Hochschule Neubrandenburg
University of Applied Sciences

Hochschule Neubrandenburg

Studiengang Geoinformatik

Entkopplung des profil GIS-Viewers vom verwendeten OGC-Geoframework am Beispiel von deegree und GeoTools

Bachelorarbeit

vorgelegt von: Erik Schmölter

Zum Erlangen des akademischen Grades

„Bachelor of Engineering“ (B.Eng.)

Erstprüfer: Prof. Dr.-Ing. Andreas Wehrenpfennig

Zweitprüfer: B. Eng. Frank Czarnikow

Eingereicht am 27.08.2012

URN: urn:nbn:de:gbv:519-thesis 2012-0635-1

Kurzfassung

Der GIS-Viewer ist ein wichtiger und zukunftsträchtiger Bestandteil des Softwareproduktes **profil c/s**, da er für die Visualisierung verschiedenster Geodaten eingesetzt werden kann und damit ein besseres Verständnis von komplexen Sachverhalten gibt. Jedoch hat der bisherige Entwicklungsstand des GIS-Viewers viele direkte Abhängigkeiten zu dem Geoframework deegree und lässt sich deshalb nur mit großem Aufwand bei Veränderungen eben dieses Frameworks anpassen. Dies führt zu einer ineffizienten Wartung des GIS-Viewers und verlangt, dass der Entwickler bei der Integration des GIS-Viewers Kenntnisse über das verwendete Geoframework hat.

Diese Bachelorarbeit untersucht deshalb, wie der GIS-Viewer von einem konkreten Geoframework entkoppelt werden kann. Dazu werden bestehende Abhängigkeiten analysiert und es wird aufgezeigt, wie mithilfe von Entkopplungsmechanismen vermeidbare Abhängigkeiten gelöst werden können. Weiterhin wird ein neues Konzept für die Nutzung von Geoframeworks im GIS-Viewer vorgestellt und für die verschiedenen Anwendungsfälle werden Lösungen präsentiert. Dies wird mithilfe einer prototypischen Umsetzung dargestellt.

Bei der Weiterentwicklung des GIS-Viewer können die gewonnen Kenntnisse direkt verwendet werden, um neue Funktionalitäten unabhängiger von Geoframeworks zu integrieren.

Abstract

The GIS-Viewer is an important and promising component of the software product **profil c/s** as it can be used for the visualization of different data and therefore grants a better understanding of complex issues. However, the current state of development of the GIS-Viewer has many dependencies on the deegree geoframework and changes can only be implemented with great efforts. This leads to an inefficient maintenance of the GIS-Viewer and requires a developer with knowledge about the used geoframework when integrating the GIS-Viewer.

Therefore this bachelor thesis examines how the GIS-Viewer can be decoupled of a concrete Geoframework. For this purpose, existing dependencies are analyzed and it is shown how to use decoupling mechanisms to resolve avoidable dependencies. A new approach to the use of geoframeworks for the GIS-Viewer is presented und solutions for the different use cases are introduced. This part is illustrated by using a prototype implementation.

The gained knowledge can be directly used to integrate new functionalities more independently from geoframeworks in the further development of the GIS-Viewer.

Inhaltsverzeichnis

1. Einleitung

1.1. Hinführung zum Thema

Geografische Informationssysteme (GIS) haben aktuell in vielen Bereichen eine zunehmende Bedeutung und werden wie auch in **profil c/s**, einem Softwaresystem der data experts gmbh (deg) zu Visualisierung verschiedener Geodaten, eingesetzt. Das Programmsystem zur Unterstützung der Förderung in der Landwirtschaft, oder eben kurz profil, hat die Aufgabe, bei Fördermittelverwaltung in den Bereichen der Landwirtschaft, Forstwirtschaft und des Naturschutzes zu unterstützen. Aufgrund dieses Einsatzes ist auch die Einbringung eines GIS-Viewers sinnvoll, da dieser dem Nutzer zum Beispiel eine anschauliche Möglichkeit bietet, um Antragsstellerparzellen zu verwalten.

Wie auch andere Bestandteile von **profil c/s** wird auch der GIS-Viewer immer komplexer. Dieser Umstand führt dazu, dass die Kopplung zu den verwendeten Frameworks zunimmt. Kopplung meint dabei die Art und den Grad für die gegenseitigen Abhängigkeiten zwischen diesen Softwaremodulen. [IEEE90] Die Verknüpfungen zwischen zwei Modulen können dabei auf unterschiedliche Weise entstehen und lassen sich damit in Gruppen unterteilen[1]:

- pathological coupling: Das Modul verlässt sich auf die konkrete inhaltliche Umsetzung eines anderen Moduls beziehungsweise modifiziert die internen Daten des anderen. [wikKop]
- content coupling: Ein Modul inkludiert den Inhalt eines anderen oder Teile davon. [wikKop]
- common-environment coupling: Zwei Module kommunizieren über gemeinsame globale Daten. [wikKop]
- hybrid coupling: In diesem Fall ist ein Parameter gleichzeitig Datum und Kontrollflussinformation. Beispielsweise könnte ein Parameter, der auf null gesetzt ist, bedeuten, dass die gegebene Information unbekannt ist. Hierbei muss zuerst herausgefunden werden, welcher Fall gegeben ist und erst hinterher kann der Parameter verwendet werden. [wikKop]
- control coupling: Zwei Module kommunizieren über eine komplexe Datenstruktur, von der aber nur ein Teil verwendet wird. [wikKop]
- data coupling: Zwei Module kommunizieren über elementare Daten, beispielsweise einfache Parameter. [wikKop]

Ziel jedes Softwareprojektes ist eine möglichst lose Kopplung. Das heißt, wenige Abhängigkeiten zwischen Modulen zu erzeugen und das Vermeiden einiger Abhängigkeitsarten anzustreben (z.B. hybrid coupling).

1.2. Motivation und Zielstellung

Ein grundlegendes Konzept von **profil c/s** ist das Bausteinkonzept, nach dem die Komponenten auf der Ebene der fachlichen Architektur in Bausteine unterteilt werden. Dies ermöglicht einen modularen Aufbau von **profil c/s**.

So ist ein Baustein die kleinste auslieferbare und installierbare Einheit in **profil c/s**. In ihm sind alle Werkzeuge, Algorithmen, Services, Vorlagen sowie Datenbank-Skripte eines fachlichen Komplexes enthalten.

[1] Hier wird die Unterteilung nach [IEEE90] vom Institute of Electrical and Electronics Engineers aufgezeigt. Eine weitere Beschreibung der Kopplungsarten ist zum Beispiel durch [Gle74] gegeben.

Das Ausgangsproblem hierbei ist, dass nicht nur der GIS-Viewer selbst, sondern auch viele weitere Bausteine, die diesen als Tool integrieren, direkt vom Framework deegree abhängig sind. Zu Problemen kommt es durch diese Konstellation, wenn das verwendete Geoframework ausgetauscht oder aktualisiert werden soll. Denn in diesem Fall müssen alle betroffenen Stellen im Code angefasst werden, sodass großer Aufwand für diese Veränderungen nötig ist. Somit erschwert diese Ausgangssituation die Wartung und Erweiterbarkeit der betroffenen Bausteine.

Dazu muss der Entwickler, der den GIS-Viewer in einen Baustein integriert, durch die Abhängigkeiten Kenntnisse über das Geoframework besitzen. Dies ist auch unerwünscht, da der Entwickler sich auf die Bearbeitung der eigentlichen Aufgabe konzentrieren soll.

Zur Lösung dieses Problems soll im Rahmen dieser Bachelorarbeit aufgezeigt werden, welche Mechanismen für eine Entkopplung von einem speziellen Geoframework eingesetzt werden können und wie eine allgemeine Lösung zur Integration beliebiger Geoframeworks aussehen kann. Das Hauptaugenmerk liegt dabei auf dem GIS-Viewer von **profil c/s**, da in diesem die wesentlichen Anwendungsfälle, die an ein Geoframework gestellt werden, vorliegen.

Die Ergebnisse dieser Arbeit sollen als Leitfaden für zukünftige Entkopplungen in **profil c/s** dienen. So geben Analyse und Konzeption einen Überblick zu den Situationen, die auftreten können, und zeigen auf, welche Lösungen für das jeweilige Problem am besten geeignet sind. In der Umsetzung stehen dann prototypische Beispiele zur Verfügung, mit denen dargestellt wird, wie eine konkrete Lösung aussehen kann. Außerdem wird anhand der Bewertung zur Erfüllung funktionaler und nicht-funktionaler Anforderungen dargelegt, welche Verbesserungen oder gegebenenfalls auch Nachteile durch das genutzte Verfahren zustande kommen.

1.3. Aufbau der Arbeit

Im zweiten Kapitel werden zunächst die Grundlagen für die Analyse erläutert, damit auftretende Begrifflichkeiten geklärt sind und die Grundideen der betrachteten Geoframeworks bekannt sind. Außerdem wird ein bereits vorhandener Lösungsansatz für das genannte Problem vorgestellt und seine Relevanz für die konkrete Situation dargestellt. Im Rahmen dieser Betrachtung wird ebenfalls kurz auf Standards und Normen eingegangen.

Im dritten Kapitel wird die Analyse des Ist-Zustandes zu Beginn der Arbeit aufgezeigt. So wird das Vorgehen bei der Analyse selbst beschrieben, ein kurzer Überblick zu **profil c/s** und dem GIS-Viewer gegeben und dann die Kopplung zum Geoframework dargelegt. Zum Abschluss der Analyse werden Anforderungen an ein Geoframework formuliert, die als Grundlage für die Entwicklung von Konzepten dienen.

Das vierte Kapitel gibt die entwickelten Konzepte wieder. Dazu wird ein Entwurf für die allgemeine Verwendung von Geoframeworks vorgestellt und dann anhand von exemplarischen Beispielen aufgezeigt, wie Lösungen im Detail gedacht sind. Für diesen Abschnitt werden besonders markante Fälle für das Vorgehen bei der Entkopplung verwendet.

Nach der Konzeption folgt das fünfte Kapitel mit der Umsetzung einiger Lösungsentwürfe. Dazu wird jeweils kurz das Vorgehen beschrieben und auf auftretende Probleme oder Besonderheiten eingegangen. Außerdem wird eine Beurteilung der umgesetzten Lösung im Rahmen der gestellten Anforderungen vorgenommen.

Diese Arbeit endet mit einer Zusammenfassung, in der noch einmal kurz die Ergebnisse beurteilt werden sollen. Weiterhin wird ein Ausblick darauf gegeben, welche Möglichkeiten die Entkopplung für zukünftige Entwicklungen am GIS-Viewer bietet und welche allgemeinen Schlüsse gezogen werden können.

2. Grundlagen

Vor der Analyse des Ist-Zustandes sollen zunächst wichtige, theoretische Grundlagen betrachtet werden. So ist die Zusammenarbeit des GIS-Viewers mit verschiedenen Geoframeworks keine Selbstverständlichkeit und nur möglich, weil es für viele Bereiche der Geoinformatik ausformulierte Standards und Normen gibt. Diese sollen in diesem Kapitel kurz erläutert werden, um dann aufzuzeigen, dass es für das Problem dieser Arbeit bereits Ansätze und Bemühungen gibt. Nach der Vorstellung dieser standardisierten Lösung wird dann auch aufgezeigt, warum diese im Rahmen dieser Arbeit nicht relevant ist.

Außerdem wird dargestellt, wo sich deegree und GeoTools einordnen, was die beiden Projekte ausmacht und in welchen Punkten sie voneinander abweichen. Durch das gewonnene Verständnis aus dieser Einführung kann dann im darauf folgenden Kapitel der Fokus komplett auf der Analyse des GIS-Viewers und der Kopplung liegen.

2.1. Standards und Normen

Nach [Bil01] ist ein Standard eine breit akzeptierte und angewandte Regel oder Norm, entweder als offizielle Norm oder Normungsarbeit hervorgegangen oder als de-facto-Standard durch seine Verbreitung gesetzt.

In vielen Bereichen werden Standards genutzt, weil sie die Flexibilität, die Funktionalität und Produktivität erhöhen. Dies gilt besonders für Informationssysteme (IS), für die eine Kommunikation untereinander ohne Standards nicht möglich wäre. [Bil01]

Für Geodaten spielen Standards eine besondere Rolle. So gibt es Bedarf für Geodaten in vielen Einrichtungen (z.B. Versorgungsunternehmen zur Verwaltung ihrer Netze oder zur Verkehrsnavigation). Diese wiederum werden von unterschiedlichen Anbietern beliefert und aufgrund dieser Situation liegen Daten mit verschiedenen Erfassungszielen, Erfassungsmethoden, Datengenauigkeiten und Datenformaten vor, sodass ein gemeinsamer Standard wie zum Beispiel in Form von Metadaten unabdingbar ist. [deL02]

2.1.1. International Organization for Standardization und Open Geospatial Consortium

Im Bereich Standardisierung von GIS sind die International Organization for Standardization (ISO) und das Geospatial Consortium (OGC) die beiden wichtigsten Organisationen, die in diesem Gebiet tätig sind.

Gegründet wurde die ISO 1945 und seitdem koordiniert sie die Normung weltweit und ist Mutterorganisation für die meisten nationalen Normungsinstitute. Auf europäischer Ebene gehören dazu zum Beispiel das Comité Européen de Normalisation (CEN) und das Deutsche Institut für Normung (DIN). [Bil01] Die eigentliche Erstellung von Normen übernehmen die Technical Comittees (TC) und speziell in Bezug auf Geodaten das TC 211 Geographic information/Geomatics. Das TC211 wurde 1994 gebildet und beschäftigt sich seitdem mit der Erstellung von Standards, die einen Bezug zum Raum oder genauer einem Ort auf der Erde haben. [IsoTc211]

Im selben Jahr wie das TC 211 wurde auch die OGC mit etwa 130 Mitgliedern gegründet. Darunter befanden sich verschiedene GIS-Anbieter, Dienstleister, DB- und IT-Firmen Datenlieferanten und Universitäten. [Bil01] Ziel der OGC ist, dass Entwickler und Nutzer alle im Netz verfügbaren Geodaten und zugehörigeren Dienste verwenden können. Um dies zu erreichen, werden verschiedene

Spezifikationen (z.B. Open GIS) vorgegeben. Um Geodaten zur Verfügung zu stellen, definiert die OGC verschiedene WebServices. Dies sind zum Beispiel:

- Web Map Service (WMS) zur Bereitstellung von Kartenabbildungen
- Web Feature Service (WFS) zur Bereitstellung von Vektordaten
- Web Coverage Service (WCS) zur Bereitstellung von Rasterdaten
- Catalogue Service (CSW) zur Bereitstellung von Metadaten
- Web Processing Service(WPS) für räumliche Analysen

Diese und andere Standards werden dann von der OGC als Implementierungsspezifikationen veröffentlicht, die dann in der Regel durch verschiedene Projekte umgesetzt werden.

2.1.2. Vorhandene Ansätze einer Domänenschicht nach den Spezifikationen der OGC

Eine gemeinsame Domänenschicht für verschiedene Projekte im Bereich Geodatenverarbeitung zu haben, ist keine neue Idee und wird teilweise mit dem ISO- und OGC-konformen Projekt GeoAPI realisiert, das verschiedene Schnittstellen für GIS-Applikationen liefert. Der durch die OGC festgelegte „GeoAPI 3.0 Implementation Standard“ definiert die zugehörige Bibliothek und alle darin enthaltenden Strukturen und Funktionen und liefert für Entwickler ein komplett dokumentiertes Modell, das unabhängig von konkreten Implementierungen ist. [OgcGApi]

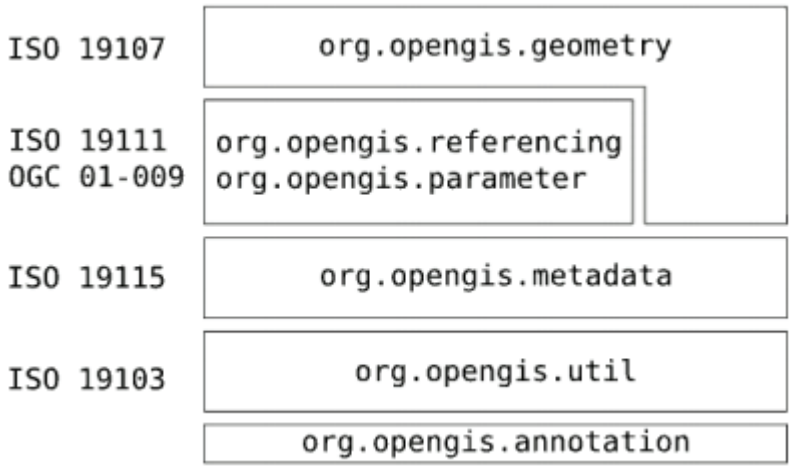

Abbildung 1 ISO-Spezifikationen und Package-Struktur von GeoAPI [OgcGApi]

Wie in der Abbildung 1 zu sehen, sind die Bestandteile von GeoAPI nach den betreffenden ISO-Spezifikationen gruppiert und in Packages abgebildet. Mit dem Projekt werden also folgende Bereiche im Umgang mit Geodaten geregelt:

- ISO 19103: Geographic Information - Conceptual Schema Language
- ISO 19107: Geographic Information - Spatial Schema
- ISO 19115: Geographic Information – Metadata
- ISO 19111: Geographic Information - Spatial Referencing by Coordinates
- OGC 01-009: OpenGIS Implementation Specification - Coordinate Transformation Services

Vorteil beim Einsatz von GeoAPI ist, dass Projekte sich gegenseitig ergänzen können und schon vorhandene Komponenten nicht erneut implementiert werden müssen. Sollten trotzdem mehrere Implementierungen für ein Interface vorliegen, kann der Entwickler diese nach seinem Belieben benutzen und ohne großen Aufwand austauschen. Zusätzlich kommt es zu einer Absicherung

gegenüber Aktualisierungen von verwendeten Projekten, denn solange diese weiterhin den Standard beibehalten, muss der Entwickler keine Veränderungen vornehmen. Nur Neuerungen der Spezifikationen selbst, also Anpassungen von GeoAPI, würden Wartungsarbeiten erforderlich machen.

Zu den Projekten, die Abschnitte von GeoAPI implementieren, gehören zum Beispiel:

- Geotk[2]
- Proj4[3]
- NetCDF[4]

Die Projektarbeit an GeoAPI wird auch aktuell weitergeführt und die regelmäßigen Veränderungen belegen, dass ein allgemeines Interesse an einer solchen Schnittstelle existiert und das Thema für den GIS-Bereich von Relevanz ist.

Das Problem des profil GIS-Viewers wird jedoch mit diesem Standard nicht gelöst, da deegree keine Implementierung der GeoAPI bietet und auch nicht von dieser beeinflusst wird. So ist zwar GeoTools bis zur Version 2.7 eine Implementierung des GeoAPI 2.0 Standards und wird noch danach von diesem beeinflusst, aber dies ändert auf Grund der Unabhängigkeit von deegree nichts an der Ausgangslage. Beide Projekte sind nicht an die GeoAPI 3.0 gebunden und es muss eine eigene Lösung gefunden werden, um die bereits erwähnten Ziele zu erreichen.

2.2. OSGeo Projects

Die Open Source Geospatial Foundation (OSGeo) ist eine gemeinnützige Organisation, die sich zum Ziel gesetzt hat, die Entwicklung von Open-Source-Software zur Verwaltung von Geodaten zu unterstützen und die Verbreitung dieser Projekte zu fördern. So erhalten Mitglieder durch die OSGeo finanzielle, organisatorische und rechtliche Unterstützung und können die verschiedenen Plattformen der Organisation zum Wissensaustausch nutzen. [Osg]

Die Projekte, die im Rahmen der OSGeo laufen, werden je nach ihren Diensten in verschiedene Kategorien unterteilt. Dabei wird folgende Unterscheidung vorgenommen:

Web Mapping

Zu dieser Kategorie gehören alle Projekte, die sich mit web-basierten Kartenanwendungen beschäftigen und Dienste verwenden, die mithilfe von OGC-Standards verschiedene Geodaten einheitlich zur Verfügung stellen.

Desktop Applications

Hierzu zählen Softwarelösungen, die dem Nutzer direkte Interaktion mit dem GIS über eine Desktopanwendung ermöglichen. So werden diese Systeme hauptsächlich zur Manipulation von großen Datenmengen und Durchführen umfangreicher Berechnungen eingesetzt. Die Nutzung web-basierter Dienste ist aber nicht ausgeschlossen.

[2] Siehe [GeoTk]
[3] Siehe [Proj4]
[4] Siehe [NetCdf]

Geospatial Libraries
Software-Bibliotheken bilden die Basis für andere Open-Source-Projekte im Rahmen der OSGeo, da sie Grundfunktionen zur Geodatenverarbeitung nach OGC-Spezifikationen bereitstellen.

Metadata Catalog
Mithilfe von Anwendungen dieser Kategorie können Ressourcen mit räumlichem Bezug verwaltet werden. Das heißt, das verschiedene Dienste zur Verwaltung von Geodaten miteinander verbunden und eingesetzt werden können. [OsgPIS]

Weitere Projekte
Da auch die Ausbildung von Fachkräften, die mit Software der OSGeo Projects umgehen können, im Interesse der Organisation liegt, beschäftigt sich das Projekt „Education and Curriculum“ damit, Bildungsstätten wie zum Beispiel Universitäten bei der Erstellung von Lehrplänen und Unterrichtsmaterial zu unterstützen. [OsgECP]

Da ein weiteres Ziel der OSGeo die freie Verfügbarkeit von Geodaten ist, gibt es auch für diesen Bereich ein Projekt. Unter „Public Geospatial Data“ fallen alle Projekte, die sich mit dem Nutzen von Geodatenformaten, dem Veröffentlichen staatlich gesammelter Daten, dem Erstellen von Datensammlungen und dem Fragen zur Lizenzen in Bezug auf Geodaten beschäftigen. [OsgGDC]

Eine Zusammenfassung aller Projekte der OSGeo steht mit OSGeo Live zur Verfügung. Dies ist ein auf Xubuntu basierendes System, das alle Technologien mit Beispieldaten und zugehörigen Dokumentationen enthält. Aufgrund dieser Ausgangslage für Entwickler und der durchgehenden Einhaltung von Standards sind die OSGeo Projects eine adäquate Grundlage, um eigene GIS-Anwendungen zu entwickeln.

2.2.1. deegree
deegree ist wie alle OSGeo Projects eine Open-Source-Software und hält sich an die vorgegebenen Standards der OGC und ISO. Das Framework kann zur Erstellung eigener Anwendungen verwendet werden. Dazu liefert deegree die wichtigsten Bausteine für den Aufbau einer Geodaten-Infrastruktur. [FosFG] Dabei steht dem Entwickler frei, ob er eine Web- oder Desktopanwendung anfertigen will und in welchem Maße er die umfangreiche API sowie damit verbundene Dienste zum Einsatz bringen will. So gibt es zum Beispiel Implementierungen für den Web Map Service, Web Feature Service, Web Coverage Service, Catalogue Service und Web Processing Service. Dies gewährt die Entwicklung von Software nach spezifischen Anforderungen.

Entwickelt wurde deegree komplett in der Programmiersprache Java und kann damit in jedem System zum Einsatz kommen, das über eine Java-Laufzeitumgebung verfügt. Dabei spielt es keine Rolle, ob deegree auf Seite des Clients oder Servers genutzt wird, da es Komponenten für beide Anwendungsfälle gibt.

2.2.2. GeoTools
GeoTools ist eine Bibliothek, welche Standard-konforme Methoden für die Bearbeitung von Geodaten bereitstellt, um zum Beispiel eine GIS-Anwendung zu implementieren. [GeoTls] GeoTools ist als Projekt der OSGeo Projects ebenfalls eine Open-Source-Software und wurde in der Programmiersprache Java entwickelt.

Zu den wichtigsten Komponenten der Bibliothek gehören verschiedene Bereiche, die für GIS-Anwendungen relevant sind. So werden durch GeoTools verschiedene Interfaces für Datenstrukturen und räumliche Konzepte vorgegeben. Außerdem gibt es eine umfangreiche API für den Umgang mit Fachobjekten, die Erstellung von Karten und Handhabung von XML sowie GML.

Im Grunde ähneln sich deegree und GeoTools sehr stark. So ist deegree zwar etwas breiter ausgelegt und bietet daher eine größere Vielfalt an Funktionalitäten, dafür geht GeoTools teilweise mehr ins Detail und liefert umfangreichere Funktionalitäten für spezielle Aufgabengebiete. Die Nutzung dieser beiden Geoframeworks ist also gerade wegen der vorhandenen Ähnlichkeit sinnvoll.

3. Analyse

3.1. Beschreibung des Vorgehens bei der Analyse

Da sich der GIS-Viewer wie alle Komponenten von **profil c/s** in einem ständigen Entwicklungsprozess befindet, wurde für die Projektarbeit im Rahmen der Bachelorarbeit ein Stand (Revision 319767) festgelegt, damit Analyse, Entwicklung von Konzepten und Umsetzung nicht ständig an andere Programmänderungen angepasst werden müssen. Da Projekte in **profil c/s** jedoch mit SVN verwaltet werden, kann der aktuelle Stand des Projektes stets mit den durchgeführten Arbeiten zusammengeführt werden.

Zu Beginn der Analyse beschränkte sich die Untersuchung auf Details, da interne Abläufe des GIS-Viewers und ein Gesamtüberblick zunächst nicht entscheidend waren. So wurde auf Klassenebene bestimmt, welche Abhängigkeiten vorliegen und zu welchem Zweck Komponenten von deegree eingesetzt werden. Auf diesem Wege ließen sich die Kommunikation und andere Beziehungen zwischen Objekten beschreiben.

Im nächsten Schritt wurde eine Generalisierung vorgenommen und Komponenten vom GIS-Viewer und deegree betrachtet. So ergaben sich bestimmende Komponenten im GIS-Viewer, wie zum Beispiel die Karte oder die Attributtabelle, die an verschiedenen Prozessen beteiligt sind. Für Bestandteile von deegree hingegen konnte dieser Part mithilfe der Dokumentation und des zugehörigen Wikis gelöst werden. Diese Analyse führte zu allgemeineren Abläufen und erlaubte eine Abstraktion von physischen zu logischen Prozessen.

