigen, dass unterschiedliche Arten von Applikationen auf dem Framework aufgesetzt und zur Visualisierung von Geosensordaten eingesetzt werden können. In Abschnitt 5.3 werden die entwickelten Anbindungskomponenten und insbesondere die Visualisierungskomponenten für Geosensordaten beschrieben.

5.1 Verwendete Techniken

5.1.1 Java

Zur Implementierung der Framework-Architektur und den darauf aufbauenden prototypischen Anwendungen sowie den Anbindungskomponenten wurde die objektorientierte Programmiersprache Java verwendet.

Ein wichtiges Kennzeichen dieser seit 1995 von der Firma *Sun Microsystems* entwickelten Sprache ist seine Plattformunabhängigkeit. Das bedeutet, dass mit Java erstellte Applikationen auf jeder Plattform ausführbar sind, für die eine *Java Virtual Machine* (JVM) existiert. Um dies zu erreichen wird Java-Quellcode nicht direkt in plattformspezifischen Maschinencode übersetzt, sondern in eine Zwischenrepräsentation, den Bytecode. Dieser kann ohne Änderungen von den unterschiedlichen JVMs ausgeführt werden[40].

In dieser Arbeit wurde als Entwicklungsumgebung das *Java 2 Standard Edition Development Kit 5.0* (JDK) (SUN MICROSYSTEMS 2007) genutzt[41]. Das JDK stellt eine Reihe von Werkzeugen und Standard-Bibliotheken zur Programmentwicklung bereit. Hierzu gehört die *Swing*-Bibliothek (LOY et al. 2002). Sie kann zur Entwicklung grafischer Benutzerschnittstellen eingesetzt werden und wird daher von der Client-Applikation verwendet.

[40] Dieses Prinzip wird von KRAMER et al. (1996) als „*Write Once, Run Anywhere*“ bezeichnet.

[41] Zur Ausführung der prototypischen Applikationen wird daher eine JVM der Version 1.5.0 benötigt.

5.1.2 JFreeChart

JFreeChart ist eine in Java geschriebene Open Source Bibliothek der Firma *Object Refinery*. Sie ermöglicht das Erzeugen verschiedener Diagrammtypen, wie z.B. Balken-, Linien- oder Streudiagrammen. Die erzeugten Diagramme können auf einfache Weise in Java Bild-Objekte gezeichnet werden (OBJECT REFINERY 2006). Diese Bibliothek wird daher von verschiedenen der implementierten Visualisierungskomponenten verwendet, um den Sachbezug der Sensordaten abzubilden.

5.1.3 Java Advanced Imaging

Java Advanced Imaging (JAI) ist eine von der Firma *Sun Microsystems* entwickelte Grafikbibliothek, deren Klassen umfangreiche Funktionalitäten zur Verarbeitung digitaler Bilder bereitstellt (SUN MICROSYSTEMS 1999). In den Implementierungen dieser Arbeit wird die JAI-Bibliothek z.B. verwendet, um die grafische Überlagerung erzeugter Visualisierungen zu berechnen (Abschnitt 4.3.4).

5.1.4 Servlets

Als Servlets[42] werden Java-Klassen bezeichnet, deren Instanzen innerhalb eines Servlet Containers laufen und von Clients ausgesandte HTTP-Anfragen entgegennehmen und beantworten können. Die Antwort (z.B. eine HTML-Seite) kann dabei zur Laufzeit erstellt werden. Servlets erweitern die abstrakte Klasse `javax.servlet.HttpServlet`. Das WMS-Frontend nutzt diese Servlet-Technologie. Als Servlet Container wird die Tomcat 5.5 Servlet Engine des *Apache Jakarta Project* (APACHE SOFTWARE FOUNDATION 2006) verwendet.

5.1.5 XMLBeans

XMLBeans ist ein Softwarepaket, das innerhalb des *Apache XML Project* (APACHE SOFTWARE FOUNDATION 2007) entwickelt wird. Es besteht aus Bibliotheken und Werkzeugen, die eine einfache Verarbeitung von XML-Daten in Java ermöglichen.

Mit Hilfe von XMLBeans kann ein XML-Schema kompiliert werden, um daraus Java Klassen zu generieren, welche die Schematypen repräsentieren. Diese Klassen erlauben ein *Marshalling* und *Unmarshalling* von XML-Daten. Beim *Unmarshalling* wird ein zum Schema konformes XML-Dokument zur Laufzeit in Java Objekte der erzeugten Klassen überführt. Die im Dokument bereitgestellten Daten können dann durch einfachen Aufruf von Java Methoden abgefragt werden. Als *Marshalling* wird der umgekehrte Weg bezeichnet, bei dem aus einem Objekt-Baum wieder XML-Daten geschrieben werden.

Die `ServiceAdapter`- sowie `FeatureStore`-Komponenten setzen XMLBeans ein, um empfangene XML-kodierte Daten zunächst in Objekte generierter Klassen zu überführen, um daraus im Anschluss Objekte der internen Modelle zu erzeugen. Dieser weitere

[42] Eine ausführliche Beschreibung der Servlet-Technologie findet sich beispielsweise bei HUNTER & CRAWFORD 2001.

Schritt im Unmarshalling-Prozess hat gegenüber einer direkten Nutzung der von XMLBeans erzeugten Klassen den Vorteil, dass Komponenten, welche die empfangenen Daten weiterverarbeiten, unabhängiger sind von den XML-Schemata. Auftretende Änderungen an den Schemata der Spezifikationen ziehen Anpassungen in den generierten Klassen nach sich. Die internen Modelle können bei geringfügigen Änderungen hingegen unberührt bleiben und somit auch die auf diesen Modellen arbeitenden Komponenten.

5.2 Implementierung der aufsetzenden Applikationen

5.2.1 Die Client-Applikation

Die im Rahmen dieser Arbeit implementierte Client-Applikation ist eine auf der Swing-Bibliothek basierende grafische Benutzerschnittstelle und kann als Rich-Client (Abschnitt 3.2.2) angesehen werden. Sie bietet dem Nutzer Möglichkeiten zur Anzeige und zur Interaktion mit den erzeugten Visualisierungen. Die Geschäftslogik dieser Anwendung wird in großen Teilen durch das OX-Framework realisiert.

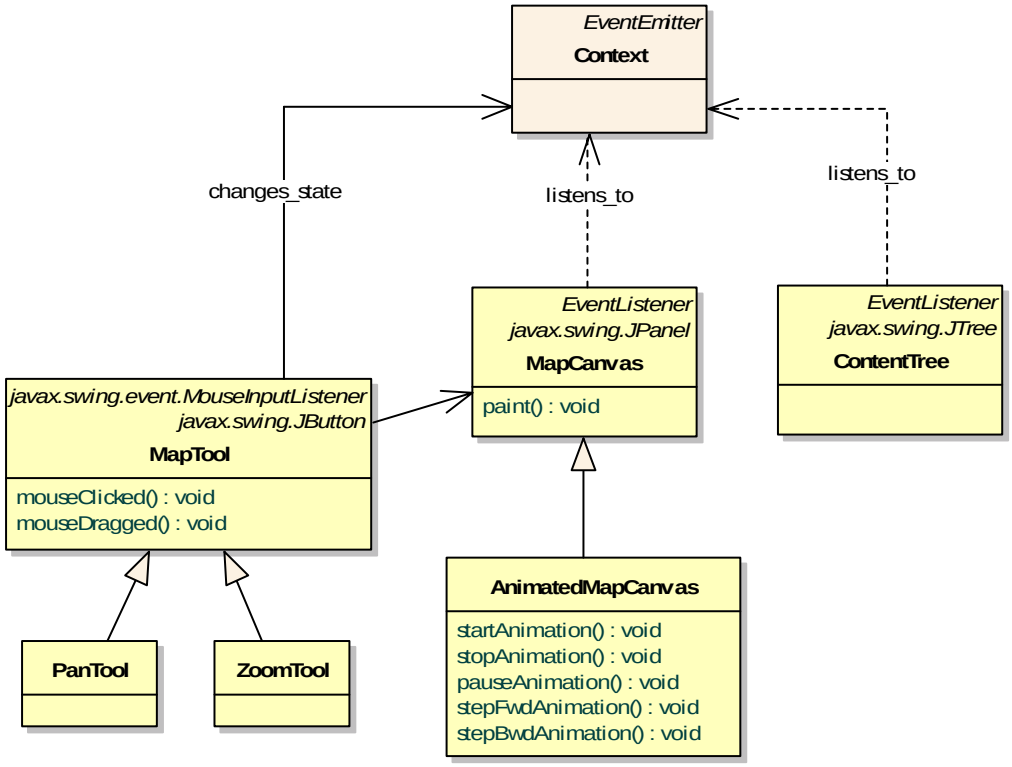

Abbildung 5-1: Zentrale Klassen der Client-Applikation in UML-Notation (vereinfachte Darstellung)

In Abbildung 5-1 sind die zentralen Klassen dieser Anwendung (gelb) vereinfacht dargestellt. Bei diesen Klassen handelt es sich jeweils um GUI-Komponenten. Abbildung 5-2 zeigt die grafischen Repräsentationen dieser Klassen kombiniert in einer grafischen Benutzeroberfläche.

Die Kartenansicht wird durch die Klasse `MapCanvas` realisiert. Sie gestattet die Präsentation von Visualisierungen, die von speziellen `MapLayerRenderern` erzeugt werden. Eine Spezialisierung dieser Klasse ist das `AnimatedMapCanvas`. Während das `MapCanvas` lediglich `StaticVisualizations` präsentieren kann, ermöglicht das `AnimatedMapCanvas` die Darstellung und das Abspielen von `AnimatedVisualizations` (Abschnitt 4.3.3). Dazu erweitert die Klasse `AnimatedMapCanvas` das `MapCanvas` um Steuerelemente und dazugehörige Methoden, so dass diese Animationen z.B. gestartet, gestoppt oder pausiert werden können.

Die Klasse `ContentTree` zeigt die Hierarchie der im `Context` befindlichen `ContextLayer` in Form einer Baumstruktur. Damit diese Ansicht stets aktuell ist, ist der `ContentTree` als `EventListener` (Abschnitt 4.4) beim `Context`-Objekt der Anwendung registriert. Er wird somit über das Hinzufügen, Entfernen und Verschieben von `ContextLayern` informiert und passt seine Darstellung automatisch an.

Um Interaktionen mit der Kartenansicht zu ermöglichen, dienen verschiedene Spezialisierungen der Klasse `MapTool`. Ein `MapTool` repräsentiert eine Schaltfläche der GUI. Durch Anklicken des Nutzers wird das `MapTool` aktiviert. Nun werden durch Mauseingaben in der Kartenansicht die Funktionalitäten des `MapTools` ausgeführt. Hierzu implementiert jedes `MapTool` die von der Swing-Bibliothek bereitgestellte `MouseInputListener` Schnittstelle.

Als Beispiele spezieller `MapTools` seien hier das `PanTool` oder das `ZoomTool` genannt. Diese Werkzeuge rufen intern die entsprechenden Geschäftsmethoden (Abschnitt 4.2.3.1) des `Contexts` auf, welche eine anschließende Prozessierung der `ContextLayer` auslösen (Abschnitt 4.3.4). Das `MapCanvas`, welches als `EventListener` auf Ereignisse des `Contexts` horcht, wird benachrichtigt sobald die visuelle Überlagerung der `ContextLayer`-Visualisierungen berechnet ist und stellt diese daraufhin dar.

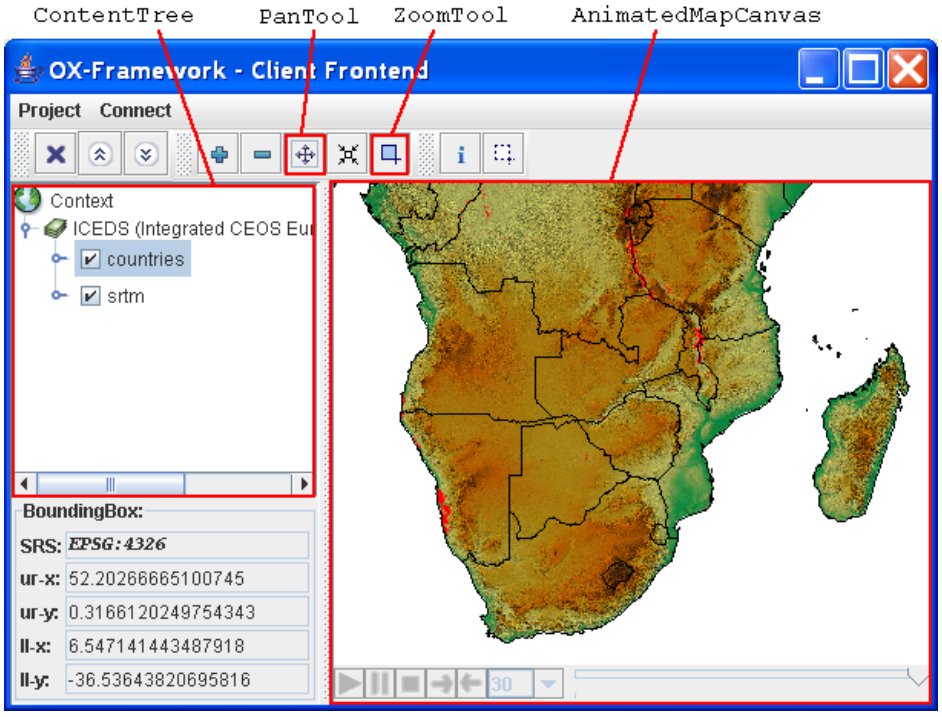

Abbildung 5-2: Die grafische Benutzeroberfläche der Client-Applikation

5.2.1.1 Einladen neuer ContextLayer

Über das Menü *Connect* (Abbildung 5-2) können Dialoge aufgerufen werden, über die der Nutzer die verschiedenen OWS Typen ansprechen kann, um neue `ContextLayer` einzubinden. Möchte der Nutzer eine SOS-Instanz ansprechen, um dessen Sensordaten zu visualisieren, so bekommt er den in Abbildung 5-3 dargestellten Dialog angezeigt[43]. Hier kann der Nutzer den `Renderer` wählen, der zur Visualisierung des neuen `ContextLayers` eingesetzt werden soll. Über die Schaltfläche *Plugin new Renderer* wird der in Abschnitt 4.3.5 beschriebenen Plugin-Mechanismus realisiert. Er befähigt den Nutzer, weitere Visualisierungskomponenten vom Dateisystem hinzuzuladen.

