Sonderforschungsbereich 637
Teilprojekt C2 – Datenintegration

TECHNICAL REPORT

Spezifikation eines Semantischen Mediators zur Unterstützung der Selbststeuerung logistischer Prozesse

Autoren: Karl A. Hribernik, Christoph Kramer, Carl Hans

BIBA / Universität Bremen
Teilprojektleiter: Prof. Dr.-Ing. Klaus-Dieter Thoben

Letzte Aktualisierung: 27.05.2011

INHALT

Inhalt I
Abbildungen III
Tabellen IV
Quelltexte V
Abkürzungen VI
1. Einführung 1
2. Systemdefinition 2
2.1. Mediator-Komponente 2
2.2. Anfrage-Schnittstelle 3
2.2.1. SPARQL 3
2.2.2. SPARQL/UPDATE 4
2.2.3. Service-Schnittstelle 4
2.3. DatenWrapper 5
3. Spezifikation 6
3.1. Package de.biba.div 6
3.2. Package de.biba.div.constraints 7
3.3. Package de.biba.exceptions 8
3.4. Package de.biba.gui 9
3.5. Package de.biba.math 9
3.6. Package de.biba.mediator.converter 9
3.7. Package de.biba.queryLanguage 9
3.8. Package de.biba.wrapper 9
4. Beispielhafte Spezifikationen von Wrapperkomponenten 11
4.1. Generische Transformationsmechanismen 11
4.1.1. SQL-Wrapper 11
4.1.2. CSV-Wrapper 15
4.2. Spezifische Transformationsmechanismen 18
4.2.1. EDIFACT EANCOM 18
4.2.2. EPCIS 19
5. Zusammenfassung und Ausblick 22
Danksagungen 23
Anhang A Vollständige Klassendiagramme 24
Klassendiagramm der Mediator-Komponente 24
Klassendiagramm der SPARQL-Schnittstelle 26
Klassendiagramm des CSV-Wrappers 27

Klassendiagramm des SQL-Wrappers 28
Klassendiagramm des EPCIS Wrappers 29
Klassendiagramm des EANCOM Wrappers 30
Anhang B EDIFACT EANCOM Transformationsregeln 31

ABBILDUNGEN

Abbildung 1: Verteilungsdiagramm 2
Abbildung 2: OntoGraf-Darstellung einer beispielhaften semantischen Beschreibung einer SQL-Datenquelle 12
Abbildung 3: OntoGraf-Darstellung einer beispielhaften semantischen Beschreibung einer CSV-Datenquelle 15
Abbildung 4: Standard EPCIS-Ereignisklassen (Quelle: GS1) 20
Abbildung 5: Ausschnitt aus dem Klassendiagramm der Mediator-Komponente 24
Abbildung 6: Klassendiagramm der Mediator-Komponente 25
Abbildung 7: Klassendiagramm der SPARQL-Schnittstelle 26
Abbildung 8: Klassendiagramm des CSV-Wrappers 27
Abbildung 9: Klassendiagramm des SQL-Wrappers 28
Abbildung 10: Klassendiagramm des EPCIS Wrappers 29
Abbildung 11: Klassendiagramm des EANCOM Wrappers 30

TABELLEN

Tabelle 1: Package de.biba.div 7
Tabelle 2: Package de.biba.div.constraints 8
Tabelle 3: Package de.biba.exceptions 9
Tabelle 4: Package de.biba.mediator.converter 9
Tabelle 5: Package de.biba.wrapper 10
Tabelle 6: Package de.biba.wrapper.SQLWrapper 15
Tabelle 7: Package de.biba.wrapper.EANCOMWrapper 18
Tabelle 8: Package de.biba.wrapper.EPCISWrapper 21

QUELLTEXTE

Quelltext 1: Eine beispielhafte RDF-Instanz 3
Quelltext 2: Eine einfache, beispielhafte SPARQL-Anfrage 4
Quelltext 3: Eine einfache, beispielhafte SPARQL/UPDATE-Anfrage 4
Quelltext 4: Beispielhaftes XML-Mappingschema für SQL 13
Quelltext 5: Auszug einers beispielhaften XML-Mapping für SQL 14
Quelltext 6: Beispielhaftes XML-Mappingschema für CSV 17
Quelltext 7: Auszug einers beispielhaften XML-Mapping für CSV 17
Quelltext 8: Beispiel einer IFTMIN –Nachricht 19
Quelltext 9: Beispiel einer Transformationsregel für EDIFACT EANCOM 19
Quelltext 10: Beispiele der Transformationsregeln für ERCIS Ereignisse – Readpoint auslesen 20
Quelltext 11: Beispiele der Transformationsregeln für ERCIS Ereignisse – Disposition auslesen 21
Quelltext 12: Beispiele der Transformationsregeln für erweiterte ERCIS Ereignisse – Temperatur auslesen 21

ABKÜRZUNGEN

CSV	Comma-Separated Values
Drools	Business Logic integration Platform
EAN	European Article Number
EANCOM	“European Article Number” und “Communication”
EDIFACT	Electronic Data Interchange For Administration, Commerce and Transport
EPC	Electronic Product Code
EPCIS	Electronic Product Code Information Services
FOSSTRAK	Free and Open Source Software for Track and Trace
JavaCC	Java Compiler Compiler
OWL-DL	Web Ontology Language – Description Logic
RDF	Resource Description Framework
SFB	Sonderforschungsbereich
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
UN/EDIFACT	United Nations Electronic Data Interchange For Administration, Commerce and Transport.
XSD	XML Schema Definition
XML	Extensible Markup Language

1. EINFÜHRUNG

In der Logistik existiert heute eine Vielfalt unterschiedlicher Systeme, um Daten verwalten und Prozesse optimieren zu können. Der überwiegende Anteil von ihnen nutzt eigene Standards und Schnittstellen. Ein systemunabhängiger Zugriff ist damit nicht ohne weiteres möglich. Standards wie Extensible Markup Language (XML) oder Web Services unterstützen zwar grundsätzlich die systemunabhängige Abbildung und den Austausch von Daten bzw. die verteilte Nutzung von Funktionalitäten und kommen einer Datenintegration damit näher - die Systemlandschaft ist jedoch nach wie vor heterogen und es ist ein erheblicher Aufwand notwendig, um einen einheitlichen, logischen Zugriff allein zwischen zwei verschiedenen Systemen zu ermöglichen.

Innerhalb selbststeuernder Prozesswelten bewegt sich eine Vielzahl autonomer Entitäten. Die Selbststeuerung erfordert dabei Daten, nicht nur bezüglich ihrer selbst sondern auch aus ihrer Umwelt, welche die Grundlage für Entscheidungsprozesse darstellen, die schließlich ihr Verhalten bestimmen. Die Quellen für diese Daten sind ebenso vielfältig wie die logistischen Entitäten: Sie werden durch zentrale IT-Systeme ebenso repräsentiert, wie durch dezentrale eingebettete Systeme, RFID-Tags oder Softwareagenten. Sie können untereinander über existierende Kommunikationskanäle und darauf aufsetzende Protokolle kommunizieren. Das wesentliche Problem im Rahmen eines Datenzugriffes ergibt sich somit aus der Auflösung struktureller und semantischer Heterogenitätskonflikte zwischen Datenquellen und -senken, d.h. der Datenintegration.

In diesem Technical Report wird auf der Basis eines ontologie-basierten Mediators ein systemübergreifender Ansatz zur Integration solcher heterogenen Datenquellen im Kontext der Selbststeuerung spezifiziert. Aufsetzend auf gängige Standarddatenaustauschformate der Logistik und generischen Schnittstellen wie etwa SQL oder CSV erlaubt der hier spezifizierte Ansatz die Bereitstellung eines schreibenden und lesenden Zugriffs auf Daten in beliebigen IT-Systemen der Logistik.

Das Dokument ist wie folgt strukturiert: Zunächst wird eine Systemdefinition präsentiert, in der die allgemeinen Systemkomponenten und ihr Zusammenhang im Gesamtsystem beschrieben wird. Die darauf folgende Systemspezifikation ist in zwei Teile gegliedert. Im ersten Teil werden die allgemeinen Systemkomponenten detailliert spezifiziert: der Mediator, die Anfrageschnittstelle und die generische Wrapperschnittstelle. Im zweiten Teil werden exemplarisch je zwei konkrete Wrapperimplementierungen für generische und spezifische Transformationsmechanismen beschrieben. Eine Zusammenfassung und Ausblick schließt den Bericht ab.

2. SYSTEMDEFINITION

Das System des semantischen Mediators soll es ermöglichen, über eine Schnittstelle lesenden und schreibenden Zugriff auf beliebige Datenquellen in selbststeuernden logistischen Prozessen zu gewährleisten. Dies wird durch ein Ontologie-System ermöglicht, welches über semantische Beschreibungen Kenntnis über die dahinter liegenden Datenquellen besitzt. Die Anfragen wird auf die spezifischen Anforderungen dieser transformiert und ermöglicht so die Bearbeitung. Die Anfrage wird an alle bekannten Datenquellen weitergeleitet und bearbeitet. Die Rückgabe erfolgt, in der Ontologie der Datenquellen und wird durch die im Vorfeld durch den Mediator, aus den Datenquellen-Ontologien, erstellte Ontologie zusammengefasst, von Duplikaten befreit und als Lösung zurückgegeben.

Die Systemarchitektur ist modular konzipiert. Die Anbindung einzelner Datenquellen erfolgt dabei über sogenannte Wrapperkomponenten, die über eine gemeinsame Schnittstelle an den Mediator angebunden sind. Die einzelnen Wrapperkomponenten kapseln dabei alle Elemente, die für die Integration der durch sie angebundenen Datenquellen notwendig sind. Im Einzelnen sind das die semantischen Beschreibungen der Datenquellen, sowie die entsprechenden Transformationsmechanismen und die Ontologiemappings.

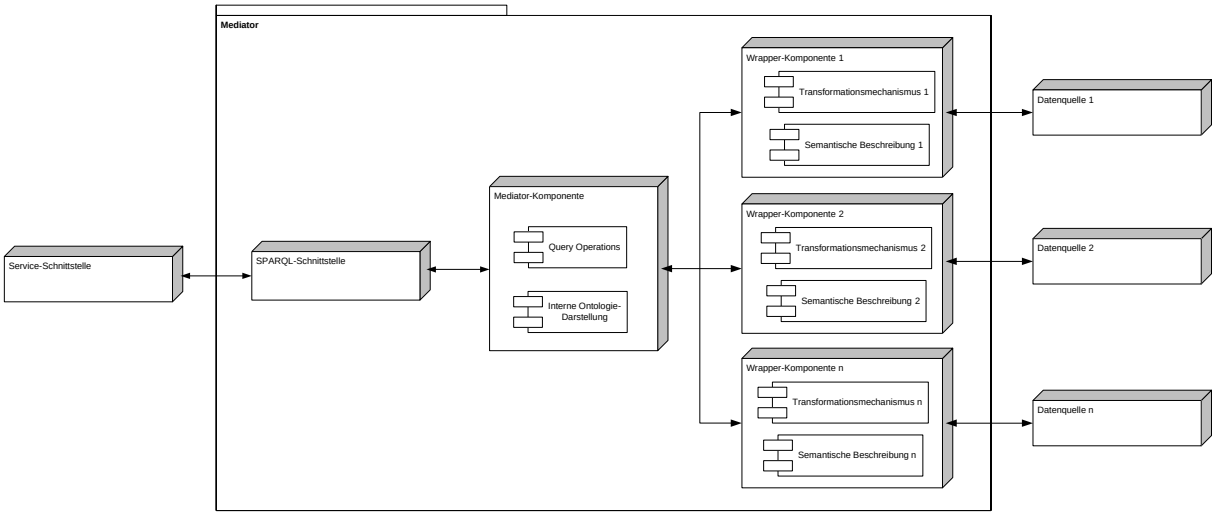

Abbildung 1: Verteilungsdiagramm

In Abbildung 1 wird der Aufbau des Mediators und dessen Komponenten dargestellt. Die verschiedenen Komponenten, die aus der Abbildung 1 zu entnehmen sind, werden im folgenden Abschnitt beschrieben.

2.1. Mediator-Komponente

Der semantische Mediator ist ontologie-basiert spezifiziert und umgesetzt worden. Ein ontologiebasierter Mediator verwendet eine oder mehrere Ontologien. So wird zwischen Mediatoren mit global und solchen mit multiplen Ontologien unterschieden. Der hier beschriebene Mediator ist mit multiplen Ontologien spezifiziert.

