Web Processing Service For Moving Objects Analysis

Tobias Fleischmann

U1249

UNIGIS MSc 2006

Oberteuringen, 16. April 2008

UNIVERSITÄT SALZBURG

Foreword/Dedication/Thanks

Foreword

This Master Thesis is about spatio-temporal analysis. An interface shall be developed allowing one to ask questions about moving objects. The approach is to carry over concepts from Moving Objects Databases and spatio-temporal modelling to develop a service that processes geodata into movement information in an Open Geospatial Consortium (OGC) compliant way. The general goal is to support users who need information about moving objects to answer queries or support decisions with a geoprocessing service in a web service environment.

Dedication

To Regina and Mirja.

Thanks

I would like to thank all who supported me in writing this Master Thesis.

Many thanks to Dr. Christian Kiehle, Dr. Markus Schneider and Prof. Dr. Zdravko Galic for providing articles and assistance in literature research.

Special thanks goes to Prof. Dr. Strobl and the whole UNIGIS team for discussing my proposal of subject and preparing the Thesis.

I also thank in particular Dr. Martin Huber for mentoring, for discussing key questions and concepts. Thanks once more to Dr. Christian Kiehle for his assistance in setting up the prototype and coping with the deegree framework.

Declaration on Independent Work and Plagiarism

With this, I declare that I have written this paper on my own, distinguished citations, and used no other than the named sources and aids. It is submitted for the degree MASTER OF SCIENCE in the UNIGIS MSc course at the Z_GIS at the University of Salzburg. No part of this research work has been submitted in the past, or is being submitted, for a degree or examination at any other University.

Abstract

OpenGIS compliant geodata registry, portrayal and access services are well established and integral part of SDIs. However, geodata manipulation services, processing the increasing volume of available geodata into value-added information, are not present. Current trends in consumer electronics and developments in the military domain afford sensors generating a huge volume of spatio-temporal data that needs to be further analysed. Here, the author presents a geoprocessing service to process movement data.

In a Model Driven Architecture (MDA) software engineering approach, starting with Use Case Analysis, concepts from spatio-temporal modelling and Moving Objects Databases are applied. A platform independent model describes a moving object data type with respective analysis operations. Finally, a service is designed to web-enable the processing functionality.

The recently approved OGC Web Processing Service Implementation Specification allows the definition of custom processes. A custom process is executing the actual data processing while the standardized WPS interface specifies how input and output data are described and how the service request is handled. Because process input data require GML encoding, an Application Schema for moving objects, based on the Dynamic Feature Core Schema, is developed.

A web application, utilizing the deegree SW framework, demonstrates the feasibility of the designed concept. A crosses process, a spatio-temporal predicate testing whether the time-varying position of a moving object first enters and then leaves a specified geometry, is implemented and deployed on an Apache Tomcat Servlet Container.

This Master Thesis connects concepts that are innately discrete: analysis of moving objects and OpenGIS Web Services (OWS). By identifying moving objects as spatio-temporal entities with a time-varying position and moving objects analysis as a geoprocessing task, the OGC standards framework contains the proper measures to implement a standardized, distributed processing service for spatio-temporal data: the OWS Implementation Specification for developing a WPS interface and the Geography Markup Language (GML) with the Dynamic Feature Schema to create an Application Schema defining a moving object data type.

Possible steps to continue this work could investigate Web Service Orchestration (WSO) to realize complex spatio-temporal processing workflows, semantic interoperability or consider performance aspects.

Kurzfassung

OpenGIS konforme Katalog-, Darstellungs- und Zugriffsdienste für verteilte Geodaten sind weit verbreitet und integraler Bestandteil von Geodateninfrastrukturen (GDIs). Daten verarbeitende Dienste zur Prozessierung in großer Menge vorliegender Geodaten in mehrwertschaffende Informationen sind jedoch kaum vorhanden. Aktuelle Entwicklungen in der Unterhaltungselektronik und im militärischen Sektor bringen Sensoren hervor, die massig räumlich-temporale Daten erzeugen und daher einer nachgelagerten Verarbeitung bedürfen. Der Autor präsentiert in dieser Arbeit einen Geoprozessierungsdienst zur Analyse beweglicher Objekte.