Auf der nächsten Abstraktionsstufe wurden der GIS-Viewer und seine Beziehung unabhängig von der bisherigen Implementierung und Nutzung von deegree untersucht. So wurde betrachtet, welche Funktionalitäten von deegree in **profil c/s** genutzt werden und in welchen Anwendungsfällen diese zum Einsatz kommen. Anhand dieser Erkenntnisse ergab sich schließlich ein Gesamtüberblick und das relativ komplexe Gefüge von Abhängigkeiten konnte auf wenige Kernprobleme beschränkt werden.

Zuletzt wurden mit den gewonnenen Kenntnissen funktionale und nicht-funktionale Anforderungen formuliert, die beim Anpassen der bestehenden Implementierung eingehalten werden sollen. Mithilfe dieser Vorgaben lässt sich dann zum Beispiel feststellen, welche Lösungsansätze am besten geeignet sind und in welchem Maße mit Verbesserungen oder Einbußen zu rechnen ist.

3.2. profil GIS-Viewer

3.2.1. Überblick

Der GIS-Viewer ist ein allgemeines Werkzeug zum Visualisieren von Geodaten. Er bildet die Grundlage für konkrete GIS-Viewer, welche diesen gegebenenfalls um spezifische Funktionalität erweitern. Für die Arbeit im GIS-Viewer werden grundlegende Navigations-, Selektions- und Informationswerkzeuge bereitgestellt. Weiterhin besteht die Möglichkeit, OGC-konforme WebServices einzubinden. [degDkG]

Technische Grundlagen

Ein Prinzip, das von **profil c/s** genutzt wird, ist der Werkzeug & Material-Ansatz[5] (WAM) in seiner Java-spezifischen Ausprägung JWAM. So setzen sich Werkzeuge entsprechend den Konventionen des

[5] Siehe [Zül12]

Frameworks in **profil c/s** aus Functional Part (FP), Interactional Part (IP) und GUI-Klasse zusammen. Dazu gibt es für jedes Werkzeug eine Tool-Klasse, die diese drei Bestandteile erzeugt und zusammenhält. Der FP kennt als einziger Teil des Werkzeugs das Material und kann Daten aus dem Material lesen sowie Änderungen am Material vornehmen. Der IP meldet Kommandos bei den GUI-Komponenten an und führt diese bei entsprechenden GUI-Events aus. Dabei wird in der Regel eine Methode des FP aufgerufen. Somit steuert der IP die GUI. Die GUI selbst ist ein JFrame, das die benötigten GUI-Komponenten enthält.

Der GIS-Viewer gehört jedoch zu einer besonderen Art von Werkzeugen, die nur aus Tool- und GUI-Klasse bestehen. FP und IP sind dabei in der Toolklasse enthalten. Diese Art von Werkzeugen wird als MonoTool bezeichnet.

Um den GIS-Viewer in **profil c/s** (z.B. im Rahmen einer Antragsmappe) zu integrieren, führt der Entwickler die gleichen Schritte wie bei anderen Werkzeugen durch und muss als einzige Besonderheit auf die Implementierung des Callback-Interfaces [6] HatGisViewer achten. Einen Überblick zur Konfiguration des Interfaces liefert die folgende Abbildung:

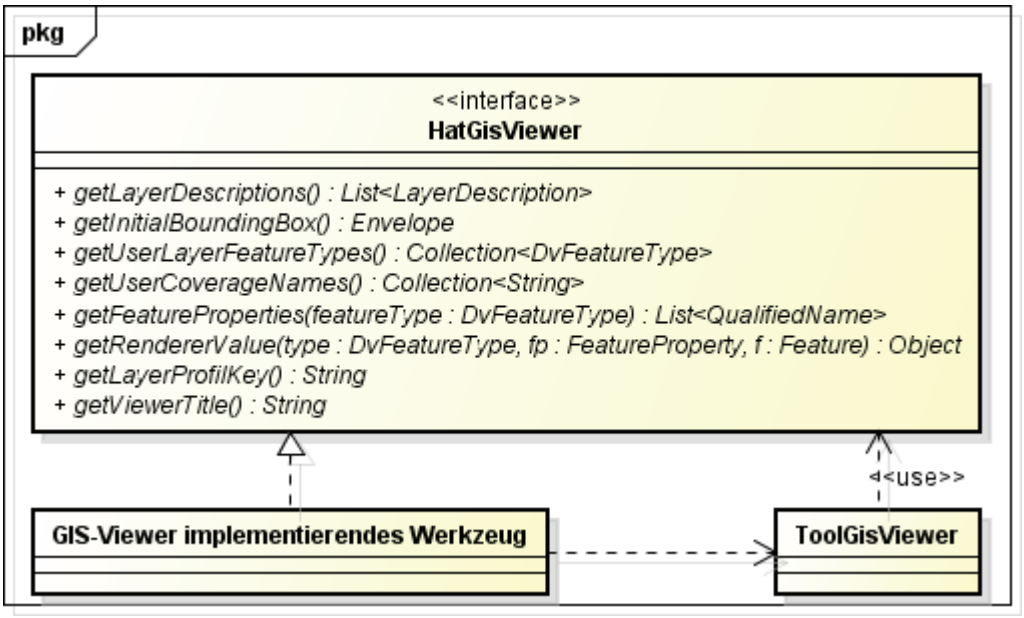

Abbildung 2 Klassendiagramm: Integration des GIS-Viewers

Durch die Nutzung dieses Interfaces kann der GIS-Viewer vor seinem ersten Aufruf mit verschiedenen Konfigurationen ausgestattet werden, die teilweise erforderlich oder auch optional sind. So müssen Titel und der anfängliche Kartenbereich auf jeden Fall angeben werden, damit der GIS-Viewer mit einer Default-Ansicht gestartet werden kann. Hingegen muss die Angabe von vordefinierten oder Nutzer-spezifischen Layern, die Vorgabe von Filtern oder die Bestimmung und Formatierung von Sachdaten nicht zwingend erfolgen, um den GIS-Viewer einzusetzen.

Die folgende Abbildung zeigt noch einmal die wesentlichen Schritte bei der Integration des GIS-Viewers:

[6] Eine Callback-Funktion bezeichnet in der Informatik eine Funktion, die einer anderen Funktion als Parameter übergeben und von dieser unter gewissen Bedingungen aufgerufen wird. [wikRrf] Da Java dieses Verfahren nicht direkt unterstützt, nutzt man stattdessen Interfaces zur Realisierung solche Aufgaben.

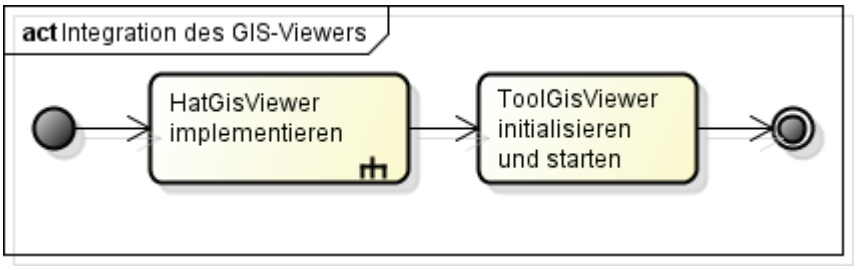

Abbildung 3 Aktivitätsdiagramm: Integration des GIS-Viewers

Da die Teilkomponente das Callback-Interface HatGisViewer implementieren muss, werden in folgender Abbildung die dazu nötigen Schritte dargestellt:

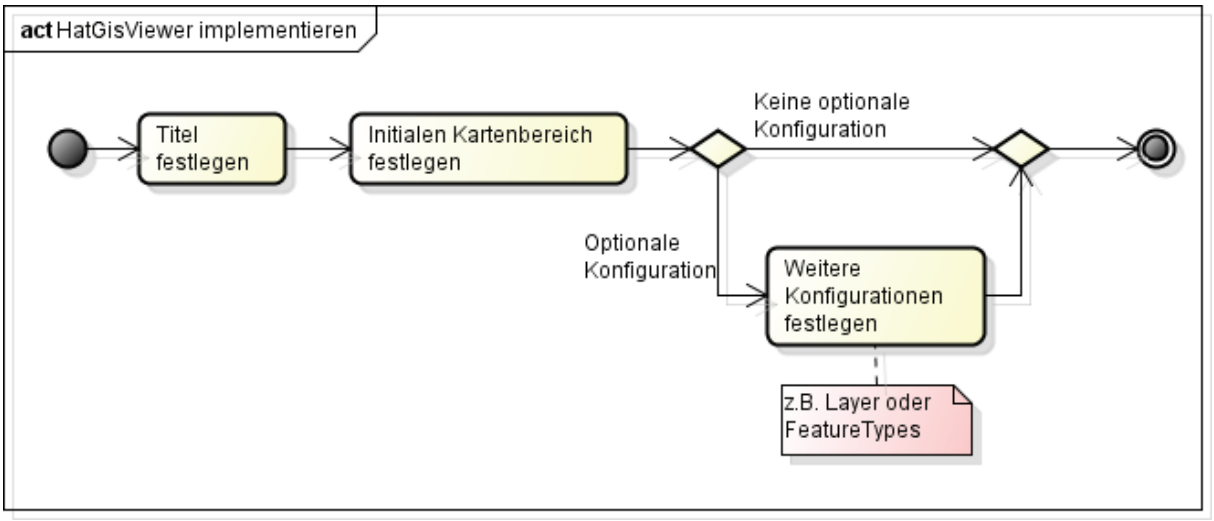

Abbildung 4 Aktivitätsdiagramm: Nutzung des Callback Interfaces HatGisViewer

Größter Vorteil dieser Vorgehensweise ist, dass der Entwickler kein eigenes Tool implementieren muss und trotzdem den GIS-Viewer nach seinen Vorstellungen anpassen kann.

GUI des GIS-Viewers

Die grundlegenden Komponenten des GIS-Viewers sind die Werkzeugleiste, die Legende, die Karte sowie die Attributtabelle. Im oberen Fensterbereich befindet sich die Werkzeugleiste, über welche ein Großteil der Werkzeuge zugreifbar ist. Innerhalb der Legende sowie der Layer-spezifischen Kontextmenüs befinden sich weitere, Ebenen-spezifische Werkzeuge, welche nicht über die Werkzeugleiste aufgerufen werden können. [degDkG]

Die Legende des GIS-Viewers wird in einer Baumstruktur abgebildet. Jeder Layer der Karte wird als Kind-Knoten des Wurzelknotens „Layer“ dargestellt. Vektorlayer haben – im Gegensatz zu Rasterlayern – zusätzlich einen Kind-Knoten, welcher die Stilinformationen beinhaltet. Die Anzahl der Stil-Informationen ist abhängig von den möglichen Geometrietypen des Vektorlayers sowie von gegebenenfalls vordefinierten Filtereinstellungen. [degDkG]

Die Anordnung der Layer ist mit der Karte synchronisiert. Das heißt, die Reihenfolge der Layer in der Legende entspricht stets der Abbildungshierarchie in der Karte. Zusätzlich werden mithilfe von Symbolen die Art (vordefinierter oder eigener Layer) und der Zustand der einzelnen Layer angezeigt.

Die Karte und Attributtabelle werden bei der Selektion ebenfalls synchronisiert. Das heißt, wird in der Karte die Geometrie eines Objektes ausgewählt, wird in der Attributtabelle die entsprechende Zeile mit den Sachdaten des Objektes selektiert. Auch der umgekehrte Weg, wenn in der Attributtabelle eine Sachdaten-Zeile selektiert wird, funktioniert und als Ergebnis wird die zugehörige Objekt-Geometrie in der Karte selektiert.

Die Attributtabelle repräsentiert die Sachdaten von Objekten aus dem in der Karte dargestellten Layer. Dabei wird für jedes sichtbare Objekt des selektierten Layers eine eigene Zeile angelegt, in der die spezifischen Sachdaten enthalten sind. Welche Attribute eines Objektes in der Tabelle angezeigt werden, wird dabei in der Regel durch das Werkzeug bestimmt, aus dem der GIS-Viewer gestartet wird oder in dem er eingebettet ist. [degDkG]

In der folgenden Abbildung ist zu sehen, wie die GUI im Allgemeinen aufgebaut ist und wie die einzelnen Komponenten im GIS-Viewer platziert sind:

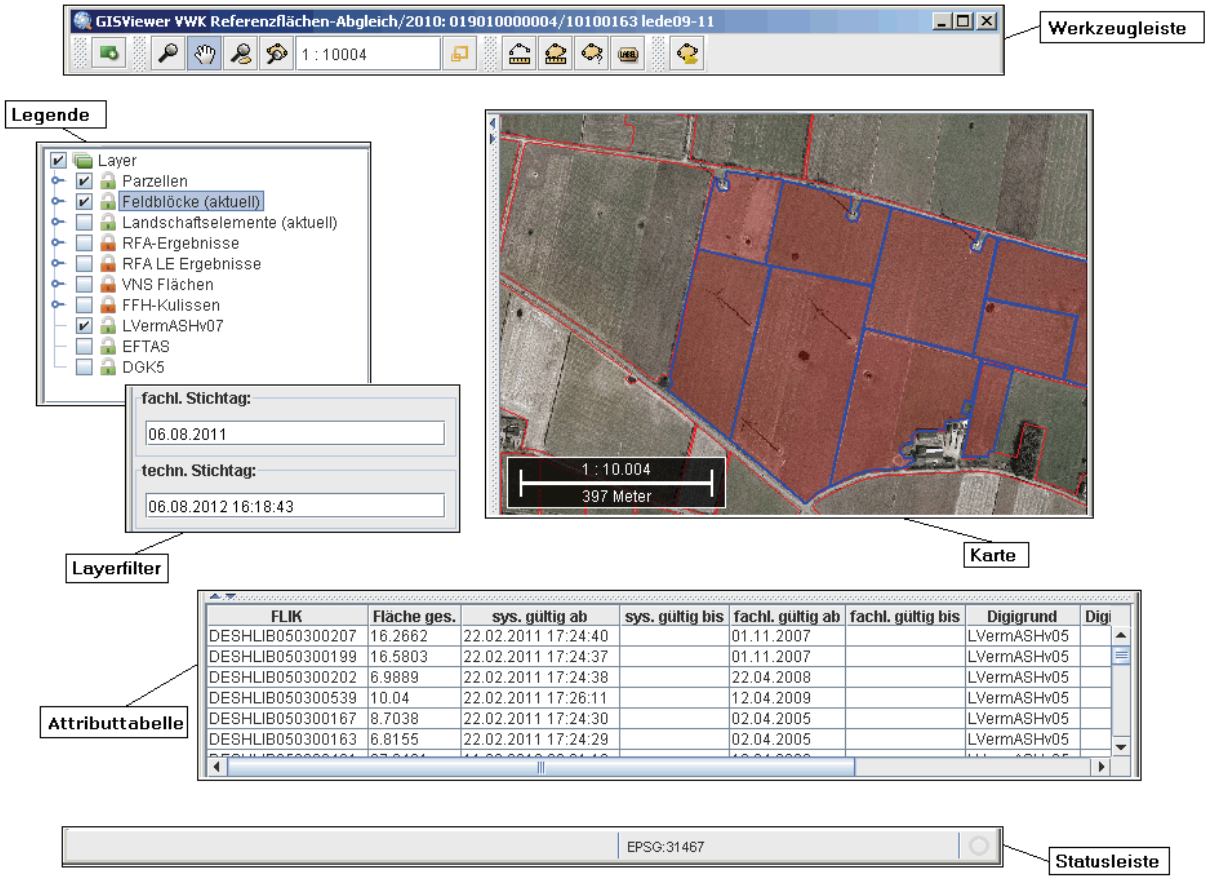

Abbildung 5 GUI des GIS-Viewers

3.2.2. Funktionsweise

Der wichtigste Vorgang im GIS-Viewer ist die Bereitstellung von Informationen in Form einer grafischen Darstellung von Geodaten auf der Karte und derer Sachdaten innerhalb der Attributtabelle. Zwar gibt es Möglichkeiten, um die Darstellungsformen zu beeinflussen (z.B. Zoomen oder Verändern eines Styles), Elemente ein- und auszublenden oder weitere Werkzeuge einzusetzen, aber diese Vorgänge sind zweitrangig und beeinflussen nur das Ergebnis des Hauptprozesses. Dieser soll im folgenden Abschnitt kurz erläutert werden.

So steht am Anfang des Prozesses die Eingabe des Nutzers oder die Konfiguration im Tool, die festlegt, was für Daten in einem Layer dargestellt werden sollen. Durch diese Vorgaben kann der GIS-Viewer eine Anfrage mithilfe eines Geoframeworks an einen Service weitergeben (z.B. ein WFS), der mit einer Datenquelle verbunden ist und das Ergebnis bestimmt. Das Resultat, was vom Service zurückgeschickt wird, ist GML.

Der erste Bestandteil, die Geography Markup Language oder kurz GML, ist eine XML Grammatik zur Beschreibung und zum Austausch von Objekten mit Raumbezug. [OgcGML] Ein Element in GML besteht immer aus Geometrie, Koordinatensystem und den Sachdaten. Damit ist eine komplette Beschreibung der Kartenelemente gegeben.

Der Styled Layer Descriptor oder kurz SLD liefert hingegen die Symbolisierung von Raster- und Vektordaten, bestimmt so die Darstellungsform für Kartenelemente und wird zum Bespiel eingesetzt, um Objekte vom WFS zu stylen. [OgcSLD]

Der GIS-Viewer übersetzt nun mithilfe des Geoframeworks diese Nachricht in ein darstellbares Element, das zum Abschluss in der Karte abgebildet wird. Neben den grafischen Informationen werden auch weitere Daten für die erhaltenen Objekte ausgewertet und können dann in der Attributtabelle dargestellt werden.

In Abbildung 6 ist der Prozess zusammengefasst. Der Feldblock in der Karte ist hierbei das Objekt, welches dargestellt werden soll und mithilfe der Anfrage durch den GIS-Viewer, der hinterlegten Daten beim Service und den Transformationsalgorithmen von GML und SLD zu einer Grafik erzeugt wird.

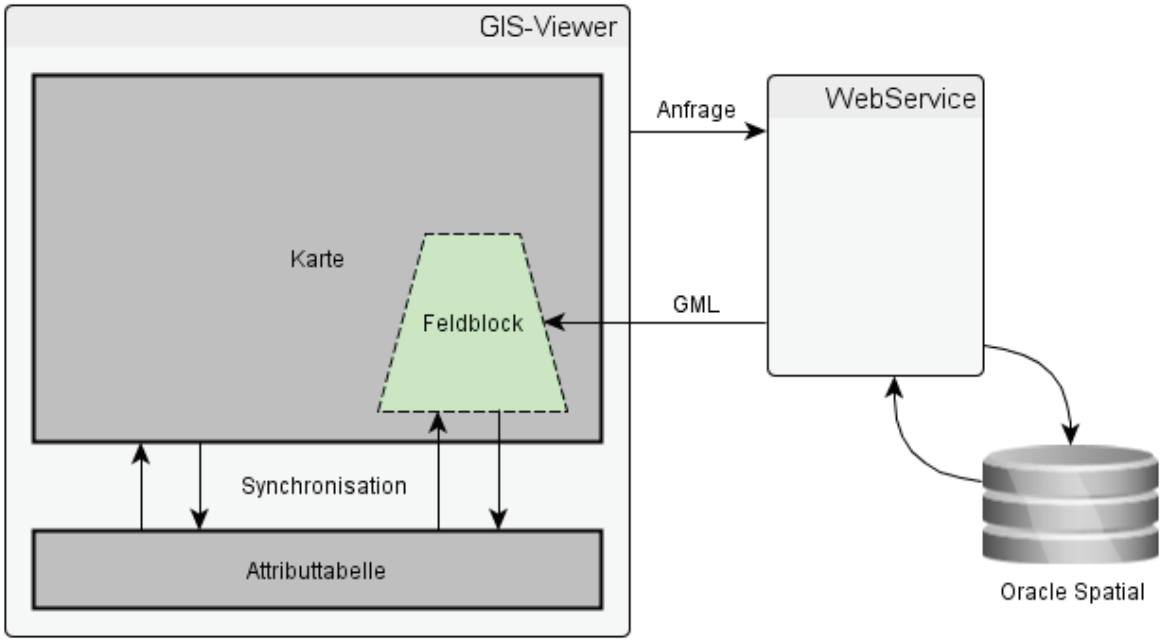

Abbildung 6 Funktionsweise GIS-Viewer

3.2.3. Integration von deegree

In diesem Abschnitt sollen die Abhängigkeiten des GIS-Viewers von deegree aufgezeigt werden. Dabei wird jedoch stets nur ein kleiner Bereich des gesamten Konstrukts betrachtet werden, da

dieses einfach zu groß und komplex ist, um es in seiner Gesamtheit abzubilden. Nachvollziehen lässt sich dies zum Beispiel, wenn nur die Importe im Baustein des GIS-Viewers betrachtet werden, denn hier finden sich, wie in Tabelle 1 zu sehen, 286 Abhängigkeiten zu deegree, wobei 131 verschiedene deegree-Klassen angesprochen werden. Insgesamt ist dies zwar nur ein geringer Anteil der gesamten API, die deegree zur Verfügung stellt, aber es offenbart trotzdem das Ausmaß der Kopplung.

	DE.data_experts.profi.gisviewer
org.deegree.graphics	96
org.deegree.ogcbase	3
org.deegree.ogcwebservices	13
org.deegree.model	155
org.deegree.framework	2
org.deegree.datatypes	17
Summe:	**286**

Tabelle 1 Abhängigkeiten zu deegree

Abhängigkeiten auf Klassenebene

Als erstes Beispiel für Abhängigkeiten auf Klassenebene wird das Interface HatGisViewer betrachtet. Wie in Kapitel 3.2.1. beschrieben, dient dieses Interface zur Konfiguration des GIS-Viewers und ist damit ein wichtiges Element des GIS-Viewers. Welche Abhängigkeiten konkret bestehen, ist in folgender Abbildung zu sehen:

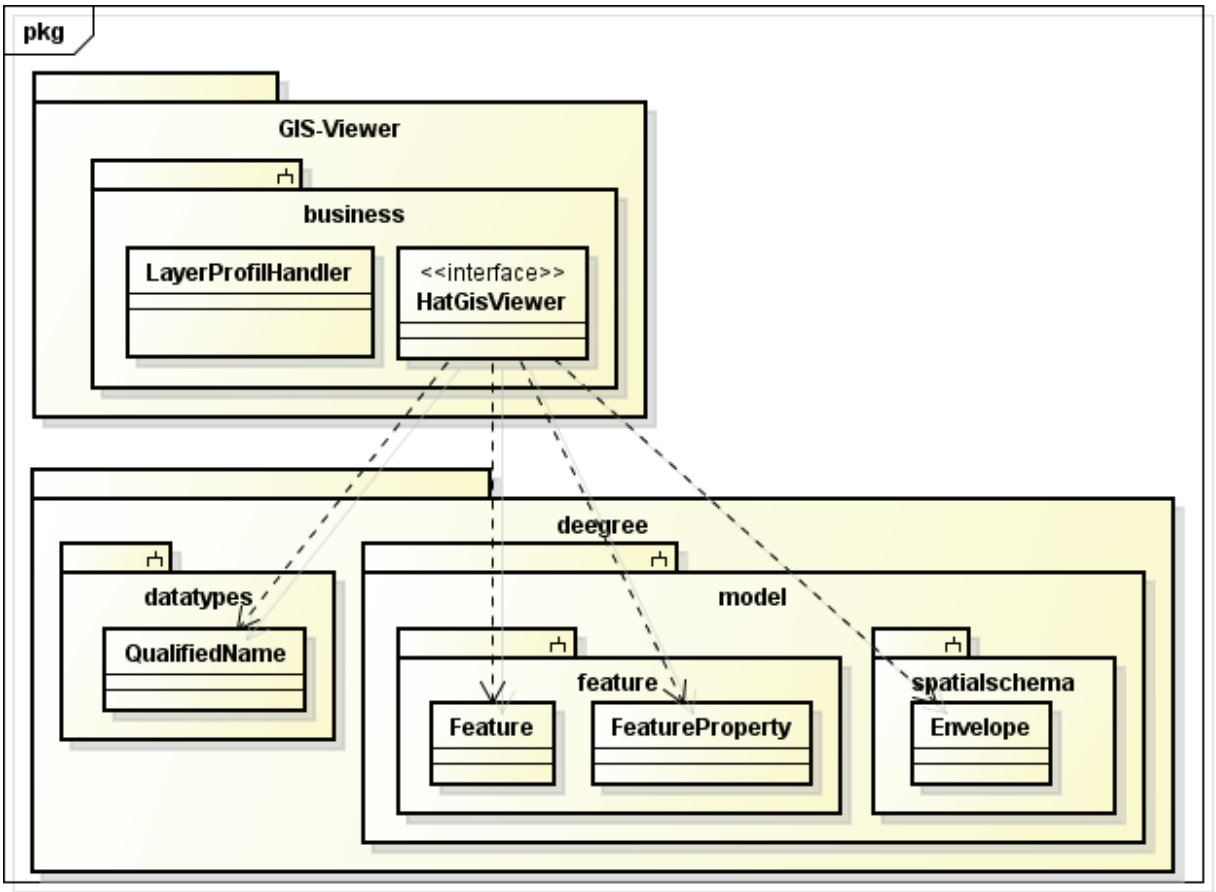

Abbildung 7 Klassendiagramm: Abhängigkeiten von HatGisViewer zu deegree

Die zusehenden Abhängigkeiten werden in erster Linie dazu gebraucht, die Konfiguration des GIS-Viewers festzulegen und sind deshalb bedeutende Bestandteile bei der Arbeit mit deegree.

Weiterhin bringen sie zusätzliche Abhängigkeiten bei der Integration mit sich, da für die Erstellung ihrer Instanzen andere Strukturen von deegree (z.B. Factorys) nötig sind. Dies wiederum sind dann jene Bestandteile, mit denen sich auch der Entwickler bei der Integration des GIS-Viewers befassen muss.