Ein Mediator mit multiplen Ontologien besitzt keine globale Ontologie. Stattdessen wird jede Datenquelle durch eine eigenständig entwickelte Ontologie beschrieben. Um eine Verbindung zwischen den einzelnen Ontologien zu schaffen, werden zwischen diesen Abbildungen definiert. Diese Abbildung, oder „Inter-Ontologie-Mapping“ identifiziert

semantisch äquivalente Terme der einzelnen Ontologien und ermöglicht so die globale Auswertung von Anfragen.

Durch die Verwendung einzelner Ontologien für die Datenquellen entfallen somit das Erstellen der globalen Ontologie. Nachteil dieser Vorgehensweise ist der signifikante Modellierungsaufwand, der mit dem Inter-Ontology-Mapping einhergeht. Vorteil ist allerdings ein Gewinn an Skalierbarkeit, Flexibilität und Adaptivität, der im Zusammenhang der Datenintegration in selbststeuernden logistischen Prozessen den Nachteil überwiegt.

Die von der Mediatorkomponente verwendeten Ontologien sind in den einzelnen Wrapperkomponenten hinterlegt. Jede Ontologie beschreibt eine Datenquelle vollständig. In den Wrapperkomponenten hinterlegt sind ebenfalls die Mappings der Ontologien aufeinander. Zur Laufzeit fügt der semantische Mediator die einzelnen Ontologien über diese Mappings zu einer internen Darstellung zusammen.

2.2. Anfrage-Schnittstelle

Die Komponente „SPARQL-Schnittstelle“ setzt eine auf die Anfragesprache SPARQL/UPDATE basierende Anfrageschnittstelle um. Über diese Schnittstelle können Datenkonsumenten (z.B. intelligente logistische Objekte und andere Logistiksysteme) den Anfragen an den Mediator stellen. Die Schnittstelle nimmt die Anfrage an und reicht sie weiter an die Mediator-Komponente. Dabei ist die Schnittstelle bidirektional ausgelegt, d.h. über die Schnittstelle können sowohl Lese- als auch Schreiboperationen ausgeführt werden.

2.2.1. SPARQL

Diese deskriptive Anfragesprache SQPARL stellt eine Empfehlung des W3C für Anfragen an RDF-basierten Ontologien dar. Dadurch das RDF die Grundlage anderer Beschreibungssprachen für Ontologien ist, kann man an diese ebenfalls Anfragen mittels SPARQL stellen. Über die vom Mediator verwendeten, in OWL-DL verfassten Ontologien können ebenfalls mit SPARQL Anfragen formuliert werden.

Die Syntax und Semantik von SPARQL ist RDF-basierten Ontologien entsprechend aufgebaut. In RDF werden Aussagen als Tripel notiert, bestehend aus Subjekt, Prädikat und Objekt. Dabei entsprechen das Subjekt einer Instanz einer Klasse aus der Ontologie, das Prädikat einer Rolle und das Objekt entweder einer Instanz oder einem Literal.

Ein Beispiel verdeutlicht dieses Prinzip: wird in einer Ontologie eine Instanz „Peter“ erstellt und dieser über die Rolle „hatNachname“ den Nachnamen „Schmidt“ zugewiesen, so wird dies in RDF wie folgt ausgedrückt (im Beispiel wird auf die Verwendung von Namensräumen verzichtet):

```
<Peter> <hatNachname> "Schmidt"
```

Quelltext 1: Eine beispielhafte RDF-Instanz

Generell lassen sich mittels SPARQL alle Tripel einer RDF-basierten Ontologie abfragen. Die Syntax ähnelt der von SQL. Einfache Anfragen über Ontologien werden analog zu SQL mit dem Schlüsselwort „SELECT“ gestellt. Eine durch Leerzeichen getrennte Variablenliste definiert die Ergebnisspalten der Rückgabe. Variablennamen werden ein Fragezeichen vorangestellt. Zuletzt folgen die Bedingungen in Form von Tripeln, eingeleitet durch das Schlüsselwort „WHERE“. Dabei werden die Bedingungen in der gleichen Form wie RDF-

Aussagen notiert, mit Subjekt, Prädikat und Objekt. Multiple Bedingungen werden durch ein Punkt („.“) getrennt.

Bei der Anfrageverarbeitung werden alle Tripel, die die Bedingungen erfüllen, gesucht. Die Variablen werden auf die entsprechenden Stellen der Anfragen gebunden. Die Belegungen der hinter dem SELECT-Schlüsselwort definierten Variablen werden zurückgegeben.

```
SELECT ?vorname ?matrikelnummer

WHERE
{
?x ?vorname hatMatrikelnummer ?matrikelnummer.
?y ?vorname hatNachname Schmidt.
}
```

Quelltext 2: Eine einfache, beispielhafte SPARQL-Anfrage

Ein sehr einfaches Beispiel (Quelltext 2) verweichlicht das Prinzip. Auch hier auf die Verwendung von Namensräumen verzichtet. Die in Quelltext 2 dargestellte Anfrage gibt alle Vornamen und Matrikelnummer zurück, unter der Bedingung, dass der dem Vornamen zugewiesene Nachname „Schmidt“ heißt.

2.2.2. SPARQL/UPDATE

Die aktuelle Version von SPARQL ist lediglich dazu konzipiert, lesende Anfragen über Ontologien auszuführen. Einen schreibenden Zugriff ist zurzeit nicht möglich. Aus diesem Grund wird die SPARQL-Schnittstelle mit der experimentellen Erweiterung SPARQL/UPDATE[1] spezifiziert und umgesetzt. Mit dieser Erweiterung ist es möglich, RDF-Tripel einer Ontologie hinzuzufügen oder sie zu löschen.

Die Erweiterung ergänzt die SPARQL-Syntax um einige neue Schlüsselwörter. So ist es zum Beispiel mit dem Schlüsselwort „INSERT“ möglich, RDF-Tripel in eine Ontologie einzufügen. Soll zum Beispiel der Beispielontologie aus Quelltext 1 die Aussage hinzugefügt werden, dass „Alex“ den Nachnamen „Meyer“ hat, kann folgende (Quelltext 3) benutzt werden.

```
INSERT
{
Alex hatNachname „Meyer“
}
```

Quelltext 3: Eine einfache, beispielhafte SPARQL/UPDATE-Anfrage

Mit der SPARQL/UPDATE-Schnittstelle wird die Anforderung umgesetzt, Datenquellen nicht nur abfragen zu können, sondern auch Schreibzugriffe zu implementieren.

2.2.3. Service-Schnittstelle

Neben der SPARQL-Anfragesprache existiert eine W3C-Empfehlung für ein SPARQL Protokoll. Das Protokoll definiert genau eine Schnittstelle – „SparqlQuery“. SparqlQuery ist so definiert, dass sie sowohl SPARQL-Anfragen, als auch RDF-Dataset-Konstrukte als Rückfragen handhaben kann.

[1] SPARQL Protocol for RDF, http://www.w3.org/Submission/2008/SUBM-SPARQL-Update-20080715/

Das SPARQL-Protokoll definiert darüber hinaus die Umsetzung der Schnittstelle als Web-Service.

Die Spezifikation der Service-Schnittstelle wird an dieser Stelle nicht ausführlich beschrieben.

2.3. DatenWrapper

Die Daten-Wrapper sind für die Kommunikation mit den entsprechenden Datenquellen und die Bereitstellung der semantisch beschreibenden Ontologien zuständig. Auf dieser Grundlage erfolgt die Transformation in die Mediator-Ontologie und die Rückgabe des Ergebnis-Datensatzes.

Es ist eine generische Schnittstelle für die Einbindung der Wrapperkomponenten spezifiziert. In der Regel wird eine Wrapperkomponente eine OWL-DL-Ontologie als semantische Beschreibung sowie ein Transformationsmechanismus zwischen der Ontologie und der Zieldatenquelle umfassen. Darüber hinaus können die individuellen Wrapper beliebig ausgeprägt sein.

In Abschnitt 4 sind beispielhafte Implementierungen von Wrapperkomponenten aufgeführt. Es werden zwei unterschiedliche Ansätze beleuchtet – einmal Ansätze mit generischen Transformationsmechanismen und einmal solche mit spezifischen, algorithmischen Mechanismen.

3. SPEZIFIKATION

Im Folgenden werden die Spezifikationen der, im semantischen Mediator enthaltenden Klassen beschrieben. Vollständige Beschreibungen der Klassendiagramme sind im Anhang A aufgeführt.

3.1. Package de.biba.div

In der folgenden „Package de.biba.div“-Tabelle sind alle Klassen die im Mediator-Package de.biba.div enthalten sind aufgeführt und beschrieben.

Klasse	Beschreibung
Abox	Verknüpft die Individuen mit Eigenschaften und Werten.
AbstractProperty	Grundklasse der Klassen „DatatypeProperty“ und „ObjectProperty“
BooleanDatatype	Implementiert den Datentyp Boolean.
CompareOperator	Aufzählungs-Klasse der Vergleichs-Operatoren.
ComplexOntClass	Erweitert die Klasse „OntClass“ um die Möglichkeit der Verwendung von „Constraints“.
DataSourceQuery	„DataSourceQuery“-Objekt welches die Datenquellen spezifischen Anfragen enthält.
Datatype	Interface für die verschiedenen Datentypen.
DatatypeProperty	Weist einer Instanz einen Wert zu und ob dieser invers Funktional oder nicht ist.
DatatypePropertyStatement	Verbindet ein Individual mit einem Datentyp. Rückgabe als String.
Datatypes	Aufzählung der vorhandenen Datentypen.
DateValue	Implementiert den Datentyp „Date“ (Datum z.B. in der Form DD.MM.YYYY) mit der Möglichkeit des Überprüfens ob es sich wirklich um eine Instanz dieses Datentyps handelt.
DuplicateRemover	Löscht Duplikate aus dem Ergebnis-Datensatz.
IMediator	Mediator-Interface implementiert die für den Mediator benötigten Funktionen.
Individual	Datentyp Individual (Instanzen einer Klasse)
InputQuery	Klasse für schreibenden Zugriff.

IQuery	Query-Interface welches die Funktion isInputQuery implementiert.
IQuerySolution	Implementiert alle Funktionen für die Klasse „QuerySolution“
ISolutionIterator	Implementiert alle Funktionen für den „SolutionIterator“
Main	Main-Klasse, fügt die unterschiedlichen Datenquellen dem Mediator hinzu und startet die GUI.
NumericDatatype	Implementiert den Datentyp Zahl mit der Möglichkeit des Überprüfens ob es sich wirklich um eine Instanz dieses Datentyps handelt.
ObjectProperty	Objekt, welches den Zusammenhang zwischen Instanzen von Klassen beschreibt.
ObjectPropertyStatement	Verknüpft zwei Individuen miteinander.
OntClass	Erweitert „OntologyElement“ um einen Namespace und einen Namen.
OntModel	Objekt eines Ontologie-Models mit der dazugehörenden A- und T-Box.
OntologyElement	Implementiert Funktion zur Rückgabe der URI (*Uniform Ressource Identifier*) bestehend aus Name und Namespace.
OutputQuery	Klasse für lesenden Zugriff.
OWLParser	Parsiert eine OWL-Datei in ein „OntModel“-Objekt.
QuerySolution	Ergebnis-Datensatz einer Abfrage.
QuerySolutionInterator	Klasse welche erlaubt durch den Ergebnis-Datensatz zu iterieren.
ReasoningMediator	Erstellt aus den vorhandenen Datenquellen- das Mediator-Ontologie-Model, durch zusammenfassen der Regelwerke der beiden Modelle.
TBox	TBox beschreibt die Ontologie-Regeln und ermöglicht das zusammenlegen mehrerer Ontologie-Regelwerke.

Tabelle 1: Package de.biba.div

3.2. Package de.biba.div.constraints

In der folgenden „Package *de.biba.div.constraints*“-Tabelle sind alle Klassen die im Mediator-Package de.biba.div.constraints enthalten sind aufgeführt und beschrieben.

Constraints beschreiben die einzelnen Anfrage-Tupel[2]. Diese werden dann in der Klasse „QuerySolution“ zusammengesetzt und an die Wrapper weitergeleitet.