In einem auf den Prinzipien von Model Driven Architecture (MDA) basierenden Softwareentwicklungsansatz, beginnend mit einer Analyse von Anwendungsfällen, werden Konzepte aus den Disziplinen der räumlich-temporalen Modellierung und Moving Objects Databases zur Ableitung der SW Architektur angewandt. Ein plattformunabhängiges Modell beschreibt einen Datentyp für bewegliche Objekte mit entsprechenden Analyseoperationen. Im nachfolgenden Schritt wird darauf aufbauend ein Dienst entwickelt, der diese Verarbeitungsfunktionalität mittels Internettechnologie zugreifbar macht.

Der erst kürzlich als Implementierungsspezifikation verabschiedet OGC Web Processing Service Standard erlaubt die Definition anwendungsspezifischer Prozesse. Solch ein individueller Prozess führt die eigentlich Datenverarbeitung durch, während die WPS Schnittstelle festlegt, wie Ein- und Ausgabedaten zu sind und Aufrufe des Dienstes abzuwickeln sind. Um Eingabedaten, wie es der Standard vorgibt, GML codiert als Parameter übergeben zu können, wird ein entsprechendes Anwendungsschema für bewegliche Objekte, basierend auf dem sogenannten Dynamic Feature Core Schema, erstellt.

Eine Webapplikation, auf Basis des deegree SW Frameworks entwickelt, weist die Umsetzbarkeit des entwickelten Konzeptes nach. Ein Crosses-Prozess, welcher den zeitlichen Verlauf einer räumlich-temporalen Beziehung zwischen einem beweglichen Objekt und einem Interessensgebiet untersucht, wird prototypisch implementiert und auf einem Apache Tomcat Servlet Container deployt.

Diese Arbeit verknüpft Konzepte, welche von Haus aus keine direkte Beziehung zueinander haben: Analyse von beweglichen Objekten und OpenGIS Web Services (OWS). Betrachtet man bewegliche Objekte als räumlich-temporale Instanzen mit einer sich im Laufe der Zeit ändernden Position und deren Analyse als eine Geoprozessierungsaufgabe, so bietet das OGC Framework mit seinen Standards die geeigneten Mittel, einen standardisierten, verteilten Dienst zur Verarbeitung räumlich-temporaler Daten zu implementieren: eine OWS Implementation Specification zur Entwicklung

einer WPS Schnittstelle und die Geography Markup Language (GML) mit dem Dynamic Feature Schema zur Erstellung eines Applikationsschemas zur Definition eines Datentyps für bewegliche Objekte.

Zur Fortsetzung dieser Arbeit bieten sich einige Themengebiete an: Analyse von Web Service Orchestrierung (WSO) zum Aufbau komplexer, räumlich-temporaler Verarbeitungsketten, Untersuchung semantischer Interoperabilität oder die Betrachtung von Performance-Aspekten zur Effizienz- und Effektivitätssteigerung von Web Service Infrastrukturen.