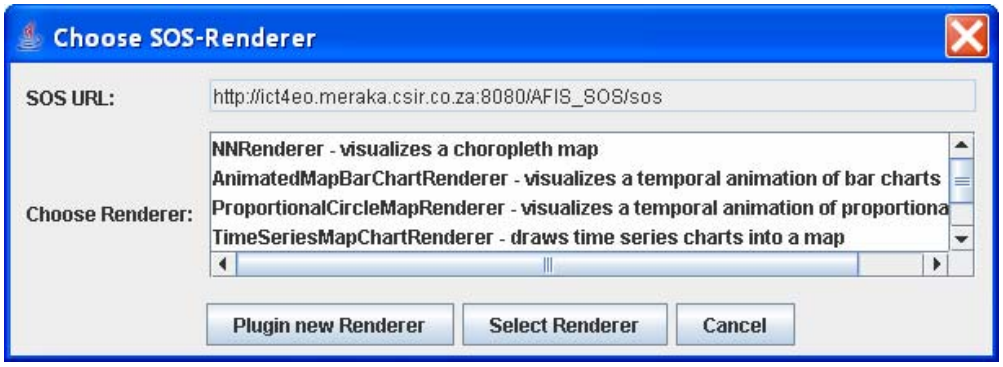

Abbildung 5-3: Auswahl der Renderer-Komponente

Hat der Nutzer einen `Renderer` ausgewählt, so wird ein für den SOS implementierter `ServiceAdapter` (Abschnitt 5.3.1) verwendet, um eine Verbindung mit der SOS-Instanz aufzubauen und das Common Capabilities Modell zu initialisieren. Im darauffolgenden Dialog (Abbildung 5-4) kann der Nutzer die zur Ausführung der GetObservation-Operation notwendigen Parameter spezifizieren[44]. Für jeden Parameter dieser Operation wird hier ein Auswahl- oder Eingabeelement abgebildet. Um diese Elemente mit zulässigen Werten zu initialisieren, wird auf die `ValueDomains` der einzelnen `Parameter`-Objekte des instanziierten Common Capabilities Modells zurückgegriffen (Abschnitt 4.2.1.1).

[43] Ähnlich zu den Dialogen zum Ansprechen einer SOS-Instanz, beinhaltet die Client-Applikation Dialoge für den Zugriff auf WMS-Instanzen. Da der Fokus dieser Arbeit jedoch auf der Visualisierung von Geosensordaten liegt, werden diese Dialoge nicht vorgestellt.

[44] Die Bedeutungen der im Dialog abgebildeten Parameter werden in Abschnitt 2.2.4.1 bzw. in der Spezifikation des SOS (NA & PRIEST 2006) beschrieben.

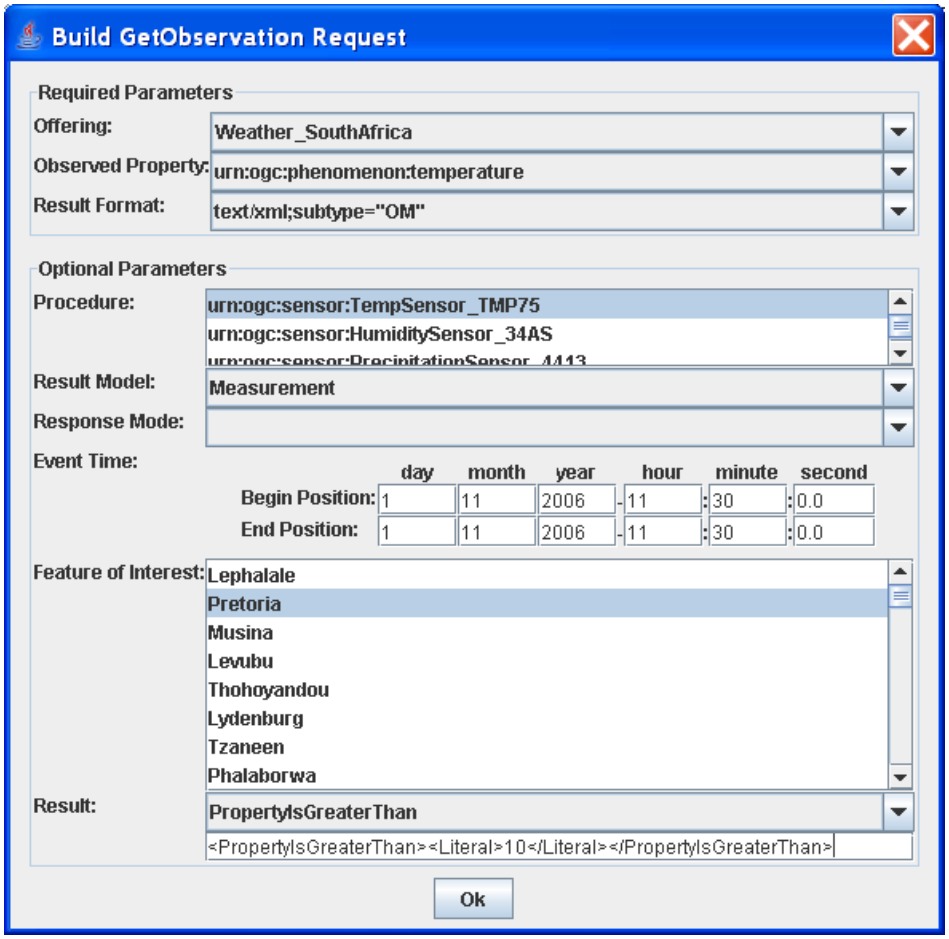

Abbildung 5-4: Dialog zum Aufbau einer GetObservation-Anfrage

Nach Drücken der Schaltfläche *Ok* wird dem aktuellen `Context` ein neues `ContextLayer`-Objekt hinzugefügt. Der `ParameterContainer` des `ContextLayers` wird gemäß den spezifizierten Parameterwerten mit `ParameterShell`-Objekten assoziiert (Abschnitt 4.2.3). Anschließend wird vom Framework die Prozessierung des `ContextLayers` angestoßen. Die im `ParameterContainer` gespeicherte Konfiguration von Parameterwerten wird dabei benutzt, um die GetObservation-Anfrage aufzubauen. Die im Dialog (Abbildung 5-4) spezifizierte Konfiguration würde den in Listing 2-2 gezeigten Request erzeugen.

5.2.1.2 Die Diagrammansicht

Die Client-Applikation bietet ebenfalls die Möglichkeit, die Visualisierung eines Diagramms zu präsentieren. Der Nutzer wählt dazu für einen `ContextLayer` einen speziellen `ChartRenderer` (Abschnitt 4.3.3) aus. Nach der Spezifikation der Operations-Parameter (Abbildung 5-4) wird die Prozessierung des `ContextLayers` durchgeführt. Die resultierende Visualisierung wird in einem separaten Fenster (Abbildung 5-5) angezeigt.

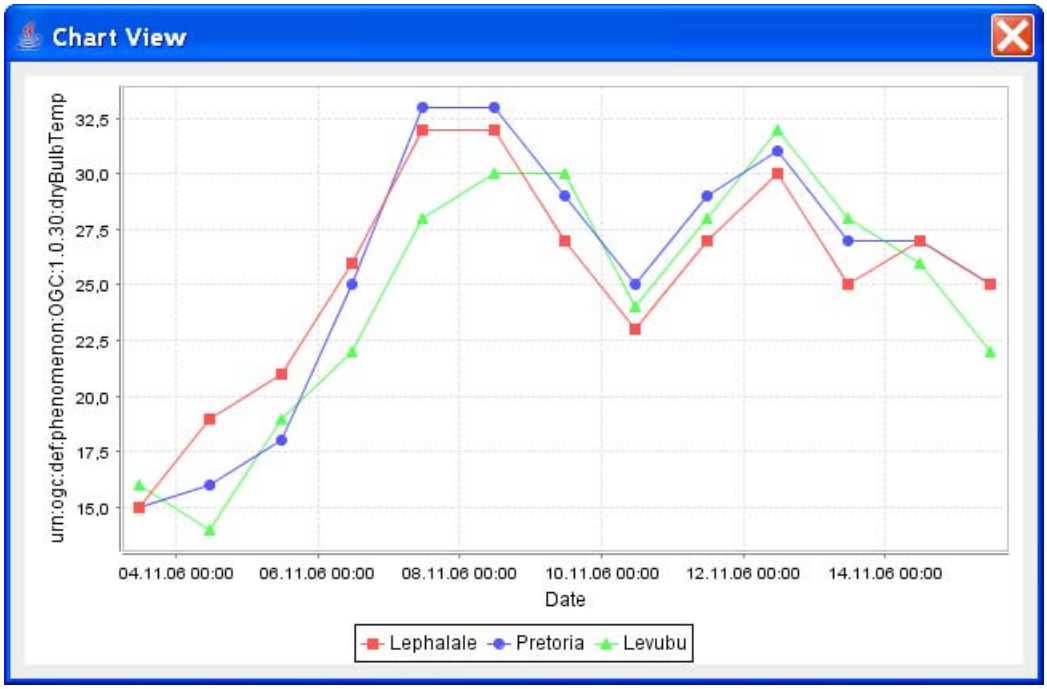

Abbildung 5-5: Die Diagrammansicht der Client-Applikation

Abbildung 5-5 zeigt eine mit Hilfe des `TimeSeriesChartRenderers` (Abschnitt 5.3.6) erzeugte Visualisierung von Geosensordaten, die von einem SOS abgefragt wurden. Der `Renderer` zeichnet für jedes in der GetObservation-Anfrage spezifizierte `featureOfInterest` einen Graphen in ein Zeitreihendiagramm.

5.2.2 Das WMS-Frontend

Wie in Abschnitt 4.5 beschrieben, aggregiert das auf dem OX-Framework aufgebaute WMS-Frontend einen oder mehrere SOS-Instanzen, um somit die Visualisierungen der Sensordaten auch mit WMS-Clients darstellen zu können. Die von diesem Kartendienst angebotenen Layer entsprechen Visualisierungen bestimmter vom SOS angebotener Sensordaten. Diese Layer können über den GetMap Request des WMS abgerufen werden.

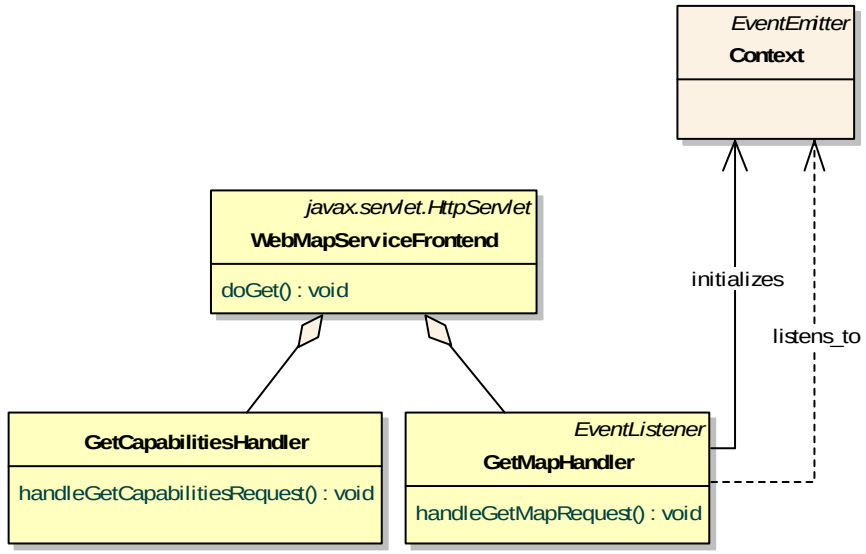

Abbildung 5-6: Zentrale Klassen des WMS-Frontends in UML-Notation (vereinfachte Darstellung)

Die Abbildung 5-6 stellt in vereinfachter Form die wichtigsten Klassen des WMS-Frontends (gelb) in UML-Notation dar. Die Klasse `WebMapServiceFrontend` ist ein spezialisiertes `HttpServlet` (Abschnitt 5.1.4), welches die Requests entgegennimmt. Die Abarbeitung der GetCapabilities- sowie der GetMap-Operation wird an die mit dem `WebMapServiceFrontend` assoziierten Handler-Klassen delegiert.

Der `GetMapHandler` verwendet die Parameterwerte einer eingehenden GetMap-Anfrage[45] um daraus das Context Modell (Abschnitt 4.2.3) zu initialisieren. Beispielsweise wird das mit dem `Context`-Objekt assoziierte `ContextBoundingBox`-Objekt sowie das `ContextWindow`-Objekt, gemäß den im GetMap Request spezifizierten Parametern `BBOX` bzw. `WIDTH` und `HEIGHT` initialisiert. Außerdem werden für die im Request angefragten WMS-Layer die entsprechenden `ContextLayer` in den `Context` geladen. Jeder dieser `ContextLayer` repräsentiert dabei eine über die GetObservation-Operation abfragbare Parameterkonfiguration eines Observation Offerings. Sämtliche Parameterwerte zur Ausführung der GetObservation-Operation, die für diesen `ContextLayer` zu verwenden sind, werden einer Konfigurationsdatei des WMS-Frontends entnommen. Eine Ausnahme bildet der `eventTime`-Parameter. Für ihn wird in der Konfigurationsdatei ein Standardwert angegeben. Dieser wird ersetzt, falls für den `TIME`-Parameter der GetMap-Anfrage ein Wert spezifiziert wurde.