Klasse	Beschreibung
BinaryConstraint	Grundklasse für das „IntersectionConstraint“ und das „UnionConstraint“.
CardinalityConstraint	Constraint welches benutzt wird wenn ein Individuum etwas mehrfach besitzen soll.
ClassConstraint	Constraint welches eine Abfrage repräsentiert, bei denen Instanzen einer Klasse mit einer bestimmten Eigenschaft gesucht werden
Constraint	Interface welches Funktionen für die Constraints implementiert.
ConstraintList	Aneinanderreihung von Constraints. Auf dem ersten Element folgende Arbeiten nur noch auf dem Ergebnis-Datensatz des vorherigen.
HasValueConstraint	Constraint wenn eine Instanz einen bestimmten Wert haben soll.
IntersectionConstraint	Constraint für das Lösen von Abfragen, die die Überschneidung zweier Datensätze benötigen.
OrConstraint	Verknüpfung zweier Constraints über oder.
PropertyConstraint	Constraint einer Abfrage, bei der alle Instanzen einer Klasse ein bestimmtes Merkmal haben sollen.
SimpleConstraint	Interface das eine Methode Implementiert welches die Komplexität der Abfrage wiedergibt.
SimplePropertyConstraint	Constraints welches benutzt wird, wen Individuen mit einer Eigenschaft, die einen bestimmten Wert hat, gesucht werden.
SomeConstraint	Constraint für eine Abfrage, die Instanzen sucht welche über eine Eigenschaft mit einer anderen Daten-Klasse verbunden sind.
UnionConstraint	Constraint zur Vereinigung zweier Datensätze.

Tabelle 2: Package de.biba.div.constraints

3.3. Package de.biba.exceptions

2 Welche die Form „*{ <Mensch>.<VerwandtMit> <name> Person.}*“ haben

Klasse	Beschreibung
de.biba.exceptions	Das Package stellt verschiedenen Klassen benötigte Exceptions zu Verfügung.

Tabelle 3: Package de.biba.exceptions

3.4. Package de.biba.gui

Das GUI-Package enthält eine GUI, welche es ermöglicht z.B. die Ausgabe- Query einzusehen.

3.5. Package de.biba.math

Das Package de.biba.math enthält Klassen, welche Methoden implementieren die vom ValueConverter genutzt werden können um Umrechnungen von Einheiten vorzunehmen.

3.6. Package de.biba.mediator.converter

In der folgenden „Package *de.biba.converter*“-Tabelle sind alle Klassen die im Mediator-Package *de.biba.converter* enthalten sind aufgeführt und beschrieben.

Klasse	Beschreibung
ComplexConverter	Für Umrechnungen die mehrere Schritte erfordern.
Converter	Implementiert Funktionen für die Klassen „ComplexConverter“, „ValueAdder“ und „ValueMultiplier“
ValueAdder	Konvertiert die Einheit der Datenquelle durch Addition mit einem festgelegten Wert.
ValueConverter	Konvertiert die Einheiten der Datenquellen nach den Regeln in der Datei „conversionRules.xml“[3].
ValueMultiplier	Konvertiert die Einheit der Datenquelle durch Multiplikation mit einem festgelegten Wert.

Tabelle 4: Package de.biba.mediator.converter

3.7. Package de.biba.queryLanguage

Das Package enthält vom JavaCC geparste Klassen nach dem Schema aus der Datei simpleQuery.jj und stellt eine Schnittstelle zur RDF-Anfragesprache SPARQL zur Verfügung.

3.8. Package de.biba.wrapper

Spezifikation der Klassen im Mediator-Package *de.biba.wrapper*

3 In der conversionRules.xml werden alle Einheiten die Umgerechnet werden können beschrieben sowie die Art der Umrechnung.

Klasse	Beschreibung
DataSource	Dieses Interface enthält Funktionen die jede Datenquelle implementieren muss.

Tabelle 5: Package de.biba.wrapper

4. BEISPIELHAFTE SPEZIFIKATIONEN VON WRAPPERKOMPONENTEN

In diesem Kapitel werden zusätzlich zur allgemeinen Wrapperschnittstelle Beispiele konkreter Wrapperkomponenten vorgestellt. Hierbei wird zwischen zwei unterschiedlichen Ausprägungen differenziert:

1. **Generische Transformationsmechanismen**

   Hierbei handelt es sich um die Spezifikationen von Wrapperkomponenten, die sich leicht durch die Änderungen von Konfigurationen auf unterschiedliche Zieldatenquellen anpassen lassen. Solche Wrapperkomponenten empfehlen sich für generische Schnittstellen, wie z.B. SQL, CSV oder Ähnliche. Die Wrapperkomponenten sind so spezifiziert, dass sie den Zugriff auf die jeweiligen Zielformate umsetzen. Zur Integration müssen lediglich die semantische Beschreibung der Zieldatenquelle und eine Konfigurationsdatei, die die Abbildung der Ontologie auf die Zieldatenquelle beschreibt, erstellt werden.

2. **Spezifische Transformationsmechanismen**

   Bei diesen Transformationsmechanismen handelt es sich um Wrapperkomponenten, die für eine bestimmte Datenquelle oder Datenformat spezifiziert sind. Der Vorteil dieses Ansatzes ist eine wesentlich höhere Performanz in der Integration, da nur die spezifischen Eigenschaften der Zieldatenquelle zu berücksichtigen sind. Zum Anderen lassen sich Wrapperkomponenten für sehr komplexe oder individuelle Standarddatenaustauschformate, wie etwa EDIFACT EANCOM, nicht oder nur sehr schwierig über generische, konfigurierbare Ansätze umsetzen.

In den folgenden Abschnitten werden zunächst Wrapperkomponenten für SQL und CSV als Beispiele generischer Transformationsmechanismen vorgestellt. Im Anschluss werden Wrapperkomponenten für EDIFACT EANCOM und EPCIS als Beispiele spezifischer Transformationsmechanismen für gängige Datenaustauschformate im Bereich der Logistik präsentiert.

4.1. Generische Transformationsmechanismen

Die folgenden Abschnitte beschreiben die Umsetzung generischer Transformationsmechanismen an den Beispielen der Datenwrapper für CSV- und SQL-Datenquellen.

4.1.1. SQL-Wrapper

Zur Konfiguration der SQL-Wrapperkomponenten werden drei Dateien herangezogen:

1. Semantische Beschreibung der SQL-Datenquelle als OWL-DL Ontologie
2. Ein XML-Schema (XSD) zur Beschreibung der Struktur des Mappings auf SQL-Datenquellen
3. Eine entsprechend dem XML-Schema strukturiertes Mappingdatei, die die konkrete Abbildung der Ontologie auf die SQL-Datenquelle beschreibt

Die folgenden Abschnitte zeigen beispielhaft eine Konfiguration einer Wrapperkomponente für eine SQL-Datenquelle in einem selbststeuernden, transportlogistischen Szenario.

Beispielontologie „Transportlogistik“

Abbildung 2 zeigt beispielhaft die semantische Beschreibung einer SQL-Datenquelle als OWL-DL-Ontologie.

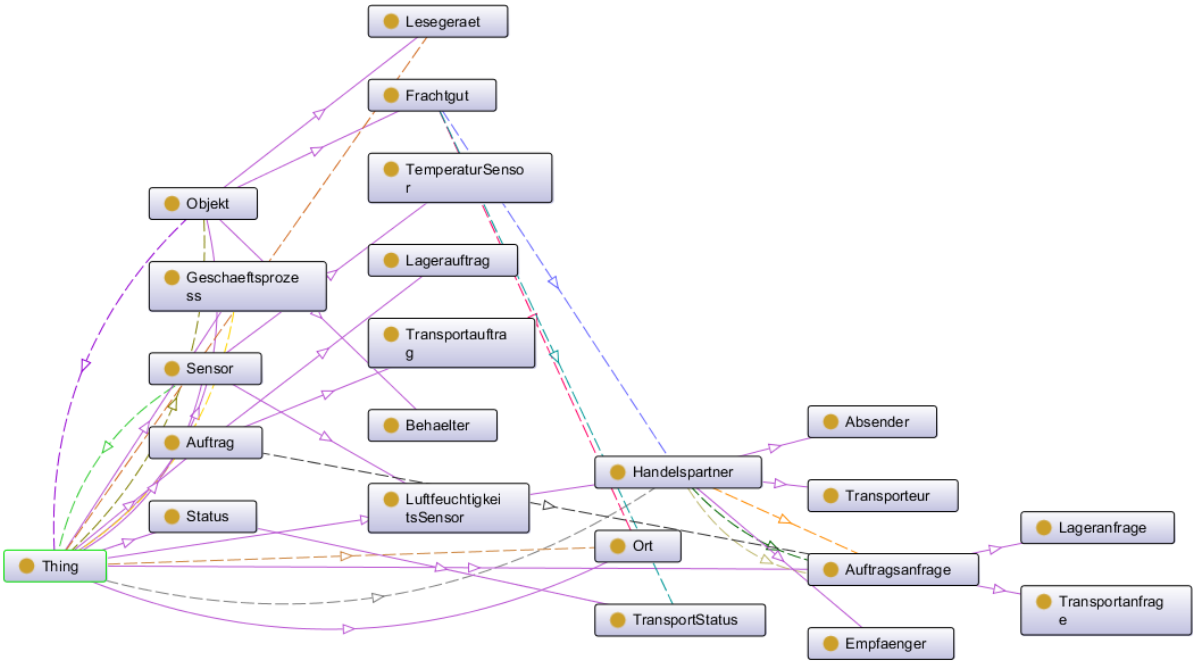

Abbildung 2: OntoGraf-Darstellung einer beispielhaften semantischen Beschreibung einer SQL-Datenquelle

Beispielhaftes XML-Mappingschema

Die folgende XSD-Definition (Quelltext 4) beschreibt beispielhaft die Struktur eines Mappings auf SQL-Datenquellen. In diesem Schema wird nicht zwischen Datatype- und Property-Mappings unterschieden, da diese bei SQL-Abfragen äquivalent sind.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://sfb637.biba.uni-bremen.de/sqlmapping"
elementFormDefault="qualified" xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:sms="http://sfb637.biba.uni-bremen.de/sqlmapping">

    <complexType name="Mapping">
          <sequence minOccurs="1" maxOccurs="1">
                    <choice minOccurs="1" maxOccurs="unbounded">
                              <element name="ClassMapping"
type="sms:ClassMapping"></element>
                              <element name="PropertyMapping"
type="sms:PropertyMapping">
                              </element>
                    </choice>
          </sequence>
          <attribute name="WrapperName" type="string"></attribute>
    </complexType>

    <complexType name="ClassMapping">
          <sequence minOccurs="1" maxOccurs="1">
                    <element name="column" type="string" maxOccurs="unbounded"
                              minOccurs="1">
                    </element>
                    <element name="tables" type="string" maxOccurs="1"
                              minOccurs="1">
                    </element>
                    <element name="conditions" type="string" maxOccurs="1"
                              minOccurs="0">
                    </element>
```

```
          </sequence>
          <attribute name="name" type="string" use="required"></attribute>
          <attribute name="unit" type="string" use="required"></attribute>
          <attribute name="valueType" type="sms:ValueType"
use="required"></attribute>
    </complexType>

    <complexType name="PropertyMapping">
          <sequence minOccurs="1" maxOccurs="1">
                    <element name="subColumn" type="string" maxOccurs="1"
minOccurs="1">
                    </element>
                    <element name="obColumn" type="string" maxOccurs="1"
minOccurs="1">
                    </element>
                    <element name="subType" type="sms:ValueType" minOccurs="1"
maxOccurs="1"></element>
                    <element name="obType" type="sms:ValueType" minOccurs="1"
maxOccurs="1"></element>
                    <choice maxOccurs="1" minOccurs="1">
                              <element name="tables" type="string"></element>
                              <element name="join"
type="sms:JoinedTable"></element>
                    </choice>
                    <element name="conditions" type="string" maxOccurs="1"
minOccurs="0">
                    </element>
          </sequence>
          <attribute name="name" type="string" use="required"></attribute>
          <attribute name="unit" type="string" use="required"></attribute>
    </complexType>

    <element name="Mapping" type="sms:Mapping"></element>

    <complexType name="JoinedTable">
          <sequence>
                    <element name="LeftTable" type="string" minOccurs="1"
maxOccurs="1">
                    </element>
                    <element name="RightTable" type="string" minOccurs="1"
maxOccurs="1">
                    </element>
                    <element name="Condition" type="string" minOccurs="1"
maxOccurs="unbounded">
                    </element>
          </sequence>
          <attribute name="JoinType" type="string" use="required"></attribute>
    </complexType>

    <simpleType name="ValueType">
          <restriction base="string">
                    <enumeration value="string"/>
                    <enumeration value="numeric"/>
                    <enumeration value="date"/>
                    <enumeration value="boolean"/>
          </restriction>
    </simpleType>
</schema>
```

Quelltext 4: Beispielhaftes XML-Mappingschema für SQL

Beispielhafte XML-Mappingdatei

Im Folgenden wird anhand von Beispielen gezeigt, wie für die semantische Beschreibung der SQL-Datenquelle mit dem Mappingschema Abbildungen darauf erzeugt werden können.

In Quelltext 5 wird die Notation zweier Mappings auf eine SQL-Datenbank gezeigt.