Als nächstes Beispiel für die Abhängigkeiten auf Klassenebene wird FeatureTableModel betrachtet, das für den Zugriff und die Verwaltung der Attributtabelle zuständig ist. Genauer werden nur die Methoden getValueAt und setValueAt untersucht, die für das Ein- und Auslesen des Wertes einer einzelnen Zelle der Tabelle eingesetzt werden.

In folgender Abbildung ist zunächst das Sequenzdiagramm für den Aufruf der Methode getValueAt zusehen:

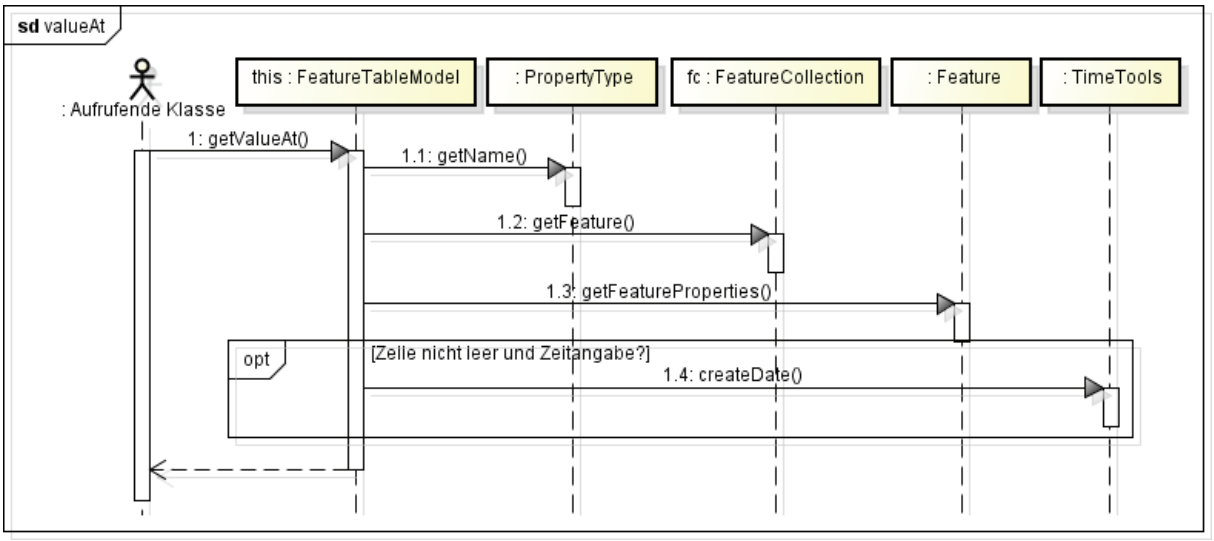

Abbildung 8 Sequenzdiagramm: FeatureTableModel - getValueAt

Für den Aufruf dieser Methode sind also verschiedene deegree-Klassen nötig, um damit bestimmte Elemente zu verwalten. Feature, FeatureCollection und PropertyType sind in diesem Fall die Klassen, welche die abgefragten Daten enthalten bzw. genauer spezifizieren. TimeTools wird hier hingegen eingesetzt, um das Ergebnis zu formatieren, wenn es sich bei dem Wert der ausgewählten Zelle um eine Zeitangabe handelt. Diese Klasse stellt also kein Objekt dar, das vom GIS-Viewer genutzt wird.

Die einzelnen Abhängigkeiten von deegree sind an dieser Stelle also eng verknüpft und eine Entkopplung wird nur sinnvoll sein, wenn die Grundfunktionalität entkoppelt wird und nicht jede Abhängigkeiten einzeln. Dieser Ansatz gilt auch für andere Stellen im GIS-Viewer, da dort eine ähnliche Konstellation vorliegt.

Zum Vergleich zeigt die nächste Abbildung das Sequenzdiagramm für die Methode setValueAt, die Unterschiede zu getValueAt aufweist:

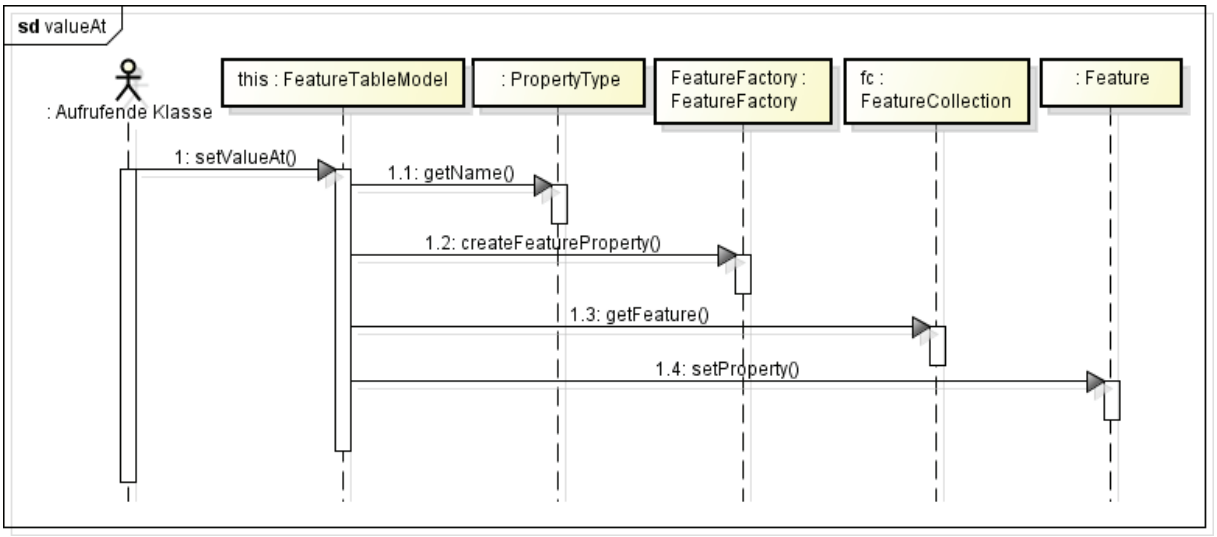

Abbildung 9 Sequenzdiagramm: FeatureTableModel - setValueAt

In dieser Methode wird TimeTools nicht mehr benötigt, da in die Eingabe nur korrekt formatierte Zeichenketten gelangen können. Jedoch wird jetzt eine Factory, nämlich die FeatureFactory, gebraucht, damit aus der erfolgten Eingabe ein Feature erzeugt werden kann. Dabei ist uninteressant, um welche Art von Feature es sich handelt, da dieses bei Bedarf über den gesetzten Property-Wert ermittelt werden kann.

Im GIS-Viewer werden noch weitere Factorys verwendet, um zum Beispiel geometrische Objekte zu erzeugen. Dabei erfolgt immer nur ein einziger Aufruf der Factory und von der Vielzahl an zur Verfügung stehenden Funktionalitäten kommt nur ein kleiner Teil zur Anwendung. In diesen Fällen könnte also neben der Entkopplung eine Vereinfachung der Schnittstelle angestrebt werden, um für mehr Übersicht im System zu sorgen.

Eine weitere Form von Abhängigkeiten liegt vor, wenn eine Klasse des GIS-Viewers von einer Klasse aus deegree erbt. Ein Beispiel hierfür ist in folgender Abbildung zu sehen:

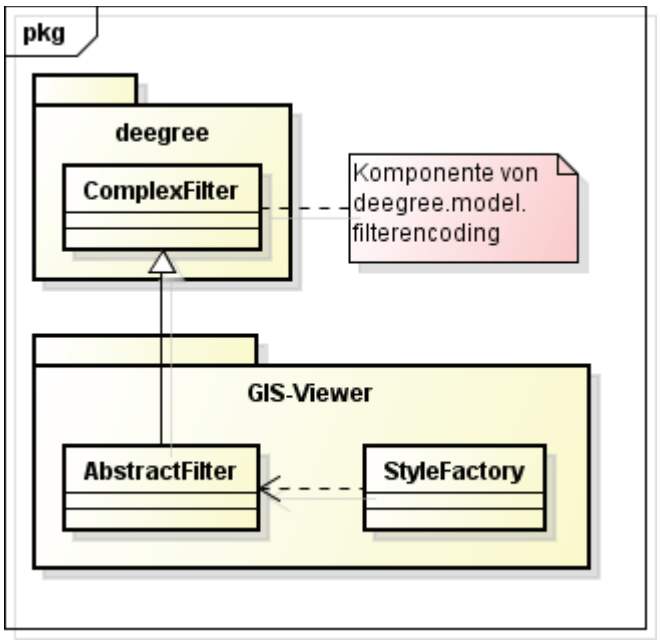

Abbildung 10 Klassendiagramm: Abhängigkeiten durch Vererbung

Filter sind Bestandteil der Definition eines Styles oder einer Gruppe von Styles und werden deshalb zu deren Erstellung benötigt. Da Styles im Ist-Zustand ebenfalls durch deegree-Komponenten gehandhabt werden, müssen dazu passende Filter verwendet werden. Aus diesem Grund erbt AbstractFilter im GIS-Viewer von der deegree-Klasse ComplexFilter, sodass GIS-Viewer-spezifische Konfigurationen bei der Filterung vorgenommen werden können. So hat zwar die StyleFactory dann keine Abhängigkeiten zu deegree bei Verwendung der Filter, trotzdem ist dann an anderer Stelle eine starke Kopplung durch die Vererbung gegeben.

Abhängigkeiten auf Ebene der Packages

Bei der Betrachtung der Kopplung zwischen GIS-Viewer und deegree eine Ebene höher zeigen sich die Abhängigkeiten zwischen Packages auf. Zwar ergeben sich auf dieser Stufe wegen der Verallgemeinerung weniger Beteiligte, jedoch bleibt die komplexe Vernetzung erhalten. Als Beispiel sind in folgender Abbildung die Abhängigkeiten des Packages model zu sehen, das ein Bestandteil der GUI beziehungsweise der Map des GIS-Viewers ist und die Layer verwaltet:

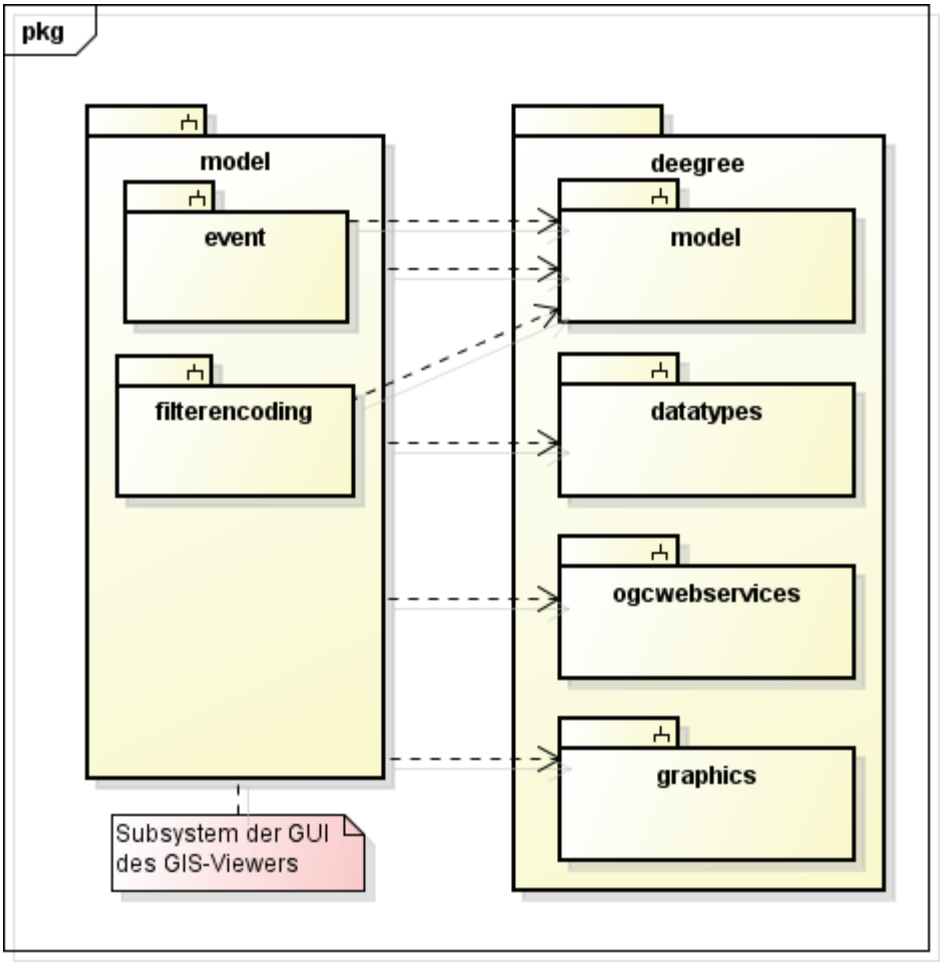

Abbildung 11 Paketdiagramm: Abhängigkeiten von gis_viewer.gui.map.model zu deegree

Wie in der Abbildung zu sehen, greifen alle Komponenten von model auf das gleichnamige Package in deegree zu. Das liegt daran, dass dort bereits Funktionalitäten liegen, die im GIS-Viewer nur noch erweitert oder spezialisiert werden. Weiterhin liegt im deegree-Package model das SpatialSchema, das die Geometrietypen von deegree kapselt und damit von zentraler Bedeutung bei der Nutzung von deegree ist. Bei den anderen Packages, zu denen in diesem Fall Abhängigkeiten bestehen, liegen ebenfalls Funktionalitäten vor, die beim Arbeiten mit Layern eine Rolle spielen. So gibt es im Package

ogcwebservie Strukturen für die Arbeit mit WCS und WFS, zu denen es im GIS-Viewer entsprechende Layer gibt.

Insgesamt ist der GIS-Viewer von folgenden Packages aus deegree abhängig:

- org.deegree.crs
- org.deegree.datatypes
- org.deegree.framework.util
- org.deegree.framework.xml
- org.deegree.graphics
- org.deegree.graphics.displayelements
- org.deegree.graphics.legend
- org.deegree.graphics.sld
- org.deegree.graphics.transformation
- org.deegree.io
- org.deegree.model.coverage
- org.deegree.model.crs
- org.deegree.model.feature
- org.deegree.model.filterencoding
- org.deegree.model.spatialschema
- org.deegree.ogcbase
- org.deegree.ogcwebservices.wcs
- org.deegree.ogcwebservices.wfs

Allgemein kann festgehalten werden, dass Abhängigkeiten je nach Problemstellung auftreten und nur wenige Ausnahmen, wie zum Beispiel QualifiedName, immer wieder vorkommen. Dies beruht jedoch auf der Tatsache, dass solche Klassen nicht für GIS-Viewer-interne Vorgänge gebraucht werden, sondern zum Aufruf vieler deegree-Komponenten nötig sind. Es gibt im Gegensatz dazu auch Abhängigkeiten, die nur einmal existieren, da in diesen Fällen eine spezielle Funktionalität gebraucht wird. Beispiel hierfür ist die Verwendung von TimeTools zur Formatierung von Zeitangaben.

Weiterhin fällt auf, dass bei deegree-Klassen vor allem Geometrie und Features im Vordergrund stehen und diese komplett durch deegree im GIS-Viewer realisiert sind. Es zeigt sich also, dass eine Entkopplung der Funktionalitäten nur durchgeführt werden kann, wenn ein alternatives Datenmodell verwendet oder ein Weg gefunden wird, darauf gänzlich zu verzichten.

Eine weitere Art von Abhängigkeiten sind Klassen, die von deegree-Klassen erben oder deegree-Interfaces implementieren. So erbt zum Beispiel AbstractFilter von ComplexFilter aus deegree, damit selbstdefinierte Filter in deegree verwendet werden können.

Es liegen also verschiedene Formen von Abhängigkeiten vor, sodass ein einzelner Prozess, der im Normalfall schon durch mehrere Klassen des Bausteins bestimmt wird, eine komplexe Verbindung zu deegree haben kann. Besonders kritisch sind hierbei die Kopplungen, bei denen Vererbungshierarchien zwischen Packages bestehen, da in diesen Fällen selbst kleine Veränderungen in deegree zu größeren Problemen führen können. Dies führt dazu, dass die Verwendung von Geoframeworks im GIS-Viewer grundlegend überdacht und eine andere Lösung gefunden werden muss, damit der Einsatz der vorgenommenen Spezifikationen weiterhin erfolgen kann.

3.2.4. Notwendige Komponenten

Bei der Vielzahl von Strukturen und Funktionalitäten, die der GIS-Viewer von deegree nutzt, kann eine Unterscheidung in notwendige und unterstützende Komponenten vorgenommen werden. Notwendige Komponenten sind hierbei Strukturen, die wesentliche Abläufe übernehmen und zum Beispiel die Kommunikation mit einem WebService ermöglichen. Hingegen sind manche Komponenten, die zum Beispiel für den Umgang mit XML-Fragmenten vorgesehen sind, keine geo-spezifischen Anwendungen und haben nur den Zweck, bei der Vorbereitung oder Bereitstellung der eigentlichen Prozesse zu helfen.

Beispielweise wird eine Vielzahl von Datentypen aus deegree im GIS-Viewer eingesetzt, für deren Implementierung es andere Alternativen gibt und die deshalb nicht notwendigerweise Komponenten des Geoframeworks sein müssen. Genauer lässt sich hierbei noch in komplette Datenmodelle oder einzelne Datentypen unterscheiden.

So arbeitet der GIS-Viewer mit den Geometrietypen von deegree, wofür sich zum Beispiel die Geometrie vom JTS Topology Suite (JTS) ebenfalls eignen würde. Hierbei handelt es sich um eine Bibliothek, die ein räumliches Objektmodell und geometrische Funktionen liefert und sich nur auf dieses Teilproblem beim Umgang mit Geodaten konzentriert. Es ergibt sich also, dass der GIS-Viewer an dieser Stelle nicht unbedingt auf deegree oder ein anderes Geoframework angewiesen ist, da ein alternatives Datenmodell für die Geometrie existiert.

Weiterhin gibt es einzelne Klassen aus deegree, für die ebenfalls Alternativen vorhanden sind. So kommt im GIS-Viewer zum Beispiel eine deegree-Klasse zur Verwaltung von Qualified Name[7] zum Einsatz, wobei hierfür eine Implementierung in der Java-Standardbibliothek existiert. Das heißt, dass in diesem Fall gänzlich auf eine zusätzliche Bibliothek verzichtet werden kann. Ähnliches gilt für weitere Fälle, die sich mit Aufgaben zu Namensräumen, Verwaltung von XML oder anderen allgemeinen Problemen beschäftigen.

Es zeigt sich also, dass der Einsatz der Komponenten eines Geoframeworks nicht in jeder Situation notwendig ist und durch andere Quellen realisiert werden könnte.

Eine weitere Art von Abhängigkeiten, die durch die Verwendung von deegree-Funktionalitäten im GIS-Viewer entsteht, ist die Nutzung von Factorys, die meist mithilfe trivialer Eingaben Instanzen von Objekten erzeugen. Diese Instanzen werden dann für die eigentliche Funktionalität eingesetzt und haben keine weitere Bedeutung für den GIS-Viewer. Das wiederum heißt, dass der GIS-Viewer auf diese Komponenten nur angewiesen ist, weil die Konvertierung von Daten an manchen Stellen im GIS-Viewer selbst stattfindet und nicht im Geoframework erfolgt. Es handelt sich damit nicht um notwendige Komponenten, da sie durch eine Entkopplung oder das Entfernen der bestimmenden Funktionalität verschwinden würden.

3.3. Anforderungen an ein Geoframework in profil c/s

Aufgrund der gewonnen Kenntnisse können allgemeine Anwendungsfälle formuliert werden, die zwischen GIS-Viewer und einem beliebigen Geoframework auftreten. Im folgenden Diagramm sind die wesentlichen Anwendungsfälle zusammengefasst, die in **profil c/s** an ein Geoframework gestellt werden:

[7] Qualified Name ist ein Name, dem sein Namenskontext vorangestellt wird. [OgcQN]

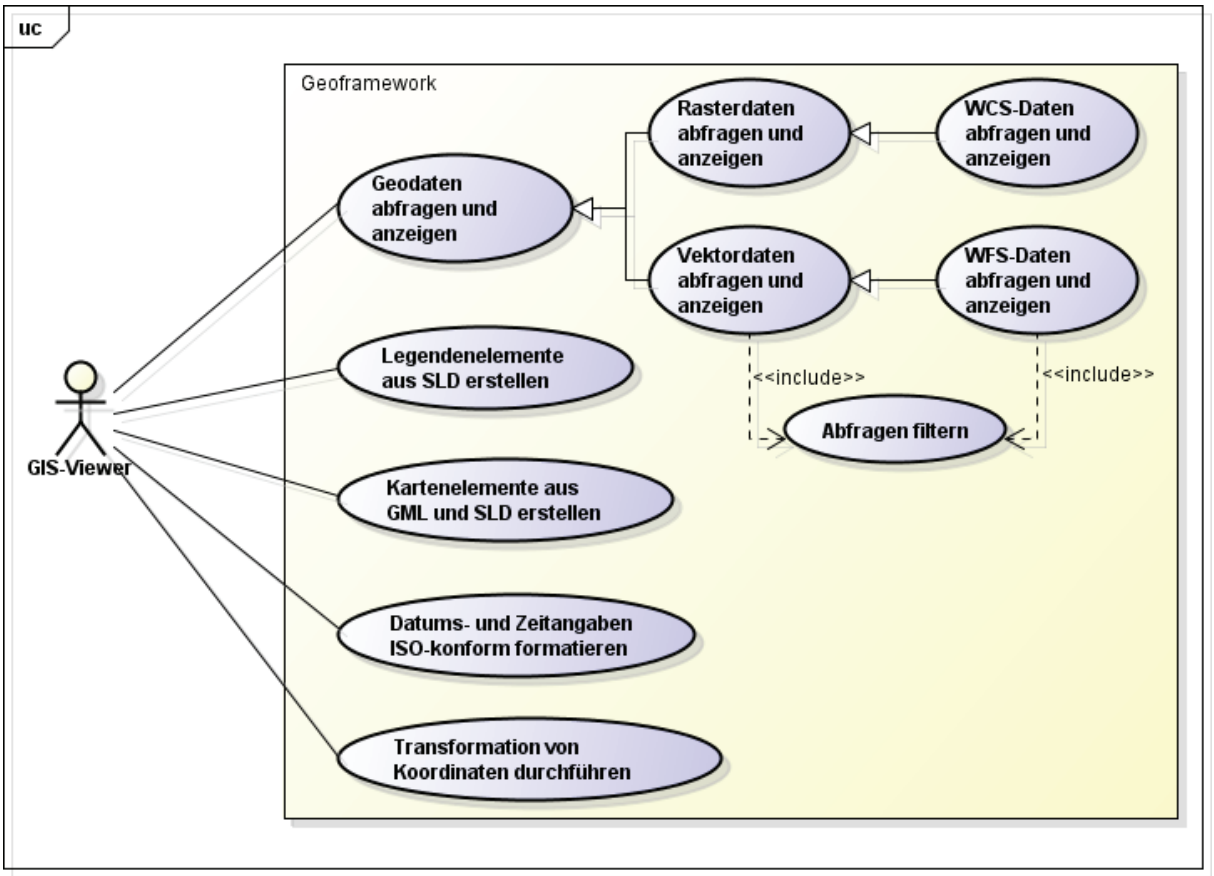

Abbildung 12 Use-Case-Diagramm: Anforderungen an ein Geoframework

Abfrage von Geodaten

Der wichtigste Aspekt eines Geoframeworks ist für den GIS-Viewer eine „Open Web Services"-Schnittstelle (OWS), die die Verwaltung von Vektor- und Rasterdaten und spezieller auch von WCS- und WFS-Daten realisiert. Das Geoframework soll Funktionen und Strukturen liefern, um mit OGC WebServices zu kommunizieren, Daten abfragen zu können, Ergebnisse zu filtern. Außerdem muss durch den Anwender festgelegt werden können, an welche Datenquelle die Schnittstelle ihre Anfragen richtet.

Da Abfragen von Vektordaten die Möglichkeit der Filterung besitzen, ist es notwendig für diesen Bereich der OWS-Schnittstelle ein entsprechendes Filter Encoding vorauszusetzen. Dies ist ein weiterer Bestandteil, der im verwendeten Geoframework realisiert sein sollte.

Kartenelemente

Um die Ergebnisse einer Anfrage an einen OGC WebService nutzen zu können, muss dieses noch weitere Funktionalitäten liefern. Denn wird zum Beispiel ein WFS genutzt, so sind die Ergebnisse von diesem keine darstellbaren Objekte, die sofort in die Karte abgebildet werden können. Stattdessen wird GML zurückgeliefert. In dieser Nachricht stecken die nötigen Informationen, um ein darstellbares Element zu erzeugen. Der SLD-Anteil wird durch den Anwender selbst oder durch den vorgegebenen Style der Anwendung konfiguriert.

Das Geoframework muss also Funktionen und Strukturen bereitstellen, mit denen die Resultate ausgewertet werden können. Das heißt, Funktionalitäten die das Umwandeln von GML und SLD in Grafiken fertigbringen. Außerdem muss der umgekehrte Weg, also die Erstellung von SLD aus

Nutzereingaben, realisiert sein, damit Veränderungen an der Darstellung und den Kartenelementen selbst durchgeführt werden können.

Legendenelemente

Eine Erweiterung der SLD-Kodierung erlaubt es, das Verfahren auch zum Erstellen von Symbolen in Legenden einzusetzen und somit auch die Gestaltung dieser standardisiert durchzuführen. Da die Legende ein entscheidender Bestandteil einer Karte ist und einheitlich gestaltet sein soll, muss auch diese Aufgabe vom Geoframework geleistet werden. Das heißt, das Geoframework muss die Umwandlung von SLD in grafische Symbole für Legenden bereitstellen.

ISO-konforme Zeit- und Datumformate

Die korrekte Formatierung von Zeitangaben kann aus verschiedenen Gründen von Bedeutung sein. So wird zum Beispiel durch die ISO-konforme Darstellung von Zeiten sichergestellt, dass es zu keinen Missverständnissen bei den Nutzern kommt. Außerdem kann durch festgelegte Formate eine automatische Verarbeitung vorgenommen werden, da den Algorithmen bekannt ist, wie das vorliegende Objekt aufgebaut ist.