Table of Contents

Foreword/Dedication/Thanks ..... **i**

Declaration on Independent Work and Plagiarism ..... **iii**

Abstract ..... **v**

Kurzfassung ..... **vii**

List of Illustrations ..... **xiii**

List of Tables ..... **xv**

List of Abbreviations Used ..... **xvii**

1 Introduction **1**

1.1 Motivation *1*

1.2 Objective *1*

1.3 Approach *2*

1.3.1 Theory *2*

1.3.2 Methods *4*

1.3.3 Tools *4*

1.3.4 Test Area / Test Data Sets *5*

1.4 Expected Results *5*

1.5 Topics Not Explored *5*

1.6 Target Audience *6*

1.6.1 Professional Language *6*

1.6.2 Familiarity with Subject *6*

1.7 Thesis Structure *7*

2 Literature Survey **9**

3 Hypothesis **13**

3.1 Theoretical Approach *13*

3.1.1 Spatio-Temporal Concepts *13*

3.1.2 Moving Objects Databases *14*

3.1.3 Spatio-Temporal Conceptual Data Modelling *21*

3.1.4 OpenGIS Standards and Specifications *22*

3.1.5 Principles Of Object Oriented Software Engineering *27*

3.2 Methods *30*

3.3 Tools *32*

Table of Contents

4 Project Description 33
4.1 Domain Model (CIM) 33
4.2 Non-Functional Requirements 35
4.3 Use Case Model 36
4.4 Analysis Model 37
4.4.1 Use Case Realization 37
4.4.2 View of Participating Classes 38
4.5 Design Model 40
4.5.1 Abstract Class Model (PIM) 40
4.5.2 GML Data Model (PSM) 43
4.5.3 OWS Model 45
4.6 Prototypical Web Service Implementation 47
4.6.1 Architectural Approach 47
4.6.2 Java Framework Deegree 49
4.6.3 Implementation, Configuration and Deployment 51
4.6.4 Test Area And Data 56
4.6.4.1 Case Study 56
4.6.4.2 Outline Of Spatio-Temporal Problem 57
5 Results 59
6 Analysis of Results 61
7 Summary, Discussion, Outlook 63
7.1 Summary 63
7.2 Discussion 63
7.3 Outlook 64
7.3.1 Generalisation of Proposed Solution 64
7.3.2 Proposed Steps For Continuation of Work 65
Bibliography 67
Annex A 69
A.1 Domain Model 69
A.2 Requirements Model 71
A.3 Use Case Model 73
A.4 Analysis Model 87
A.5 Abstract Class Model 88
A.6 GML Data Model 113
A.7 OWS Model 118

Table of Contents

A.8 Java Model..... 142
A.9 Component Model..... 156
A.10 Deployment Model..... 158
Annex B..... 159
B.1 Moving Objects GML Application Schema..... 159
Annex C..... 163
C.1 Deployment Descriptor..... 163
C.2 Capabilities Document..... 166
C.3 Process Description Document..... 169
C.4 Execute Request Document..... 171
C.5 Test Data (GML)..... 175
Annex D..... 178
D.1 Process Class..... 178

List of Illustrations

Figure 3.1: Type System..... 16
Figure 3.2: Predicates..... 17
Figure 3.3: Set Operations..... 17
Figure 3.4: Aggregate Operations..... 17
Figure 3.5: Numeric Operations..... 18
Figure 3.6: Distance And Direction Operation..... 18
Figure 3.7: Projection Operations..... 19
Figure 3.8: Interaction With Values In Domain And Range..... 19
Figure 3.9: Derivative Operations..... 19
Figure 3.10: Basic Spatio-Temporal Predicates..... 21
Figure 3.11: Methods applied..... 31
Figure 4.1: Domain Model..... 34
Figure 4.2: Use Case Model..... 37
Figure 4.3: Use Case Realization..... 38
Figure 4.4: View of Participating Classes..... 39
Figure 4.5: Moving Objects Analysis Subsystem & Interface..... 40
Figure 4.6: Interface Operations & Implementation..... 42
Figure 4.7: Dynamic Feature GML Core Schema..... 43
Figure 4.8: Moving Objects Application Schema..... 45
Figure 4.9: Web Processing Service..... 46
Figure 4.10: Describe Process Request & Response..... 47
Figure 4.11: Multi-Tier Architecture..... 48
Figure 4.12: Crosses Process..... 49
Figure 4.13: Deegree Framework For Building Web Applications..... 50
Figure 4.14: Deegree WPS Configuration..... 52
Figure 4.15: Runtime Environment..... 53
Figure 4.16: Web Application Component View..... 53
Figure 4.17: Web Application Business Logic..... 54
Figure 4.18: Generic OGC Web Service Client..... 55
Figure 4.19: Web Application Deployment View..... 56
Figure 4.20: Case Study..... 57