```
<?xml version="1.0" encoding="UTF-8"?>
<sms:Mapping xmlns:sms="http://sfb637.biba.uni-bremen.de/sqlmapping"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://sfb637.biba.uni-bremen.de/sqlmapping/SQLMapping.xsd
">


// Beispiel eines DatatypeMappings auf ein Element einer Tabelle

  <sms:PropertyMapping name="http://sfb637.biba.uni-
```

```
bremen.de/ontologien/transportszenario.owl#transportbeginn" unit="unknown">
          <sms:subColumn>auftraege.transportid</sms:subColumn>
          <sms:obColumn>auftraege.transportbeginn</sms:obColumn>
          <sms:subType>string</sms:subType>
          <sms:obType>date</sms:obType>
          <sms:tables>auftraege</sms:tables>
  </sms:PropertyMapping>

// Beispiel eines PropertyMappings über zwei Tabellen

  <sms:PropertyMapping name="http://sfb637.biba.uni-
bremen.de/ontologien/transportszenario.owl#AuftragsAbsender" unit="unknown">
          <sms:subColumn>auftraege.auftragsnummer</sms:subColumn>
          <sms:obColumn>kontaktdatenbank.partnerid</sms:obColumn>
          <sms:subType>string</sms:subType>
          <sms:obType>string</sms:obType>
          <sms:join JoinType="inner join">
                    <sms:LeftTable>auftraege</sms:LeftTable>
                    <sms:RightTable>kontaktdatenbank</sms:RightTable>
                    <sms:Condition>auftraege.absender =
kontaktdatenbank.partnerid</sms:Condition>
          </sms:join>
  </sms:PropertyMapping>


</sms:Mapping>
```

Quelltext 5: Auszug einers beispielhaften XML-Mapping für SQL

Im ersten Mapping wird die das Subjekt der DataTypeProperty „Transportbeginn“ auf die Spalte „transportid“ einer Tabelle „auftraege“ gemappt. Das Objekt wird auf die Spalte „transportbeginn“ derselben Tabelle gemappt. Die Datentypen sind als „string“ respektive „data“ definiert.

Das zweite Beispiel veranschaulicht das Mapping einer Property über zwei Tabellen, die mit einem Inner Join miteinander verbunden werden. Hier wird das Subjekt der Property „AuftragsAbsender“ mit der Spalte „auftraege“ in der Tabelle „auftragsnummer“ verbunden. Das Objekt wird auf die Spalte „partnerid“ in der Tabelle „kontaktdatenbank“ verbunden. Als Bedingung für den Inner Join wird angegeben, dass die Werte in der Spalte „absender“ der Tabelle „auftraege“ mit denen der Spalte „partnerid“ in der Tabelle „kontaktdatenbank“ übereinstimmen.

Entsprechend des in Quelltext 4 dargestellten Mappingschemas können so beliebige Transformationen zwischen der Ergebnisontologie und SQL-Schemata konfiguriert werden.

Package de.biba.wrapper.SQLWrapper

In der folgenden „Package *de.biba.wrapper.SQLWrapper*“-Tabelle sind alle Klassen die im Mediator-Package *de.biba.wrapper.SQLWrapper* enthalten sind aufgeführt und beschrieben.

Klasse	Beschreibung
ClassMapping	Die Klasse übernimmt das Auslesen des SQL-Mapping-Files (welches die Ontologie enthält), sowie auf dieser Grundlage das Erstellen der SQL-Abfrage.
DatatypePropertyCreator	Erstellt eine neue „DatatypeProperty“ und fügt dies dem Ontologie-Model hinzu
IndividualCreator	Erstellt Instanzen einer Daten-Klasse
JoinedTableModifier	Erlaubt der SQL-Abfrage einen „Join“ von links oder rechts.

ObjectPropertyCreator	Erstellt eine Eigenschaft und fügt diese dem Ontologie-Model hinzu.
PropertyMapping	Erstellt die SQL-Abfrage.
QueryFinisher	Container-Klasse, welche die Daten der Abfrage bündelt.
SimpleTableModifier	Einfache SQL-Abfragen ohne „SQL-Join“.
SQLDataSource	Realisiert die Verbindung zu einer SQL-Datenquelle, sowie das Erstellen des Ontologie-Models.
SQLMapping	Liest ein Ontologie-Mappingfile ein und ermöglicht die Transformation in diese.
TableModifier	Implementiert Funktionen für die Klassen „JoinedTableModifier" und „SimpleTableModifier“

Tabelle 6: Package de.biba.wrapper.SQLWrapper

4.1.2. CSV-Wrapper

Zur Konfiguration der CSV-Wrapperkomponenten werden analog zur SQL-Wrapperkomponente drei Dateien herangezogen:

1. Semantische Beschreibung der CSV-Datenquelle als OWL-DL Ontologie
2. Ein XML-Schema (XSD) zur Beschreibung der Struktur des Mappings auf CSV-Datenquellen
3. Eine entsprechend dem XML-Schema strukturiertes Mappingdatei, die die konkrete Abbildung der Ontologie auf die CSV-Datenquelle beschreibt

Die folgenden Abschnitte zeigen beispielhaft eine Konfiguration einer Wrapperkomponente für eine CSV-Datenquelle in einem selbststeuernden, transportlogistischen Szenario.

Beispielontologie „Transportlogistik“

Abbildung 2 zeigt beispielhaft die semantische Beschreibung einer CSV-Datenquelle als OWL-DL-Ontologie.

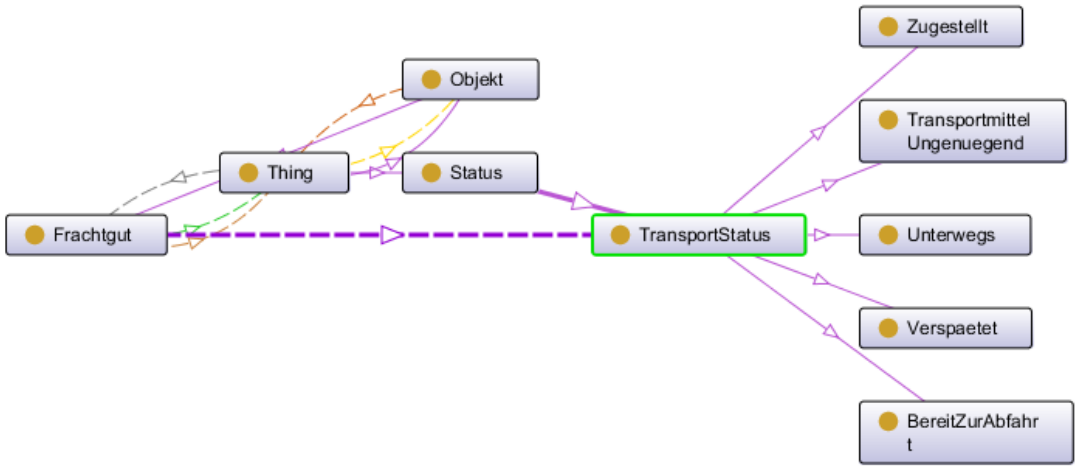

Abbildung 3: OntoGraf-Darstellung einer beispielhaften semantischen Beschreibung einer CSV-Datenquelle

Beispielhaftes CSV-Mappingschema

Die folgende CSV-Definition beschreibt beispielhaft die Struktur eines Mappings auf CSV-Datenquellen.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:complexType name="ClassMapping">
          <xsd:sequence>
                    <xsd:element name="ID" type="xsd:int" minOccurs="1"
maxOccurs="1"/>
                    <xsd:element name="Condition" type="Condition" minOccurs="0"
maxOccurs="1"/>
          </xsd:sequence>
          <xsd:attribute name="OntClass" type="xsd:string"  use="required"/>
</xsd:complexType>

<xsd:complexType name="Condition">
          <xsd:sequence>
                    <xsd:element name="ColumnNumber" type="xsd:int" minOccurs="1"
maxOccurs="1"/>
                    <xsd:element name="Value" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                    <xsd:element name="Operator" type="comparator" minOccurs="1"
maxOccurs="1"/>
          </xsd:sequence>
          <xsd:attribute name="ValueType" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:simpleType name="comparator">
          <xsd:restriction base="xsd:string">
          <xsd:enumeration value="="/>
          <xsd:enumeration value=">="/>
          <xsd:enumeration value="&lt;="/>
          <xsd:enumeration value="&lt;"/>
          <xsd:enumeration value=">"/>
          <xsd:enumeration value="!="/>
          </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="DatatypePropertyMapping">
          <xsd:sequence>
                    <xsd:element name="SubID" type="xsd:int" minOccurs="1"
maxOccurs="1"/>
                    <xsd:element name="SubClass" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                    <xsd:element name="Column" type="xsd:int" minOccurs="1"
maxOccurs="1"/>
          </xsd:sequence>
          <xsd:attribute name="Property" type="xsd:string" use="required"/>
          <xsd:attribute name="Unit" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="ObjectPropertyMapping">
          <xsd:sequence>
                    <xsd:element name="SubjectID" type="xsd:int" minOccurs="1"
maxOccurs="1"/>
                    <xsd:element name="SubClass" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                    <xsd:element name="ObjectID" type="xsd:int" minOccurs="1"
maxOccurs="1"/>
                    <xsd:element name="ObClass" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

          </xsd:sequence>
          <xsd:attribute name="Property" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:element name="Mapping">
          <xsd:complexType>
                    <xsd:sequence>
                              <xsd:element name="ClassMapping"
type="ClassMapping" minOccurs="0" maxOccurs="unbounded"/>
                              <xsd:element name="DatatypePropertyMapping"
type="DatatypePropertyMapping" minOccurs="0" maxOccurs="unbounded"/>
                              <xsd:element name="ObjectPropertyMapping"
type="ObjectPropertyMapping" minOccurs="0" maxOccurs="unbounded"/>
                    </xsd:sequence>
```

```
          <xsd:attribute name="namespace" type="xsd:string" />
          </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Quelltext 6: Beispielhaftes XML-Mappingschema für CSV

Beispielhaftes CSV-Mapping

Im Folgenden wird anhand von Beispielen gezeigt, wie für die semantische Beschreibung der CSV-Datenquelle mit dem Mappingschema Abbildungen darauf erzeugt werden können.

In Quelltext 7 wird die Notation zweier Mappings auf eine CSV-Datenbank gezeigt.