Der GIS-Viewer benötigt formatierte Zeitangaben in erster Linie für die Darstellung verschiedener Daten (z.B. in der Attributtabelle) und für WFS-Anfragen. Daher ergibt sich auch, dass diese Funktionalität nicht unbedingt mithilfe eines Geoframeworks eingebracht werden muss. Entscheidend ist nämlich, dass die Standards eingehalten werden, die für GIS-Anwendungen vorgesehen sind.

Koordinatentransformation

Ein notwendiger Schritt bei der Darstellung von Raster- oder Vektordaten ist die Umwandlung der Ausgangskoordinaten (z.B. Landeskoordinaten) in Bildkoordinaten, die das Abbilden auf dem Bildschirm ermöglichen. Für eine Bearbeitung oder Selektion von Daten muss der umgekehrte Weg ebenfalls gewährleistet werden.

Mathematisch lässt sich die Umwandlung von Koordinaten zwischen zwei Systemen mithilfe der affinen Transformation beschreiben, die zu verschiedenen Zwecken eingesetzt wird. Dies führt dazu, dass der Algorithmus in verschiedenen Bibliotheken implementiert vorliegt, weshalb der GIS-Viewer an dieser Stelle nicht zwingend auf ein Geoframework angewiesen ist. In diesem Fall ließe sich zum Beispiel die Java-Standardbibliothek nutzen, die ebenfalls eine Klasse zur affinen Transformation enthält.

Zusammenfassend lässt sich festhalten, dass für die gestellten Anwendungsfälle des GIS-Viewers nur ein Bruchteil der Funktionalitäten benötigt wird, die ein Geoframework wie deegree oder GeoTools mit sich bringt. Zusätzlich sind manche Anwendungsfälle so allgemein, dass sie nicht notwendigerweise ins Aufgabengebiet eines Geoframeworks fallen, aber optional durch dieses bereitgestellt werden können. Dies wäre zum Beispiel erforderlich, wenn durch weitere Anforderungen eine spezielle Implementierung benötigt wird.

3.4. Anforderungen an den Lösungsentwurf

Für die Entkopplungen am GIS-Viewer sollen verschiedene Forderungen gelten, damit sich die vorgenommen Änderungen bewerten lassen und die Ziele der Arbeit noch einmal präzise

ausformuliert sind. Im Folgenden wird außerdem in Anforderungen unterschieden, die sich speziell für den GIS-Viewer und die angestrebte Lösung ergeben.

3.4.1. Nichtfunktionale Anforderungen

Die wichtigsten Kriterien für die Bewertung der Arbeit sind die Flexibilität, Offenheit und Integrierbarkeit des Systems. Das heißt, das Änderungen zukünftig schnell und einfach durchgeführt und Funktionalitäten ohne großen Aufwand hinzugefügt werden können. Die Lösung soll nicht nur für den GIS-Viewer eingesetzt werden können und in verschiedenen Stellen von **profil c/s** integrierbar sein. Auf der anderen Seite sollte die Lösung auch neue Komponenten, also weitere Geoframeworks, aufnehmen können, ohne größere Änderungen im bestehenden Code zu erfordern.

Außerdem soll der GIS-Viewer weiterhin verlässlich funktionieren. Damit ist gemeint, dass die Ergebnisse von Anfragen weiterhin korrekt sind und die geforderte Qualität einhalten. Zu diesem Punkt gehört auch die Testbarkeit der verschiedenen Komponenten, wobei sich unabhängig vom verwendeten Geoframework die gleichen Resultate ergeben müssen. So kann mit Hilfe von ausreichenden Tests sichergestellt werden, dass Änderungen im eigenen Code keine Fehler verursachen und Neuerungen in den verwendeten Frameworks auffallen. Dies gewährt auch auf lange Sicht eine hohe Qualität.

Zuletzt spielt noch die Performance eine Rolle. Im Rahmen dieser Anforderung sollte beachtet werden, in welchem Maße die Bearbeitungszeit von Aufgaben beeinflusst wird und wie sich der Speicherbedarf zur Haltung bestimmter Daten verändert.

3.4.2. Maßzahlen zur Bewertung der Entkopplung

Zur Untersuchung von Software stehen verschiedene Metriken zur Verfügung. So gibt es zum Beispiel klassische Metriken wie Lines-Of-Code[8], Halstead-Metrik[9] oder McCabe-Metrik[10], mit denen sich allgemeine Aussagen zur Qualität des Codes treffen lassen. Zwar können die grundsätzlichen Ideen hinter diesen Kennzahlen nicht unberücksichtigt bleiben, aber da im Rahmen dieser Arbeit die Entkopplung des GIS-Viewers alleiniges Ziel ist, sollen Metriken in Betracht gezogen werden, die sich auch in Bezug auf diese Thematik sinnvoll interpretieren lassen bzw. direkt eine Aussage dazu treffen.

Coupling Between Objects

Eine erste Metrik zum Beschreiben der Kopplung ist Coupling Between Objects (CBO), die der Anzahl der gekoppelten Klassen zur betrachteten Klasse entspricht. Mit Kopplung sind dabei Fälle gemeint, bei denen Variablen einer instanziierten Klasse oder Methoden einer Klasse verwendet werden. Angewendet wird CBO vorwiegend in Zusammenhang mit der Testbarkeit und Wartbarkeit von Software. Dem liegt die Überlegung zugrunde, dass eine Klasse umso komplexer zu verstehen und zu behandeln ist, je grösser die Anzahl ihrer Kopplungen mit anderen Klassen ist. Die Komplexität zieht dann auch einen höheren Aufwand – einen erhöhten Schwierigkeitsgrad – bei Änderungen nach sich. [IwCBO]

Folgendes Beispiel soll noch einmal kurz erläutern, wie sich der CBO-Wert ergibt. Die in Abbildung 13 dargestellten Klassen sind dabei unterschiedlich stark gekoppelt. Es ergeben sich folgende Werte:

[8] Die Lines-Of-Code-Metrik, kurz LOC, misst die Anzahl der Zeilen im Quellcode eines Programms. [Glo05]
[9] Unter dem Namen Halstead-Metrik fasst man eine Reihe von Kennzahlen zusammen, die Größe und Komplexität des Codes widerspiegeln sollen. [Glo05]
[10] Mit der McCabe-Metrik will man die strukturelle Komplexität eines Programmes ermitteln. [Glo05]

- CBO(A) = 1
- CBO(B) = 2
- CBO(C) = 1
- CBO(D) = 0

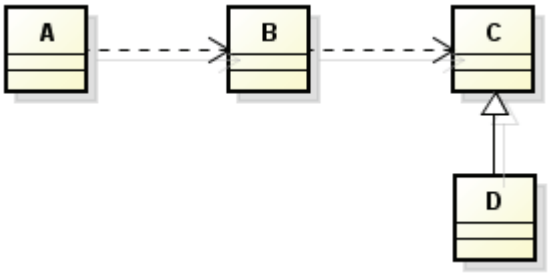

Abbildung 13 Beispiel CBO

Number Of Services

Eine weitere Metrik zur Beschreibung der Kopplung ist Number Of Services (NOS). Diese gibt wieder, wie viele verschiedene Dienste aus der betrachteten Klasse heraus genutzt werden. Mit Diensten sind in diesem Zusammenhang alle Methodenaufrufe gemeint. [Glo05]

Number Of Accesses

Unter gegebenen Umständen kann es auch interessant sein, alle Verbindungen einer Klasse zu anderen Klassen zu untersuchen. In diesem Fall entspricht die Menge der Zugriffe der Number Of Accesses (NOA). [Glo05]

4. Konzeption

4.1. Entkopplungsmechanismen

Zur Entkopplung eines Systems können verschiedene Verfahren eingesetzt werden, die mehr oder weniger für mehrere Arten der Kopplung geeignet sind. Im Rahmen der Entkopplung des GIS-Viewers von deegree werden zunächst Extension Points betrachtet, die sich zum Beispiel mit einer Lösung wie dem ServiceLoader[11] der Java-Standardbibliothek zu einem vollwertigen Plug-In-basierten System ausbauen lassen. Weiterhin wird das Entwurfsmuster der Anwendungsfassade, eine spezielle Form des Mediators[12], als Möglichkeit für die Entkopplung vorgestellt.

4.1.1. Extension Points und Plug-In-Mechanismen

Extension Points können optional von einer Software angeboten werden und erlauben, dass während der Laufzeit eine Funktionalität geladen oder gewechselt wird. Somit lässt sich die Software beliebig erweitern. Das Gegenstück zum Extension Point ist die Extension selbst, die die Anwendung erweitern kann. Beide Komponenten zusammen ergeben den Plug-In-Mechanismus.

Der Extension Points fungiert bei einem Plug-In als Registrierung. Diese ist gefüllt mit Einträgen der Extensions, die für eine Komponente als Erweiterung infrage kommen. Wird jetzt nach einer Funktionalität gesucht, wird die Registrierung auf die Existenz dieser geprüft und bei positivem Ergebnis erfolgt der Aufruf der zutreffenden Funktion. Die aufrufende Komponente kennt dabei nicht die ausgeführte Realisierung, sodass es an dieser Stelle zu keiner Abhängigkeit kommt. Grundlage für dieses Vorgehen ist jedoch eine Zugriffvereinbarung, die zum Beispiel in Form eines Interfaces definiert wird. [Hen08]

In folgender Abbildung ist ein Package zu sehen, das eine Komponente enthält, die zwei Extension Points nutzt. Die jeweiligen Extensions liegen dabei in zwei externen Packages:

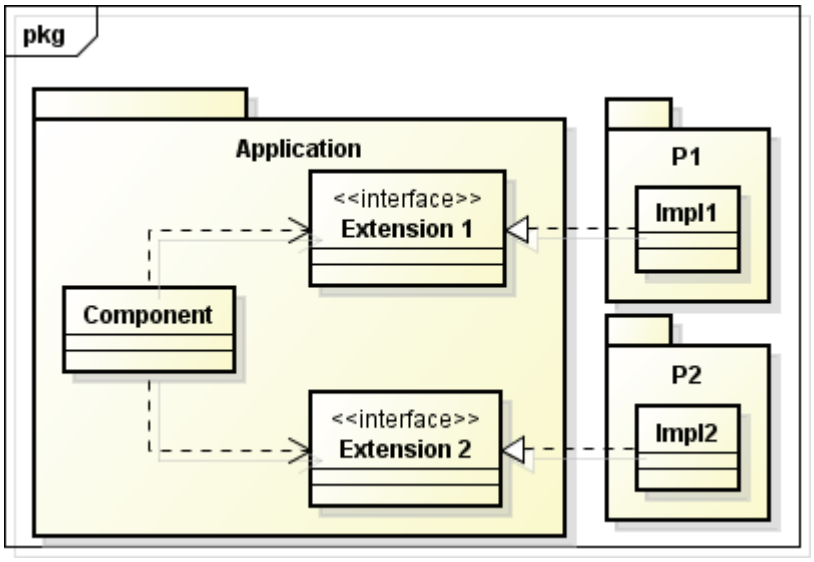

Abbildung 14 Klassendiagramm: Beispiel Extension Points

Um mithilfe eines Plug-In-Mechanismus die Entkopplung des GIS-Viewers vom verwendeten Geoframework durchzuführen, müssten zwei neue Schichten zwischen Geoframework und GIS-

[11] Siehe [JavaSL]
[12] Siehe [OodMed]

Viewer eingefügt werden. Die erste Schicht würde dabei die Extension Points enthalten, welche genau eine Funktionalität abbilden. In der zweiten Schicht würden sich die Extensions befinden. Somit würde die Abhängigkeit zu deegree oder GeoTools nur in den Extensions bestehen, wohin gegen die Extension Points unabhängig von den verwendeten Geoframeworks sind. In folgender Abbildung ist zu sehen, wie die Lösung aussehen könnte:

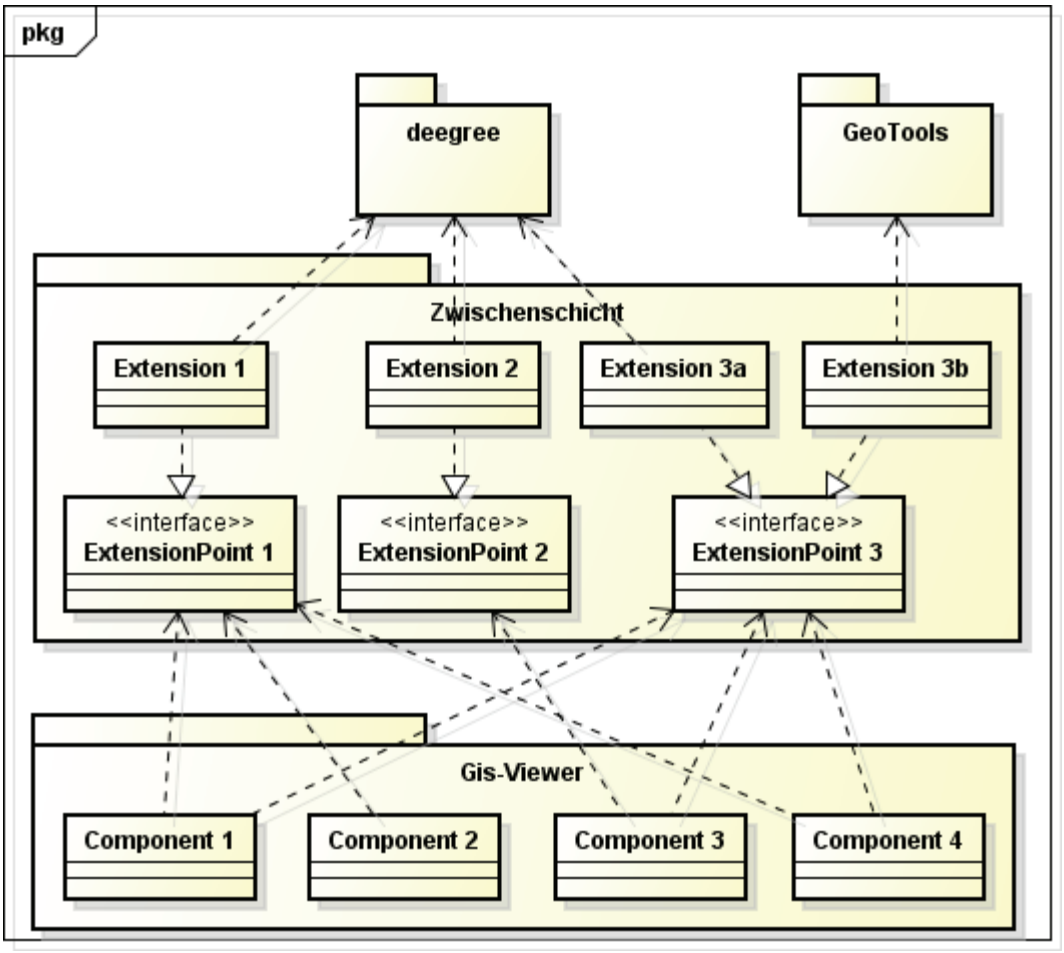

Abbildung 15 Klassendiagramm: Einsatz des ExtensionPoint-Mechanismus

Durch den Einsatz von Extension Points könnte also der GIS-Viewer mithilfe neuer Schichten lose an verschiedene Geoframeworks gekoppelt werden. So würde der GIS-Viewer nicht mehr an eine bestimmte Extension gebunden sein und stattdessen mit den ExtensionPoints arbeiten, deren konkrete Implementierungen nicht bekannt sein müssen.

Der Plug-In-Mechanismus ist also ein Mittel, mit dem sich die Entkopplung durchführen lässt und mit dem eine einfachere Schnittstelle erzeugt wird, die verbirgt, welche Komplexität hinter einen Aufruf steckt. Jedoch würde der Zugriff auf die Extension Points aus mehreren Stellen des GIS-Viewers erfolgen, sodass zum Beispiel nicht nur mehrere Komponenten von einem Extension Point abhängig sein können, sondern auch eine Komponente von mehreren Extension Points.

Um dies zu verhindern und die Schnittstelle einfacher zu gestalten, bietet sich ein weiteres Verfahren zur Entkopplung an. Dieses wird im folgenden Kapitel beschrieben und wird kombiniert mit dem Plug-In-Mechanismus zum tatsächlichen Entkopplungsverfahren führen.

4.1.2. Anwendungsfassade

Die Anwendungsfassade[13] (facade pattern) ist ein Entwurfsmuster, das eine vereinfachte Schnittstelle zu komplexeren Bibliotheken liefert und speziell für die objektorientierte Programmierung konzipiert ist. Eine Anwendungsfassade ermöglicht im Idealfall unter anderem eine leichtere Benutzung und Testbarkeit der Softwarebibliothek. Zusätzlich werden die Abhängigkeiten zur inneren Bibliothek verringert und es ist für den Anwender nicht mehr nötig, sich mit der Komplexität des System zu beschäftigen. Ein weiterer Vorteil ist die Erhöhung der Flexibilität. Bei der Nutzung mehrerer Schnittstellen lassen sich diese außerdem vereinigen und vereinfachen. [wikFac]

Nachteil einer solchen Lösung ist, dass eine neue Schicht in die bestehende Softwarearchitektur eingefügt wird. Dies erfordert einen einmaligen Aufwand und bewirkt eine Verschiebung der Wartung von der nutzenden Schicht in die Anwendungsfassade.

In folgender Abbildung ist ein Beispiel für eine Anwendungsfassade gegeben. In diesem Fall handelt es sich um eine Fassade, die den Zugriff auf die beiden Systeme Package1 und Package2 ermöglicht:

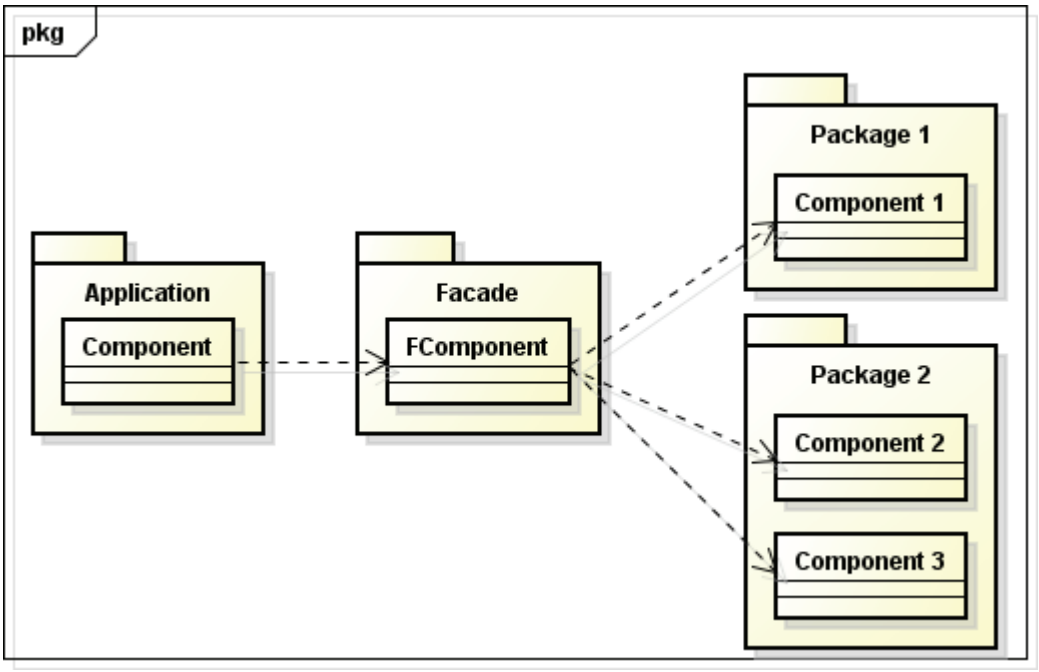

Abbildung 16 Klassendiagramm: Beispiel Anwendungsfassade

Das Entwurfsmuster der Anwendungsfassade scheint trotz der genannten Einschränkung eine geeignete Erweiterung für die Entkopplung zwischen GIS-Viewer und deegree zu sein. So entspricht die Ausgangslage genau den vorgegebenen Anwendungsfällen und die Ergebnisse nach der Umsetzung des Musters stimmen genau mit den Anforderungen überein, die an die Lösung gestellt werden. So kann mit der Anwendungsfassade eine unveränderliche Schnittstelle zum GIS-Viewer geschaffen werden, die sich aber mit wenig Aufwand an beliebige Geoframeworks anpassen lässt.

Die in Abbildung 15 gezeigte Lösung soll also durch die Einführung einer Anwendungsfassade ergänzt werden. Dafür muss eine weitere Schicht eingeführt werden, die eine Klasse enthält, welche die Funktionalität der verschiedenen Extension Points bündelt und beim Aufruf einer Methode an die benötigte Komponente delegiert.

[13] Siehe [Gam04]

Weiterhin könnte in der neuen Schicht ein Interface implementiert werden, welches wiederum die konkrete Implementierung der Anwendungsfassade verbirgt und bei Bedarf auch den Zugriff auf die Fassade mithilfe des Plug-In-Mechanismus ermöglicht. Dies würde zusätzlich für eine lose Kopplung zwischen GIS-Viewer und Fassade sorgen.

Die angepasste Lösung mit hinzugefügter Anwendungsfassade könnte dann folgendermaßen aussehen:

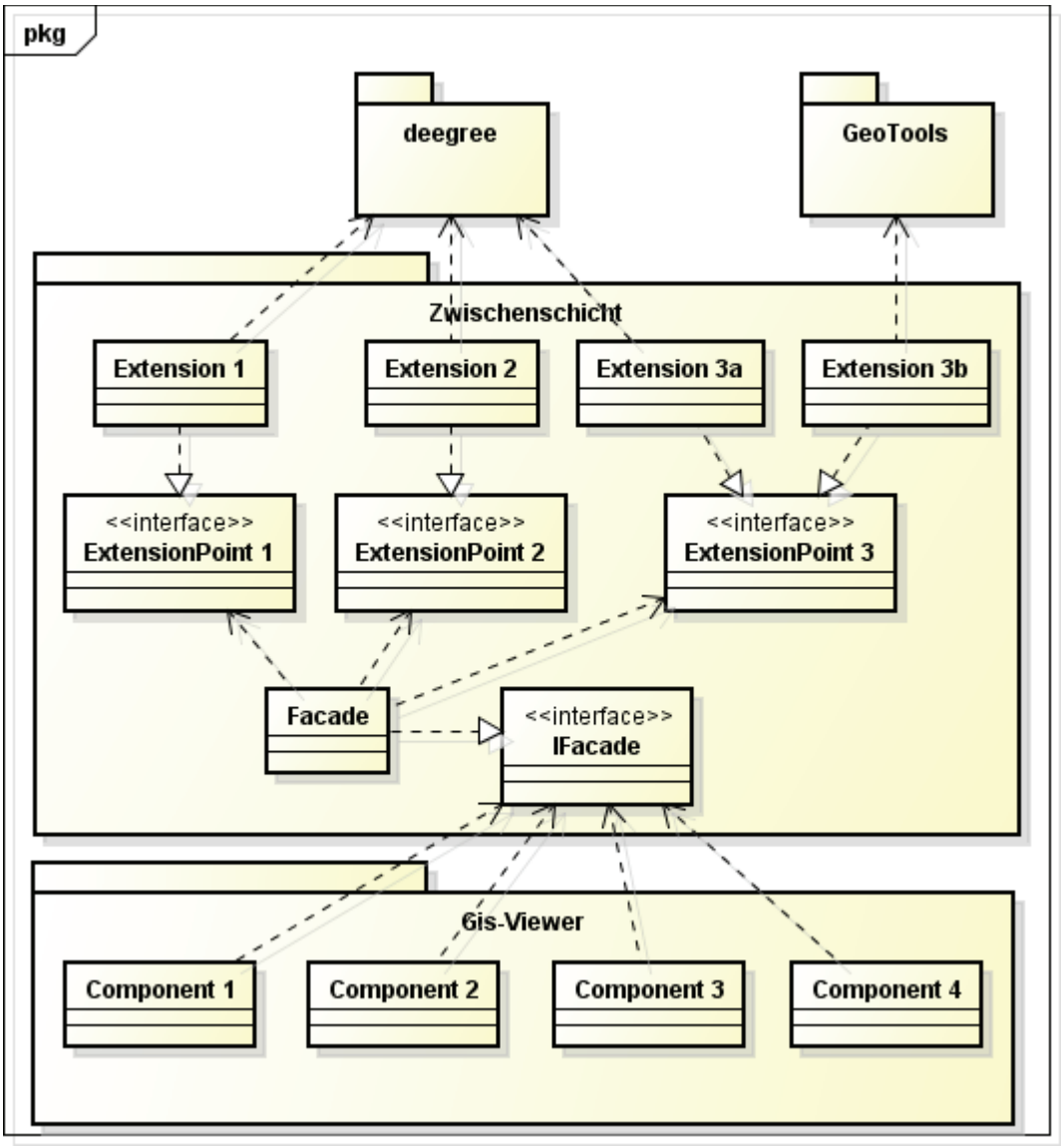

Abbildung 17 Klassendiagramm: Erweiterung mit einer Anwendungsfassade

Im Gegensatz zur Lösung ohne die Anwendungsfassade, besitzt nun eine Komponente nicht mehr Abhängigkeiten zu verschiedenen Extension Points, sondern greift stattdessen auf die Anwendungsfassade zu. Somit gibt es eine zentrale Schnittstelle, die für den Anwender alle Funktionalitäten der Extension Points zusammenfasst.

Zwar kommt es durch die Einführung zu einer Erhöhung der Komplexität innerhalb der Zwischenschichten und zu einem weiteren Aufruf, der die eigentliche Anfrage weiter delegiert, jedoch ist dies für den Anwender nicht sichtbar. Außerdem sind durch die Änderungsstabilität, den geringen Änderungsaufwand beim Austausch von Geoframeworks sowie durch die vereinfachte

Schnittstelle mehrere Vorteile gegeben, die genau den gestellten Zielen entsprechen und daher die Einschränkungen bei weitem aufwiegen.