Ebenso wie die Parameterwerte werden auch die für den `ContextLayer` zu verwendenden `ServiceAdapter`-, `FeatureStore`- und `Renderer`-Komponenten in der Konfigurationsdatei spezifiziert. Der Name eines WMS-Layers stellt also den Schlüssel zu einer bestimmten Konfiguration von Parameterwerten und Anbindungskomponenten dar.

Der mit dem `ContextLayer` verknüpfte `Renderer` muss dabei vom Typ `MapLayerRenderer` sein – da lediglich Visualisierungen von Kartenansichten über das WMS-Frontend zurückgegeben werden können. Außerdem sind in der aktuellen prototypischen Imple-

[45] Eine vollständige Beschreibung der Parameter der GetMap-Operation findet bei DE LA BEAUJARDIERE (2006).

mentierung des WMS-Frontends lediglich `Renderer` erlaubt, die `StaticVisualizations` erzeugen. Zukünftig wäre es aber auch möglich `AnimatedVisualizations` über die WMS-Schnittstelle zurückzugeben, beispielsweise im Format eines animierten GIFs.

Im Anschluss an die Initialisierung des Context-Modells wird durch das Framework die Prozessierung der `ContextLayer` angestoßen (Abschnitt 4.3.4). Sobald diese beendet ist, wird dies vom `Context`-Objekt publiziert. Der `GetMapHandler` ist als `EventListener` bei diesem `Context`-Objekt registriert. Er empfängt das Ereignis und damit die berechnete Visualisierung der Kartenansicht. Als Reaktion darauf, gibt er das Bild der Kartenansicht im vom Nutzer spezifizierten Format zurück.

Der `GetCapabilitiesHandler` gibt ein zur WMS Spezifikation konformes GetCapabilities-Dokument zurück, welches eine Beschreibung der angebotenen Layer beinhaltet. Die Informationen zu den Layern, wie Name, Titel und geografischer Raumausschnitt, werden ebenfalls der erwähnten Konfigurationsdatei entnommen.

Die optionale GetFeatureInfo-Operation der WMS Schnittstelle, die den Nutzer befähigt, Informationen über dargestellte Features abzurufen, wird in der aktuellen Implementierung des WMS-Frontends nicht unterstützt.

Neben dem beschriebenen Vorteil des Verfügbarmachens der implementierten Visualisierungsmethoden für WMS-Clients, bringt das Konzept des WMS-Frontends allerdings den Nachteil des Verlusts der Funktionalität der SOS-Schnittstelle mit sich. Denn die Parameter der GetObservation-Anfrage werden mit Ausnahme des `eventTime`-Parameters bereits zur Konfigurationszeit festgelegt. Der Nutzer kann lediglich zwischen vordefinierten Konfigurationen wählen. Abhilfe könnte das Konzept des *O&M Portrayal Service* schaffen, welches in Abschnitt 4.5.3 vorgestellt wurde.

5.3 Implementierung der Anbindungskomponenten

Im Rahmen dieser Arbeit werden neben den Anbindungskomponenten für den SOS auch Anbindungskomponenten für den WMS implementiert. Die implementierten `Renderer` für die vom SOS empfangenen Geosensordaten dienen als Proof-of-Concept und sollen die Möglichkeiten des Frameworks zur Entwicklung von Visualisierungskomponenten aufzeigen. Die realisierten Visualisierungsmethoden für Geosensordaten können auf den Fall stationärer, in-situ Sensoren angewandt werden, für die die in Abschnitt 3.1.1 definierte Restriktion gilt.

Die Funktionsweise dieser Visualisierungsmethoden wird beispielhaft an Geosensordaten gezeigt, die an Wetterstationen in Südafrika gemessen wurden. SOS-Instanzen des *Advanced Fire Information Systems* (AFIS)[46] stellen die Daten bereit. Die Implementierungen können auf Szenarien mit äquivalenten Vorraussetzungen übertragen werden. Als Beispiel seien hier stationäre Sensornetzwerke genannt, die Luft- oder Boden-Belastungen bestimmter chemischer Stoffe messen.

[46] AFIS (McFERREN et al. 2006) ist ein Gemeinschaftsprojekt zwischen dem südafrikanischen Forschungsinstitut *Council for Scientific and Industrial Research* (CSIR) und dem Stromversorgungsunternehmen *Eskom*. Das AFIS-Projekt verwendet Fernerkundungsdaten, um auf deren Grundlage, Buschbrände zu detektieren. Sobald diese die Infrastruktur der Stromversorgung (Überlandleitungen oder Strommasten) bedrohen, findet eine automatische Benachrichtigung der verantwortlichen Person statt. Zur Realisierung des AFIS-Projekts wird das SWE-Framework genutzt.

Eine Diskussion über die Realisierbarkeit von Visualisierungsmethoden für mobil- und/oder remote-erfasste Geosensordaten erfolgt in Abschnitt 5.3.8. Inwieweit das Framework Visualisierungsmethoden unterstützt, für welche die in Abschnitt 3.1.1 definierten Einschränkungen zurückgenommen werden können, wird in Abschnitt 5.3.9 diskutiert.

5.3.1 ServiceAdapter

Um die Service-Operationen ausführen zu können und die Metadaten der Dienste verfügbar zu machen, wurden für den SOS und den WMS entsprechende `ServiceAdapter` (Abschnitt 4.3.1) implementiert. In Abbildung 5-7 ist ihre Klassenstruktur vereinfacht dargestellt. Die `ServiceAdapter`-Implementierungen unterstützten nur bestimmte Versionen der OWS Spezifikationen, da das Metadaten-Modell sowie die Signatur der Service-Operationen zwischen den Versionen differieren[47].

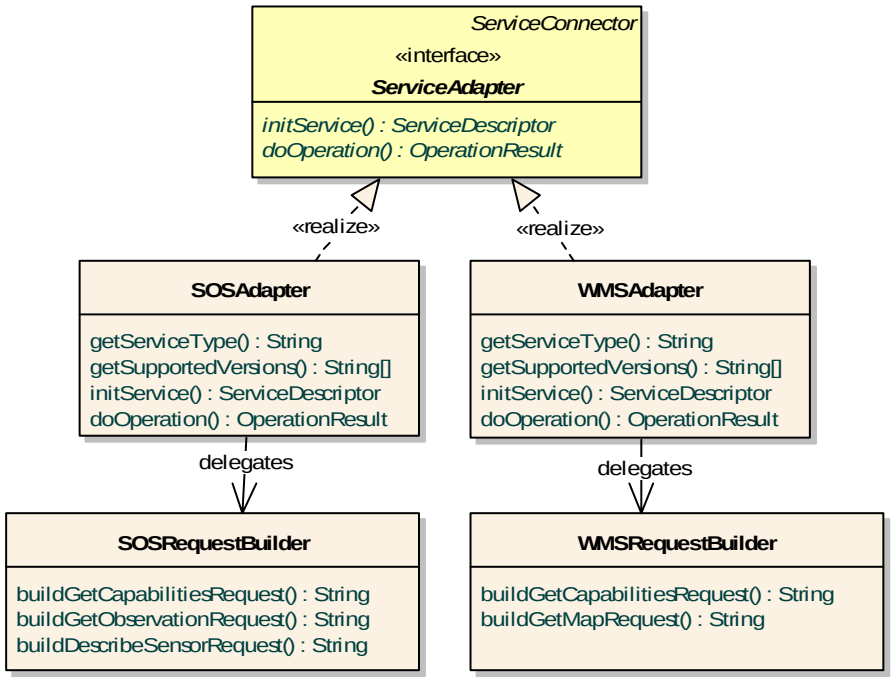

Abbildung 5-7: Implementierte ServiceAdapter in UML-Notation

Zur Ausführung der Service-Operationen durch die `doOperation()`-Methode wird zunächst der Request aufgebaut, um diesen anschließend an den Dienst zu senden. Der Aufbau des Requests wird an die mit den `ServiceAdaptern` assoziierten `RequestBuilder` delegiert. Die `build*`-Methoden der `RequestBuilder`-Klassen erwarten einen `ParameterContainer` (Abschnitt 4.2.3) als Argument[48]. Diesem werden die zur Ausführung

[47] In der aktuellen Implementierung unterstützt der `SOSAdapter` die Spezifikationsversion 0.1.5 und der `WMSAdapter` das Ansprechen von WMS-Instanzen der Versionen 1.0.0, 1.1.0, 1.1.1 und 1.3.0.

[48] Aus Gründen der Übersichtlichkeit wurde auf die Darstellung der Methodenparameter im Diagramm der Abbildung 5-7 verzichtet.

der Operation festgelegten Parameterwerte entnommen. Gegenwärtig unterstützen die `ServiceAdapter` lediglich die von den OWS Spezifikationen verbindlich geforderten Operationen.

Die Implementierungen der `initService()`-Methoden greifen auf die `doOperation()`-Methode zurück, um zunächst die GetCapabilities-Operation des Dienstes auszuführen. Falls bei der Ausführung keine Fehler oder Ausnahmen auftreten, kapselt das zurückgegebene `OperationResult` das Capabilities-Dokument. Aus diesem wird mit Hilfe der XMLBeans-Bibliothek (Abschnitt 5.1.5) ein Objekt-Baum generiert. Die in diesen Objekten enthaltenen Informationen werden genutzt, um das Common Capabilities Modell (Abschnitt 4.2.1) zu initialisieren.

5.3.2 FeatureStore

Um die vom SOS offerierten Daten nutzen zu können, wurde für diesen Dienst ein `FeatureStore` (Abschnitt 4.3.2) implementiert. Dieser erwartet als Eingabeargument der `produceFeatures()`-Methode ein `OperationResult`-Objekt, welches das Antwortdokument der GetObservation-Operation beinhaltet. Nach dem Unmarshalling des Antwortdokuments durch XMLBeans wird das O&M Modell (Abschnitt 5.3.3) instanziiert. Auf den erzeugten Feature-Objekten können für den SOS implementierte `Renderer` arbeiten und Visualisierungen erzeugen.

5.3.3 O&M Modell

Das hier implementierte Modell zur Abbildung O&M-kodierter Daten, ist eine Erweiterung des internen Feature Modells. Während sich das Feature Modell im Core Subsystem befindet, ist das darauf aufbauende O&M Modell im Service-Connector Subsystem enthalten, da lediglich die Anbindungskomponenten des SOS auf dieses Modell zugreifen. Der Core wird durch diese Erweiterung des Frameworks nicht verändert.

Durch die in Abschnitt 3.1.1 festgelegte Einschränkung kann hier zunächst auf eine vollständige Implementierung des Schemas der O&M Spezifikation (Abschnitt 2.2.1) verzichtet werden. Es werden die vier skalaren Observation-Typen sowie der Typ `ObservationCollection` abgebildet. Der „generische“ Typ `Observation` sowie die `CommonObservation` werden nicht umgesetzt. Eine Abbildung komplexer Ergebnisse (siehe die in Abschnitt 2.2.1.3 beschriebenen Fälle) ist auch mit diesem reduzierten Modell durch die Nutzung der `ObservationCollection` und der Aggregation beliebig vieler skalarer Observations möglich, solange sich die von den Sensoren gemessenen Resultate auf skalare Werte aufspalten lassen[49].

Abbildung 5-8 zeigt ein vereinfachtes Klassendiagramm des implementierten O&M Modells. Die in gelb dargestellten Klassen stellen die instanzierbaren Observation-Typen

[49] Der Vorteil der `CommonObservation` besteht in ihrer kompakteren Form. Werden die gleichen Informationen zum einen durch Aggregation mehrerer skalarer Observations in einer `ObservationCollection` und zum anderen durch eine `CommonObservation` ausgedrückt, so ist das serialisierte XML-Dokument der zweiten Variante im Allgemeinen kleiner. Hierdurch wird eine schnellere Übertragung der Daten erreicht. Da in dieser Arbeit Aspekte der Performance jedoch eine untergeordnete Rolle spielen, bleibt die Unterstützung der `CommonObservation` zunächst aus.

dar. Die Klasse `StationType` (grün) wird als Typ eines `featureOfInterest` verwendet. Da die Implementierungen der SOS-`Renderer` auf die Anwendung des beschriebenen Falls (Abschnitt 5.3) beschränkt werden, reicht die Implementierung dieses `featureOfInterest`-Typs, der ebenfalls in der O&M Spezifikation definiert ist (Abschnitt 2.2.1.1), aus. Sobald jedoch Daten von SOS-Instanzen verwendet werden sollen, die andere Schemata zur Beschreibung der `featureOfInterest` nutzen, müssen diese ebenfalls auf Basis des internen Feature Modells implementiert werden.

Abbildung 5-8: Das implementierte O&M Modell in UML-Notation (vereinfachte Darstellung)

Die Klasse `FeatureType` (rot) des Feature Modells (Abschnitt 4.2.2) ist die zentrale Oberklasse sämtlicher Feature-Typen. Die davon abgeleitete Klassenhierarchie bildet die Struktur des O&M-Schemas nach. Für die im O&M-Schema definierten Attribute besitzen die einzelnen Feature-Typen jeweils `AttributeDescriptoren`. Gegenwärtig beschränkt sich das implementierte Modell auf die wichtigsten Attribute. Zur Erzeugung von `Feature`-Objekten stellt jeder Feature-Typ eine `createFeature()`-Methode bereit. Diese nimmt als Eingabeparameter[50] die Attributwerte des zu erzeugenden `Features` entgegen. Die Attributwerte werden über die `setAttributeValue()`-Methode des zu

[50] Aus Gründen der Übersichtlichkeit wurde auf die Darstellung der Methodenparameter im Diagramm der Abbildung 5-8 verzichtet.

initialisierenden Features für den korrespondieren AttributeDescriptor gesetzt. Die Datentypen der Attributwerte werden den Java Standard-Bibliotheken entnommen. Ist dies nicht möglich, wie z.B. für das Attribut eventTime der EventType-Klasse, so werden die Datentypen im O&M Modell realisiert.