```
<?xml version="1.0" encoding="UTF-8"?>
<Mapping namespace="http://sfb637.biba.uni-bremen.de/csvmapping">



          <ObjectPropertyMapping Property="http://sfb637.biba.uni-
bremen.de/ontologien/transportszenario.owl#HatTransportStatus" Unit="unknown">
                    <SubjectID>0</SubID>
                    <SubjClass>Frachtgut</SubClass>
                    <ObjectID>12</SubID>
                    <ObjClass>Status</SubClass>
          </ObjectPropertyMapping>




</Mapping>
```

Quelltext 7: Auszug einers beispielhaften XML-Mapping für CSV

Im ersten Mapping wird eine Property „HatTransportStatus“ auf die Inhalte eines CSV-Formats gemappt. Das Subjekt „Frachtgut“ wird auf die erste Spalte (0) des Formats gemappt. Das Objekt „Transportstatus“ wird entsprechend mit der dreizehnten (12) Spalte des CSV-Formats verbunden.

Mappings zu Class und DatatypeProperties werden analog beschrieben entlang der im Mappingschema festgelegten Regeln.

Package de.biba.wrapper.tableWrapper

In der folgenden „Package *de.biba.wrapper. EANCOMWrapper*“-Tabelle sind alle Klassen die im Mediator-Package *de.biba.wrapper. EANCOMWrapper* enthalten sind aufgeführt und beschrieben.

Klasse	Beschreibung
EANCOMWrapper	Setzt den lesenden und schreibenden Zugriff auf EANCOM-Nachrichten um
LegacyStyledElementAnnotations	Setzt die Transformationsregeln für EANCOM-Nachrichten algorithmisch um.
OntologyGenerator	Überträgt Anfrageergebnisse in die Ergebnisontologie
Message	Verarbeitet EANCOM-Nachrichten
Segment	Verarbeitet EANCOM-Segmente
SegmentIterator	Behandelt die Iteration der Verarbeitung von EANCOM-Segmenten

DataElement	Verarbeitet EANCOM-Datenelemente

Tabelle 7: Package de.biba.wrapper.EANCOMWrapper

4.2. Spezifische Transformationsmechanismen

Die folgenden Abschnitte beschreiben anhand von Beispielen die Umsetzung der zwei Wrapperkomponenten, die konkrete Datenaustauschformate der Logistik behandeln. Im Einzelnen sind diese EDIFACT EANCOM und EPCIS.

Es wurden zunächst regelbasierte Ansätze erprobt, die eine weitestgehend generische, regelbasierte Konfiguration der Wrapperkomponenten für diese Formate erlauben. Allerdings wurden in weiteren Entwicklungsiterationen diese Ansätze allerdings verworfen. Spezifische, algorithmische Ansätze erwiesen sich als weitaus performanter. Dennoch bieten sich die ursprünglichen, in Drools (Drools Business Logic integration Platform) verfassten Transformationsregeln an, um die Funktionsweise der Wrapperkomponenten zu veranschaulichen, da in der spezifischen Umsetzung die im ersten Schritt notierten Regeln optimiert in Java umgesetzt wurden.

4.2.1. EDIFACT EANCOM

Eine umfangreiche Beschreibung des Nachrichtenformats EDIFACT EANCOM ist im Technical Report „Technical Report Standard Data Exchange Formats in the Field of Logistics“ zu finden. An dieser Stelle soll an einer Beispielnachricht „IFTMIN“ die Funktionsweise des EDIFACT EANCOM-Wrappers veranschaulicht werden.

Quelltext 8 zeigt ein Beispiel einer IFTMIN-Nachricht. Die NAD-Segmente („Name and Adress“ – Name und Adresse) sind fett hervorgehobenen und gelb hinterlegt und werden im folgenden Beispiel einer Transformationsregeln genutzt.

```
UNH+ME000001+IFTMIN:D:01B:UN:EAN004'     Message header
BGM+610+569952+9'                        Transport instruction number
DTM+137:20020301:102'                    Message date/time 1st March 2002
DTM+2:200203081100:203'                  Delivery date/time requested, 8th March
                                         2002 at 11:00
CNT+11:1'                                Total number of packages 1
RFF+CU:TI1284'                           Consignor's reference number TI1284
TDT+20++30+31'                           Details of transport, by truck
DTM+133:200203051100:203'                Estimated departure of truck 5th March
                                         2002 at 11am
LOC+9+5412345678908::9'                  Place of truck loading identified with
                                         GLN 5412345678908
NAD+CZ+5412345123453::9'                 Consignor identified with GLN
                                         5412345123453
NAD+CA+5411234512309::9'                 Carrier identified with GLN
                                         5411234512309
NAD+CN+5411234444402::9'                 Consignee identified with GLN
                                         5411234444402
NAD+DP+5412345145660::9'                 Delivery party identified with GLN
                                         5412345145660
GID+1+1:09::9+14:PK'                     First occurrence of goods in one
                                         returnable pallet with 14 packages
HAN+EAT::9'                              The goods are foods stuffs
TMP+2+000:CEL'                           Transport temperature 0 degrees Celsius
RNG+5+CEL:-5:5'                          The range of temperature must be between
                                         -5 and 5 degrees Celsius
MOA+44:45000:EUR'                        Declared valued of the carriage 45.000
                                         EUR
PIA+5+5410738377117:SRV'                 Product identification of the goods
                                         using GTIN 5410738377117
MEA+AAE+X7E+KGM:250'                     Gross weight of returnable pallet plus
                                         14 packages on the pallet is 250
                                         Kilograms
PCI+33E'                                 Marked with the EAN.UCC serial shipping
                                         container code
GIN+BJ+354123450000000014'               Identification of marked serial shipping
```

```
                                        container code
UNT+23+ME000001'                        Total number of segments in the message
                                        equals 23
```

Quelltext 8: Beispiel einer IFTMIN –Nachricht[4]

Genauso wie bei den generischen Transformationsmechanismen dienen in OWL-DL notierte Ontologien der Beschreibung der Datenquellen.

Der folgende Quelltext 9 zeigt einen Auszug aus einer Drools-Regel zur Transformation von EDIFACT EANCOM „IFTMIN"-Nachrichten in eine Ergebnisontologie. Im Besonderen wird hier die Regel zur Transformation desjenigen Nachrichtsegments dargestellt, das zur Definition des Absenders oder Empfängers einer Sendung dient.

```
rule "IFTMIN -NAD "
          when
                    $m : Message ( type == " IFTMIN ")
                    $nad : Segment ( segmentName == "NAD ") from $m. getSegments
()
                    $ind : Individual ( ontClass . toString == (NS +"
Transportauftrag ")
                    || == (NS +" Lagerauftrag "))
                    $qualifier : String ( this != "") from $nad . getDataElement
(0).
                    getElement (0)
                    $party : String ( this != "") from $nad . getDataElement (1).
                    getElement (0)
          then
                    String prop = null ;
                    if( $qualifier . equals (" CN ")){
                    prop = NS +" AuftragsEmpfaenger ";
                    }
          else if( $qualifier . equals (" CZ ")){
                    prop = NS +" AuftragsAbsender ";
                    }
          else
           return ;

          Individual i = insertPropertyToIndividual ( result ,$ind ,NS +"
          Handelspartner ", $party . replaceAll ("\\?:" , ":") ,prop );
end
```

Quelltext 9: Beispiel einer Transformationsregel für EDIFACT EANCOM

Die Wrapperkomponente setzt diese sowie alle anderen zur Transformation der IFTMIN Nachrichten notwendigen Regeln innerhalb des betrachteten Transportlogistikszenarios algorithmisch um. Der vollständige Regelsatz ist in Anhang B aufgeführt.

4.2.2. EPCIS

Die Wrapperkomponente für EPCIS ist in der Lage; XML-formatierte EPCIS-Ereignisse zu lesen und zu schreiben. Dabei erfolgt der Zugriff auf die Ereignisse über die in der EPCglobal Architecture Framework spezifizierten Capture und Query Control Schnittstellen. Als Testumgebung dient dabei die Open Source Umsetzung einer EPCIS Repository im Rahmen des Free and Open Source Software for Track and Trace, FOSSTRAK.[5]

Eine umfangreiche Beschreibung der einzelnen EPCIS Ereignisse ist im Technical Report „Technical Report Standard Data Exchange Formats in the Field of Logistics" zu finden. An dieser Stelle

Genauso wie bei den oben beschriebenen Transformationsmechanismen dienen in OWL-DL notierte Ontologien der Beschreibung der Datenquellen.

[4] Quelle: GS1 Schweden (http://gs1.se/eancom_2002/ean02s4/user/part2/iftmin/examples.htm)

[5] FOSSTRAK Homepage http://www.fosstrak.org/

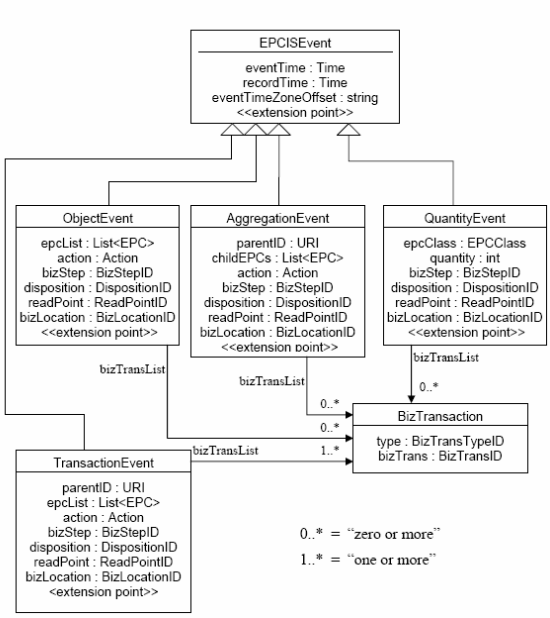

Abbildung 4: Standard EPCIS-Ereignisklassen (Quelle: GS1[6])

Abbildung 4 zeigt ein UML Klassendiagramm, in dem die Kernereignistypen von EPCIS abgebildet sind. In der Wrapperkomponente EPCIS sind zunächst die für das Transportlogistikszenario notwendigen Ereignistypen ObjectEvent und AggregationEvent spezifiziert und umgesetzt worden. Es sind zusätzlich zu den Standarddefinitionen Erweiterungen zur Aufnahme von Sensordaten spezifiziert worden. Hierbei handelt sich um die Felder „Temperatur“ und „Luftfeuchtigkeit“.

Die folgenden Quelltexte zeigen einen Auszug aus den Drools-Regeln zur Transformation von EPCIS Ereignissen in eine Ergebnisontologie.

```
rule " ReadPoint "
          when
                    $event : EPCISDataTypeWrapper ($rp : readPoint != null ,
                    $ind : Individual ()
          then
                    insertObjectPropertyToIndividual ( resultModel , $ind , NS +"
                    Lesegeraet ", $rp ,NS +" WurdeGelesenVon ");
 End
```

Quelltext 10: Beispiele der Transformationsregeln für ERCIS Ereignisse – Readpoint auslesen

Die in Quelltext 10 dargestellte Regel liest den „ReadPoint“ aus einem EPCIS-Ereignis.

```
// Mit dieser Regel wird die Disposition aus einem Event ausgelesen

 rule " Disposition "
          when
                    $event : EPCISDataTypeWrapper ( $dispo : disposition != null
,date : eventTime )
                    $ind : Individual ()
          then
                    insertDatatypeProperty ( resultModel , $ind , NS +" zustand
", null ,dispo );
 end

 // Mit dieser Regel wird der BizStep aus einem Event ausgelesen
```

[6] Quelle: EPC Information Services (EPCIS) Version 1.0.1
Specification http://www.gs1.org/gsmp/kc/epcglobal/epcis/epcis_1_0_1-standard-20070921.pdf

```
 rule " BizStep "
          when
                    $event : EPCISDataTypeWrapper ( $bizStep : bizStep != null ,
date : eventTime )
                    $ind : Individual ()
 then
                    insertObjectPropertyToIndividual ( resultModel , $ind , NS +"
                    Geschaeftsprozess ", $bizStep ,NS +" TeilVonProzess ");
 end
```

Quelltext 11: Beispiele der Transformationsregeln für ERCIS Ereignisse – Disposition auslesen

Die in Quelltext 10 dargestellte Regel liest die „Disposition“ aus einem EPCIS-Ereignis.