4.2. Allgemeiner Lösungsentwurf

Da der GIS-Viewer Komponenten aus verschiedenen Stellen von deegree nutzt, mit ihnen verschiedene Aufgaben erledigt und in Zukunft auch Bestandteile andere Geoframeworks nutzen soll, lässt sich die Entkopplung nicht mit einer einzelnen Anwendungsfassade lösen. Stattdessen werden mehrere Fassaden ein neues System bilden, das zusätzliche Klassen enthält. Diese helfen dann bei der Zusammenarbeit der einzelnen Abläufe.

Die Komponenten innerhalb der Fassade lassen sich wiederum kapseln und untergeordneten Fassaden zuteilen. Auf diesem Weg lässt sich dann zum Beispiel leicht bestimmen, welche Elemente nur intern zugänglich sein sollen. Durch Einführung abstrakter Oberklassen kann außerdem erreicht werden, dass die Anwendung nicht mehr wissen muss, welche Implementierung gerade verwendet wird.

In folgender Abbildung ist vereinfacht dargestellt, wie die Anwendungsfassade den Zugriff auf die Geoframeworks regeln könnte:

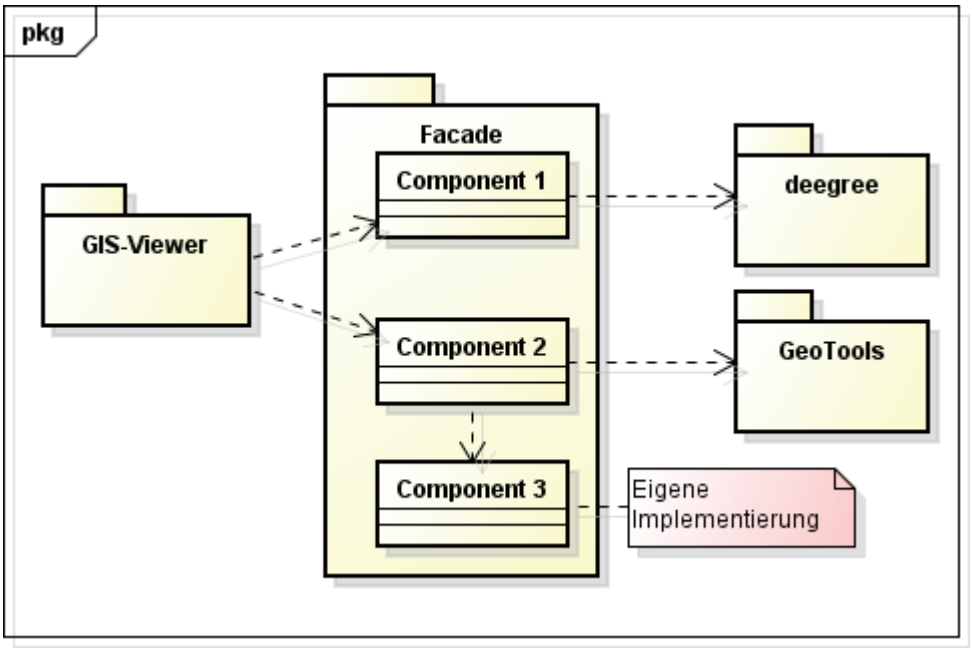

Abbildung 18 Klassendiagramm: Allgemeiner Lösungsentwurf

Es ist so vorgesehen, dass der GIS-Viewer in Zukunft nicht mehr von deegree oder einer anderen Geoframework abhängt und stattdessen zentrale Aufrufe in der Fassade bereitgestellt werden. Durch diese ermittelten Ergebnisse sollen dann nur noch in Datentypen der Java-Standardbibliothek oder von JTS an den GIS-Viewer geliefert werden. Aus diesem Grund werden zusätzliche Funktionen nötig sein, mit denen zum Beispiel eine Geometrie von deegree zu JTS umgewandelt wird.

Weiterhin sollen manche Aufgaben nicht mehr durch ein Geoframework gelöst werden und stattdessen durch eigene Implementierungen bereitgestellt werden. Auch diese Komponenten müssen in der Anwendungsfassade untergebracht werden, sodass sie sich in die restliche Schnittstelle einfügen.

Neben dem Aufbau verändert sich auch die Funktionsweise des GIS-Viewers durch die Einführung der Anwendungsfassade. Der in Abbildung 6 dargestellte Hauptprozess im GIS-Viewer verändert sich, wie in folgender Abbildung zu sehen ist:

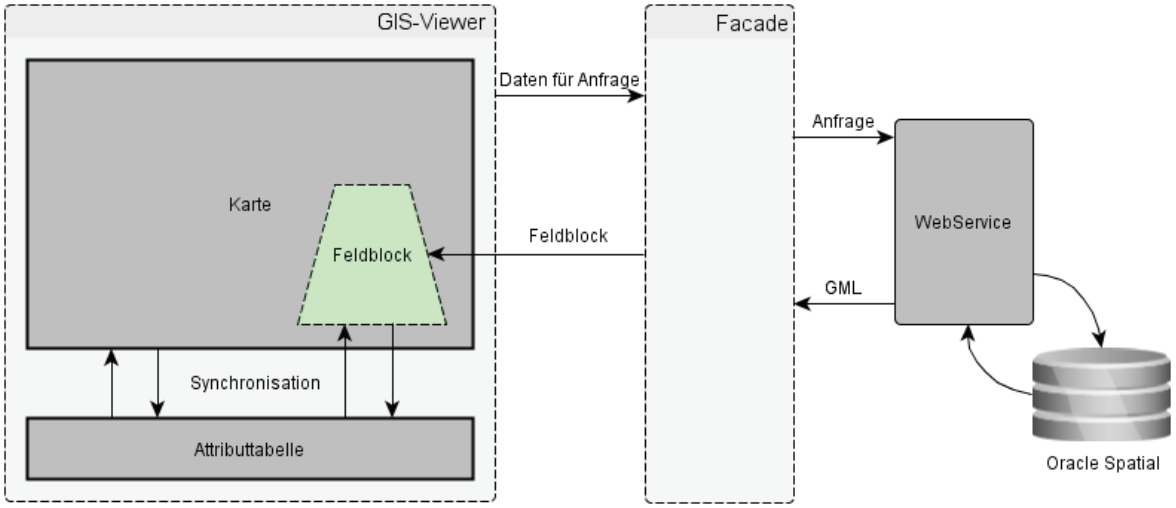

Abbildung 19 Funktionsweise des GIS-Viewer mit Anwendungsfassade

Wie in der Abbildung zu sehen, übernimmt die Anwendungsfassade verschiedene Aufgaben, die vorher im GIS-Viewer gelöst wurden. Somit hat der GIS-Viewer auch keinen direkten Kontakt mehr zum Service mit der Datenquelle und stattdessen verwendet er nur noch die fertigen Objekte, die er aus der Fassade erhält. Durch dieses Vorgehen kann neben der losen Kopplung die Code-Duplizierung vermieden wird.

4.3. Exemplarische Detailentwürfe

Nach der Vorstellung des Grobkonzeptes soll nun anhand einiger Bespiele aufgezeigt werden, wie mit verschiedenen Situationen der Kopplung verfahren werden kann. So werden zunächst Beispiele für die Entkopplung einzelner Objekte vorgestellt, um die einfachste Form der Kommunikation zwischen GIS-Viewer und Geoframeworks mithilfe der Fassade zu verdeutlichen.

Darauf wird ein Beispiel mit höherer Komplexität dargestellt, sodass nicht mehr das Entkoppeln eines einzelnen Objektes ausreicht und stattdessen eine Funktionalität innerhalb der Fassade formuliert werden muss.

Weiterhin wird im nächsten Schritt ein anderes Beispiel angebracht, in dem eine neue Funktionalität formuliert und mithilfe des ServiceLoaders entkoppelt wird. Anhand dieses Beispiels wird dann auch aufgezeigt, wie Komponenten von deegree oder GeoTools zum Erfüllen einer Aufgabe eingesetzt werden können. Somit handelt es sich dann auch um einen sinnvollen Einsatz des Plug-In-Mechanismus und den praktischen Beleg für die Austauschbarkeit der Geoframeworks.

Zuletzt wird ein Fall präsentiert, in dem die Entkopplung durch eine Eigenimplementierung erreicht wird, sodass auch diese Möglichkeit beschrieben ist.

4.3.1. Entkopplung eines Datenobjekts

Für den Aufruf von Methoden aus deegree-Klassen sind manche Datenobjekte unabdingbar, da keine Alternativen geboten werden und die Schnittstelle von deegree für das eigene Datenmodell

ausgelegt ist. Somit lässt sich zum Beispiel nicht umgehen, die Klasse QualifiedName zu benutzen, da diese für viele Aufgaben benötigt wird. Dies führt dazu, dass der Aufruf einer deegree-Komponente immer weitere Abhängigkeiten mit sich bringt und sich eine Situation ergeben kann, die wie in folgender Abbildung aufgebaut ist:

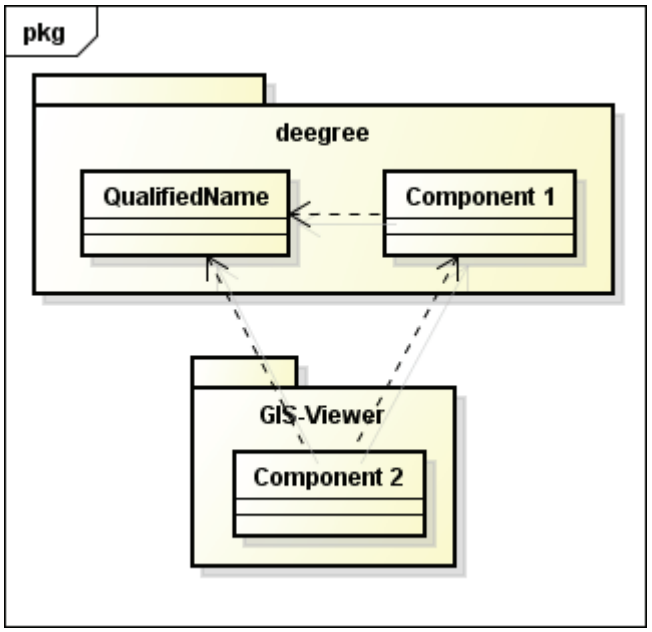

Abbildung 20 Klassendiagramm: Abhängigkeiten zu QualifiedName

Zur Lösung des Problems gibt es mehrere Möglichkeiten. So ließe sich zunächst innerhalb der Fassade ein eigenes Datenobjekt formulieren oder aber es findet sich eine alternative Implementierung. Dies könnte für QualifiedName zum Beispiel QName aus der Java-Standardbibliothek sein, welche den Vorteil mitbringt, dass GeoTools diese Implementierung nutzt.

Trotz der Verwendung einer vorgegebenen Klasse muss in der Fassade eine Struktur implementiert werden, die den Austausch zwischen QName und QualifiedName erlaubt und daher eine Transformation zwischen beiden Objekten ermöglicht. Ob dafür eine gänzlich neue Datenstruktur eingeführt wird oder nur eine Klasse, die über die nötigen Konvertierungsmethoden verfügt, hängt dann von weiteren Faktoren ab. So könnte zum Beispiel eine Rolle spielen, ob wirklich alle Bestandteile der vorliegenden Datenobjekte gebraucht werden oder welche weiteren Abhängigkeiten zu einem Geoframework bestehen.

Eine Klasse zur Handhabung von QName und QualifiedName könnte zum Beispiel folgendermaßen aufgebaut sein:

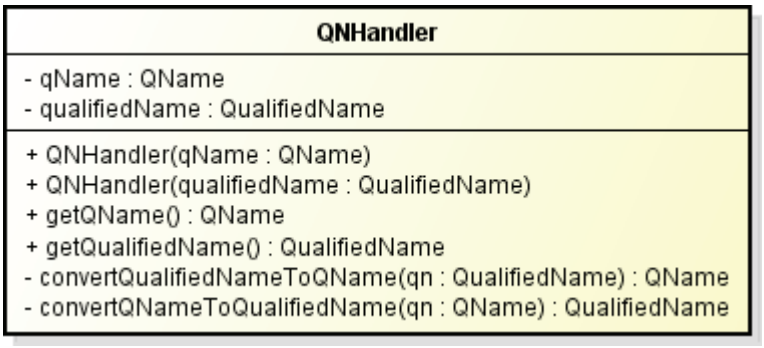

Abbildung 21 QNHandler

In folgender Abbildung ist dargestellt, wie mithilfe dieser neuen Klasse QNHandler die Entkopplung durchgeführt werden könnte:

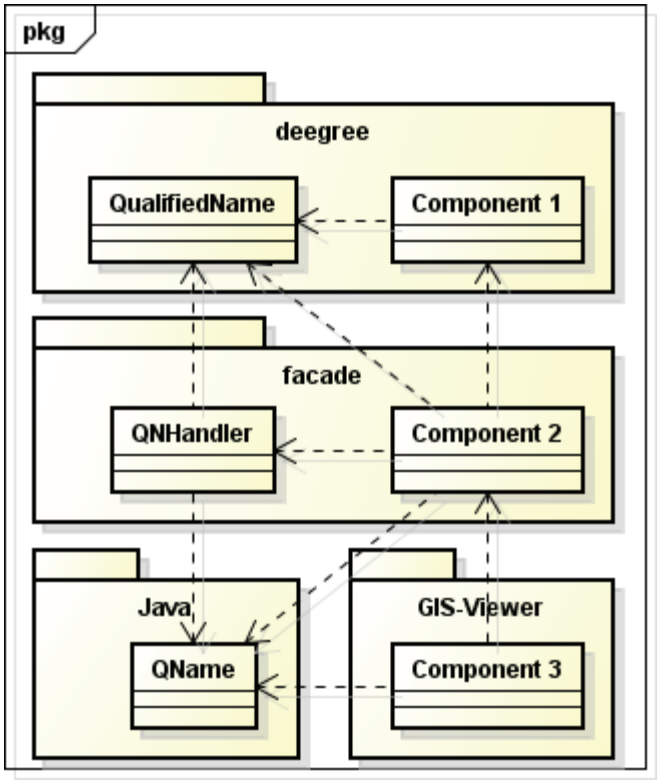

Abbildung 22 Klassendiagramm: Entkopplung von QualifiedName

Ergebnis einer solchen Entkopplung wären also die vollständige Trennung von GIS-Viewer und verwendeten Geoframework und die Nutzung einer allgemeineren Klasse aus der Java-Standardbibliothek. Außerdem müsste beim Hinzukommen eines neues Frameworks, das eventuell noch eine andere Implementierung von QName benutzt, nur der QNHandler ergänzt werden, um den Austausch mit den bereits vorhandenen Komponenten zu ermöglichen.

Zwar bleibt bei diesem Entwurf in der Fassade die Abhängigkeit zu QualifiedName bestehen, aber der wesentliche Gewinn der Entkopplung ist in diesem Fall, dass weitere Geoframeworks ohne großen Aufwand hinzugefügt werden können und zum GIS-Viewer hin eine vereinfachte Schnittstelle geboten wird. Diese verbirgt, welches Framework und speziell welches Datenobjekt eigentlich dahintersteht.

4.3.2. Entkopplung einer Funktionalität durch Zusammenführen von Diensten

An manchen Prozessen im GIS-Viewer sind viele deegree-Klassen beteiligt, da für die Bewältigung bestimmter Aufgabe verschiedene Zwischenschritte notwendig sind. Eine solche Situation liegt zum Beispiel bei der Erstellung der Legendenelemente vor, da hierfür mehrere Objekte, dazugehörige Factorys und weitere Strukturen nötig sind. Es handelt sich in diesem Fall also um eine Vielzahl von Abhängigkeiten, wobei sich diese an verschiedene Stellen im GIS-Viewer verteilen, weshalb es daher auch zur Codeduplizierung kommt.

So hängt der LegendTableRenderer, der das Zeichnen der Legende übernimmt, von mehreren deegree-Klassen ab und enthält gewisse Funktionalitäten, die auch in anderen Komponenten anzutreffen sind. Ein Beispiel wäre die Überprüfung, ob ein Legendenelement vorliegt. Dieser Test wird im LegendTableRenderer gebraucht, um die Elemente mit Style zu bestimmen, und im ModelAdapter eingesetzt, um festzulegen, welche Elemente selektierbar sein sollen.

In folgender Abbildung ist dargestellt, welche Abhängigkeiten die beiden benannten Klassen betreffen:

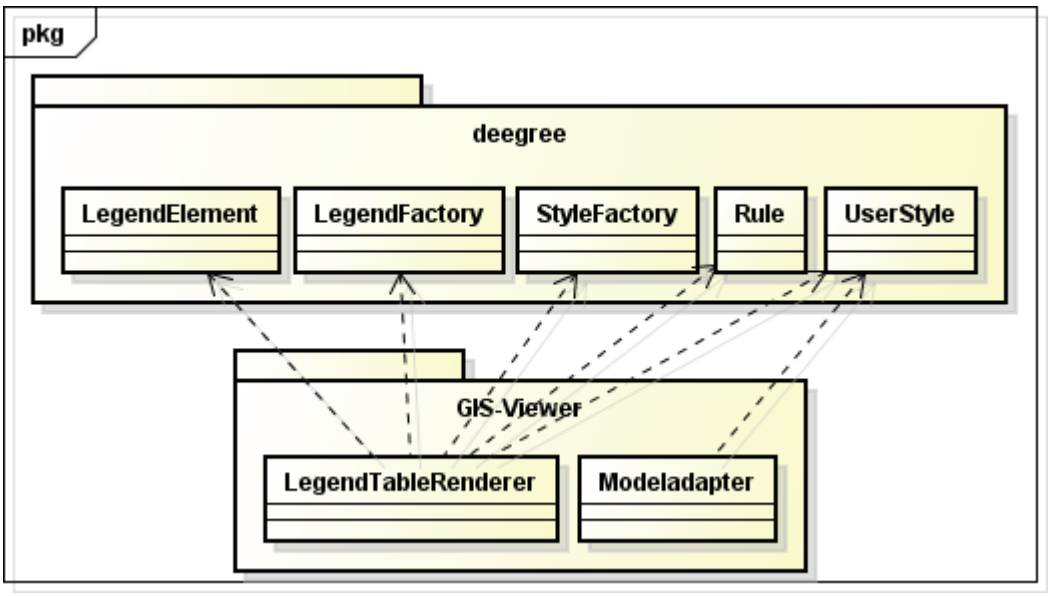

Abbildung 23 Klassendiagramm: Abhängigkeiten im Rahmen der Legende

Im Rahmen der Anwendungsfassade werden die aufgerufenen Dienste zusammengefasst, sodass eine zentrale Schnittstelle entsteht, die alle benötigten Aufrufe liefert. Diese Vereinfachung führt auch zu einer höheren Kohäsion des Systems, da die Aufgabengebiete klarer definiert sind.

Konkret könnte das Problem im aufgezeigten Beispiel durch die Einführung einer Klasse LegendTools innerhalb der Fassade gelöst werden. Diese Klasse würde kein eigenes Objekt repräsentieren, sondern nur Methoden für die Erzeugung oder Bearbeitung anderer Objekte enthalten. Zwar würden durch diese Lösung die Abhängigkeiten zu deegree nun in der Fassade liegen, jedoch würde die Codeduplizierung komplett wegfallen und im Rahmen der Austauschbarkeit und Wartung müsste nun im Vergleich zur Ausgangssituation nur noch eine Stelle angepasst werden.

Mithilfe der Fassade könnte die Lösung zum Beispiel folgendermaßen aussehen:

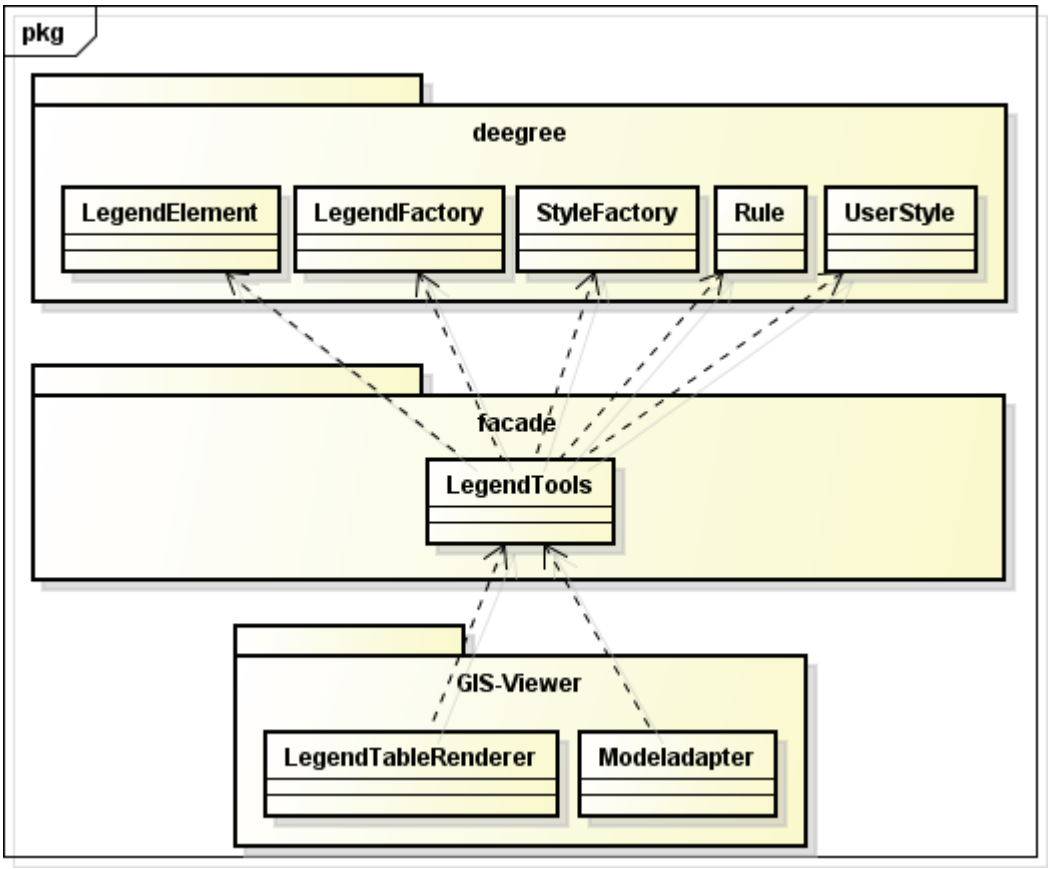

Abbildung 24 Klassendiagramm: Entkopplung mithilfe der Klasse LegendTools

Da eine solche Klasse verbirgt, welches Framework für die Bearbeitung einer Aufgabe genutzt wird, kann es durchaus zum Einsatz mehrerer Geoframeworks kommen. In diesem Fall würde sich jedoch die Unterteilung in untergeordnete Fassaden für jedes Geoframework anbieten, damit die Übersicht erhalten bleibt und zum Beispiel beim Austausch eines Geoframeworks nur die betroffene Fassade angepasst werden muss. Die untergeordneten Fassaden werden bei einer solchen Lösung dann wiederum von der eigentlichen Fassade verwendet, sodass die erhöhte Komplexität für den GIS-Viewer nicht sichtbar ist.

4.3.3. Entkopplung einer Funktionalität mithilfe neuer Strukturen

Da nicht alle Prozesse, wie zum Beispiel in Kapitel *4.3.2. Entkopplung einer Funktionalität durch Zusammenführen von Diensten* vorgestellt, mit simplen Datenobjekten als Ergebnis enden oder sich auf einen einzelnen Methodenaufruf beschränken, kommt es in den meisten Fällen zu wesentlich komplexeren Abhängigkeiten zwischen GIS-Viewer und dem verwendeten Geoframework. Dies führt dazu, dass sich nicht alle Probleme mithilfe einer einzigen Klasse zentralisieren lassen und stattdessen die Schaffung neuer Strukturen in der Fassade notwendig ist, die die Kommunikation der betroffenen Komponenten regeln.

Ein Beispiel für diese Art von Problem sind die vorliegenden SymbolizerPanel des GIS-Viewers, die eine Vorschau bei der Konfiguration von Styles liefern. So gibt es zwei Ausprägungen dieser SymbolizerPanel, wobei diese jeweils für die Bearbeitung des Styles von Polygonen oder Punkten vorgesehen sind. Die Elternklasse hingegen regelt, die Umwandlung der Styles in ein Bild für die Vorschau und wie auf Nutzereingaben reagiert werden soll.

Aufgrund dieser Aufgaben bestehen verschiedene Abhängigkeiten zu deegree, da Styles und Symbole erstellt werden müssen. Außerdem wird die Umwandlung der Symbole in ein Bild ebenfalls mithilfe von deegree-Komponenten gelöst.

In folgender Abbildung werden alle Abhängigkeiten aufgezeigt, die in der Ausgangssituation eine Rolle spielen:

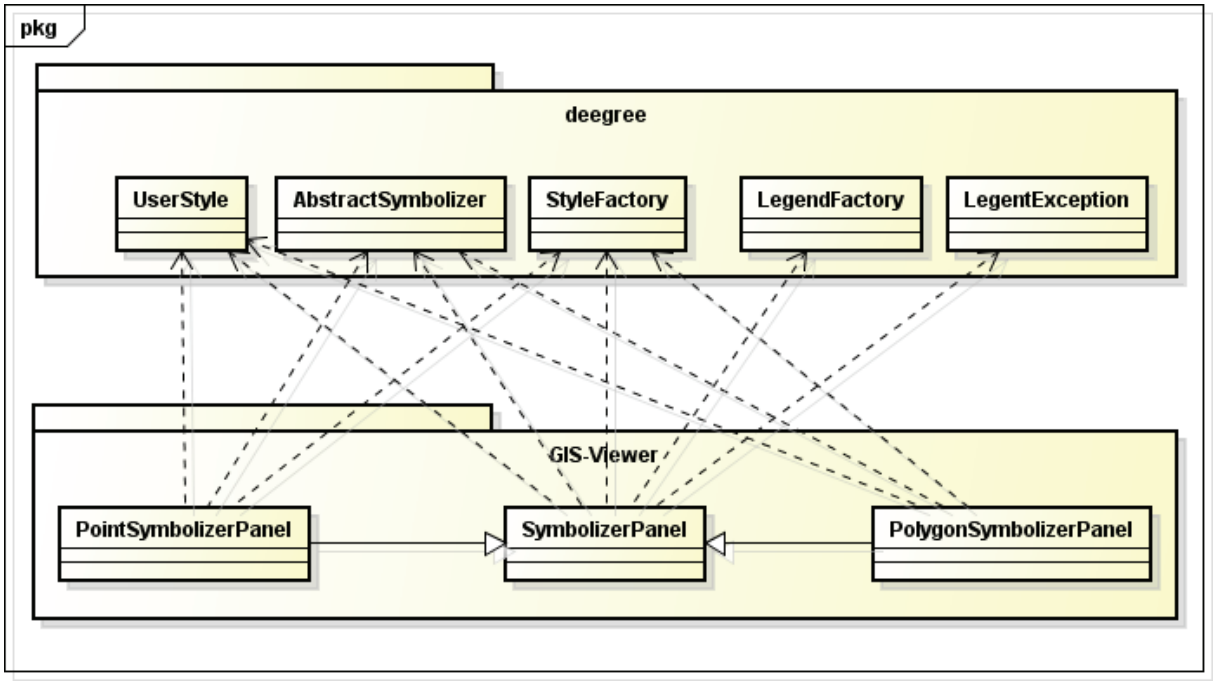

Abbildung 25 Klassendiagramm: Abhängigkeiten der SymbolizerPanel

Das entscheidende Problem ist hierbei, dass die Symbole vom Typ AbstractSymbolizer in den speziellen Klassen PolygonSymbolizerPanel und PointSymbolizerPanel definiert werden und über eine Methode der Elternklasse SymbolizerPanel ausgewertet und somit zu einem Bild umgewandelt werden. Dieser Prozess muss also, obwohl er auf mehrere Klassen verteilt ist, zusammen entkoppelt werden.