5.3.4 WMS-Renderer

Um die vom WMS zurückgegebenen Raster-Daten in aufsetzenden Applikationen darstellen zu können, steht die Klasse WMSRenderer als Unterklasse des MapLayerRenderers (Abschnitt 4.3.3) bereit. Die render()-Methode dieser Klasse erwartet ein RenderDataInput-Objekt vom Typ OperationResult. Dieses Objekt enthält die vom WMS empfangenen binären Bild-Daten, die mit Hilfe der JAI-Bibliothek (Abschnitt 5.1.3) in ein Java Bild-Objekt überführt werden. Der WMSRenderer unterstützt die Bild-Formate JPEG, PNG und GIF. Abbildung 5-2 zeigt zwei überlagerte, vom WMSRenderer erzeugte Visualisierungen.

5.3.5 SOS-Renderer zur Kartenlayer-Visualisierung

Nun folgend werden die implementierten Visualisierungsmethoden für Geosensordaten beschrieben. Wie in Abschnitt 3.1.3 gefordert, berücksichtigen diese Renderer auf unterschiedliche Weise den Raumbezug, den Zeitbezug und den Sachbezug der Observations bzw. der sie betreffenden Geoobjekte.

In diesem Abschnitt werden zunächst SOS-Renderer beschrieben, die von der Klasse MapLayerRenderer erben und somit Kartenlayer-Visualisierungen erzeugen. Dies geschieht unter Zuhilfenahme der räumlichen Beschreibung der Geoobjekte bezüglich derer die Daten gemessen wurden. Abhängig von der implementierten Visualisierungsmethode werden ebenfalls zeitliche und thematische Aspekte einbezogen. In Abschnitt 5.3.6 werden SOS-Renderer vorgestellt, die Diagramm-Visualisierungen erzeugen und in erster Linie den Sachbezug der Sensordaten berücksichtigen. Sie erben von der Klasse ChartRenderer.

Beide Arten von Renderer nehmen in ihrer render()-Methode ein FeatureDataInput-Objekt (Abschnitt 4.3.3) entgegen. Dieses enthält die vom FeatureStore produzierte FeatureCollection, welche die vom SOS zurückgegebenen Observations in Form von Feature-Objekten zusammenfasst.

5.3.5.1 Visualisierung von Diagrammkarten

Zur Darstellung von Diagrammen in einer Kartenansicht, wurde zum einen der TimeSeriesMapChartRenderer implementiert. Dieser zeichnet für jedes featureOfInterest, das in den übergebenen Observations enthalten ist, ein Zeitreihendiagramm in eine Karte. Die Diagramme stellen die zeitliche Entwicklung der Resultatwerte einer im GetObservation Request spezifizierten observedProperty dar.

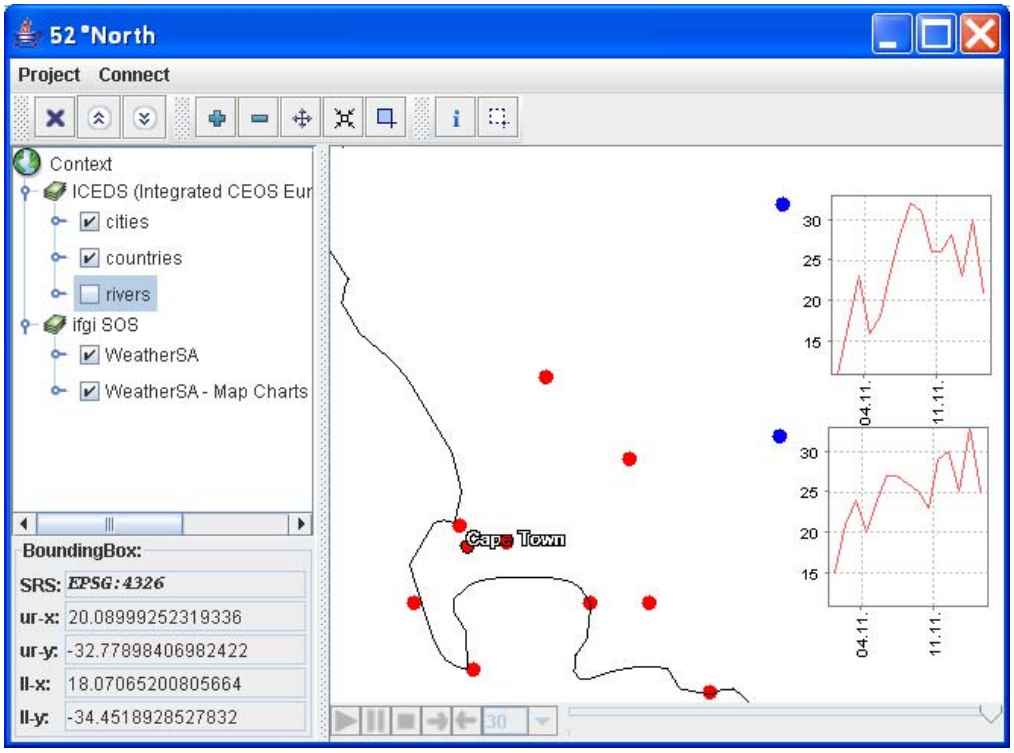

Abbildung 5-9: Entwicklung der Temperatur an zwei Wetterstationen in Südafrika

Aus den Realwelt-Koordinaten der geometrischen Beschreibung des `featureOfInterest` lassen sich entsprechend dem vom `MapLayerRenderer` (Abschnitt 4.3.3) geerbten Attributen `boundingBox`, `mapWidth` und `mapHeight` die Bildschirmkoordinaten berechnen, an deren Position die Diagramme gezeichnet werden. Ist die Geometrie des `featureOfInterest` nicht punkthaft, sondern stellt z.B. eine Linie oder ein Polygon dar, so wird der Schwerpunkt der Geometrie verwendet.

Abbildung 5-9 stellt eine mit dem `TimeSeriesMapChartRenderer` erzeugte Visualisierung dar. Sie zeigt die Entwicklung der Beobachtungsvariablen *Temperatur* an zwei Wetterstationen (den `featureOfInterest`). Die vom SOS zurückgegebenen Observations, welche die Resultate der Beobachtungsvariablen *Temperatur* wiedergeben, sind vom Typ `Measurement`. Demzufolge werden die von der `FeatureStore`-Komponente des SOS erzeugten `Feature`-Objekte mit Hilfe der Klasse `MeasurementType` (Abbildung 5-8) des O&M Modells erzeugt. Während für die `CountObservation` eine Visualisierung ebenso möglich ist, werden die Observation-Typen `Category`- und `TruthObservation` von diesem `Renderer` nicht unterstützt, da ihr Wertebereich nicht metrisch skaliert ist.

Zum anderen wurde der `AnimatedMapBarChartRenderer` implementiert, welcher zu jedem Geoobjekt ein Balkendiagramm in die Karte zeichnet. Jeder Balken eines der dargestellten Balkendiagramme, steht für eine Beobachtungsvariable. Somit können die Resultatwerte mehrerer Beobachtungsvariablen gleichzeitig visualisiert werden. Damit auch bei dieser Visualisierungstechnik die zeitliche Entwicklung der Resultatwerte verdeutlicht werden kann, wird für jeden Zeitschritt eine Kartenlayer-Visualisierung berechnet, die dann als Frames in einer `AnimatedVisualization` dienen.

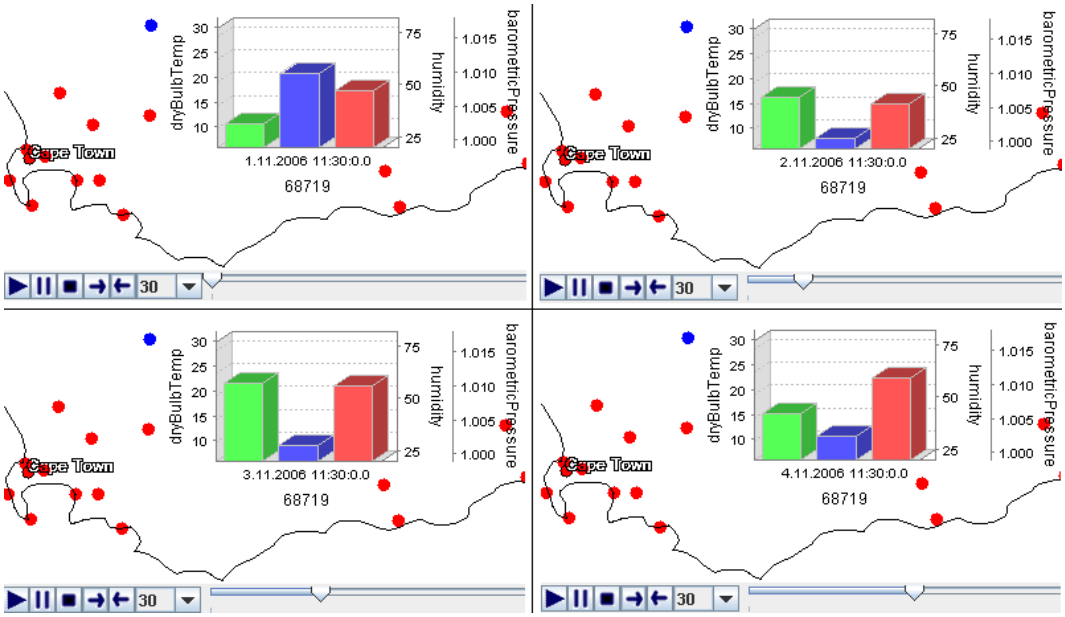

Abbildung 5-10: Animierte Visualisierung von Temperatur (grün), Luftfeuchte (blau) und Luftdruck (rot)

Abbildung 5-10 zeigt vier Frames einer animierten Visualisierung, die mit Hilfe des `AnimatedMapBarChartRenderers` erzeugt wurde. Die Animation stellt die an einer Wetterstation gemessenen Werte der Beobachtungsvariablen *Temperatur*, *Luftfeuchtigkeit* sowie *Luftdruck* dar.

Beide Visualisierungstechniken berücksichtigen sowohl den Sach- als auch den Zeitbezug der Geosensordaten. Da die Diagramme an die geografische Position der Geoobjekte gezeichnet werden, wird ebenso der Raumbezug ersichtlich. Somit geben die erzeugten Visualisierungen dem Nutzer einen Eindruck, um Aussagen über eventuelle raumzeitliche Trends oder Muster in den Daten treffen zu können.

Der zweite vorgestellte `Renderer` ermöglicht im Gegensatz zum ersten die Darstellung mehrerer Beobachtungsvariablen in einer Visualisierung, nämlich als einzelne Balken. Diese *multivariate Repräsentation* (MACEACHREN et al. 1998), also die gleichzeitige Darstellung mehrerer thematischer Eigenschaften sowie der zeitlichen und räumlichen Dimension, ermöglichen dem Nutzer, vergleichende Aussagen über die abgebildeten Variablen zu treffen.

Nachteilig bei der Visualisierung von Diagrammkarten ist allerdings, dass die Größe der einzelnen Diagramme stark limitiert ist, da es ansonsten zu Überlappungen zwischen den Diagrammen oder anderer Informationsinhalte der Karte kommen kann. Weiterentwicklungen dieser Visualisierungskomponenten könnten die Größe und Position der Diagramme so anpassen, dass Überlappungen ausbleiben. Eine Kopplung der Diagrammgröße an den aktuellen Maßstab der Karte wäre ebenfalls denkbar.

5.3.5.2 Visualisierung von interpolierten Werteoberflächen

Häufig werden raumzeitliche Phänomene durch Geosensoren punkthaft aufgenommen. Der Nutzer ist im Allgemeinen allerdings an einer Visualisierung interessiert, die eine

flächenhafte Oberflächenrepräsentation des Phänomens zeigt. Um eine solche Visualisierung zu erzeugen, müssen Verfahren der räumlichen Interpolation eingesetzt werden. In Abschnitt 4.5.2 wurde bereits die Nutzung eines Processing Service vorgeschlagen, der aus punkthaft gemessenen Daten eine Werteoberfläche für das gesamte Beobachtungsgebiet interpoliert, die anschließend von einem `Renderer` visualisiert wird. Im Folgenden werden `Renderer` beschrieben, welche sowohl die Interpolation als auch die Visualisierung durchführen.

Zur Berechnung der Werteoberfläche für das gesamte Untersuchungsgebiet werden hier die beiden nichtstatistischen Interpolationsverfahren der *Nearest Neighbour*- (NN) sowie der *Inverse Distance*-Methode (IDW)[51] unterstützt[52]. Die Klassen `IDWRenderer` und `NNRenderer` realisieren die Visualisierung von Werteoberflächen mit Hilfe der genannten Verfahren. Abbildung 5-11 zeigt eine mit Hilfe des `IDWRenderers` erzeugte Visualisierung der Beobachtungsvariablen *Temperatur*.