```
rule " Extension - Temperatur "
          when
                    $event : EPCISDataTypeWrapper ( eventType == "
                    ObjectEventType ", $date : eventTime )
                    $epc : EPC () from $event . getEPC ()
                    $temp : String () from $event . getExtension (" temperature
")
          then
                    Individual ind = createIndividualAndLinkObjekt ( resultModel
,
                    NS +" TemperaturSensor ", $epc . getValue () ,NS +"
                    SensorVerbindung ",NS +" SensorZuVerbindung ");
                    DatatypeProperty dp = resultModel . createDatatypeProperty
(NS+" gemesseneTemperatur ");
                    SimpleDateFormat dfTo = new SimpleDateFormat (" yyyy -MM -dd
'T' HH:mm:ss 'Z '") ;
                    insertDatatypeProperty ( resultModel , ind , " http ://
localhost:8080/ ontologien / transportszenario .owl# informationsDatum ",
                    XSDDatatype . XSDdateTime , dfTo . format ( $event .
getEventTime ()));
end
```

Quelltext 12: Beispiele der Transformationsregeln für erweiterte ERCIS Ereignisse – Temperatur auslesen

In Quelltext 12 wird beispielhaft verdeutlicht, wie Erweiterungen von EPCIS-Ereignissen durch die Wrapperkomponente erfasst werden können. In diesem Beispiel handelt es sich um die Erweiterung „Temperatur“, die den Standardereignistyp „Object“ um einen Feld erweitert, der Temperaturwerte von Sensoren aufnehmen kann.

Package de.biba.wrapper.tableWrapper

In der folgenden „Package *de.biba.wrapper.EPCISWrapper*“-Tabelle sind alle Klassen die im Mediator-Package *de.biba.wrapper. EPCISWrapper* enthalten sind aufgeführt und beschrieben.

Klasse	Beschreibung
EPCISWrapper	Realisiert die Schnittstelle zum EPCISWrapper und setzt lesende und schreibende Zugriffe um.
VocabularyElement	Greift auf die Elemente der in der EPCIS-Repository definierten Erweiterungen zu.
VocabularAttribute	Greift auf die Attribute der in der EPCIS-Repository definierten Erweiterungen zu.
OntologyGenerator	Überträgt Anfrageergebnisse in die Ergebnisontologie
LegacyStyledElementAnnotations	Setzt die Transformationsregeln für EPCIS algorithmisch um.

Tabelle 8: Package de.biba.wrapper.EPCISWrapper

5. ZUSAMMENFASSUNG UND AUSBLICK

Die hier dokumentierte Spezifikation beschreibt einen semantischen Mediator zur Unterstützung selbststeuernder logistischer Prozesse. Er ist modular und erweiterbar konzipiert und umfasst eine zentrale Mediatorkomponente, eine SPARQL-Anfrageschnittstelle, sowie eine generische Schnittstelle für Wrapperkomponenten gegenüber zu integrierenden Datenquellen. Die Spezifikation von Ontologien bildet die Grundlage zur Beschreibung von Datenquellen und basiert auf der Ontologiebeschreibungssprache OWL-DL. Im Mediator definieren semantische Beschreibungen die in den Zieldatenquellen enthaltenen Daten. Diese Beschreibungen werden vom Mediator über Methoden des Ontologie-Mappings zu einer Gesamtbeschreibung vereinigt, über die Anfragen gestellt werden können. Für den Anfragenden bleibt die Beschaffenheit der einzelnen Datenquellen transparent.