Zur Lösung des Problems bietet sich die Einführung drei neuer Klassen innerhalb der Fassade an, mit denen der Prozess verschoben werden kann, sodass die SymbolizerPanel nur noch für die Eingabe der Daten und die Darstellung des fertigen Vorschaubildes benötigt werden. Die erste hinzugefügte Klasse ClassifiedGeometrySymbolizer ist eine abstrakte Klasse und enthält nur eine Methode zur Umwandlung eines Styles in ein Vorschaubild. Dabei wird bewusst darauf verzichtet, den Style wie in der Ausgangssituation von einem AbstractSymbolizer abzufragen, da diese Abhängigkeit mit der neuen Lösung nicht mehr notwendig sein wird.

Weiterhin werden zwei weitere Klassen eingeführt, die beide von ClassifiedGeometrySymbolizer erben und den Style für das Vorschaubild bereitstellen. Die Klassen PointSymbolizer und PolygonSymbolizer selbst enthalten jeweils nur einen Konstruktor, der mithilfe einfacher Eingaben einen Style erzeugt, der verborgen für den GIS-Viewer gespeichert wird. Dieser Style kann nun jeder Zeit über die Methode der Elternklasse in ein Bild umgewandelt werden.

Die Klassen PolygonSymbolizerPanel und PointSymbolizerPanel im GIS-Viewer müssen nun nur noch die Nutzereingaben an den zugehörigen Symbolizer in der Fassade geben und haben damit keine

weiteren Abhängigkeiten. Für den SymbolizerPanel gilt das gleiche Prinzip. So ruft dieser nur noch die Methode von ClassifiedGeometrySymbolizer auf, um das Vorschaubild zu erhalten.

Die Lösung ist in folgender Abbildung dargestellt:

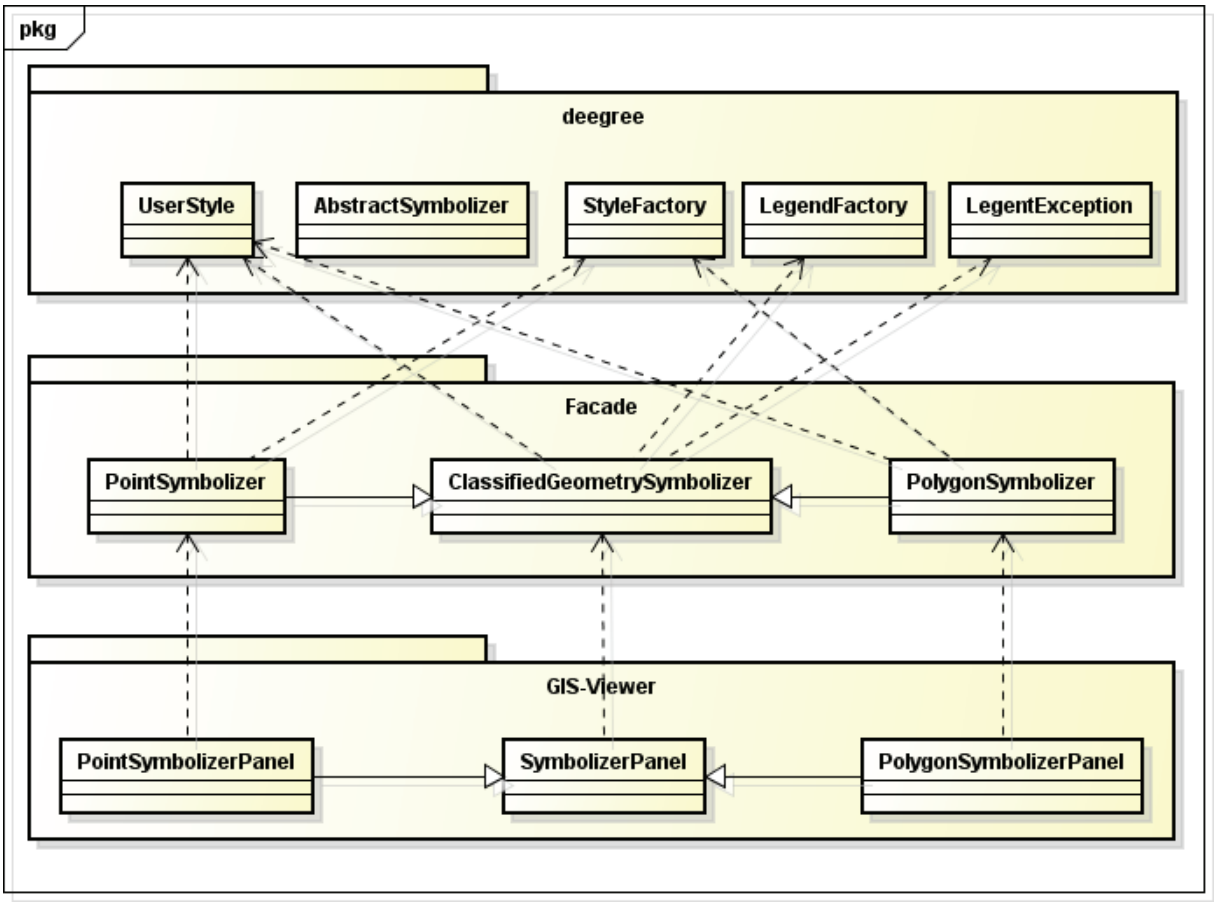

Abbildung 26 Klassendiagramm: Entkopplung der SymbolizerPanel

Vorteil dieses Entwurfs ist, dass zunächst die Abhängigkeiten minimiert werden und besser strukturiert, sodass neben der losen Kopplung auch mehr Übersichtlichkeit bewirkt wird. Außerdem wird eine einfache Schnittstelle erzeugt, die an die Vorgaben des GIS-Viewers angepasst ist und trotzdem mit wenig Aufwand erweitert werden kann.

Bei der Bewertung einer solchen Lösung muss jedoch bedacht werden, dass diese Lösung nur für eine spezielle Konstellation funktioniert und jedes weitere Problem einen Anfangsaufwand erfordert, um eine ähnliche Entkopplung zu finden, die dann auch ihren Zweck erfüllt. Im Allgemeinen kann aber festgestellt werden, dass auch die Entkopplung von Prozessen möglich ist, die auf verschiedene Komponenten verteilt sind.

4.3.4. Entkopplung mithilfe des ServiceLoaders

Neben der Entkopplung des GIS-Viewers von deegree ist die Nutzung von GeoTools für manche Aufgaben ein weiteres Ziel dieser Arbeit. Dies kann wiederum dazu führen, dass gewisse Funktionalitäten doppelt auftreten, weil für sie Lösungen mithilfe beider Geoframeworks erstellt werden können. Dies könnte zum Beispiel auftreten, wenn die Varianten eine unterschiedliche Performance oder Robustheit haben, sodass sich je nach Anwendungsfall nur eine der Lösungen anbietet.

An dieser Stelle bietet sich der Einsatz eines Plug-In-Mechanismus innerhalb der Fassade an, denn auf diesem Wege lässt sich zur Laufzeit bestimmen, welche Implementierung vorliegen und für die Bewältigung der Aufgabe genutzt werden soll. So ist dieses Vorgehen beispielsweise bei der Konvertierung von GML in eine Geometrie interessant, weil dadurch die Möglichkeit bestände, Ergebnisse der verschiedenen Geoframeworks zu vergleichen, und je nach Situation das optimale Verfahren einzusetzen.

Bevor es jedoch zur Verwendung des ServiceLoaders, wie in Kapitel *5.2.3. Einsatz des ServiceLoaders* beschrieben, kommen kann, muss zunächst jeweils eine untergeordnete Fassade für deegree und GeoTools erzeugt werden, die eine Methode zur Konvertierung von GML bereitstellt. Dies geschieht nach dem Muster aus Kapitel *4.3.2. Entkopplung einer Funktionalität durch Zusammenführen von Diensten* und wird durch ein Interface erweitert, das beide Klassen implementiert. Dadurch ist sichergestellt, dass beide Klassen die gleichen Methodenaufrufe besitzen und ein einheitlicher Zugriff möglich ist. Das Interface bestimmt zudem den Rückgabetyp der Geometrie, die von JTS stammen soll. Durch diese Festlegung kann die Forderung eingehalten werden, im GIS-Viewer ein Datenmodell, also eben das von JTS, zu verwenden, das unabhängig von den Geoframeworks ist.

Die Lösung könnte für deegree also folgendermaßen aussehen:

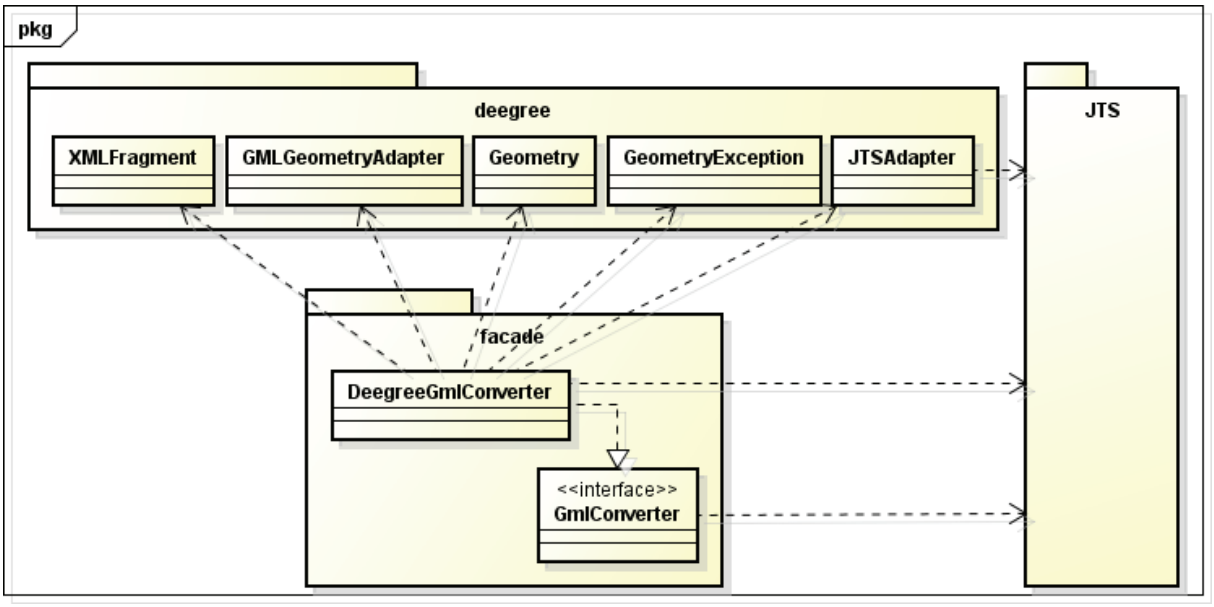

Abbildung 27 Klassendiagramm: GML-Konverter mit deegree

Bei der Lösung mit GeoTools kommt noch ein weiteres Framework hinzu. Es handelt sich um SAX - Simple API for XML[14], das für das Auswerten von XML genutzt wird. Mit GeoTools und SAX könnte die Lösung folgendermaßen aussehen:

[14] Siehe [SAX]

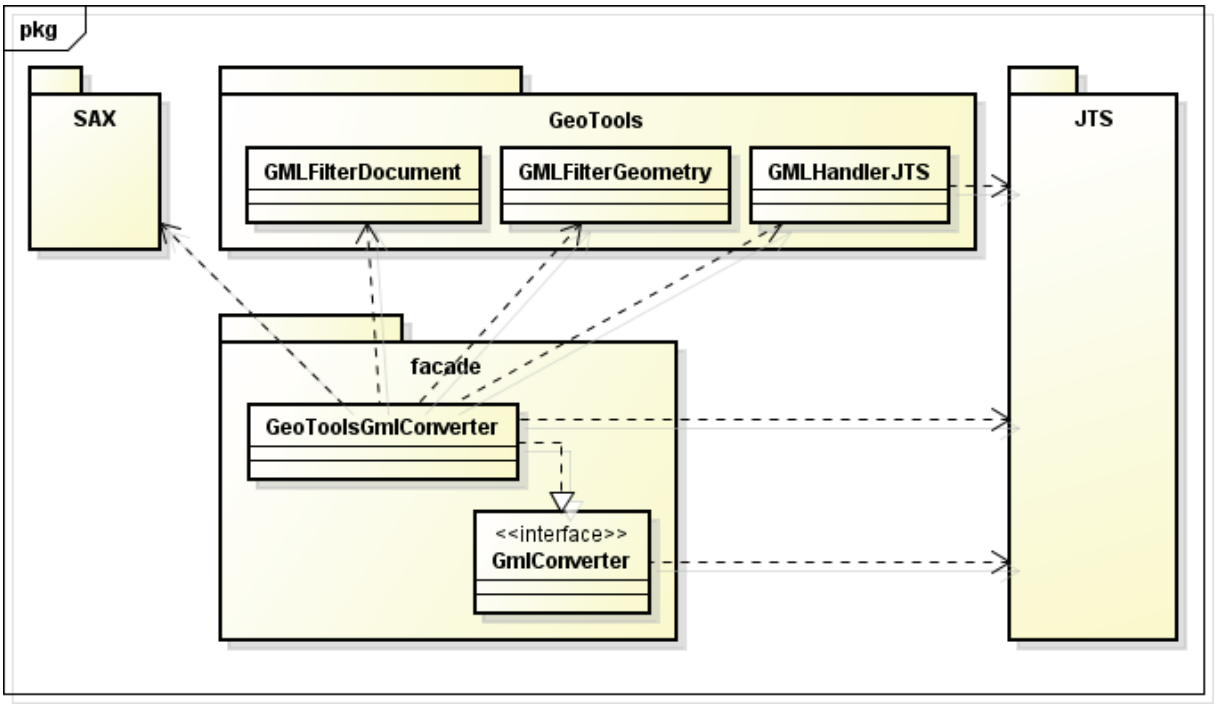

Abbildung 28 Klassendiagramm: GML-Konverter mit GeoTools

Wenn beide Lösungen in der Fassade vorliegen, kann nun das eigentliche Tool verfasst werden. Dies stellt nun die tatsächliche Schnittstelle der Fassade dar, auf die der Zugriff erfolgen soll. Der erste Vorteil dieser Lösung ist nun, dass schon dieses Tool keine Verbindung mehr zu den Geoframeworks hat. Das heißt, sollten sich deegree oder Geotools in Zukunft ändern, ist selbst dieser Teil der Fassade wartungsfrei. Weiterhin besteht, durch den Einsatz des ServiceLoaders auch keine Abhängigkeit zu den konkreten Implementierungen von GMLConverter, sodass auch innerhalb der Fassade eine möglichst lose Kopplung gegeben ist.

In folgender Abbildung ist dargestellt, wie der Gesamtentwurf für die Situation aussehen könnte, wobei aus Gründen der Übersichtlichkeit auf die Einbringung von JTS und der Abhängigkeiten auf Klassenebene zu den verwendeten Geoframeworks verzichtet wird:

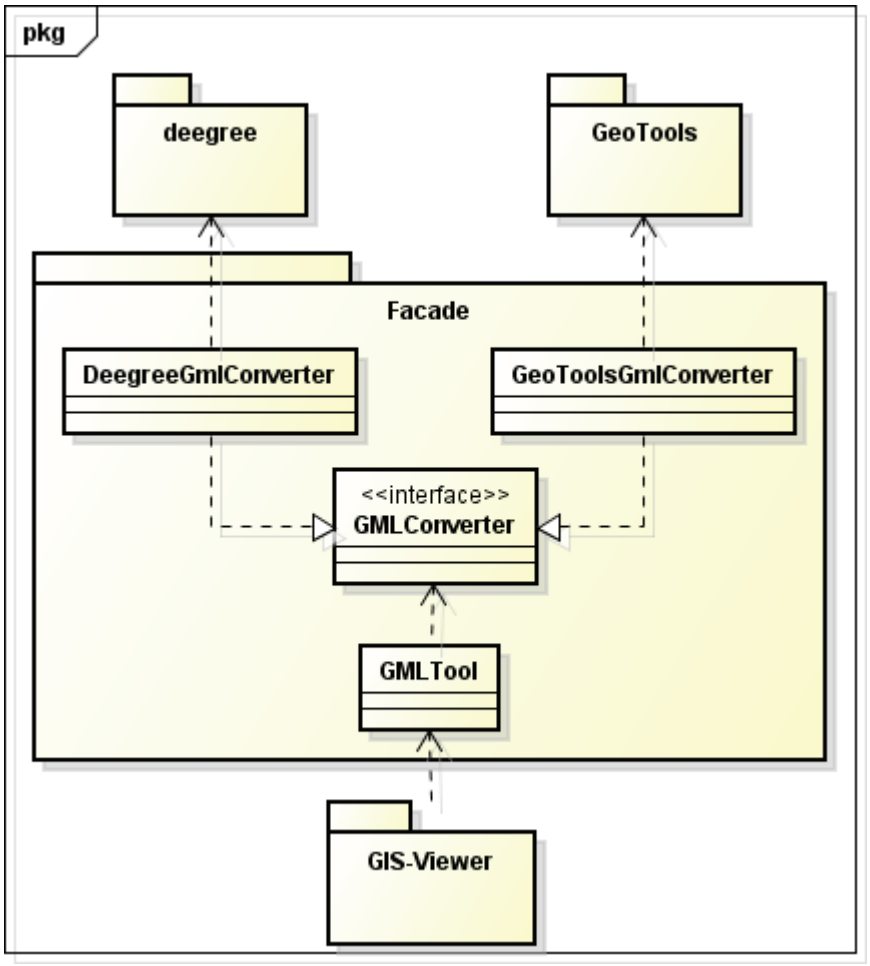

Abbildung 29 Klassendiagramm: Entkopplung der GML-Konvertierung mithilfe des ServiceLoaders

Dieses Beispiel zeigt also, dass die Unterteilung der Anwendungsfassade in Schichten durchaus sinnvoll ist und weitere Vorteile im Rahmen der loseren Kopplung und Kohäsion bewirkt. So können mithilfe des ServiceLoaders Klassen entwickelt werden, die unabhängig von Geoframeworks sind und zur Laufzeit die Implementierungen für geforderte Aufgaben bestimmen. Der GIS-Viewer selbst erhält dann Zugriff auf eine Schnittstelle, die wiederum alle Komplexität verbirgt und nur noch mit Objekten arbeitet, die aus der Java-Standardbibliothek oder JTS stammen.

Das hier vorgestellte Konzept wird erneut in Kapitel *5. Umsetzung* betrachtet und genauer erläutert werden, da es die wichtigsten Schritte aller vorgestellten Entkopplungsmöglichkeiten beinhaltet und ein sehr praxisrelevantes Beispiel ist. So eignet sich das Verfahren ebenfalls für andere Bereiche bei der Arbeit mit Geoframeworks (z.B. die Arbeit mit Shapefiles).

4.3.5. Entkopplung durch eigene Implementierungen

Ein weiteres Mittel zur Entkopplung des GIS-Viewers vom verwendeten Geoframework ist die Entwicklung eigener Lösungen für bestimmte Problemstellungen. Dieses Vorgehen lohnt sich zum Beispiel, wenn die gesuchte Funktionalität trivial ist oder die vorhandenen Ansätze in den Frameworks Nachteile mit sich bringen.

Ein Beispiel hierfür ist die Verwendung von TimeTools. Diese Klasse wird genutzt, um eine Zeichenkette einer ISO-konformen Zeitangabe in ein Datum als Date der Java-Standardbibliothek zu transformieren. Dabei liest TimeTools nur die betreffenden Stellen der Zeichenkette aus und setzt die

Werte zum Erstellen des Datums ein. Diese simple Funktionalität führt also unnötigerweise zu einer Abhängigkeit zu deegree, die in folgender Abbildung dargestellt ist:

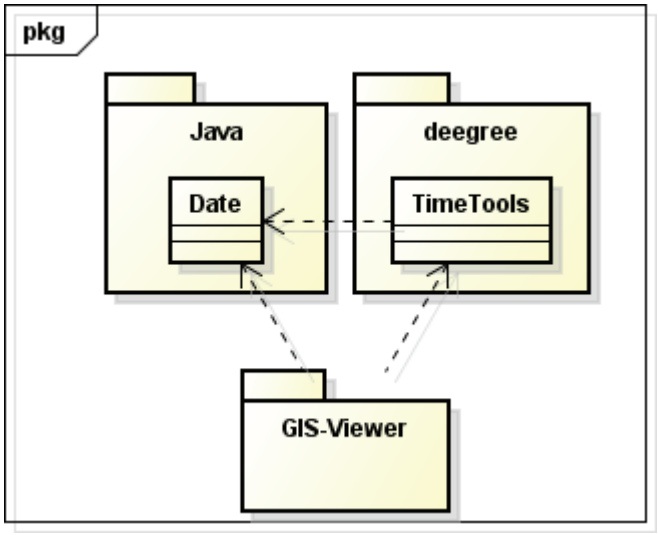

Abbildung 30 Klassendiagramm: Abhängigkeiten TimeTools

Die eigene Implementierung kann dann je nach Anforderungen oder weiteren Einsatzgebieten verschieden aussehen. So könnte beispielsweise eine Klasse entwickelt werden, die von Date erbt und nur über einen modifizierten Konstruktor aufgerufen wird. Dieser erwartet als Eingabe eine Zeichenkette mit Zeitangabe und wandelt diese dann in das zugehörige Datum um. Mit dieser Lösung würde dann auch nur eine Abhängigkeit zur neuen Klasse im GIS-Viewer bestehen.

Die Lösung könnte folgendermaßen aussehen:

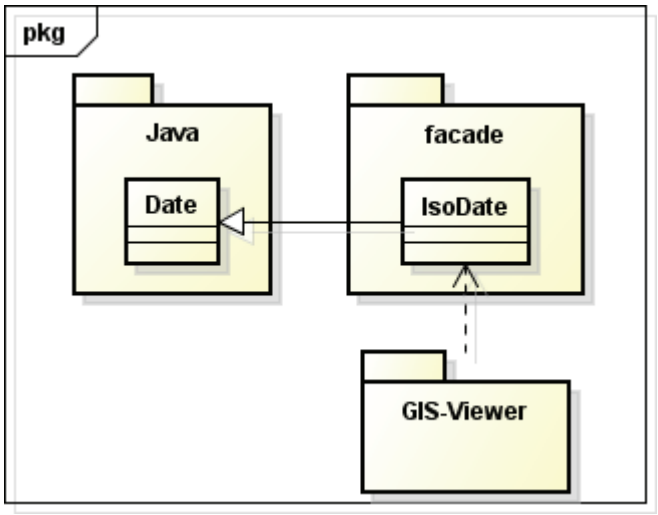

Abbildung 31 Klassendiagramm: Einsatz von IsoDate

Der größte Nachteil eigener Implementierungen ist der verhältnismäßig große Aufwand, der nötig ist, da neben dem Entwurf und der Umsetzung auch Tests durchgeführt werden müssen und zum Beispiel bei Änderungen von Standards die daraus folgenden Anpassungen komplett in der eigenen Verantwortung liegen.

Vorteil einer solchen Lösung ist hingegen die vollständige Unabhängigkeit von Frameworks und die damit verbundene Unbeschränktheit. Das heißt, es kann eine Lösung entwickelt werden, die an genau an die geforderten Aufgaben angepasst ist.

5. Umsetzung

Dieser Abschnitt behandelt die Umsetzung der Lösung aus dem Kapitel *4.3.4. Entkopplung mithilfe des ServiceLoaders* und beschreibt, welche Arbeiten notwendig sind, um die angestrebte Fassade zu erstellen. Außerdem wird dargestellt, welche Probleme oder Besonderheiten mit dem jeweiligen Geoframework aufgetreten sind.

Der Quellcode für die hier gezeigte Lösung befindet sich im Anhang dieser Arbeit.

5.1. Vorbereitende Arbeiten

Vor der Umsetzung spezieller Lösungen innerhalb der Fassade mussten verschiedene andere Schritte durchgeführt werden, um mit der eigentlichen Arbeit beginnen zu können. So ist GeoTools bisher nur teilweise in **profil c/s** integriert, weshalb einige Komponenten fehlen, die für die Arbeit mit GML und Geometrie entscheidend sind. Daher wurde auch als erste Aktion, sichergestellt, dass zumindest lokal alle benötigten Komponenten vorhanden sind und GeoTools genauso wie deegree in ihrem vollen Umfang zur Verfügung stehen.