List of Tables

Domain Model..... 35
View of Participating Classes..... 40

List of Abbreviations Used

WWW: World Wide Web..... 1
OGC: Open Geospatial Consortium..... 1
OWS: OpenGIS Web Service..... 1
SDI: Spatial Data Infrastructure..... 1
LBS: Location Based Services..... 1
WMS: Web Map Service..... 1
WCTS: Web Coordinate Transformation Service..... 1
AOI: Area Of Interest..... 2
CASE: Computer Aided Software Engineering..... 4
IDE: Integrated Development Environment..... 4
SQL: Structured Query Language..... 9
MADS: Modeling of Application Data with Spatio-temporal Features..... 9
GIS: Geographic Information System..... 22
ORM: OpenGIS Reference Model..... 23
XML: Extensible Markup Language..... 24
URL: Uniform Resource Locator..... 24
KVP: Keyword Value Pair..... 25
MIME: Multipurpose Internet Mail Extensions..... 25
WCS: Web Coverage Service..... 26
WFS: Web Feature Service..... 26
WSO: Web Service Orchestration..... 26
MDA: Model Driven Architecture..... 27
OOA: Object Oriented Analysis..... 28
OOD: Object Oriented Design..... 28
OMG: Object Management Group..... 29
W3C: World Wide Web Consortium..... 30
GML: Geography Markup Language..... 30
XSD: XML Schema Definition..... 30
CIM: Computation Independent Model..... 30
PIM: Platform Independent Model..... 31
PSM: Platform Specific Model..... 31

List of Abbreviations Used

MOD: Moving Objects Databases..... 31
OO: Object Orientation.....32
URI: Uniform Resource Identifier.....52
GPX: GPS Exchange Format..... 57

1 Introduction

1.1 Motivation

In the geoinformation science, interoperability is the crucial factor. As a result of the big success of the WWW, distributed processing and service oriented architectures are the determining trends in the information technology. This development has conceived the geoinformation technology as well and monolithic systems are going to be replaced by collaborating services. The OGC is an organization which develops standards specifying OpenGIS Web Services (OWS). The importance of the OGC service framework increases by the ongoing activities for building SDIs on various administrative levels. Geoprocessing services extend current available SDI services, focusing on the provision of geospatial data, by providing a method to process geodata into value-added information.

Analysis of moving targets is playing a major role in the military surveillance and reconnaissance domain: sensors are tracking man-made objects like vehicles and low flying aerial platforms. Non military sensors like the upcoming European satellite system GALILEO will foster current trends in consumer electronics to equip mobile devices with GPS receivers. RFID tags are used for indoor tracking of products. All theses trends contribute that a huge volume of movement data, also called trajectories, will become available that need to be managed and analysed. This will lead to the development of many kinds of new applications, such as LBS.

This Thesis examines if moving objects analysis can be performed by means of OGC concepts and therefore encounters objectives of current research.

1.2 Objective

Until recently, the range of OGC standards covered only portrayal (e.g. WMS) and information providing (e.g. WFS, WCS) services and lacked of *true* geoprocessing services which process raw geodata into valuable information. Simple processing services executing arithmetic operations like WCTS were specified very well but don't represent *true* geoprocessing in the aforementioned sense. The OGC Web Processing Service (WPS) overcomes the aforementioned restrictions and offers access to geospatial processing functionality.

The major goal of this work is to apply the concepts of spatio-temporal modelling and Moving Objects Databases to the development of a geoprocessing service. By associating moving objects analysis functionality with the OGC standard framework, an OpenGIS compliant WPS shall be de-

veloped. The resulting service shall describe an interoperable interface providing processes to conduct spatial and temporal analysis of moving objects, e.g. query moving objects crossing a specified Area Of Interest (AOI), which goes beyond the first approaches to define processes by implementing standard GIS functions like buffering as a WPS.

A prototype shall prove the interface concept by implementing a selected process.

1.3 Approach

1.3.1 Theory

Spatio-temporal concepts that accommodate well known spatial concepts to include time fail. Because of the differences between spatial and temporal dimensions, simple dimensioning-up strategies work poorly. Alike, the enhancement of temporal concepts to include spatial data types works not for moving objects. Spatio-temporal data live in a 3D space which is the cross-product of the spatial and the time domain.

To describe continuous movement, Moving Object Databases, as described in (Güting & Schneider, 2005), offer abstract data types with suitable operations. Moving objects are geometries changing over time, moving points, in contrast to moving regions, are entities for which only the time-dependent position in space is relevant and not the extent. The spatio-temporal data perspective in a database is looking at histories of movement concerning the valid time. Querying histories of movement or evolutions of spatial objects over time requires a spatio-temporal model providing respective data types and operations as well as spatio-temporal predicates.