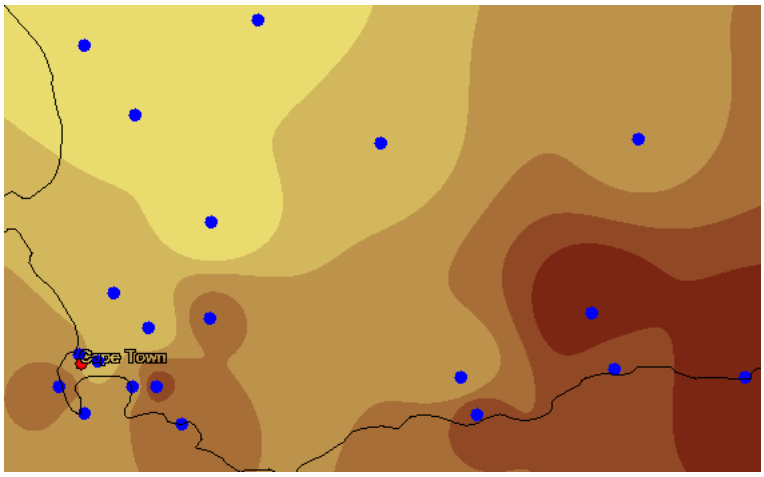

Abbildung 5-11: Werteoberflächen-Visualisierung der Temperatur (erzeugt mit dem IDWRenderer)

Das vom `NNRenderer` produzierte Resultat ist in Abbildung 5-12 dargestellt.

[51] Auf eine ausführliche Beschreibung der beiden Verfahren soll in dieser Arbeit verzichtet werden, kann aber z.B. bei BURROUGH & MCDONNELL (1998) nachgelesen werden.

[52] Künftige Implementierungen könnten komplexere Verfahren aus der Geostatistik realisieren. Als Beispiel sei hier das Kriging-Verfahren genannt, welches z.B. bei LONGLEY et al. (2001) beschrieben wird.

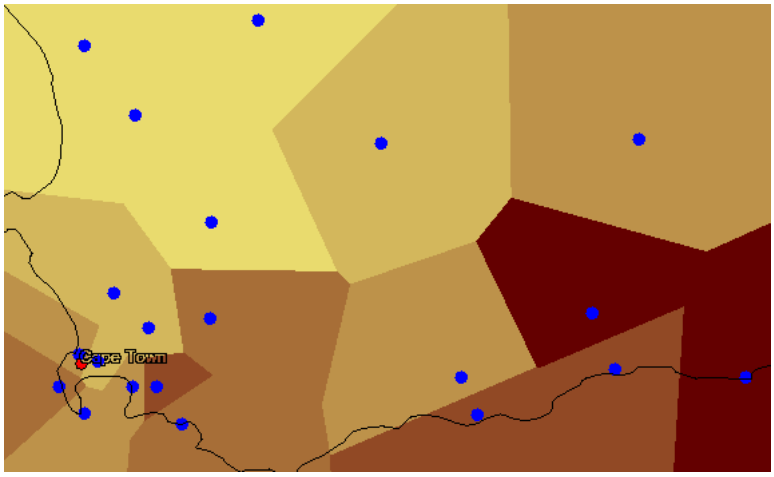

Abbildung 5-12: Werteoberflächen-Visualisierung der Temperatur (erzeugt mit dem NNRenderer)

Da für beide Klassen große Teile der Implementierung gleich sind, besitzen sie die gemeinsame abstrakte Oberklasse `InterpolationRenderer` (Abbildung 5-13).

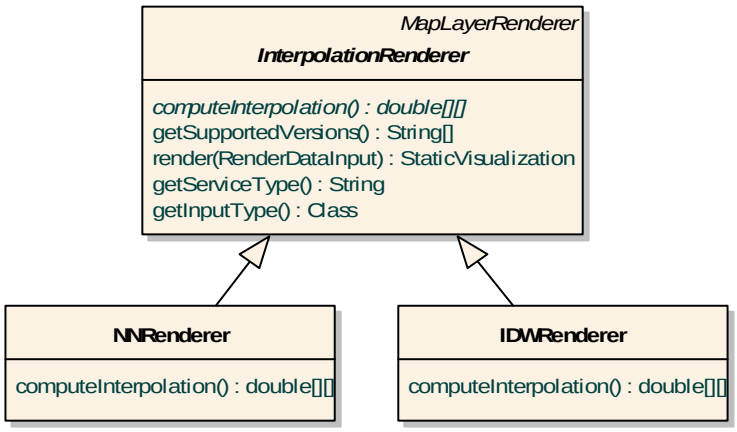

Abbildung 5-13: Renderer zur Visualisierung von Werteoberflächen in UML-Notation

In der geerbten `render()`-Methode wird zunächst durch Aufruf der `computeInterpolation()`-Methode aus den Observations eine Werteoberfläche[53] berechnet. Sie wird repräsentiert durch ein Raster, ähnlich einer Bildmatrix, dessen Ausmaße den Dimensionen der Kartenansicht entsprechen. Die konkrete Realisierung der `computeInterpolation()`-Methode, in der für jede Zelle des Rasters ein Wert geschätzt wird, findet sich jeweils in den beiden Unterklassen.

Um aus dem berechneten Raster ein Bild zu erzeugen, muss jeder Rasterzelle im Anschluss eine Klasse zugewiesen werden. Sollen `Truth`- oder `CategoryObservations` visualisiert werden, so verwendet der `Renderer` die Kategorien des Wertebereichs als Klassen. Sind die Observations jedoch vom Typ `Measurement` oder `CountObservation`, so muss eine Klassifizierung des Wertebereichs der Beobachtungsvariablen durchgeführt

[53] Beide Klassen erzeugen eine statische Visualisierung. Die Observations müssen sich daher auf die gleiche Messzeit beziehen, da die Interpolation gegenwärtig nur über den Raum nicht aber über die Zeit stattfindet.

werden[54]. Zur Visualisierung der Werteoberfläche wird jeder Klasse eine Farbe zugewiesen. Die vorgenommene Klassifikation und die zu den Klassen gehörigen Farben werden in einem Legendenbild dargestellt, welches mit der erzeugten `StaticVisualization` assoziiert wird und über die `getLegend()`-Methode abfragbar ist (Abschnitt 4.3.3). Abbildung 5-14 stellt die Legendenansicht der Client-Applikation dar, welche die von einem `InterpolationRenderer` erzeugte Legende zeigt.

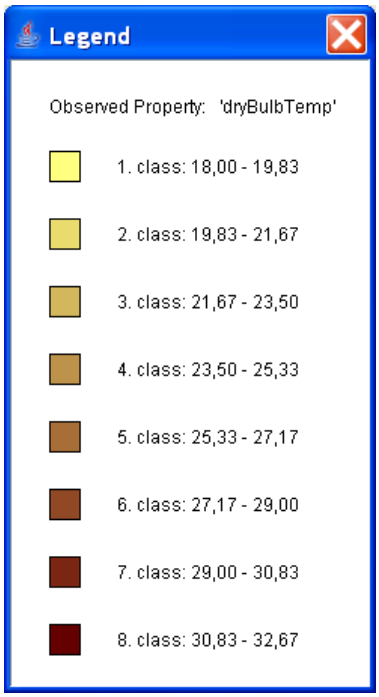

Abbildung 5-14: Legendenansicht der Client-Applikation

Da die gegenwärtigen Implementierungen der `InterpolationRenderer` statische Visualisierungen generieren, wird die Dynamik des beobachteten räumlichen Prozesses nicht berücksichtigt. Dies kann sich in zukünftigen Implementierungen, durch die Erzeugung temporaler Animationen ändern. Eine andere Möglichkeit den Zeitbezug zu berücksichtigen, kann darin bestehen, die Werteoberflächen zweier Messzeiten zu „subtrahieren“ und somit die Veränderung zwischen diesen darzustellen.

5.3.5.3 Visualisierung von Proportionalen Symbolkarten

Der `ProportionalCircleMapRenderer` erlaubt es, Proportionale Symbolkarten (KRAAK & ORMELING 1997) zu visualisieren. Dazu wird der Wertebereich einer Beobachtungsvariablen[55] nach einer Klassifikation auf Kreissymbole bestimmter Größen abgebildet. Für jedes Geoobjekt wird dann gemäß dem vom Sensor aufgenommenen Resultatwert ein

[54] Die aktuelle Implementierung teilt den Wertebereich in gleich große Klassen. Abschnitt 5.3.7 diskutiert hierzu die Einbeziehung des Nutzers.

[55] Dieser `Renderer` ist auf die Visualisierung der Observation-Typen `Measurement` und `CountObservation` beschränkt, da deren metrische Wertebereiche auf die proportionalen Symbole abgebildet werden können.

Kreissymbol entsprechender Größe in die Karte gezeichnet. Um bei dieser Visualisierungsmethode die zeitliche Entwicklung der Resultatwerte beobachten zu können wird auch hier für jede Messzeit ein eigener Frame generiert und das Ergebnis als `AnimatedVisualization`-Objekt zurückgegeben.

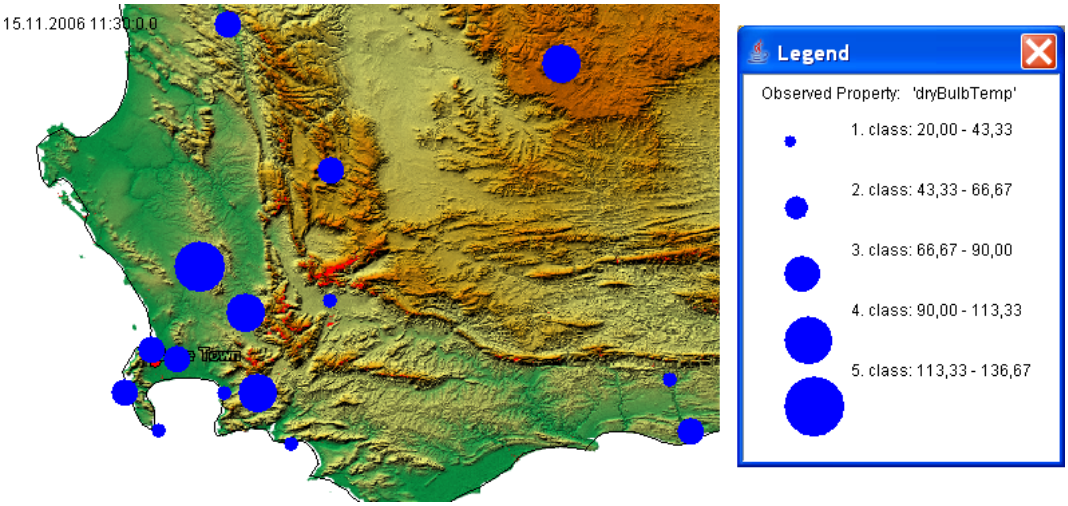

Abbildung 5-15: Proportionale Symbolkarte der Temperatur

Abbildung 5-15 zeigt die von einem `ProportionalCircleMapRenderer` generierte Visualisierung sowie die zugehörige Legende von proportionalen Kreissymbolen für die Beobachtungsvariable *Temperatur*.

5.3.6 SOS-Renderer zur Diagramm-Visualisierung

Die bisher beschriebenen `Renderer` produzieren direkt raumbezogene Visualisierungen in Form von Kartenlayern. Das OX-Framework erlaubt allerdings auch die Visualisierungen von Diagrammen, indem es die Implementierung von speziellen `ChartRenderern` (Abschnitt 4.3.3) ermöglicht. Exemplarisch werden hier der `TimeSeriesChartRenderer` sowie der `ScatterPlotChartRenderer` vorgestellt.

Der `TimeSeriesChartRenderer` produziert die Visualisierung eines Zeitreihendiagramms. Eine mit diesem `Renderer` erzeugte Visualisierung zeigt Abbildung 5-5. Diese Visualisierung gibt die Entwicklung der Temperatur über einen gewissen Zeitraum an drei Wetterstationen wieder. Diese Diagrammform kann verwendet werden, um die Geosensordaten nach zeitlichen Trends oder Mustern zu erkunden.

Der `ScatterPlotChartRenderer` zeichnet ein Streudiagramm (engl. „scatterplot“), welches zwei Beobachtungsvariablen auf zwei senkrecht aufeinander stehende Achsen abbildet. Diese sind entsprechend dem zugehörigen Wertebereich skaliert. Dazu wird jedem Paar aus `featureOfInterest` und Messzeit das entsprechende Tupel von Resultatwerten der beiden Beobachtungsvariablen zugeordnet. Diese Tupel werden dann als Punkte in das Streudiagramm eingezeichnet. Aus der entstehenden Punktwolke lassen sich eventuelle Korrelationen zwischen den beiden Variablen erkennen. Abbildung 5-16 zeigt ein so erzeugtes Streudiagramm. Es stellt die Beobachtungsvariablen *Temperatur* und *Luftdruck* gegenüber.