Weiterhin wurde für die Fassade ein neues Projekt angelegt und in **profil c/s** integriert werden, damit der GIS-Viewer Zugriff auf die neuen Komponenten hat und die Fassade wie jeden anderen Baustein erkennt. Um dies zu erreichen, waren verschiedene Konfigurationen notwendig, die beispielweise festlegen, welche externen Bibliotheken genutzt werden, wo sich das Projekt in SVN einordnet oder welche Targets in der build.xml[15] gesetzt werden müssen.

Das Projekt, das den Namen gisfacade erhielt, musste dann noch eine interne Struktur nach den Vorgaben von **profil c/s** erhalten, damit zum Beispiel Tests vom Produktivcode getrennt sind und auch die umgesetzten Lösungen in thematische Bereiche unterteilt vorliegen. Durch dieses Vorgehen ist gewährleistet, dass der neue Baustein einen übersichtlichen Aufbau hat.

Mit dem Abschluss dieser Arbeiten konnten die eigentlichen Lösungen umgesetzt werden, von denen im folgenden Abschnitt jene vorgestellt wird, die die wesentlichen Schritte verschiedener Entkopplungen enthält und daher als Muster für ein allgemeines Vorgehen angesehen werden kann.

5.2. Erstellung einer Anwendungsfassade

5.2.1. Vorbereitung für den Einsatz mehrerer Geoframeworks und des ServiceLoaders

Die Umwandlung von standardisierten GML, das in Form einer Zeichenkette vorliegt, in eine Geometrie ist ein entscheidender Vorgang bei der Auswertung einer Anfrage an einen WFS, weil nur so der GIS-Viewer die Elemente erhält, die in der Karte dargestellt werden sollen. Da es sich also um ein grundlegendes Problem bei der Arbeit mit Geodaten handelt, liefern deegree und auch GeoTools Werkzeuge, um eine solche Transformation vornehmen zu können.

Jedoch unterschieden sich die Lösungsansätze in beiden Fällen sehr und auch die Ergebnisse sind verschiedene Objekte, die jeweils auf ihre Art eine Geometrie repräsentieren. Daher ist es notwendig, dass nicht nur die eigentliche Umwandlung von GML zu Geometrie in der Fassade vorgenommen wird, sondern die Umwandlung von deegree- oder GeoTools-Geometrie zur angestrebten JTS-Geometrie ebenfalls durchgeführt wird.

[15] Mithilfe der build.xml wird die Erstellung eines Softwareprojektes unter Apache Ant gesteuert. [wikAnt]

Aufgrund dieser Anforderung und der Voraussicht auf den Einsatz des ServiceLoaders bietet sich daher die Definition eines Interfaces an, sodass die Lösungen für deegree und GeoTools an einheitliche Methodenaufrufe und Rückgabeparameter, also eben die geforderte JTS-Geometrie, gebunden sind. In folgender Abbildung ist zu sehen, welche Methoden durch das Interface definiert werden:

Abbildung 32 Interface GmlConverter

Neben der Methode für die Transformation, wird noch eine weitere Methode im Interface festgelegt, sodass die Identifizierung eines Services möglich ist, da dieser gezwungen ist, einen Namen zu besitzen. Dies führt dazu, dass ein Service mithilfe seines Namens nicht nur während der Laufzeit bestimmt werden kann, sondern auch eine Auswahl zu treffen leichter möglich ist. Mit dieser Methode wird also eine Schwäche der ServiceLoaders ausgeglichen, da dieser selbst nicht die Möglichkeit bietet, die Bezeichnung der konkreten Implementierung und zugehörige Versionsnummer herauszufinden. Mit getName könnte eine Extension hingegen selbst diese Informationen liefern. So zum Beispiel könnte eine Extension, die mit deegree arbeitet, als Rückgabewert der Methode folgenden String liefern: „GmlConverter, deegree 2.2“.

5.2.2. Einsatz von deegree und GeoTools

Nach der Definition von GmlConverter werden die jeweiligen Lösungen für deegree und GeoTools entwickelt, wobei beide Klassen das Interface implementieren müssen. Die Lösungen, wie in Abbildung 27 und Abbildung 28 zu sehen, sollen in diesem Abschnitt genauer erläutert werden, um die Unterschiede darzustellen, die im Endeffekt durch die Fassade verborgen werden.

DeegreeGmlConverter

Um mit deegree aus einem GML-String eine Geometrie zu erzeugen, wird der String zunächst in ein Objekt der Klasse XMLFragment übergeben, sodass statt der einfachen Zeichenkette ein Objekt vorliegt, das ein Element eines XML-Dokumentes präsentiert. Durch dieses Vorgehen wird das Laden der Daten für die in deegree verwendeten Parser möglich.

Das Wurzelelement des erstellten XMLFragments ist die übergebene Geometrie, die sich mithilfe des GMLGeometryAdapter dann einfach in eine deegree-Geometrie umwandeln lässt. Zum Abschluss wird diese Geometrie dann noch mit dem JTSAdapter als JTS-Geometrie exportiert, sodass das endgültige Ergebnis vorliegt. Der Ablauf der Methode getGeometryFromGML ist in folgender Abbildung zusammengefasst:

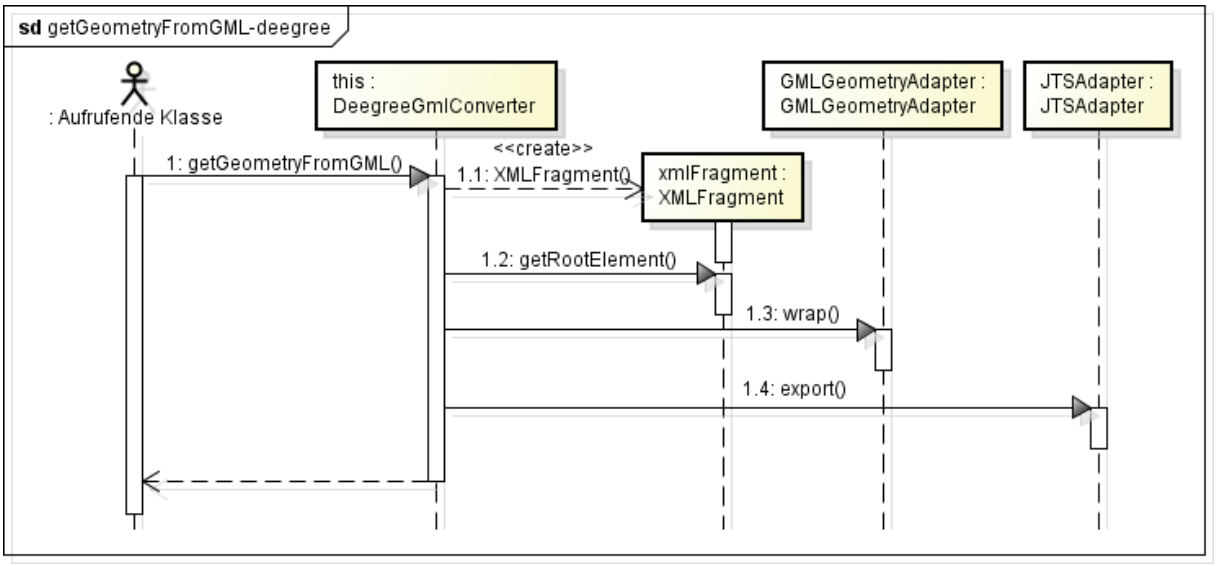

Abbildung 33 Sequenzdiagramm: getGeometryFromGML - deegree

Eine Besonderheit, die beachtet werden muss, ist die Ersetzung aller Einträge von „gml:PolygonPatch“ mit „gml:Polygon“ vor der Übergabe des Strings an XMLFragment. Dieser Schritt muss durchgeführt werden, da momentan im GIS-Viewer eine alte Version von deegree (Version 2.2) verwendet wird, die noch Polygon und nicht PolygonPatch erwartet. Sollte also ein Wechsel auf eine neuere Version von deegree vorgenommen werden, muss dieser Schritt entfernt oder eine Klasse hinzugefügt werden, die mithilfe der neuen Version eine Lösung anbietet. In diesem Fall ließen sich dann beide Varianten durch ihre Namen unterscheiden, die durch den Aufruf von getName verfügbar sind.

GeoToolsGmlConverter

Unter GeoTools wird zur Umwandlung von GML in eine Geometrie ein weiteres Framework benötigt. Es handelt sich um SAX, das erlaubt die Handhabung auftretender Ereignisse innerhalb eines Dokuments (z.B. das Auftreten eines bestimmten Elements) selbst zu steuern. Dies ermöglicht dann zum Beispiel, eine Geometrie aus einem Dokument zu filtern und direkt als Ergebnis zurückzugeben.

Zunächst muss dazu ein Klasse erzeugt werden, die von DefaultHandler aus SAX erbt und GMLHandlerJTS aus GeoTools implementiert, damit der spezifische Handler von den Komponenten beider Frameworks genutzt werden kann. Außerdem wird auf diesem Wege festgelegt, welche Ergebnisse am Ende geliefert werden sollen. Somit werden alle anderen Aufgaben den Frameworks überlassen.

Der umgesetzte Handler ist in folgender Abbildung zu sehen:

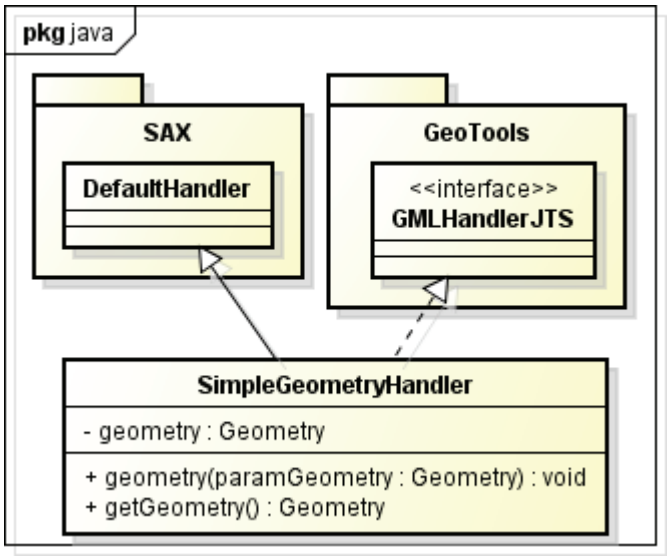

Abbildung 34 Klassendiagramm: SimpleGeometryHandler

Vom selbstdefinierten SimpleGeometryHandler wird eine Instanz an GMLFilterGeometry, einer Implementierung für einen XML-Filter, übergeben, sodass zunächst ein Filter vorliegt, dessen Ergebnisse mithilfe von SimpleGeometryHandler immer den Anforderungen entsprechen[16]. Dieser Filter wird jetzt an GMLFilterDocument übergeben, welches den eigentlichen ContentHandler zur Auswertung von GML liefert.

Um die Umwandlung durchzuführen, wird ein XMLReader erzeugt. Dieser erhält dann den ContentHandler in Form des angelegten GMLFilterDocument und GML als InputSource. Die Erstellung einer InputSource aus dem gegebenen String oder anderen Daten dient dazu, auch komplexere Eingaben innerhalb eines Objektes zu kapseln. Nach Ausführung der Umwandlung kann dann das Ergebnis über die Instanz des SimpleGeometryHandler abgefragt werden.

Der Ablauf der Methode getGeometryFromGML für GeoTools ist in folgender Abbildung zusammengefasst:

[16] Siehe [GtGeom]

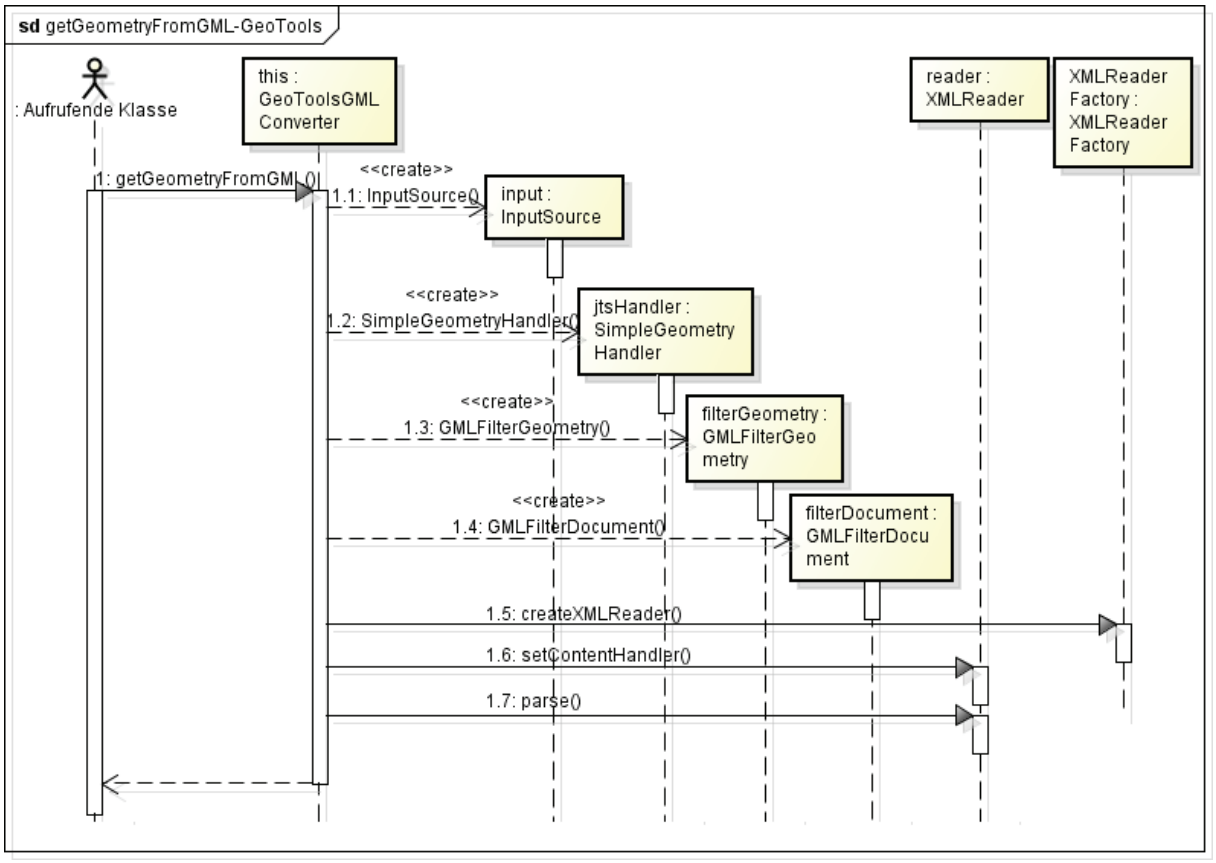

Abbildung 35 Sequenzdiagramm: getGeometryFromGML - GeoTools

Um die logische Zusammengehörigkeit von SimpleGeometryHandler mit GeoToolsGmlConverter erkennbar zu machen und eine bessere Kapselung zu erreichen, wird SimpleGeometryHandler als innere Klasse eingesetzt.

Besonderheiten wegen der verwendeten Version von GeoTools (Version 2.7.2) treten in diesem Beispiel nicht auf. Außerdem gibt es wie bei deegree keine Probleme mit JTS (Version 1.7).

Vergleich der Lösungen von deegree und GeoTools

Das eigentliche Problem bei der Arbeit mit mehreren Geoframeworks ist nicht, dass die Implementierung für ein und dasselbe Problem sehr verschieden aussieht. Viel problematischer ist hingegen, wenn sich die Ergebnisse am Ende unterscheiden.

So sind die beiden Implementierungen von GmlConverter im vorgestellten Stand ohne Kapselung durch eine weitere Fassade oder Einsatz des ServiceLoaders nutzbar und lassen sich somit auch testen. Dabei fällt auf, dass Geometrien, die aus GML mit deegree oder GeoTools erzeugt werden, verschiedene Orientierungen haben. Das heißt, vor der Nutzung der Geometrie muss zunächst noch sichergestellt werden, dass eine einheitliche Orientierung vorliegt. Jedoch ist es leicht, solche Unterschiede festzustellen und für die Lösung solcher Probleme gibt es entsprechende Werkzeuge, die zum Beispiel durch Module in **profil c/s** bereitgestellt werden.

5.2.3. Einsatz des ServiceLoaders

Die Einführung des ServiceLoaders kann im Rahmen der Implementierungen von GmlConverter und im Allgemeinen aus verschiedenen Gründen interessant sein. So lässt sich zum Beispiel mithilfe eines Plug-In-Mechanismus zur Laufzeit festlegen, welche konkrete Implementierung genutzt werden soll.

Im umgekehrten Fall, wenn es nicht relevant ist, welches Framework verwendet wird, ist es dann dem ServiceLoader überlassen, eine passende Variante einzusetzen. Das heißt, für den Entwickler und Anwender spielen die konkreten Abläufe keine Rolle mehr.

Der größte Vorteil dieses Verfahren ist jedoch, dass keine Anpassungen im Quellcode vorgenommen werden müssen, wenn Implementierungen ausgetauscht, hinzugefügt oder entfernt werden.

Wie in der Abbildung 29 dargestellt, wird für die Zusammenführung der verschiedenen Lösungen die Klasse GmlTool eingesetzt. Diese Klasse bildet die tatsächliche Fassade und damit auch Schnittstelle nach außen, die dann zum Beispiel vom GIS-Viewer angesprochen werden kann.

GmlTool ist hierbei als Klasse vorgesehen, die kein Objekt repräsentiert und stattdessen nur eine Menge an Methoden liefert, die zur Arbeit mit GML gebraucht werden. In folgender Abbildung ist zu sehen, welchen Umfang an Methoden die Klasse für die Konvertierung von GML zu Geometrie bereitstellen muss:

Abbildung 36 GmlTool

Der Zugriff auf GmlTool ist also so einfach wie nur möglich gehalten und für die Anforderungen dieses Beispiels reicht es aus, das nur der String übergeben wird, welcher GML enthält. Weiterhin hat die Klasse einen Konstruktor, der von außen nicht sichtbar ist, sodass kein Exemplar vom GmlTool erzeugt werden kann. Auf diese Möglichkeit wird verzichtet, da der Aufruf der statischen Methode auch ohne ein Objekt erfolgen kann.

Innerhalb der Methode getGeometryFromGml erfolgt als erstes das Laden aller Implementierungen des Services GmlConverter. Die geladenen Klassen können danach über den ServiceLoader aufgerufen werden. Um nun die Methode einer Implementierung zu nutzen, kann eine beliebige Implementierung ausgewählt und mithilfe von GmlConverter aufgerufen werden. Die erhaltene Geometrie kann dann zurückgegeben werden und bei Bedarf ist es möglich, auszugeben, welches Geoframework für die durchgeführte Aufgabe verwendet wurde.

In folgender Abbildung ist dargestellt, welche Aufrufe in der Methode getGeometryFromGml erfolgen:

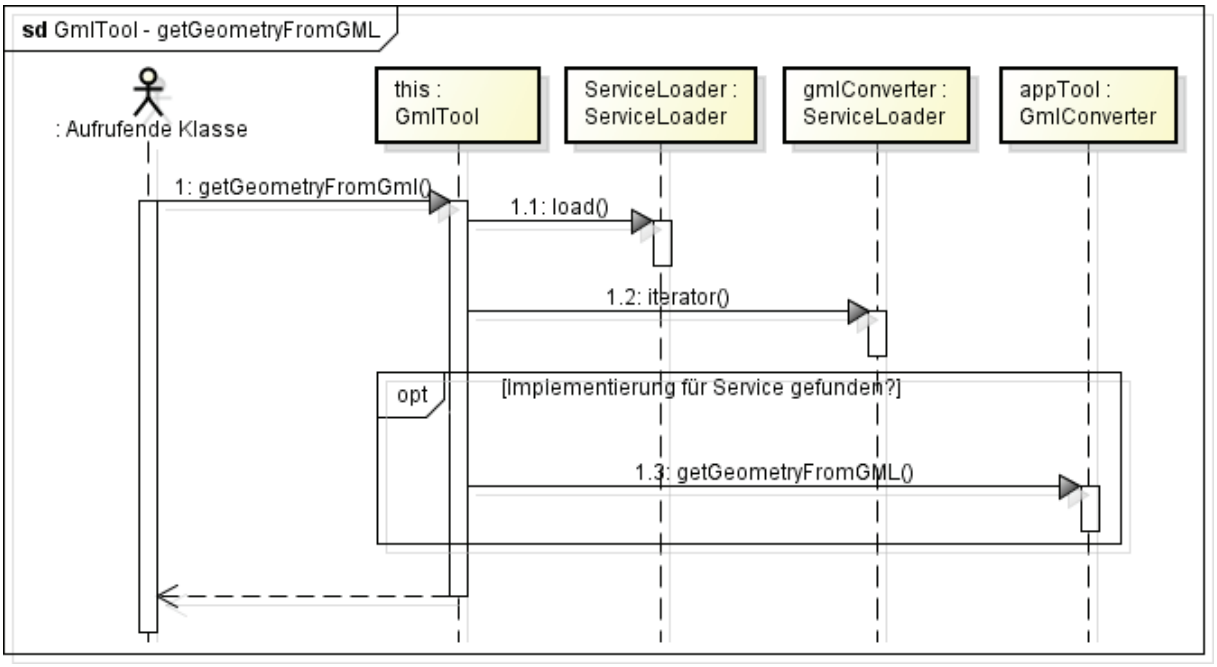

Abbildung 37 Sequenzdiagramm: GmlTool - getGeometryFromGML

Damit der ServiceLoader auch die gewünschten Implementierungen finden kann, muss nun noch die build.xml des Bausteins angepasst werden. Dies ist notwendig, damit beim Erzeugen des Jar-Files im Ordner Meta-inf ein Unterordner services erstellt wird. Dieser Ordner wiederum muss eine Datei enthalten, die als Namen den vollständigen Pfad der Klasse erhalten muss, die mithilfe des ServiceLoaders andere Klassen laden will. Innerhalb der Datei werden nur die Pfade zu den vorhandenen Implementierungen gespeichert, damit diese zur Laufzeit gefunden werden können.

Nach Abschluss der Anpassungen an der build.xml ist dann GmlTool einsatzfähig, findet die Implementierungen GeoToolsGmlConverter und DeegreeGmlConverter und liefert das gewünschte Ergebnis.

Ein Problem, das der Einsatz des ServiceLoaders bewirkt, ist eine Minderung der Performance, da das Suchen der betreffenden Klassen bei jedem Aufruf einen gewissen Zeitaufwand erfordert und sich mit zunehmenden Aufrufen dann immer deutlicher bemerkbar macht.

In folgender Abbildung ist zu sehen, dass die Nutzung des ServiceLaoders zwar nicht den Großteil der Bearbeitungszeit ausmacht, jedoch kann der Einfluss nicht unberücksichtigt bleiben. Dabei ist im Diagramm dargestellt, welche Bearbeitungszeiten auftreten, wenn mehrere Strings nacheinander in dazugehörige Geometrien umgewandelt werden.

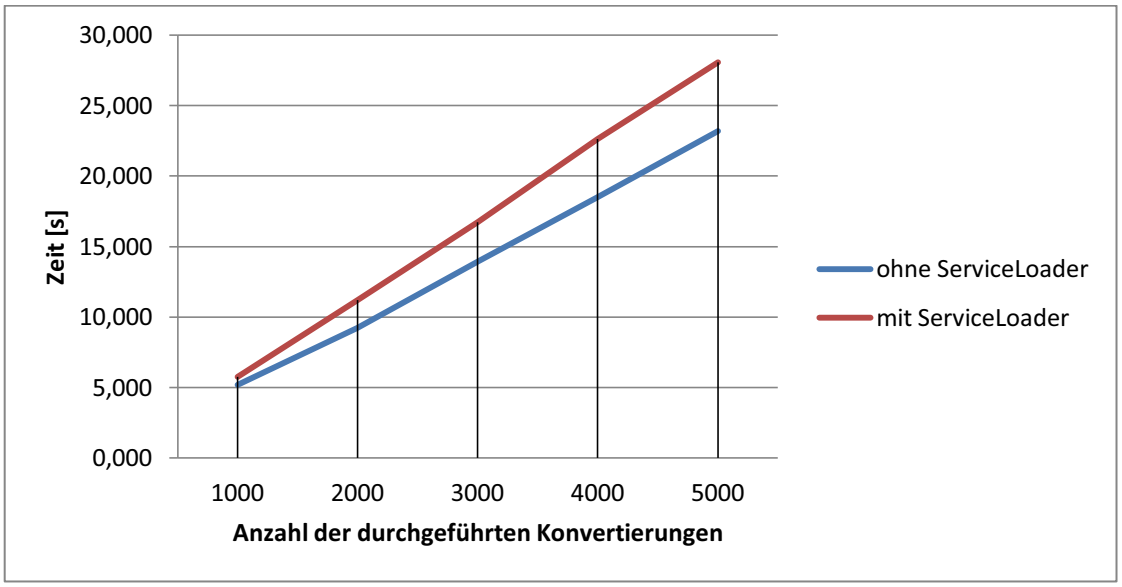

Abbildung 38 Beeinträchtigung der Performance durch Einsatz des ServiceLoaders

Um zu umgehen, dass die zutreffenden Implementierungen immer wieder gesucht werden, könnte die Klasse GmlTool erweitert werden, sodass die Suche der Implementierungen nur beim ersten Aufruf der Klasse durchgeführt wird. Danach könnte auf die gespeicherten Ergebnisse dieser einmaligen Suche zugegriffen werden und somit wäre der zusätzliche Zeitaufwand nur bemerkbar, wenn die Instanz der Klasse erneut angelegt wird.

Weiterhin sind keine zusätzlichen Unterschiede zwischen den Aufruf mit oder ohne ServiceLoader zu bemerken, weshalb sich der Einsatz der gezeigten Lösung auf jeden Fall lohnt und auch für andere Gebiete empfiehlt.

5.3. Ergebnisse der Umsetzung

Im Rahmen der Umsetzung wurden erste Bestandteile der Fassade eingerichtet und damit einige der Anwendungsfälle gelöst, die in Kapitel *3.3. Anforderungen an ein Geoframework in profil c/s* formuliert wurden. In folgender Abbildung ist dargestellt, welcher Stand dabei erreicht wurde. Anwendungsfälle, die in der Abbildung grün dargestellt sind, wurden den Anforderungen entsprechend umgesetzt, gelb dargestellte Anwendungsfälle müssen noch weiter bearbeitet werden und rot dargestellte Anwendungsfälle wurden noch nicht bearbeitet.