Für strukturierte und semi-strukturierte Standarddatenquellen, wie zum Beispiel relationale Datenbanken, Comma-Separated-Value-Files (CSV) oder XML-Dateien wurden algorithmische Transformationsmechanismen spezifiziert und für ausgewählte Datenquellen als Wrapperkomponenten umgesetzt, erprobt und validiert. Es hat sich gezeigt, dass eine adäquate allgemeingültige Lösung nicht existiert. Die algorithmischen Transformationsmechanismen zeigten sich für die ausgewählten, generischen Integrationsziele SQL und CSV als praxistauglicher Ansatz. Der regelbasierte Ansatz erwies sich in der Umsetzung und Evaluierung für komplexe Nachrichtenformate als sehr komplex und nicht performant. Für solche spezifischen, komplexen Schnittstellentypen wie etwa EDIFACT EANCOM oder EPCIS besteht die Notwendigkeit, dedizierte Algorithmen zu spezifizieren.

Über die SPARQL/UPDATE-Schnittstelle lassen sich sowohl lesende als auch schreibende Zugriffe auf beliebige Datenquellen verwirklichen. Für die Anfrageschnittstelle ist über das SPARQL Protokoll eine Web Service-Umsetzung definiert. Um eine vollständige Umsetzung der Funktionalität des semantischen Mediators als Web Service zu verwirklichen, sind an dieser Stelle Anpassungen der Servicedefinition notwendig. Dies ist insbesondere auf die Nutzung der experimentellen Erweiterung SPARQL/UPDATE zur Gewährleistung des Schreibzugriffs auf die Datenquellen zurückzuführen. Infolgedessen muss die Servicedefinition ebenfalls um die durch SPARQL/UPDATE eingeführten Syntaxerweiterungen ergänzt werden.

DANKSAGUNGEN

Diese Arbeit wurde durch die Deutsche Forschungsgemeinschaft im Rahmen des Sonderforschungsbereichs 637 "Selbststeuerung logistischer Prozesse – Ein Paradigmenwechsel und seine Grenzen" unterstützt.

Einen besonderen Beitrag zu dieser Arbeit haben auch Dennis Stahlhut und Moritz von Stietencron geleistet.

ANHANG A VOLLSTÄNDIGE KLASSENDIAGRAMME

Klassendiagramm der Mediator-Komponente

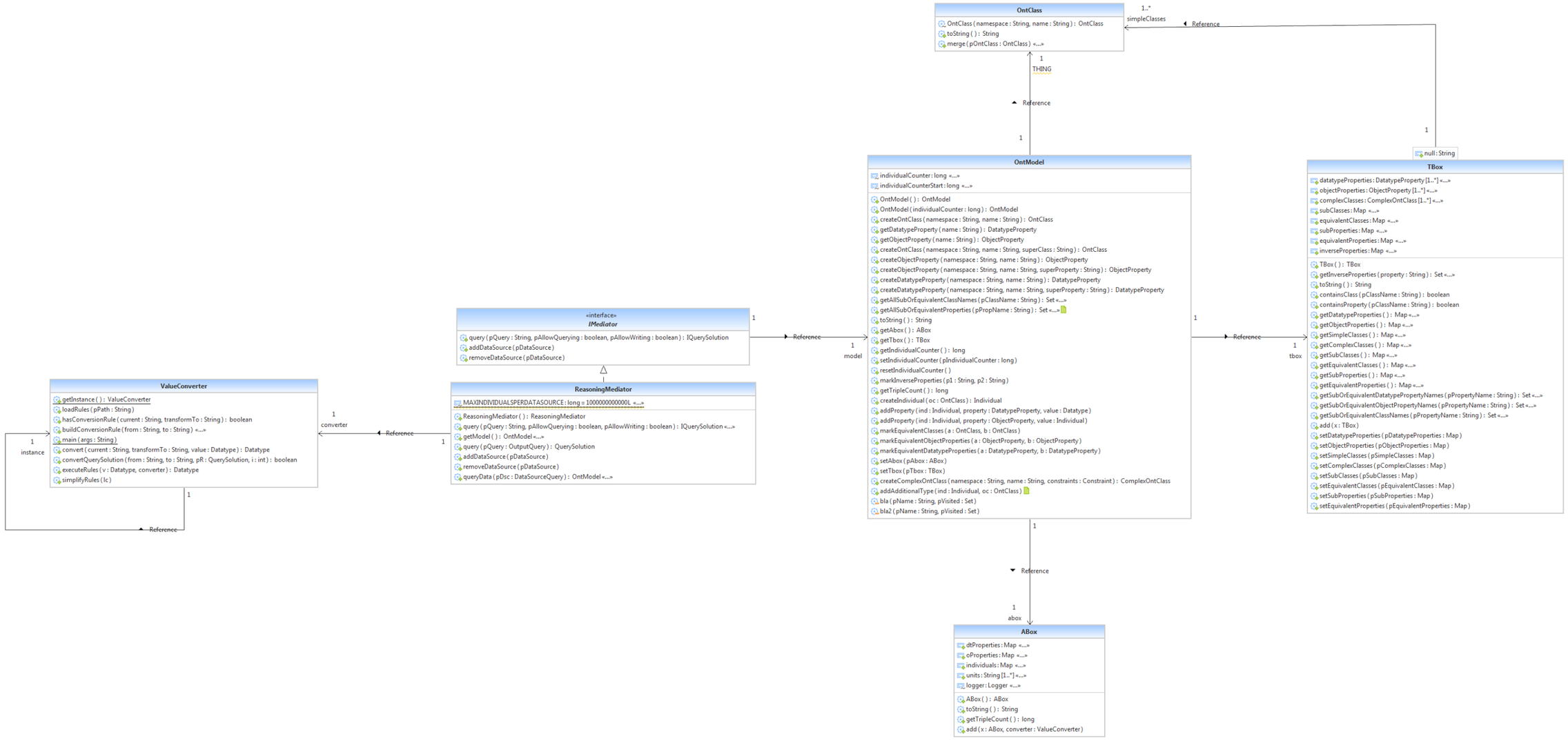

Abbildung 5: Ausschnitt aus dem Klassendiagramm der Mediator-Komponente

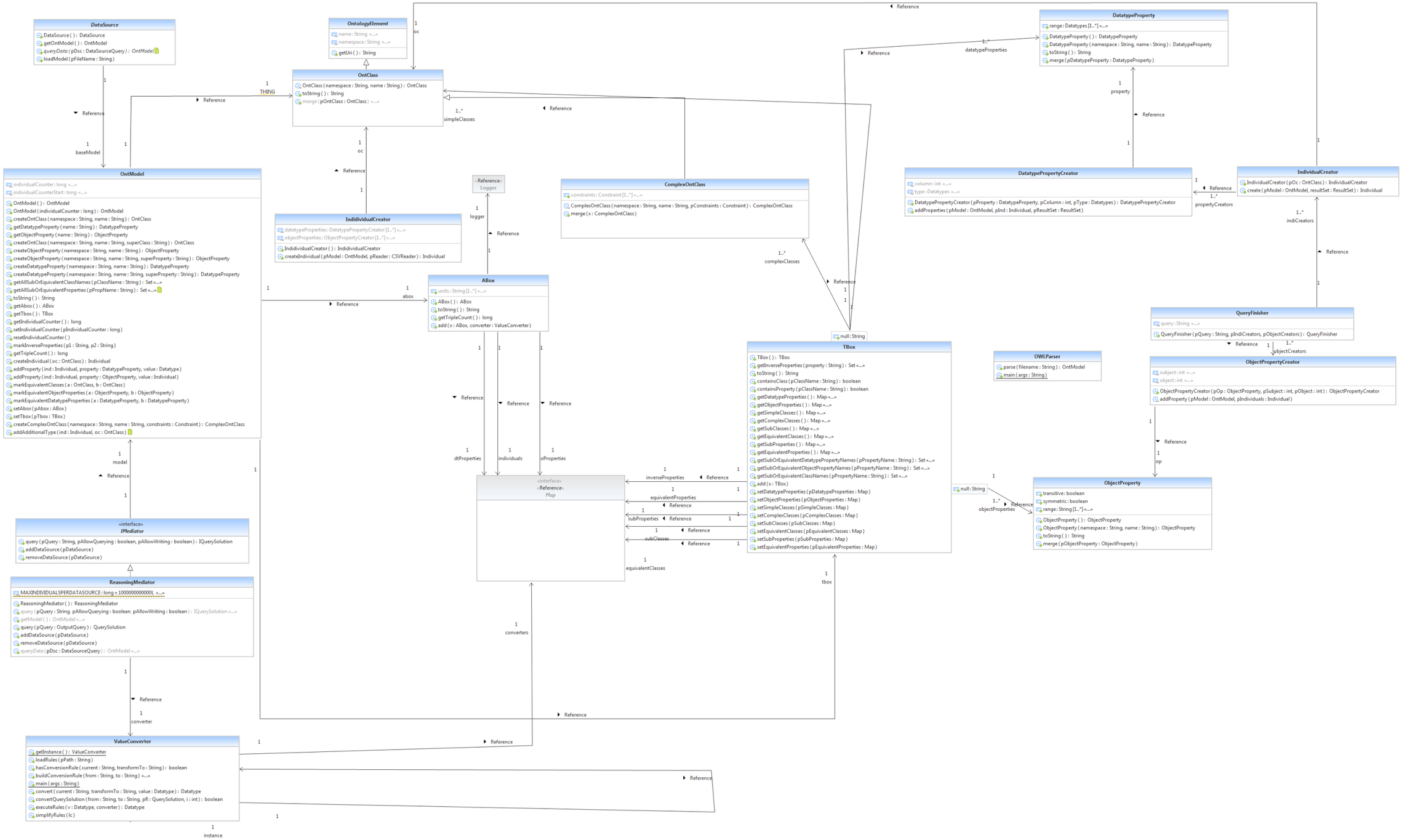

Abbildung 6: Klassendiagramm der Mediator-Komponente

Klassendiagramm der SPARQL-Schnittstelle

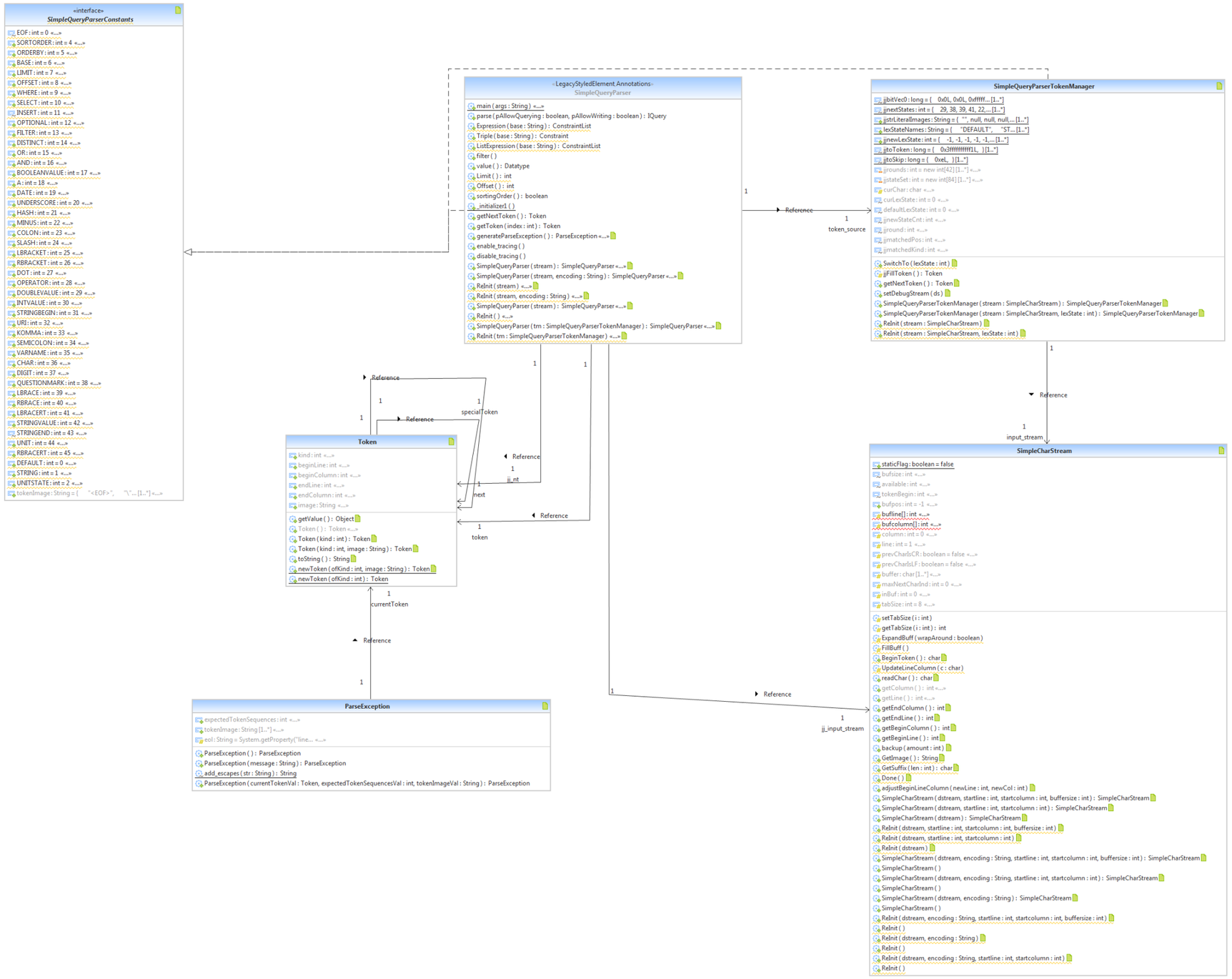

Abbildung 7: Klassendiagramm der SPARQL-Schnittstelle

Klassendiagramm des CSV-Wrappers

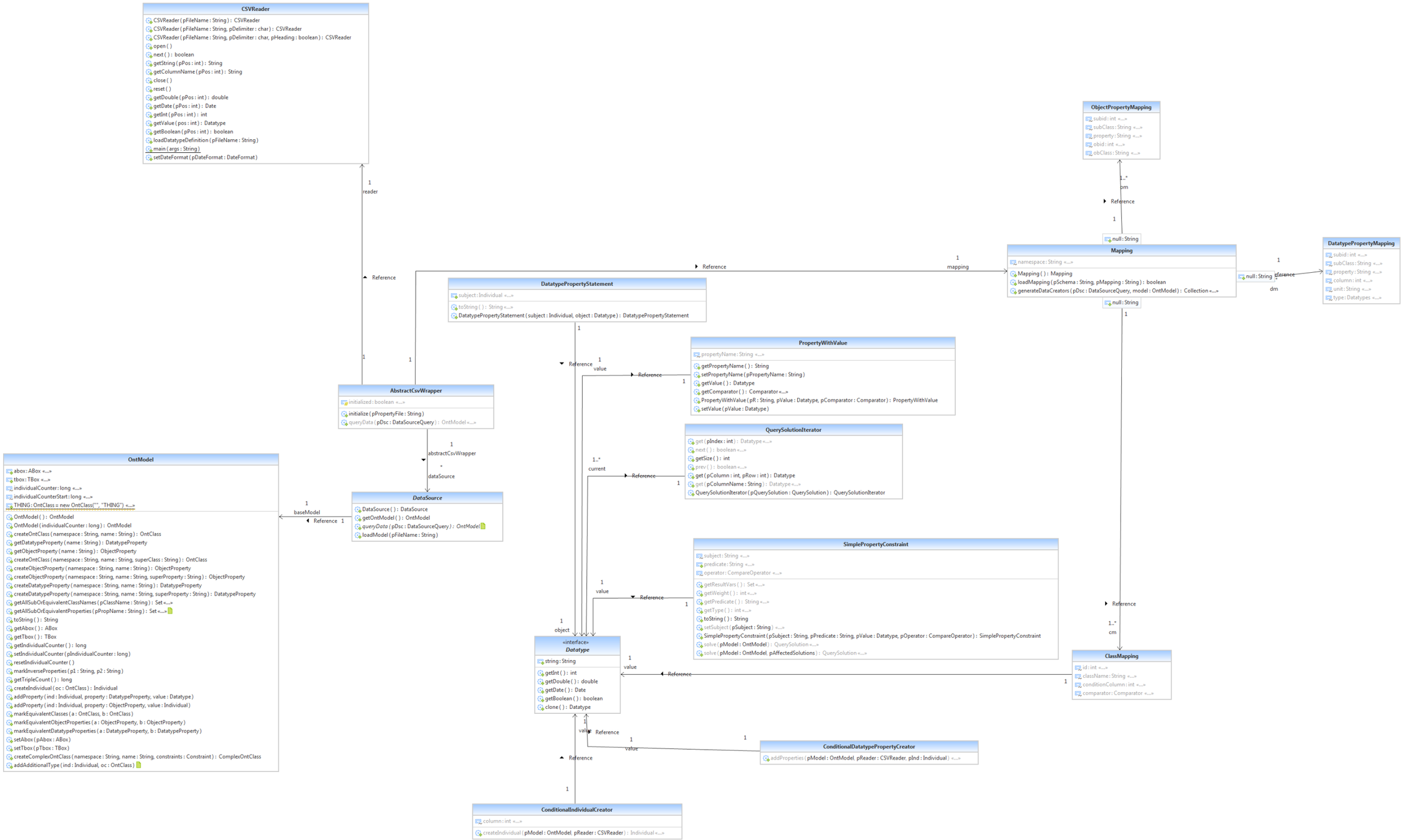

Abbildung 8: Klassendiagramm des CSV-Wrappers

Klassendiagramm des SQL-Wrappers

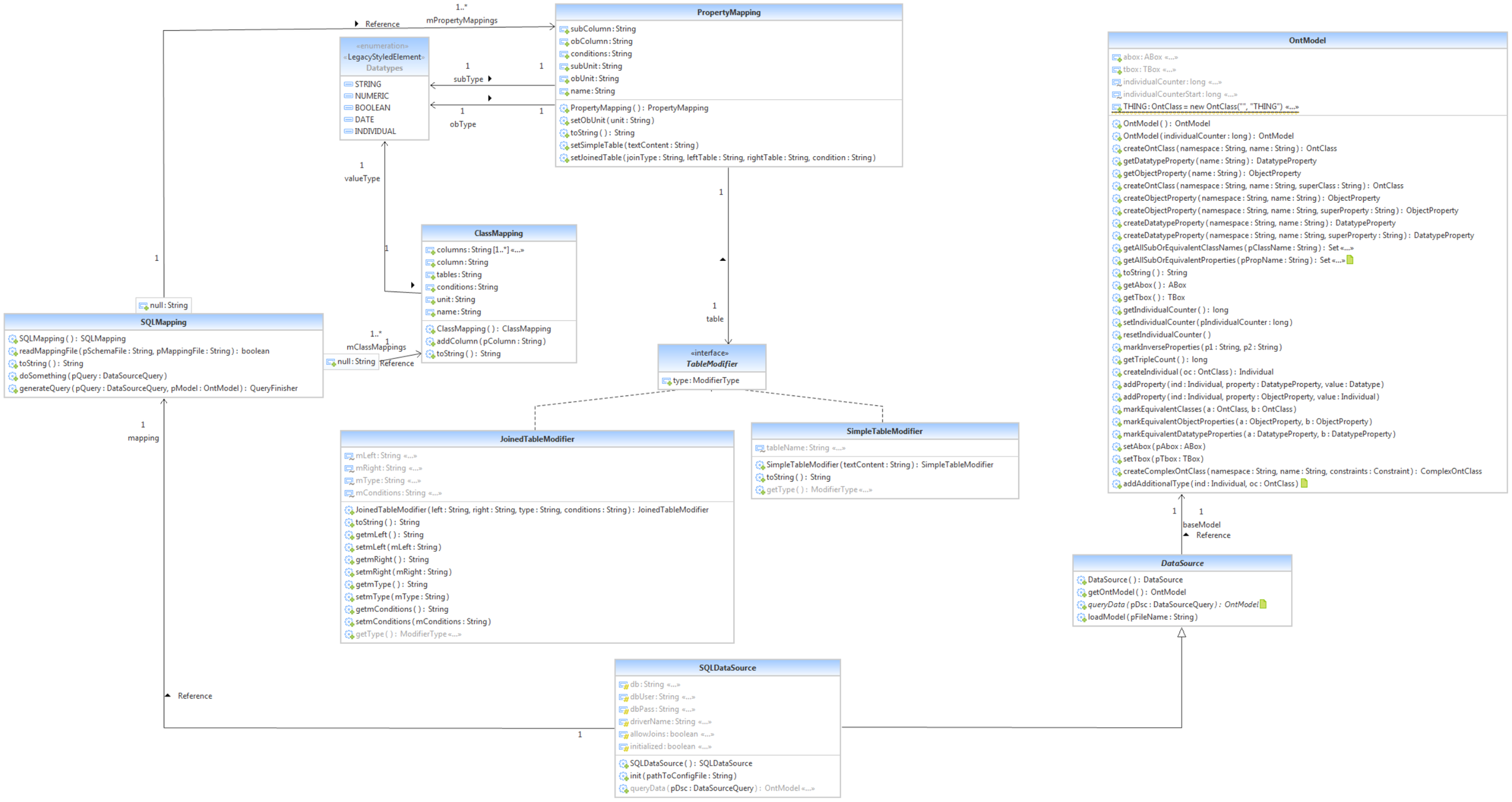

Abbildung 9: Klassendiagramm des SQL-Wrappers

Klassendiagramm des EPCIS Wrappers

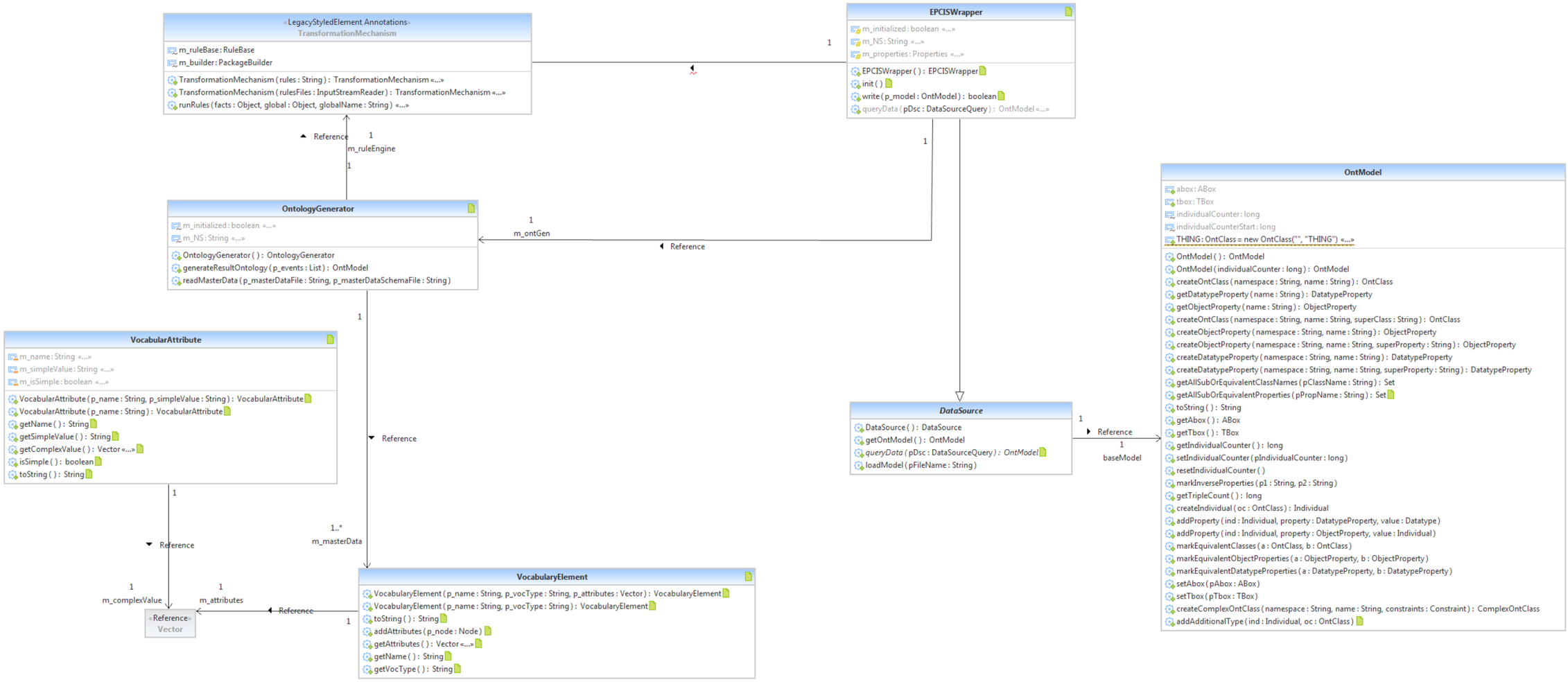

Abbildung 10: Klassendiagramm des EPCIS Wrappers

Klassendiagramm des EANCOM Wrappers

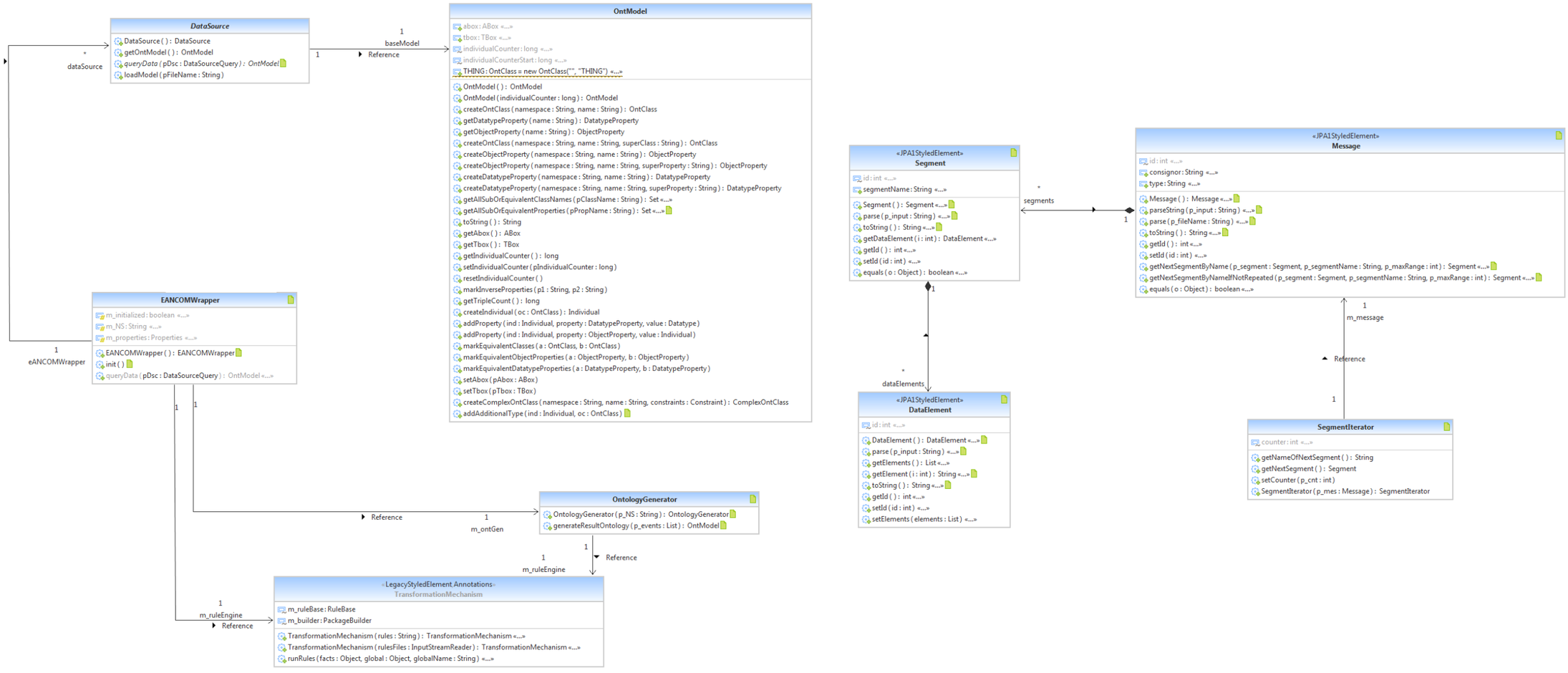

Abbildung 11: Klassendiagramm des EANCOM Wrappers

ANHANG B EDIFACT EANCOM TRANSFORMATIONSREGELN

Beispiel: IFTMIN-Nachricht