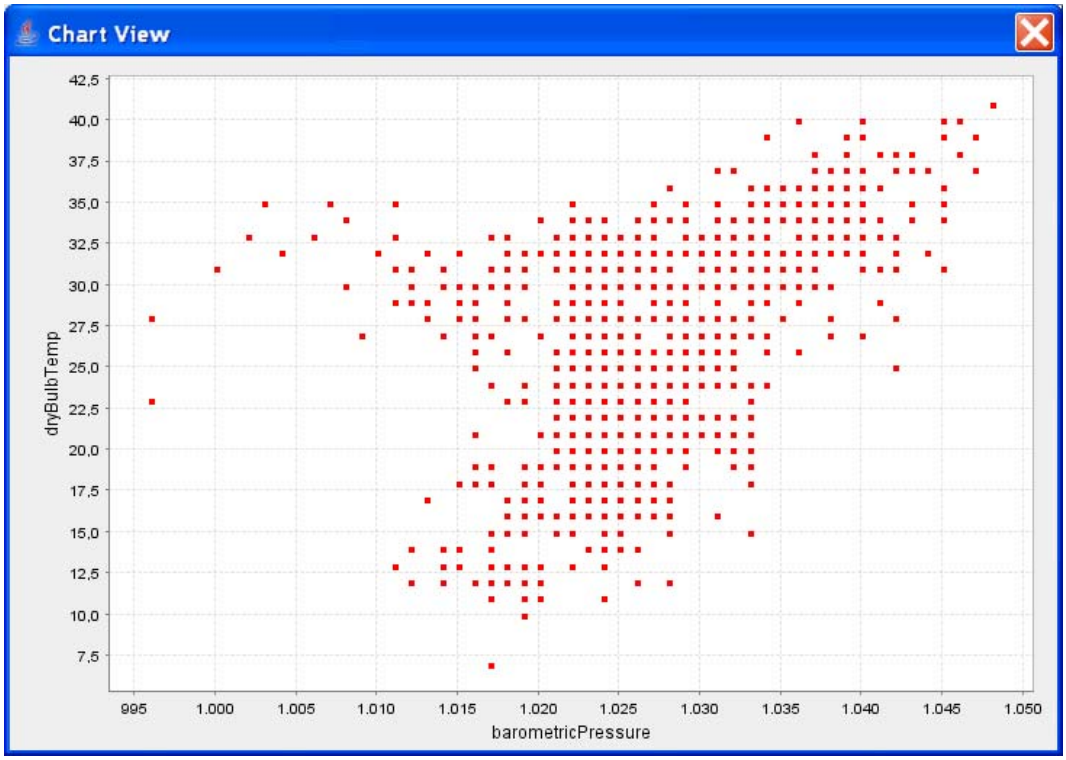

Abbildung 5-16: Streudiagramm von Temperatur (y-Achse) und Luftdruck (x-Achse)

Die aktuellen Implementierungen von `ChartRenderern` erzeugen statische Visualisierungen. Ebenso ist es möglich, dass `ChartRenderer` animierte Diagramme, also Objekte vom Typ `AnimatedVisualization`, produzieren. Um diese z.B. in der Client-Applikation abspielen zu können, müsste der Dialog der Diagrammansicht (Abschnitt 5.2.1.2) um Eingabeelemente zur Steuerung von Animationen erweitert werden.

5.3.7 Parametrisierung des Rendervorgangs

Gegenwärtig sieht das Framework keine Parametrisierung des Rendervorgangs durch den Nutzer vor. Sinnvoll wäre, z.B. die Klassifizierung des Wertebereichs einer Beobachtungsvariablen oder die Auswahl von Farben durch den Nutzer bestimmbar zu machen. Diese Parameter werden aktuell zur Implementations- bzw. zur Konfigurationszeit der einzelnen `Renderer` definiert. Um Eingabeparameter für den Rendervorgang zur Laufzeit festlegen zu können, müsste die `render()`-Methode der `Renderer`-Schnittstelle erweitert werden. Ebenso müssten aufsetzenden Applikationen entsprechende Mechanismen zur Spezifikation der Eingabeparameter bereitstellen.

5.3.8 Visualisierung mobil- und/oder remote-erfasster Geosensordaten

Die beschriebenen SOS-`Renderer` wurden für Geosensordaten implementiert, die von stationären, in-situ Sensoren aufgenommen werden. Aufbauende Arbeiten müssen zeigen, dass auf Basis des entwickelten Frameworks ebenso Visualisierungsmethoden für mobil- und/oder remote-erfasste Geosensordaten implementiert werden können. Bei diesen

Geosensordaten ändert sich die Position des Sensors über die Zeit, und die Position von Sensor und beobachtetem Geoobjekt unterscheidet sich zum Messzeitpunkt. Die Grundlagen für die Implementierung solcher Visualisierungsmethoden bietet das Framework bereits:

Die zur Visualisierung mobil-erfasster Daten notwendigen Informationen über die Position des Sensors zur Zeit des Messereignisses können im O&M Modell für eine Observation gespeichert werden (Abbildung 5-8)[56]. Werden für mehrere Messzeiten Observations eines mobilen Sensors abgefragt, so können daraus die Positionsveränderungen abgeleitet werden. Beispielsweise können dann die vom Framework bereitgestellten Mechanismen zur Erzeugung temporaler Animationen benutzt werden, um die Positionsveränderungen visuell zu veranschaulichen. Die Möglichkeit zur separaten Beschreibung der Geometrie von Sensor und beobachtetem Geoobjekt innerhalb einer Observation[57] stellen die Informationen bereit, die eine Visualisierung der besonderen Eigenschaft remote erfasster Geosensordaten zulassen.

5.3.9 Visualisierung komplexerer Formen von Geosensordaten

Das Framework wurde in Abschnitt 3.1.1 auf die Unterstützung von Visualisierungsmethoden für Geosensordaten beschränkt, die zu einer diskreten Messzeit erfasst werden und deren Resultate in skalare Werte aufgespalten werden können. Über den vom O&M-Schema bereitgestellten generischen Typ `Observation` (Abschnitt 2.2.1.3) können allerdings beliebig komplexe Typen als Resultat verwendet werden. Lediglich das XML-Schema des Resultattyps muss in der Observation referenziert werden. Als Beispiel sei hier die Instanz einer `Observation` genannt, deren Resultatwert die Beschreibung eines geometrischen Objekts speichert (vgl. COX 2006, S. 44). Dieses kann nicht als skalarer Wert aufgefasst oder ohne Informationsverlust in skalare Werte aufgespalten werden. Das Framework unterstützt allerdings auch Visualisierungsmethoden, die auf Geosensordaten mit diesen komplexen Resultatwerten angewandt werden können. Allerdings müssen die komplexen Typen dazu a priori bekannt sein. D.h. das implementierte O&M Modell sowie die `FeatureStore`-Komponente für den SOS müssen zur Unterstützung dieser Typen erweitert werden. Für das erweiterte O&M Modell können dann `Renderer` entwickelt werden, die auf Basis der neuen Feature-Typen Visualisierungen erzeugen.

Eine andere Art, komplexe Resultatwerte mit Hilfe der O&M zu kodieren, ist die Referenzierung externer Ressourcen auf denen sich die vom Sensor erfassten Daten befinden (Abschnitt 2.2.1.4). Aufgrund der Vielfalt der Formen von Sensordaten, die mit diesem Mechanismus referenzierbar sind, muss die Eignung des Frameworks für die Unterstützung sinnvoller Visualisierungsmethoden im Einzelfall geprüft werden. Eine diesbezügliche Analyse und eventuelle Erweiterung der Architektur kann in aufbauenden Arbeiten erfolgen.

[56] Die geerbten Attribute `time` und `location` einer Observation können dazu verwendet werden.

[57] Dies ist über die Attribute `location` und `featureOfInterest` möglich.

6 Diskussion und Ausblick

Das Ziel der Arbeit ist der Entwurf eines (Software-)Frameworks, welches den Zugriff auf verschiedene OGC Web Services unterstützt und das flexible Einbinden unterschiedlicher Visualisierungsansätze für integrierte Geodaten und insbesondere Geosensordaten erlaubt. Im nächsten Abschnitt werden die zentralen Anforderungen, die sich aus der Zielsetzung (Abschnitt 1.1) ergeben und in Kapitel 3 präzisiert wurden, wieder aufgegriffen und der entworfenen Architektur gegenübergestellt. Aus den Eigenschaften des Frameworks werden Ideen für zukünftige Arbeiten entwickelt.

6.1 Unterstützung vielfältiger Visualisierungsansätze

Eine zentrale Anforderung an das Framework ist, dass vielfältige Visualisierungsansätze unterstützt werden und die verschiedenen Aspekte der Geosensordaten, Raum-, Sach- und Zeitbezug, in die Visualisierungen einfließen können (Abschnitt 3.1.2 und 3.1.3).

Die entwickelten Konzepte zur Realisierung von Visualisierungskomponenten (Abschnitt 4.3.3) sind so flexibel, dass das Framework diese Anforderung erfüllt. Dies wurde anhand beispielhafter Implementierungen gezeigt. Während diese Implementierungen gegenwärtig auf Sensordaten beschränkt sind, die von stationären, in-situ Sensoren aufgenommen werden, bietet das Framework ebenso die Grundlage zur Visualisierung mobil- und/oder remote-erfasster Geosensordaten (Abschnitt 5.3.8).

Um den *Sachbezug* der Geosensordaten separat darzustellen, können auf Basis des Frameworks Komponenten entwickelt werden, die Diagramme oder Grafiken visualisieren. Im Rahmen dieser Arbeit wurden so Visualisierungskomponenten implementiert, die Streu- oder Zeitreihendiagramme generieren. Künftig können neben diesen einfachen Diagrammen auch komplexere grafische Techniken zur Visualisierung der Resultatwerte realisiert werden. Als Beispiele seien hier Ikonenbasierte Techniken, Pixelbasierte Techniken, Hierarchische Techniken oder Streckenzüge genannt, die neueren Forschungsbereichen, wie der Exploratory Visual Analysis (GAHEGAN 2001) oder dem Visual Data Mining (HINNEBURG et al. 1999) entstammen.

Zur Darstellung des *Raumbezugs* können Visualisierungskomponenten implementiert werden, die Kartenansichten erzeugen. Über die Techniken der thematischen Kartografie kann in diesen Kartenansichten neben dem Raum- auch der Sachbezug veranschaulicht werden. Außerdem können für ausgewählte Geosensordaten im gleichen Kontext Kartenlayer- und Diagramm-Visualisierungen generiert werden. Somit wird eine indirekte Darstellung des Raumbezugs hergestellt. Um diese indirekten Darstellungen aussagekräftiger zu gestalten, könnten künftige Applikationen die Technik des *Linking & Brushing* unterstützen. Dabei kann der Nutzer durch Selektion von Feature-Objekten der einen Darstellung die korrespondierenden Objekte verknüpfter Darstellungen grafisch hervorheben (SCHUMANN & MÜLLER 2000). Die Grundlagen zur Realisierung dieser Technik sind bereits gegeben, da Visualisierungskomponenten darüber informiert werden können, welche Feature-Objekte als selektiert betrachtet werden sollen (Abschnitt 4.3.3).

Weiterhin lässt das Framework die Erzeugung temporaler Animationen zu. Somit kann die Dynamik der beobachteten Geoobjekte bzw. der *Zeitbezug* der Geosensordaten explizit zum Ausdruck gebracht werden.

Die im Rahmen dieser Arbeit implementierten Visualisierungskomponenten können weiterentwickelt und verbessert werden. Mögliche Ansätze hierzu wurden bereits skizziert (Abschnitt 5.3.4ff). Ebenso können neue Visualisierungsmethoden umgesetzt werden, die wiederum andere Charakteristiken der Geodaten zum Ausdruck bringen. Stehen mehrere alternative Repräsentationen der Daten bereit, erleichtert dies die visuelle Exploration.

Für einzelne Projekte können auf einfache Weise eigene Visualisierungskomponenten realisiert werden. Das OX-Framework und die Client-Applikation werden bereits in dem südafrikanischen Forschungsprojekt AFIS (Abschnitt 5.3) genutzt und weiterentwickelt. Für dieses Projekt wurden Visualisierungskomponenten implementiert, die Windfeldkarten auf Basis von Windrichtungs- sowie Windgeschwindigkeits-Daten generieren. Diese unterstützen den Nutzer bei der Interpretation und der Validierung von empfangenen Buschbrand-Alarmierungen.

Beachtet werden muss, dass die Anwendung der Visualisierungskomponenten im Regelfall nur für bestimmte Observation-Typen Sinn macht bzw. fehlerfrei funktioniert. Die in dieser Arbeit implementierten SOS-Renderer sind auf die Anwendung der skalaren Observation-Typen beschränkt. Im Einzelfall können Einschränkungen hinzukommen, die z.B. die Anzahl angefragter Beobachtungsvariablen betreffen. Werden dem Nutzer in einem Anwendungsprogramm Visualisierungskomponenten zur Auswahl gestellt, so müssen Informationen zur Verfügung stehen, welche die erlaubten Eingabedaten beschreiben. Dies ist notwendig damit ein Anwendungsprogramm Dialoge zur Anfrage der zu visualisierenden Eingabedaten entsprechend anpassen kann. Außerdem müssen im Falle einer Eingabe nicht unterstützter Daten aussagekräftige Fehlermeldungen erzeugt werden können. Dies wird gegenwärtig noch nicht vom Framework berücksichtigt. Ein Konzept zur Lösung dieses Problems könnte die Verknüpfung einer Visualisierungskomponente mit einer Metadaten-Beschreibung vorsehen, die entsprechende Informationen für Applikationen bereitstellt. Das Prozess-Modell der SensorML Spezifikation (Abschnitt 2.2.2) könnte für eine solche Metadaten-Beschreibung genutzt werden (siehe dazu auch Abschnitt 4.5.3). Die Realisierung dieses Konzepts würde den Austausch und die Verbreitung von Visualisierungskomponenten erleichtern. Diese Metadaten-Beschreibungen könnten in Katalogen veröffentlicht werden, um den Nutzern eine Suche nach passenden Visualisierungskomponenten zu ermöglichen.

6.2 Anbindung verschiedener OWS Typen

Eine weitere Anforderung an das Framework ist, dass die Datenressourcen verschiedener OGC Web Service Typen eingebunden und visualisiert werden können (Abschnitt 3.1.3.2).

Um dieser Anforderung gerecht zu werden, wurden auf der Grundlage von OGC Spezifikationen das Common Capabilities Modell (Abschnitt 4.2.1) sowie das Feature Modell (Abschnitt 4.2.2) in den Kern des Frameworks übernommen. Diese Datenmodelle erlauben die Integration der Service Metadaten bzw. angebotener Feature-Daten und bieten

den unterschiedlichen Komponenten des Frameworks eine einheitliche Basis zur Weiterverarbeitung dieser Daten.

Die Aufgaben zur Anbindung eines Dienstes wurden in drei separaten Schnittstellen (Abschnitt 4.3) gekapselt. Konkrete Anbindungskomponenten können für beliebige OWS Typen implementiert werden und als Plugins in Applikationen eingebunden werden. Im Rahmen dieser Arbeit wurden Anbindungskomponenten für den SOS und den WMS implementiert. In Zukunft könnten weitere OWS Typen (z.B. WCS oder WFS) unterstützt werden.