Main spatio-temporal data types are moving point and moving region. Additionally to this main types, related data types with corresponding time-dependent (also called temporal or moving) types are needed. Operations can be classified into operations on temporal types, operations on non-temporal types and operations on set of objects. An approach, called temporal lifting, first designs operations on non temporal types and then extends temporal variants. Topological predicates alter when temporal changes of spatial objects occur. The subject of spatio-temporal predicates therefore is to examine topological predicates over time. The application of temporal lifting to topological predicates requires additionally temporal aggregation to avoid undefined spatio-temporal predicates. An undefined result would violate the essence of predicates that have to be either true or false. Temporal aggregation is realized by defining an aggregate operator. An universal quantifier aggregates only over a restricted time interval and can therefore avoid that a moving object is undefined when examining a spatio-temporal relationship. Developments represent the change of spatial situations over time. Observations on developments reveal however that predicate constrictions have to be defined and predicates need to be distinguished between instant predicates and period predicates. A concise

syntax for developments denotes temporal composition of spatial and spatio-temporal predicates by a sequence of predicates linked by an infix operator. Beside temporal composition, there are other logical connectives to combine predicates and obtain new complex or compound predicates.

The MADS approach describes a conceptual model, free from implementation-based limitations, addressing data modelling and manipulation for space and time dimension. Objects that are subjects to change over time are called time-varying phenomena. According to the MADS definition, a spatio-temporal object type has both, a spatial and a temporal extent, separately, or has a time-varying spatial extent. The life cycle of an object describes the history of his evolution. MADS supports basic spatial and temporal data types and spatio-temporal data types for moving objects along with associated operations and predicates. MADS extends the concept of time-varying or moving data types and operations, introduced in (Güting & Schneider, 2005), to the space dimension by supporting additional space-varying data types and associated operations.

The Open Geospatial Consortium is leading the development of standards and specification for geospatial services. The technical documents of the OGC detail interfaces of software components to enable GIS interoperability.

The conceptual foundation for the Implementation Specifications, which describe implementation details, is provided by the Abstract Specifications. Herein, the feature concept, which is a central term in the “OpenGIS world”, is introduced. Furthermore, the possibility for an information community to define own Application Schemas based on the provided conceptual schemas is explained. GML, an XML grammar written in XML Schema, is the language for describing and encoding geospatial information. The paper also provides key definitions for the service framework, that is a service, an interface and an operation is defined. The interfaces of OGC Web Services have explicit bindings for HTTP. The OGC Web Service Common Specification specifies many aspects that are common to all OWS interface Implementation Specifications, most notably the mandatory GetCapabilities operation.

The WPS Implementation Specification specifies the interface to a Web Processing Service which facilitates the publishing of geospatial processes and the discovery of and binding to those processes by clients. A WPS is able to offer any sort of GIS functionality. The WPS interface standardizes the way, how processes and their input and output data are described, how a client can request a process execution and how the output is handled. The WPS interface therefore offers three mandatory operations: GetCapabilities, DescribeProcess and Execute.

The GML Core Schemas include the Dynamic Feature Schema, which defines a number of types and relationships to represent the time-varying properties of geographic features.

The MDA approach follows three main goals, namely portability, interoperability and re-usability. To accomplish these goals, models were built and

regarded from different views. A CIM contains no details about a systems structure, a PIM no details about concrete platforms and the PSM finally extends the PIM about such concrete platform details. The principles of object orientation, basically abstraction, encapsulation, modularity and hierarchy, foster the MDA approach. OOA and OOD shall transform system requirements into a system design, evolve a robust architecture for the system and adapt the design to match the implementation environment. The UML is a de-facto standard in OOD, just a language, process independent and optimally used in a Use Case driven development process. The OO programming language Java finally is used to implement the prototype.