```
package rules
// Diese Regel initialisiert das Einlesen einer IFTSTA - Nachricht, indem es ein Transport -
oder Lagerauftragsinstanz
// erstellt und dem Workingspace hinzufügt , welche Bedingung für die weiteren Regeln ist
rule " IFTMIN "
          salience -5
          when
                    $m : Message ( type == " IFTMIN ")
                    $bgm : Segment ( segmentName == "BGM ") from $m. getSegments ()
                    $messageName : String ( this != "") from $bgm . getDataElement (1).
                              getElement (0)
          then
                    Segment ftx = $m. getNextSegmentByName ($bgm , "FTX", 5);
                    OntClass oc;
                    if( ftx != null && ftx . getDataElement (0) . getElement (0). equals ("
                              AAI ") && ftx. getDataElement (3). getElement (0). equals ("
                              LAGERAUFTRAG "))
                              oc = result . createClass (NS +" Lagerauftrag ");
                    else
                              oc = result . createClass (NS +" Transportauftrag ");
                              Individual ind = result . createIndividual ( $messageName ,oc);
                              insert (ind );
end

 // Diese Regel liest die Priorität eines Auftrags aus einer Nachricht aus.

rule "IFTMIN -TSR "
          when
                    $m : Message ( type == " IFTMIN ")
                    $tsr : Segment ( segmentName == "TSR ") from $m. getSegments ()
                    $ind : Individual ( ontClass . toString == (NS +" Transportauftrag ")
                    || == (NS +" Lagerauftrag "))
          then
                    if( $tsr . getDataElement (2) . getElement (0) . equals ("1") || $tsr .
                    getDataElement (2) . getElement (0) . equals ("2") || $tsr .
                    getDataElement (2) . getElement (0) . equals ("3") ){
                     insertDatatypeProperty2 (result , $ind ,NS +" prioritaet ",
                    XSDDatatype . XSDint , $tsr . getDataElement (2) . getElement (0) );
          }
End

// Diese Regel liest den Transport - oder Lagerbeginn einer Nachricht aus.

rule "IFTMIN -DTM "
          when
                    $m : Message ( type == " IFTMIN ")
                    $dtm : Segment ( segmentName == "DTM ") from $m. getSegments ()
                    $ind : Individual ( ontClass . toString == (NS +" Transportauftrag ")
                    || == (NS +" Lagerauftrag "))
                    String ( this == "235") from $dtm . getDataElement (0). getElement
                    (0)
                    String ( this == "203") from $dtm . getDataElement (0). getElement
                    (2)
          then
                    String propName = " transportbeginn ";
                    if( $ind . getOntClass (). toString (). equals (NS +" Lagerauftrag "))
                    propName = " lagerbeginn ";
                    SimpleDateFormat dfTo = new SimpleDateFormat (" yyyy -MM -dd 'T'HH:mm:ss 'Z
'") ;
                    SimpleDateFormat dfFrom = new SimpleDateFormat (" yyyyMMddHHmm ");
                    insertDatatypeProperty2 (result , $ind ,NS+ propName , XSDDatatype .
                    XSDdateTime , dfTo . format ( dfFrom . parse ( $dtm . getDataElement(0) .
getElement (1))));
end

// Diese Regel liest die Referenz eines Auftrags aus.

rule "IFTMIN -RFF "
          when
                    $m : Message ( type == " IFTMIN ")
                     dtm : Segment ( segmentName == "RFF ") from $m. getSegments ()
                    ind : Individual ( ontClass . toString == (NS +" Transportauftrag ")
```

```
                    || == (NS +" Lagerauftrag "))
                    String ( this == "BN ") from $dtm . getDataElement (0). getElement (0)
                    String ( this != "") from $dtm . getDataElement (0). getElement (1)
          then
                    insertPropertyToIndividual (result ,$ind ,NS +" Auftragsanfrage ",
                    $dtm . getDataElement (0) . getElement (1) . replaceAll ("\\?:" ,
                    ":") ,NS +" HatReferenz ");
          end

// Diese Regel liest den Auftragsempfänger oder Absender aus.

rule "IFTMIN -NAD "
          when
                    $m : Message ( type == " IFTMIN ")
                    $nad : Segment ( segmentName == "NAD ") from $m. getSegments ()
                    $ind : Individual ( ontClass . toString == (NS +" Transportauftrag ")
                    || == (NS +" Lagerauftrag "))
                    $qualifier : String ( this != "") from $nad . getDataElement (0).
                    getElement (0)
                    $party : String ( this != "") from $nad . getDataElement (1).
                    getElement (0)
          then
                    String prop = null ;
                    if( $qualifier . equals (" CN ")){
                    prop = NS +" AuftragsEmpfaenger ";
                    }
          else if( $qualifier . equals (" CZ ")){
                    prop = NS +" AuftragsAbsender ";
                    }
          else
           return ;

          Individual i = insertPropertyToIndividual ( result ,$ind ,NS +"
          Handelspartner ", $party . replaceAll ("\\?:" , ":") ,prop );
end

// Diese Regel liest Informationen über das Frachtgut aus. Dadurch das Folgesegmente ins
Workingspace kopiert werden ,
// werden Regeln aus IFTMBF .drl zur weitern Verarbeitung benutzt .

rule "IFTMIN -GID "
          when
                    $m : Message ( type == " IFTMIN ")
                    $gid : Segment ( segmentName == "GID ") from $m. getSegments ()

                    $ind : Individual ( ontClass . toString == (NS +" Transportauftrag ")
          || == (NS +" Lagerauftrag "))

          then
                    Individual item ;
                    String itemname ;
                    Segment pia = $m. getNextSegmentByNameIfNotRepeated ($gid ," PIA" ,5);
                    if( pia != null && pia . getDataElement (0) . getElement (0). equals("5") )
                    itemname = pia. getDataElement (1). getElement (0). replaceAll("\\?:" ,
":") ;
          else

          item = insertPropertyToIndividual (result , $ind , NS +" Frachtgut", itemname , NS
+" HatFrachtobjekt ");

// Anzahl der Kartons und Paletten lessen

          for(int i =1; i <=5; i++){String packageQuantity = $gid . getDataElement (i).
getElement(0);
                    if( packageQuantity . equals ("") )
                    break ;
          String packageType = $gid . getDataElement (i). getElement (1);
          String propName ="";
          if( packageType . equals ("09") ){
                    propName = NS +" anzahlPfandPalettenDesFrachtguts ";}
          else if( packageType . equals (" CT ")){
                    propName = NS +" anzahlKartonsDesFrachtguts ";}
          else
           continue ;
          insertDatatypeProperty2 (result , item , propName , XSDDatatype .XSDint ,
packageQuantity );
          }
```

```
// Alle Folgesegment bis zum nächsten GID - Segment in Workingspace kopieren

         List segments = $m. getSegments ();
         for(int i=0; i< segments . size ();i++){
                  Object o = segments .get (i);
                  if(o. equals ( $gid )){
                           for(int j=1; j+i < segments . size (); j++){
                                    Segment s = ( Segment ) segments .get(i+j);
                                    if (!s. getSegmentName (). equals (" GID ")){
                                    insert (s);
                           }
                  else {
                  break ;
                           }
                  }
         }
}
insert ( item );
end
```