Beim Einbinden der Datenressourcen verschiedener Dienste wird in der aktuellen Implementierung des Frameworks auf eine Transformation zwischen unterschiedlichen räumlichen Bezugssystemen verzichtet. Visuell zu überlagernde Datenressourcen müssen daher im gleichen räumlichen Bezugssystem vorliegen. Die Erweiterung des Systems um die Funktionalität der Koordinatentransformation ist denkbar. Um auf eine eigene Implementierung zu verzichten, kann die notwendige Logik über einen Web Coordinate Transformation Service (WCTS) (WHITESIDE et al. 2005) bezogen werden. Für diesen OWS Typ kann dazu eine Zugriffskomponente (Abschnitt 4.3.1) implementiert werden.

Am Beispiel des WCTS wird deutlich, dass das Framework nicht auf das Anbinden datenofferierender Dienste beschränkt ist. Im Hinblick auf die Nutzung des Sensor Webs, können Zugriffskomponenten für die übrigen Dienste der SWE-Initiative, SPS, SAS und WNS, implementiert werden, um diese über das Framework ansprechen zu können. Für den SAS wäre ebenso die Entwicklung von Visualisierungskomponenten sinnvoll, um auftretende Ereignisse z.B. in einer Kartenansicht darstellen zu können. Basierend auf dem OX-Framework könnten Applikationen entwickelt werden, die dem Nutzer neben der Visualisierung von Geosensordaten auch die Steuerung von Sensoren sowie den Erhalt von Alarmierungen und Benachrichtigungen ermöglichen.

6.3 Entwicklung der Architektur als Framework

Die wesentliche technische Anforderung dieser Arbeit ist, die Architektur als Framework zu entwerfen, um es somit in unterschiedlichen Applikationen einsetzen zu können (Abschnitt 3.2.1). Diese Anforderung wurde durch die Umsetzung der zentralen Charakteristiken einer Framework-Architektur erfüllt:

Durch die Konzeption der Schnittstellen der Anbindungskomponenten als *Variationspunkte* des Frameworks (Abschnitt 4.3) wurde erreicht, dass der Kernbereich des Frameworks unabhängig ist von der Datenverwaltungs-Schicht, also den OGC Web Services. Die Implementierung neuer Anbindungskomponenten verlangt dadurch keine Änderungen des Kernbereichs.

Das entwickelte Event-Listener-Konzept (Abschnitt 4.4) realisiert die *Umkehrung des Hauptkontrollflusses*. Es bewahrt die Unabhängigkeit des Frameworks von der Präsentations-Schicht. Es können neue Applikationen auf dem OX-Framework aufgebaut werden, ohne Anpassungen vornehmen zu müssen.

Da das entworfene Framework lediglich *Aufgaben der Geschäftslogik* erfüllt und unabhängig ist von der Präsentations-Schicht, wird die Wiederverwendung in verschiedenen Applikationen möglich. Die Geschäftslogik, die das Framework bereitstellt, bezieht sich

im Wesentlichen auf die Integration und Visualisierung von Geodaten. Sollen aufsetzende Anwendungen nicht unterstützte Funktionalitäten erfüllen, so müssen diese von frameworkexternen Komponenten bzw. von der Anwendung selbst realisiert werden.

Die Implementierungen der Client-Applikation sowie des WMS-Frontends haben gezeigt, dass das Framework in verschiedenartigen Anwendungen zum Einsatz kommen kann. In Zukunft können weitere Applikationen entwickelt werden. Denkbar ist eine Thin-Client-Applikation deren Präsentations-Schicht durch ein HTML-Frontend realisiert wird. Ebenso ist es möglich, das OX-Framework auf einem mobilen Endgerät mit einer entsprechenden Präsentations-Applikation zu nutzen. In beiden Fällen können die vom Framework bereitgestellte Geschäftslogik sowie die implementierten Anbindungskomponenten wiederverwendet werden.

Die Client-Applikation kann beispielsweise um Analyse- oder Prozessierungs-Funktionalitäten erweitert werden. Das Feature Modell bietet eine Basis, um räumliche Operatoren, Puffer-Berechnungen oder Simulationen für Feature- bzw. Sensordaten durchführen zu können. Derartige Prozessierungs-Funktionen können auch über externe Web Processing Services (SCHUT & WHITESIDE 2005) eingebunden werden, indem hierzu spezifische Zugriffskomponenten implementiert werden.

Eine Weiterentwicklung des Konzepts des WMS-Frontends zu einem komplexeren Workflow Service wurde bereits in Abschnitt 4.5.2 beschrieben. Die Eigenschaft des OX-Frameworks, verschiedene OGC Web Services anbinden zu können, befähigt zur Entwicklung von Applikationen, die mehrere Dienste unterschiedlichen Typs in Verkettungen einbeziehen.

Eine andere mögliche Weiterentwicklung des WMS-Frontends zu einem Kartendienst, der eine *loosely-coupled* Beziehung mit der SOS-Instanz eingeht wurde in Abschnitt 4.5.3 vorgestellt. Dieses Konzept eines *O&M Portrayal Service* gestattet die Visualisierung von Geosensordaten beliebiger SOS-Instanzen und die Auswahl der zu verwendenden Visualisierungsmethode durch den Client.

7 Zusammenfassung

Die Technologien der Sensor Web Enablement-Initiative ebnen den Weg für die Integration der von Sensornetzwerken aufgenommenen raumzeitlichen Daten in Geodateninfrastrukturen. Die Web Service Spezifikation des Sensor Observation Service sowie das Datenschema der Observations & Measurements Spezifikation gestatten eine interoperable Abfrage und Verbreitung dieser Geodaten. Die Auswertung zur Informationsextraktion der von den Sensoren erfassten Daten kann mit Hilfe von Visualisierungstechniken erfolgen. Sie erlauben dem Nutzer eventuelle Trends, Muster, Zusammenhänge oder Anomalien in den Daten zu entdecken.

Im Rahmen dieser Arbeit wird daher eine Systemarchitektur entwickelt, auf deren Basis vielfältige Visualisierungsmethoden für Geosensordaten realisiert werden können. Bevor das System eine Visualisierung ausführen kann, kommt es zum Zugriff auf den Web Service und zur Integration der abgefragten Daten. Die Architektur beinhaltet dazu Datenmodelle, die auf verschiedenen OGC Spezifikationen basieren, und die Integration der Geodaten ermöglichen. Die Aufgaben des *Zugriffs*, der *Integration* sowie der *Visualisierung* werden in separaten Anbindungskomponenten gekapselt, deren Implementierungen über einen Plugin-Mechanismus zur Laufzeit hinzugefügt werden können. Diese Anbindungskomponenten können nicht nur für den SOS, sondern für beliebige OGC Web Services entwickelt werden. Somit ist die Architektur nicht auf die Implementierung von Visualisierungsmethoden für die vom SOS bereitgestellten Geosensordaten beschränkt und erlaubt die visuelle Überlagerung mit unterstützenden Geodaten, die dem Nutzer die Interpretation der Geosensordaten erleichtern.
Damit die einmal implementierten Anbindungskomponenten und insbesondere die Visualisierungskomponenten in unterschiedlichen Anwendungen wiederverwendet werden können, wird die Systemarchitektur als Software-Framework konzipiert, welches die zentralen Geschäftsmethoden zur Visualisierung von Geodaten bereitstellt.

Anhand der prototypischen Implementierung einer Client- sowie einer Server-Applikation wird demonstriert, wie das Framework auf der Ebene der Geschäftslogik in verschiedenen Anwendungen eingesetzt werden kann. Die Applikationen realisieren dabei die Präsentations-Schicht des Systems, um die Darstellung erzeugter Visualisierungen zu ermöglichen.
Konkrete Anbindungskomponenten werden für die Spezifikationen des SOS sowie des WMS implementiert. Exemplarische Realisierungen von Visualisierungskomponenten für die vom SOS angebotenen und im O&M-Format kodierten Geosensordaten zeigen, dass das Framework die Generierung von Visualisierungen zulässt, die Sach-, Raum- und Zeitbezug der Geosensordaten einbeziehen und den Nutzer zur Exploration der Daten befähigen.

Indem zahlreiche Möglichkeiten zur Weiterentwicklung der Architektur sowie der implementierten Anbindungskomponenten und Applikationen skizziert werden, wird im Abschluss der Arbeit die vielfältige Verwendbarkeit und Erweiterbarkeit des Frameworks aufgezeigt.

Literaturverzeichnis

ANDRIENKO, G.L. & N.V. ANDRIENKO (1999): *Interactive maps for visual data exploration*. In: Geographical Information Science, 1999, 13(4). S. 355-374.

APACHE SOFTWARE FOUNDATION (2006): *Apache Tomcat*. Online unter: http://tomcat.apache.org/. (abgerufen am 27.01.2007).

APACHE SOFTWARE FOUNDATION (2007): *XMLBeans Overview*. Online unter: http://xmlbeans.apache.org/overview.html. (abgerufen am 27.01.2007).

BALZERT, H. (2001): *Lehrbuch der Software-Technik. Software-Entwicklung.* 2. Auflage. Heidelberg. Spektrum Akademischer Verlag.

BERNARD, L., I. SIMONIS & A. WYTZISK (2002): *Monitoring und Simulation raumzeitlicher Prozesse in Geodateninfrastrukturen.* In: Schubert, Reusch & Jesse (Hrsg.) (2002): Informatik bewegt: Informatik 2002 - 32. Jahrestagung der Gesellschaft für Informatik. Dortmund. S. 756-761.

BIRRER, A., W.R. BISCHOFBERGER & T. EGGENSCHWILER (1995): *Wiederverwendung durch Frameworktechnik - vom Mythos zur Realität*. In: Objektspektrum, 1995, 5. S. 18-26.

BOTTS, M. (2006): *Sensor Model Language (SensorML)*. OpenGIS® (Draft) Implementation Specification. Open Geospatial Consortium: 05-086r2, Version: 1.0.

BOTTS, M., G. PERCIVALL, C. REED & J. DAVIDSON (2006a): *OGC® Sensor Web Enablement: Overview And High Level Architecture.* OGC White Paper. Open Geospatial Consortium: 06-046r2, Version: 2.0.

BOTTS, M., A. ROBIN, J. DAVIDSON & I. SIMONIS (2006b): *OpenGIS® Sensor Web Enablement Architecture Document.* OGC Discussion Paper. Open Geospatial Consortium: 06-021r1, Version: 1.0.

BOOCH, G., J. RUMBAUGH, I. JACOBSON (2005): *Unified Modeling Language User Guide*. Amsterdam. Addison-Wesley Longman.

BLOK, C. (2005): *Dynamic visualization variables in animation to support monitoring of spatial phenomena*. Utrecht, Enschede. Universiteit Utrecht, ITC (= Netherlands Geographical Studies, 328).

BRÖRING, A.H., T. FÖRSTER & I. SIMONIS (2006): *An Integrated Software Framework for OGC Web Services*. Online unter: http://www.foss4g2006.org-/contributionDisplay.py?contribId=134&sessionId=37&confId=1. (abgerufen am: 11.02.2007).

BUEHLER, K. & L. MCKEE (1998): *The OpenGIS® Guide. Introduction to Interoperable Geoprocessing and the OpenGIS Specification*. Wayland. Open Geospatial Consortium.

BURROUGH, P.A. & R.A. MCDONNELL (1998): *Principles of Geographical Information Systems. Spatial Information Systems and Geostatistics*. New York, USA. Oxford University Press.

COX, S., P. DAISEY, R. LAKE, C. PORTELE & A. WHITESIDE (2004): *OpenGIS® Geography Markup Language (GML) Implementation Specification*. OpenGIS® Recommendation Paper. Open Geospatial Consortium: 03-105r1, Version: 3.1.0.

COX, S. (2006): *Observations and Measurements.* OGC Discussion Paper. Open Geospatial Consortium: 05-087r3, Version: 0.13.0.

DALY M. & T. KRALIDIS (2005): *OGC Web Services Context Documents (OWS Context) Interoperability Experiment: Final Report*. OGC™ Interoperability Experiment Report. Open Geospatial Consortium: 05-062, Version: 0.0.3.

DELIN, K. (2005): *Sensor Webs in the Wild*. In: N. Bulusu & S. Jha, (Hrsg.) (2005): Wireless Sensor Networks: A Systems Perspective. Artech House.

DE LA BEAUJARDIERE, J. (2006): *OpenGIS® Web Map Server Implementation Specification*. OGC Implementation Specification. Open Geospatial Consortium: 06-042, Version: 1.3.0.

DIBIASE, D. (1990): *Visualization in the Earth Sciences.* In: Earth and Mineral Sciences, Bulletin of the College of Earth and Mineral Sciences, 59(2). S. 13-18.

DRANSCH, D. (1997): *Computer-Animation in der Kartographie. Theorie und Praxis*. Berlin. Springer.

FAYAD, M., D.C. SCHMIDT & R.E. JOHNSON (1999): *Building application frameworks. object-oriented foundations of framework design*. New York, USA. Wiley.

FOWLER, M., D. RICE & M. FOEMMEL (2003): *Patterns für Enterprise Application-Architekturen*. Bonn. mitp-Verlag.

GAHEGAN, M. (2001): *Visual Exploration in geography. Analysis with light.* In: H.J. Miller and J. Han (Hrsg.) (2001). Geographic Data Mining and Knowledge Discovery. London. Taylor & Francis. S. 260-287.

GAMMA, E., R. HELM, R. JOHNSON & J. VLISSIDES (1995): *Design Patterns. Elements of Reusable Object-Oriented Software*. Reading, USA. Addison-Wesley.

HAVENS, S. (2006): *Transducer Markup Language.* OpenGIS® (Draft) Implementation Specification. Open Geospatial Consortium: 06-010r6.