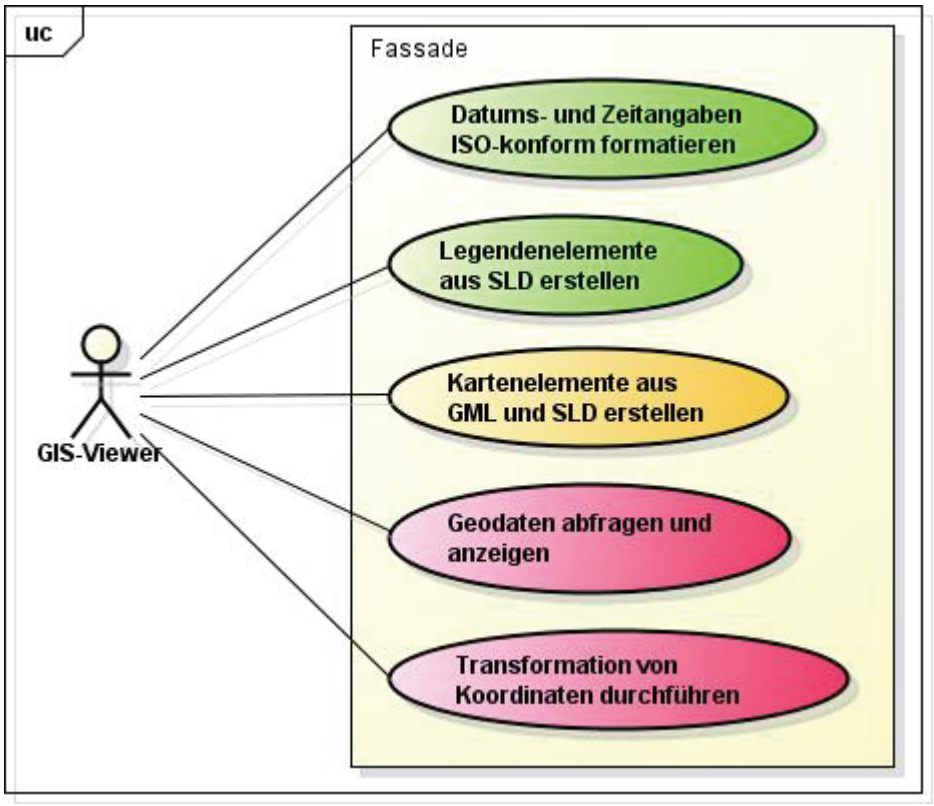

Abbildung 39 Use-Case-Diagramm: Ergebnisse der Umsetzung

Aus technischer Sicht sind die Ergebnisse der Entkopplung in folgender Abbildung zusammengefasst. So ist in dem Diagramm dargestellt, welche Abhängigkeiten zu deegree-Klassen ganz und teilweise gelöst wurden sowie welche Abhängigkeiten weiterhin bestehen.

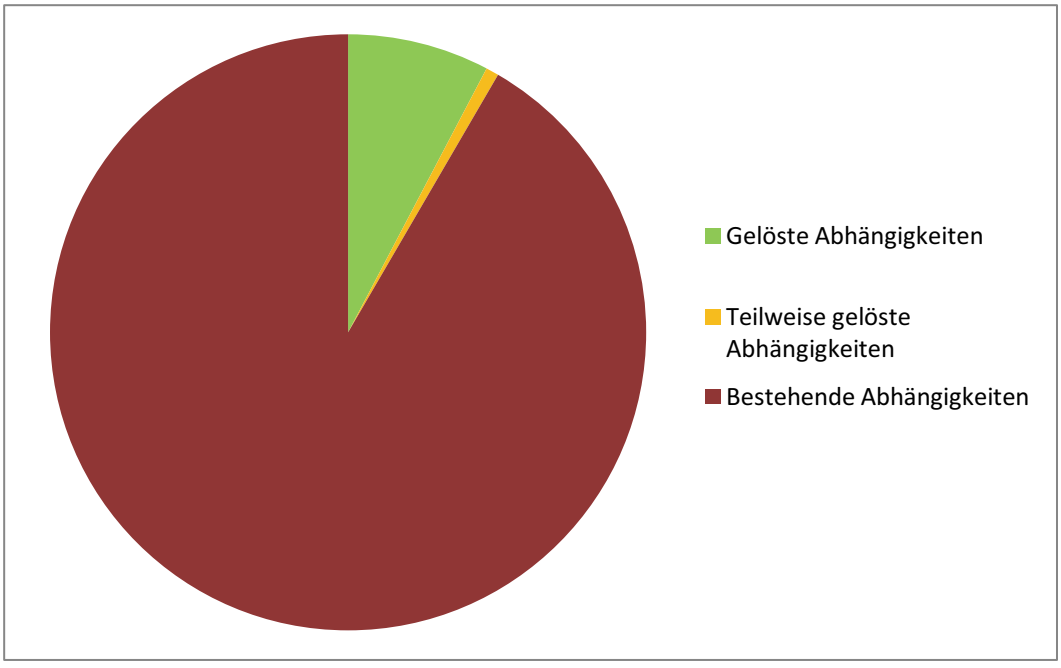

Abbildung 40 Abhängigkeiten zu deegree nach der Umsetzung

6. Zusammenfassung

6.1. Fazit

Bei einer Betrachtung der gestellten Anforderungen aus Kapitel *3.4.1. Nichtfunktionale Anforderungen* und Untersuchung der umgesetzten Lösungsentwürfe in Bezug auf diese lässt sich feststellen, dass die Ziele der Entkopplung erfüllt werden konnten. So wurde mithilfe des ExtensionPoint-Mechanismus und der Anwendungsfassade eine flexible Zwischenschicht geschaffen, die sich auf JTS als Metamodell stützt und für Erweiterungen in verschiedenster Form offen ist. Neue Komponenten können nach dem aufgezeigten Muster aus Kapitel *5. Umsetzung* in das System integriert werden und die Erweiterung des Systems durch neue Extensions, die an bereits vorhandene Extension Points anschließen, ist ebenfalls möglich.

Die Änderungen des Systems durch Anwendung von ExtensionPoint-Mechanismus und Anwendungsfassade sind in folgender Abbildung zusammengefasst:

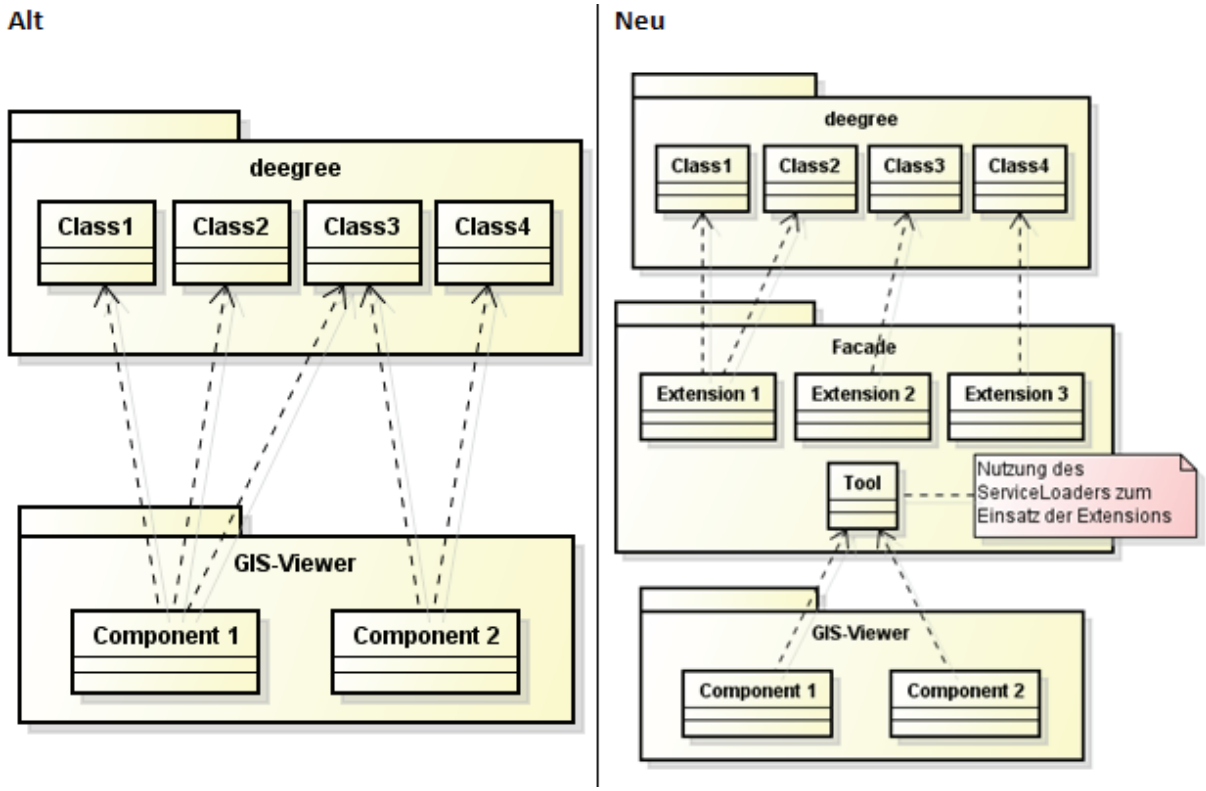

Abbildung 41 Gegenüberstellung des Systems ohne und mit Fassade

Weiterhin funktioniert der GIS-Viewer nach Umsetzung der Entwürfe verlässlich und in manchen Bereichen ließen sich auftretende Unterschiede zwischen den Frameworks durch die nun vorhandene Vergleichsmöglichkeit und mithilfe von Tests schnell bestimmen und damit auch berücksichtigen (z.B. die Korrektur der auftretenden Orientierung von Polygonen).

Im Rahmen der Entkopplung wurde also mithilfe der Anwendungsfassade eine einheitliche und vereinfachte Schnittstelle geschaffen, die jegliche Komplexität vor dem Entwickler verbirgt und stattdessen Funktionalitäten für die anzutreffenden Anwendungsfälle liefert. Diese Veränderung lässt sich auch mit den vorgestellten Kennzahlen aus Kapitel *3.4.2. Maßzahlen zur Bewertung der Entkopplung* feststellen. Dazu ist in folgenden Tabellen dargestellt, wie sich die Werte von Coupling

Between Objects, Number Of Services und Number Of Accesses beim Aufruf mit oder ohne Fassade verhalten:

Konvertierung von GML zu Geometrie

	Alt	Neu
CBO	3	1
NOS	3	1
NOA	5	1

Tabelle 2 CBO, NOS und NOA - Konvertierung von GML zu Geometrie

Legendenelemente erstellen

	Alt	Neu
CBO	4	1
NOS	14	2
NOA	18	2

Tabelle 3 CBO, NOS und NOA – Legendenelemente erstellen

Vorschau von Styles anzeigen

	Alt	Neu
CBO	5	1
NOS	5	2
NOA	9	2

Tabelle 4 CBO, NOS und NOA – Vorschau von Styles anzeigen

Zeitangabe in Date umwandeln

	Alt	Neu
CBO	1	1
NOS	1	1
NOA	1	1

Tabelle 5 CBO, NOS und NOA – Zeitangabe in Date umwandeln

Zu diesen Anwendungsfällen lassen sich folgende Werte für die Abhängigkeiten innerhalb der Fassade nach deegree feststellen:

	CBO	NOS	NOA
Konvertierung von GML zu Geometrie	5	4	9
Legendenelemente erstellen	5	14	8
Vorschau von Styles anzeigen	3	3	3
Zeitangabe in Date umwandeln	0	0	0

Tabelle 6 CBO, NOS und NOA – Anwendungsfassade zu deegree

Diese Werte lassen sich jedoch nicht sinnvoll mit dem Ausganszustand gegenüberstellen, da zum Beispiel zusätzliche Komponenten für den Umgang mit JTS gebraucht werden und durch das Zusammenführen von Funktionalitäten manche Abhängigkeiten hinzugekommen sind oder andere

wiederum entfernt wurden. Aufgrund dieser neuen Verhältnisse lässt sich also in dieser Hinsicht kein direkter Vergleich ziehen.

Zusammenfassend lässt sich folglich feststellen, dass mit dem vorgestellten, allgemeinen Lösungsentwurf und den dazugehörigen Detailentwürfen eine Vorlage für weitere Arbeiten gegeben ist, die alle gestellten Anforderungen erfüllt. Dabei beschränkt sich die Anwendung der Fassade nicht nur auf den GIS-Viewer, sodass die Ergebnisse dieser Arbeit auch von **profil c/s** im Allgemeinen genutzt werden können.

6.2. Ausblick

In Zukunft könnte die Entkopplung des GIS-Viewers und der damit verbundene Ausbau der Fassade fortgeführt werden, sodass an weiteren Stellen der Einsatz von GeoTools möglich wird. Weiterhin könnte eines der vorstellten Geoframeworks aus Kapitel *2.1.2. Vorhandene Ansätze einer Domänenschicht nach den Spezifikationen der OGC* eingebracht und damit die Fassade hinzu GeoApi gearbeitet werden. Somit könnte dieser Standard zu einer Basis für die Arbeit in **profil c/s** werden, wenn es um Aufgaben des Bereiches GIS geht. Durch dieses Vorgehen könnten dann weitere Geoframeworks in das System eingeführt werden. Einige dieser Frameworks, die ähnlich wie deegree oder GeoTools den Standard nicht umsetzen, müssten zwar durch eigene Implementierungen angepasst werden, dafür könnten aber andere Frameworks, die GeoApi von sich aus realisieren, ohne weitere Änderungen hinzugefügt werden. Dieser Zusatz würde also einerseits die Erweiterung der Fassade durch das Einbringen neuer Frameworks erleichtern und andererseits die Fassade um ein standardisiertes Modell für den Umgang mit Geodaten ergänzen.

Trotz der Einführung der Fassade gibt es immer noch bestimmte Probleme, die beim Arbeiten mit den Geoframeworks berücksichtigt werden müssen. So ist beim Austausch oder Aktualisieren von Geoframeworks darauf zu achten, ob die verwendeten, externen Bibliotheken kompatibel sind und keine Konflikte durch die verschiedenen Versionen entstehen. In erster Linie sollte hierbei darauf geachtet werden, dass die verwendeten Geoframeworks mit der vorliegenden Version von JTS zusammenarbeiten können, da diese Bibliothek übergreifend eingesetzt wird.

Dennoch hat sich gezeigt, welche Möglichkeiten die Anwendungsfassade und der ExtensionPoint-Mechanismus für weitere Arbeiten bieten, weshalb sich auch der Aufwand für das Ausbauen dieser Lösung längerfristig lohnen wird. So könnte das in dieser Arbeit vorgestellte Vorgehen auch in anderen Bausteinen von **profil c/s** zum Einsatz kommen, sodass Module bereitgestellt werden, die vom nutzenden Entwickler kein Wissen über interne Vorgänge verlangen. Außerdem würde das gesamte System so stabiler gegenüber Änderungen werden und damit weniger Wartungsarbeiten erfordern. Ferner wird auch die Coderedundanz gemindert werden. Im Endeffekt könnte eine neue Schicht in **profil c/s** entstehen, die mehrere vereinfachte Schnittstellen für verschiedene Aufgabengebiete liefert.

Glossar

API	Application Programming Interface, Programmierschnittstelle
GIS	Geoinformationssystem, Geografisches Informationssystem
GML	Geography Markup Language
GUI	Graphical User Interface, grafische Benutzeroberfläche als Teil einer Software zur Interaktion zwischen Nutzer und Anwendung
ISO	International Organization for Standardization
Java	Objektorientierte Programmiersprache
JTS	JTS Topology Suite, Bibliothek mit räumlichen Objektmodell und zugehörigen Funktionen
OGC	Open Geospatial Consortium
OWS	Open Web Service
profil	Programmsystem zur Unterstützung der Förderung in der Landwirtschaft
profil c/s	profil Client/Server
SAX	Simple API for XML
SLD	Styled Layer Discriptor
SVN	Apache Subversion ist eine Software zur Versionsverwaltung
Xubuntu	Abwandlung von Ubuntu und damit eine Linux-Distribution, die auf Debian basiert
XML	Extensible Markup Language, Auszeichnungssprache

Literaturverzeichnis

[Bil01] R. Bill, Lexikon der Geoinformatik, Heidelberg: Herbert Wichmann Verlag, 2001.

[deegr] deegree, „deegree,“ [Online]. Available: http://www.deegree.org/. [Zugriff am 03 07 2012].

[degDkG] deg, „Detailkonzept GIS-Viewer,“ Neubrandenburg, 2012.

[deL02] N. de Lange, Geoinformatik in Theorie und Praxis, Berlin: Springer, 2002.

[FosFG] FOSSGIS e.V., „FreeGIS.org,“ [Online]. Available: http://freegis.org/database/viewobj?obj=1220. [Zugriff am 02 07 2012].

[Gam04] E. Gamma, R. Helm, R. Johnson und J. Vlissides, Entwurfsmuster, München: Addison-Wesley Verlag, 2004.

[GeoTk] „Geotoolkit.org,“ [Online]. Available: http://www.geotoolkit.org/. [Zugriff am 10 07 2012].

[GeoTls] GeoTools, „About GeoTools,“ [Online]. Available: http://geotools.org/about.html. [Zugriff am 04 07 2012].

[Gle74] G. J. Myers, Reliable Software through Composite Design, New York: Mason and Lipscomb Publishers, 1974.

[Glo05] W. Globke, „Software-Metriken,“ 2005.

[GtGeom] GeoTools, „GeoTools - Geometry,“ [Online]. Available: http://docs.geotools.org/latest/userguide/library/xml/geometry.html. [Zugriff am 15 08 2012].

[Hen08] M. Hennig und H. Seeberger, „Einführung in den "Extension Point"-Mechanismus von Eclipse,“ *JavaSpektrum,* pp. 19-24, 01 2008.

[IEEE90] IEEE: Standard Glossary of Software Engineering Terminology, New York, 1990.

[IsoTc211] ISO, „ISO/TC 211,“ [Online]. Available: http://www.isotc211.org/. [Zugriff am 05 07 2012].

[IwCBO] ITWissen.info, „CBO,“ [Online]. Available: http://www.itwissen.info/definition/lexikon/CBO-coupling-between-objects.html. [Zugriff am 02 07 2012].

[JavaSL] Oracle, „Java™ Platform, Standard Edition 6,“ [Online]. Available: http://docs.oracle.com/javase/6/docs/api/java/util/ServiceLoader.html. [Zugriff am 09 08 2012].

[NetCdf] „NetCDF Java,“ [Online]. Available: http://www.unidata.ucar.edu/software/netcdf-java/. [Zugriff am 10 07 2012].

[OgcGApi] Open Geospatial Consortium, „GeoAPI 3.0 Implementation Standard,“ 2011.

[OgcGML] OGC, „Geography Markup Language,“ [Online]. Available: http://www.opengeospatial.org/standards/gml. [Zugriff am 03 07 2012].

[OgcQN] OGC, „OGC Glossary,“ [Online]. Available: http://www.opengeospatial.org/ogc/glossary/q. [Zugriff am 09 07 2012].

[OgcSLD] OGC, „Styled Layer Descriptor,“ [Online]. Available: http://www.opengeospatial.org/standards/sld. [Zugriff am 03 07 2012].

[OodMed] OODesign, „Mediator Pattern,“ [Online]. Available: http://www.oodesign.com/mediator-pattern.html. [Zugriff am 17 07 2012].

[Osg] OSGeo, „About the Open Source Geospatial Foundation,“ [Online]. Available: http://www.osgeo.org/content/foundation/about.html. [Zugriff am 02 07 2012].

[OsgDgr] OSGeo, „deegree - OSGeo-Live 5.5,“ [Online]. Available: http://live.osgeo.org/de/overview/deegree_overview.html. [Zugriff am 03 07 2012].

[OsgECP] OSGeo, „The OSGeo Education and Curriculum Project,“ [Online]. Available: http://www.osgeo.org/education. [Zugriff am 02 07 2012].

[OsgGDC] OSGeo, „OSGeo - Public Geospatial Data Committee,“ [Online]. Available: http://www.osgeo.org/geodata. [Zugriff am 02 07 2012].

[OsgPIS] OSGeo, „GeoNetwork opensource Project Info Sheet,“ [Online]. Available: http://www.osgeo.org/geonetwork. [Zugriff am 02 07 2012].

[Proj4] „PROJ.4 - Cartographic Projections Library,“ [Online]. Available: http://trac.osgeo.org/proj/. [Zugriff am 10 07 2012].

[SAX] „SAX,“ [Online]. Available: http://www.saxproject.org/. [Zugriff am 31 07 2012].

[wikAnt] wiki, „Wikipedia,“ [Online]. Available: http://de.wikipedia.org/wiki/Apache_Ant. [Zugriff am 07 08 2012].

[wikFac] wiki, „Wikipedia,“ [Online]. Available: http://en.wikipedia.org/wiki/Facade_pattern. [Zugriff am 16 07 2012].

[wikKop] wiki, „Wikipedia,“ [Online]. Available: http://de.wikipedia.org/wiki/Kopplung_(Softwareentwicklung). [Zugriff am 10 07 2012].

[wikRrf] wiki, „Wikipedia,“ [Online]. Available:

http://de.wikipedia.org/wiki/R%C3%BCckruffunktion. [Zugriff am 05 07 2012].

[Zül12] H. Züllighoven, W.-G. Bleek und G. Gryczan, „http://www.inf.fu-berlin.de," [Online]. Available: http://www.inf.fu-berlin.de/lehre/WS01/SWT/VL03Ueberblick.pdf. [Zugriff am 09 08 2012].

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang

GmlConverter.java:

```
package DE.data_experts.profi.gisfacade.gmlencoding.tool;
// ---- Importe ------------------------------------------------
import com.vividsolutions.jts.geom.Geometry;

public interface GmlConverter {
      // ---- public Methoden ---------------------------------------
      public String getName();
      public Geometry getGeometryFromGML( String gmlString );
}
```

GmlTool.java:

```
package DE.data_experts.profi.gisfacade.gmlencoding.tool;
//---- Importe ------------------------------------------------
import java.util.Iterator;
import java.util.ServiceLoader;
import com.vividsolutions.jts.geom.Geometry;

public class GmlTool {
      // ---- Konstruktoren -----------------------------------------
      private GmlTool() {}

      // ---- public Methoden ---------------------------------------
      public static Geometry getGeometryFromGml( String gmlString ) {
            ServiceLoader<?> gmlConverter = ServiceLoader.load(
            GmlConverter.class );
            Iterator<GmlConverter> it = (Iterator<GmlConverter>)
            gmlConverter.iterator();
            if ( it.hasNext() ) {
                  GmlConverter appTool = it.next();
                  return appTool.getGeometryFromGML( gmlString );
            }
            return null;
      }
}
```

DeegreeGmlConverter.java:

```
package DE.data_experts.profi.gisfacade.gmlencoding.worker;
// ---- Importe ------------------------------------------------
import java.io.StringReader;
import org.deegree.framework.xml.XMLFragment;
import org.deegree.model.spatialschema.GMLGeometryAdapter;
import org.deegree.model.spatialschema.Geometry;
import org.deegree.model.spatialschema.GeometryException;
import org.deegree.model.spatialschema.JTSAdapter;
import DE.data_experts.profi.gisfacade.gmlencoding.tool.GmlConverter;
import DE.data_experts.util.degexception.DegRTException;

public class DeegreeGmlConverter implements GmlConverter {
      // ---- public Methoden ---------------------------------------
      @Override
      public String getName() {
            return "GMLConverter -> deegree";
      }
```

```
	@Override
	public com.vividsolutions.jts.geom.Geometry getGeometryFromGML(
	String inputString ) {
		String gmlString = inputString.replaceAll( "gml:PolygonPatch",
		"gml:Polygon" );
		XMLFragment xmlFragment = null;
		Geometry geometry = null;
		try {
			xmlFragment = new XMLFragment( new StringReader(
			gmlString ), "http://www.data-experts.de" );
			geometry = GMLGeometryAdapter.wrap(
			xmlFragment.getRootElement(), null );
		}
		catch ( Exception e ) {
			throw new DegRTException( e );
		}
		try {
			return JTSAdapter.export( geometry );
		}
		catch ( GeometryException e ) {
			throw new RuntimeException( e );
		}
	}
}
```

GeoToolsGmlConverter.java:

```
package DE.data_experts.profi.gisfacade.gmlencoding.worker;

//---- Importe -------------------------------------------------------------
import java.io.StringReader;
import org.geotools.gml.GMLFilterDocument;
import org.geotools.gml.GMLFilterGeometry;
import org.geotools.gml.GMLHandlerJTS;
import org.xml.sax.InputSource;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.helpers.XMLReaderFactory;
import DE.data_experts.profi.gisfacade.gmlencoding.tool.GmlConverter;
import com.vividsolutions.jts.geom.Geometry;

public class GeoToolsGmlConverter implements GmlConverter {
	// ---- public Methoden ------------------------------------------
	@Override
	public String getName() {
		return "GMLConverter -> GeoTools";
	}

	@Override
	public Geometry getGeometryFromGML( String gmlString ) {
		InputSource input = new InputSource( new StringReader(
		gmlString ) );
		SimpleGeometryHandler jtsHandler = new SimpleGeometryHandler();
		GMLFilterGeometry filterGeometry = new GMLFilterGeometry(
		jtsHandler );
		GMLFilterDocument filterDocument = new GMLFilterDocument(
		filterGeometry );
		try {
			XMLReader reader = XMLReaderFactory.createXMLReader();
			reader.setContentHandler( filterDocument );
			reader.parse( input );
```

```
                return jtsHandler.getGeometry();
            }
            catch ( Exception e ) {
                throw new RuntimeException( e );
            }
        }

        static class SimpleGeometryHandler extends DefaultHandler implements
        GMLHandlerJTS {

            @Override
            public void geometry( Geometry paramGeometry ) {
                this.geometry = paramGeometry;
            }

            public Geometry getGeometry() {
                return geometry;
            }

            private Geometry geometry;
        }
}
```

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Neubrandenburg, den

Unterschrift