HEILER, S. & P. MICHELS (1994): *Deskriptive und explorative Datenanalyse*. München. Oldenbourg.

HINNEBURG, A., D. KEIM & M. WAWRYNIUK (1999): *HD-Eye: Visual Mining of High-Dimensional Data*. In: Computer Graphics and Applications, 19(5). S. 22-31.

HUNTER, J. & W. CRAWFORD (2001): *Java Servlet Programming.* O'Reilly.

IEEE (2007): *About the IEEE*. Online unter: http://www.ieee.org/web/aboutus/home-/index.html. (abgerufen am: 10.02.2007).

IETF (2004): *RFC 3935. A Mission Statement for the IETF*. Online unter: http://www.ietf.org/rfc/rfc3935.txt. (abgerufen am: 10.02.2007).

ISO (2002): *ISO 19108:2002. Geographic information - Temporal schema.* ISO TC211.

ISO (2003): *ISO 19107:2003. Geographic information - Spatial schema.* ISO TC211.

ISO (2004): *ISO 8601:2004. Data elements and interchange formats - Information interchange: Representation of dates and times*. ISO TC154.

KEIM, D. A. (2002): *Datenvisualisierung und Data Mining*. In: Datenbank-Spektrum, 2002, 2. S. 30-39.

KOTTMAN, C. (1999): *The OpenGIS™ Abstract Specification - Topic 5: Features*. Open Geospatial Consortium: 99-105r2, Version: 4.

KOTTMAN, C. & C. ROSWELL (2000): *The OpenGIS® Abstract Specification - Topic 6: The Coverage Type and its Subtypes*. Open Geospatial Consortium: 00-106, Version: 4.

KRAAK, M.J. (2002): *Current trends in visualisation of geospatial data with special reference to cartography*. In: Indian Cartographer, 2002(22). S. 319-324.

KRAAK, M.J. & F.J. ORMELING (1997): *Cartography. Visualization of spatial data*. Harlow. Addison Wesley Longman Limited.

KRAAK, M.J., A. SLIWINSKI & A. WYTZISK (2005): *What happens at 52N? An Open source approach to education and research*. In: Proceedings of the Joint ICA Commissions Seminar "Internet-Based Cartographic Teaching and Learning: Atlases, Map Use, and Visual Analytics" (6.-8. Juli 2005) im Rahmen der 22. ICC Conference: Mapping Approaches into a Changing World. La Coruna, Spanien. S. 16-20.

KRAMER, D., B. JOY & D. SPENHOFF (1996): *The Java Platform. A White Paper*. Online unter: http://java.sun.com/docs/white/platform-/javaplatformTOC.doc.html. (abgerufen am: 27.01.2007).

LALONDE, W. (2002): *Styled Layer Descriptor Implementation Specification*. OpenGIS® Implementation Specification. Open Geospatial Consortium: 02-070, Version: 1.0.0.

LANSING, J. (2002): *OWS1 Coverage Portrayal Service*. OpenGIS© OGC Interoperability Program Report-Engineering Specification. Open Geospatial Consortium: 02-019r1, Version: 0.0.2.

LIANG, S.H.L. & V. TAO (2005): *Design of an integrated OGC spatial sensor web client*. Online unter: http://sensorweb.geoict.net/Assets/publications-/LiangTao2005Geoinformatics_final.pdf. (abgerufen am: 27.01.2007).

LONGLEY, P.A., M.F. GOODCHILD, D.J. MAGUIRE & D.W. RHIND (2001): *Geographic Information. Systems and Science*. New York, USA. John Wiley & Sons.

LOY, M., R. ECKSTEIN, D. WOOD, J. ELLIOTT & B. COLE (2002): *Java Swing. Developing GUIs in Java*. O'Reilly.

MACEACHREN, A.M. (1995): *How Maps Work. Representation, Visualization, and Design*. New York, USA. Guilford Press.

MACEACHREN, A.M. & M.J. KRAAK (1997): *Exploratory Cartographic Visualization: Advancing the Agenda*. In: Computers & Geosciences, 23(4). S. 335-343.

MACEACHREN, A.M., F.P. BOSCOE, D. HAUG & L.W. PICKLE (1998): *Geographic Visualization: Designing Manipulable Maps for Exploring Temporally Varying Georeferenced Statistics*. In: Proceedings of the IEEE Symposium on Information Visualization (18.-19. Oktober 1998). New York, USA. S. 87-94.

MACEACHREN, A.M. & M.J. KRAAK (2001): *Research Challenges in Geovisualization*. In: Cartography and Geographic Information Systems, 28(1). S. 3-12.

MCCARTHY, T. (2003): *Sensor Collection Service.* OpenGIS Interoperability Program Report. Open Geospatial Consortium: 03-023r1, Version: 0.9.0.

MCFERREN, G., S. ROOS & A. TERHORST (2006): *Fire Alerts on the Geospatial Semantic Web*. Online unter: http://www.ordnancesurvey.co.uk/partnerships/research-/research/TerraCognita_Papers_Presentations/McFerrenRoosTerhorst.pdf. (abgerufen am 06.03.2007).

MENG, L. (2003): *Rahmenbedingungen beim Einsatz von Methoden und Techniken der Geovisualisierung*. In: Kartographische Schriften, 7: Visualisierung und Erschließung von Geodaten. Bonn. S. 3-11.

MÜLLER, M. (2006a): *Styled Layer Descriptor profile of the Web Map Service Implementation Specification.* OpenGIS® Implementation Specification. Open Geospatial Consortium: 05-078r2, Version: 1.1.0.

MÜLLER, M. (2006b): *Symbology Encoding Implementation Specification*. OpenGIS® Implementation Specification. Open Geospatial Consortium: 05-077r4, Version: 1.1.0.

NA, A. & M. PRIEST (2006): *OpenGIS® Sensor Observation Service Implementation Specification*. OpenGIS® (Draft) Implementation Specification. Open Geospatial Consortium: 06-009r1, Version: 0.1.5.

NEBERT, D. & A. WHITESIDE (2005): *OGC™ Catalogue Services Specification.* Open Geospatial Consortium: 04-021r3, Version: 2.0.0.

OBJECT REFINERY (2006): *JFreeChart*. Online unter: http://www.jfree.org/jfreechart/. (abgerufen am 27.01.2007).

OGC (2006): *FAQs - OGC and "Openness"*. Online unter: http:-//www.opengeospatial.org/resource/faq/openness/#2. (abgerufen am 10.02.2007).

PERCIVALL, G. (2002): *The OpenGIS Abstract Specification – Topic 12: OpenGIS Service Architecture*. Open Geospatial Consortium: 02-112, Version: 4.3.

PERCIVALL, G. (2003): *OGC Reference Model*. Open Geospatial Consortium: 03-040, Version: 0.1.3.

POLASEK, W. (1994): *Explorative Datenanalyse. Einführung in die deskriptive Statistik*. Berlin. Springer.

REED, C. (2005): *The OGC Abstract Specification - Topic 0: Abstract Specification Overview*. Open Geospatial Consortium: 04-084, Version: 5.

REYNOLDS, G. (2005). *GO-1 Application Objects*. OGC Implementation Specification. Open Geospatial Consortium: 03-064r10, Version: 1.0.

SCHERP, A. & S. BOLL (2006): *Framework-Entwurf*. In: R. Reussner & W. Hasselbring (Hrsg.) (2006): Handbuch der Software-Architektur. Heidelberg. dpunkt. S. 395-417.

SCHUMANN, H. & W. MÜLLER (2000): *Visualisierung. Grundlagen und allgemeine Methoden*. Berlin. Springer.

SCHUT, P. & A. WHITESIDE (2005): *OpenGIS® Web Processing Service*. OpenGIS® Discussion Paper. Open Geospatial Consortium: 05-007r4, Version: 0.4.0.

SKONNARD, A. & M. GUDGIN (2001): *Essential XML Quick Reference. A Programmer's Reference to XML, XPath, XSLT, XML-Schema, SOAP, and More.* Amsterdam. Addison-Wesley Longman.

SIMONIS, I. (2004): *Sensor Webs: A Roadmap*. In: Proceedings of the 1st Goettinger GI and Remote Sensing Days. Goettingen.

SIMONIS, I. (2005): *OpenGIS® Sensor Planning Service Implementation Specification*. OpenGIS® Implementation Specification. Open Geospatial Consortium: 05-089r3, Version: 0.0.30.

SIMONIS, I. (2006a): *OGC® Sensor Alert Service Implementation Specification*. OpenGIS® (Draft) Implementation Specification. Open Geospatial Consortium: 06-028r3, Version: 1.0.0.

SIMONIS, I. (2006b): *Integration raumbezogenen Wissens in interoperable Geodiensteinfrastrukturen.* Verlag Natur & Wissenschaft. Solingen. (= IfGIprints, 26).

SIMONIS, I., U. STREIT & A. WYTZISK (2003): *Integrating simulation models into SDI's*. In: Conference proceedings: AGILE. Lyon, Frankreich.

SIMONIS, I. & J. ECHTERHOFF (2006): *Draft OpenGIS® Web Notification Service Implementation Specification.* OpenGIS® Best Practices Paper. Open Geospatial Consortium: 06-095, Version: 0.0.9.

SLIWINSKI, A., I. SIMONIS, A. REMKE, U. STREIT & A. WYTZISK (2005): *Boosting the OGC Sensor Web Enablement Initiative by Open Source Web Services – The Case of 52°North*. In: J. Strobl, T. Blaschke & G. Griesebner (Hrsg.) (2005): Angewandte Geoinformatik 2005 - Beiträge zum 17. AGIT Symposium. Salzburg, Österreich. Wichmann Verlag. S. 686 - 695.

SONNET, J. (2005): *Web Map Context Documents*. OGC™ Implementation Specification. Open Geospatial Consortium: 05-005, Version: 1.1.0.

STREIT, U. (2004): *Einführung in die Geoinformatik. Kapitel 4: Geoobjekte und ihre Modellierung*. Online unter: http://ifgivor.uni-muenster.de/vorlesungen-/Geoinformatik. (abgerufen am: 12.02.2007).

SUN MICROSYSTEMS (1999): *Programming in Java Advanced Imaging*. Online unter: http://java.sun.com/products/java-media/jai/forDevelopers/jai1_0_1guide-unc/. (abgerufen am: 27.01.2007).

SUN MICROSYSTEMS (2007): *J2SE 5.0*. Online unter: http://java.sun.com/j2se/1.5/. (abgerufen am: 27.01.2007).

TILLMAN, S. & J. GARNETT (2006): *OWS Integrated Client Architecture, Design, and Experience*. OGC Discussion Paper. Open Geospatial Consortium: 05-116, Version: 0.0.3.

THOMAS, S. (2006): *Alternativen zu AJAX*. In: Java Spektrum, 2006(6). S. 16-18.

VAN DER VLIST, E. (2002): *XML-Schema. The W3C's Object-Oriented Descriptions for XML*. Beijing. O'Reilly.

VASILIEV, I.R. (1997): *Mapping Time*. In: Cartographica, 34(2). S. 1-51.

VRETANOS, P.A. (2005a): *OpenGIS® Filter Encoding Implementation Specification*. OpenGIS® Implementation Specification. Open Geospatial Consortium: 04-095, Version: 1.1.0.

VRETANOS, P.A. (2005b): *Web Feature Service Implementation Specification*. OpenGIS® Implementation Specification. Open Geospatial Consortium: 04-094, Version: 1.1.0.

W3C (2007): *About the World Wide Web Consortium (W3C)*. Online unter: http://www.w3.org/Consortium/. (abgerufen am: 10.02.2007).

WHITESIDE, A., M.U. MÜLLER, S. FELLAH & F. WARMERDAM (2005): *Web Coordinate Transformation Service (WCTS) draft Implementation Specification*. OGC™ Discussion Paper. Open Geospatial Consortium: 05-013, Version: 0.3.0.

WHITESIDE, A. (2006): *OGC Web Services Common Specification.* OGC Implementation Specification. Open Geospatial Consortium: 06-121r2.

WHITESIDE, A. & J.D. EVANS (2006): *Web Coverage Service (WCS) Implementation-Specification*. OpenGIS® Implementation Specification. Open Geospatial Consortium: 06-083r8, Version: 1.1.0.

WOODWARD, B. & A. WHITESIDE (2005): *Feature Portrayal Service*. OpenGIS® Discussion Paper. Open Geospatial Consortium: 05-110, Version: 0.0.30.

WYTZISK, A. (2003): *Interoperable Geoinformations- und Simulationsdienste auf Basis internationaler Standards*. Verlag Natur & Wissenschaft. Solingen. (= IfGIprints, 20).

CD-Anhang

Pfad: *DA.pdf*	PDF-Version dieser Diplomarbeit.
Pfad: *Development\Doc*	Mit Java-Doc generierte Quellcode-Dokumentation des OX-Frameworks sowie der implementierten Applikationen.
Pfad: *Development\Source*	Quellcode und zugehörige Bibliotheken für die Applikationen, den Core sowie das Service-Connector Subsystem.
Pfad: *Distribution\Client-Application*	Ausführbarer Prototyp der Client-Applikation. Die Applikation kann über die Datei *client-application.bat* gestartet werden. Lediglich eine Java Runtime Environment muss vorinstalliert sein.
Pfad: *Distribution\WMS-Frontend*	Enthält das WMS-Frontend, welches durch Kopieren in das */webapps*-Verzeichnis eines Apache Tomcat Servlet Containers zur Ausführung gebracht werden kann.

Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit in vollem Umfang, einschließlich der beigefügten Abbildungen, von mir angefertigt worden ist und keine anderen als die angegebenen Hilfsmittel verwandt wurden. Alle Stellen, die dem Wortlaut oder dem Sinn nach aus anderen Werken stammen, sind unter Angabe der Quelle kenntlich gemacht.

Münster, den 13. März 2007

Arne Henrik Bröring