1.3.2 Methods

The MDA approach represents the engineering guideline for software development. To reduce SW system's complexity, a UML model is implemented to abstract reality. A set of diagrams provides different perspective on the model. First of all, the business domain of moving object analysis is described in a CIM. The goal of the analysis phase is to understand the problem and to develop a first analysis model, independent of implementation and technology concerns. A Use Case driven approach is used to translate functional requirements into a system design. In the design phase, the model is refined by applying concepts from spatio-temporal modelling: Moving Objects Databases and the MADS approach describe data types, operations and predicates for time-varying phenomena in general and moving objects in particular. Finally, the developed geoprocessing functionality is adapted to the OWS framework to be offered as an OpenGIS Web Processing Service. The WPS interface provides the three mandatory operations, described in the WPS Implementation Specification. A GML Application Schema is developed to describe the moving objects data structure in a process execution request. The whole development process is planned and controlled by respective project management methods.

1.3.3 Tools

- Citavi: bibliography
- FreeMind, EverNote: brainstorming
- OpenOffice Writer: text processing
- Microsoft Visio: graphical illustrations
- Enterprise Architect: software engineering by means of UML modelling (CASE tool)
- XML Spy: XML encoding
- Eclipse: prototype implementation (Java IDE)
- Microsoft Project: project management
- Garmin MapSource: test data generation

1.3.4 Test Area / Test Data Sets

As test area, the Lake Constance region in the south of Germany was chosen.

The following test data sets, encoded as GML documents, are required:

- a track, describing a moving object as GML Application Schema, which has to be developed in this work
- the boundary of the Lake Constance as GML polygon geometry

For test data generation, a track was manually created using a GPS planning tool. Such a track, consisting of a list of track points, can be stored in the GPX format. The single track point coordinates, supplemented by a time stamp, are finally carried over in the GML document.

The same test data generation approach, as described for the track, was utilized to roughly vectorize the shape of the Lake Constance and to get the respective boundary coordinates.

1.4 Expected Results

In short, the result of this thesis shall be an OGC compliant interface description for a Web Processing Service which facilitates geospatial processes for moving objects analysis along with a prototypical implementation of a sample process. The method of resolution shall answer the following questions:

Is it possible to apply methods from project management and software engineering to elaborate a geospatial processing problem?

Is it possible to take over concepts from conceptual spatio-temporal modelling and Moving Objects Databases in the software design?

Is it possible to define processes for moving objects analysis based on the OGC WPS Implementation Specification?

Is it possible to use the GML Dynamic Feature Core Schema to model a moving object Application Schema?

Is it possible to utilize the deegree SW framework to implement a prototypical WPS to proof the defined concept?

1.5 Topics Not Explored

There are some topics, touching web-enabled geoprocessing in the broader sense, but going beyond the scope of this work.

Data acquisition is a crucial prerequisite in order to collect the data processed into valuable information afterwards. The Sensor Web, a sensor network consisting of spatially distributed sensor platforms, is not further considered in this thesis.

A “Moving Objects Algebra”, offering a complete set of spatio-temporal

types and operations for querying and manipulating time-varying data, is not part of this work. Instead, the focus is on defining a moving point type, disregarding types with a spatial extent called moving regions. Actual processing algorithms behind such operations are not picked out as a central theme.

The second major perspective on moving objects besides the spatio-temporal data perspective, the location management perspective, dealing with frequently changing location information, real-time position updates and data processing, is not explored in the following chapters.

Usage of Web Service Orchestration to enhance WPS in order to define complex processing workflows or the WSO Framework are not considered in this work. This could be interesting for service chaining, e.g. WFS integration for data acquisition.

Regarding the prototype, a sophisticated client application with a map front end embedded in a Graphical User Interface, offering interactive request definition and map based result visualisation, is not implemented.

Performance evaluation and improvement for geoprocessing services is a crucial fact for effective and efficient Web Service infrastructures but beyond the scope of this work. To ensure a usable and scalable service infrastructure, developers have to tackle standard Internet technology intrinsic bottlenecks and restrictions caused by HTTP and Web Services.

1.6 Target Audience

This paper primarily addresses to researchers, analysts and developers focusing on geoprocessing, moving objects analysis or OpenGIS Web Services.

1.6.1 Professional Language

This Master Thesis contains concepts from both, the mainstream Information Technology and the Geographic Information Science.

A professional perspective on spatio-temporal concepts in general and moving objects in particular, a brief overview about the OGC's mission and OpenGIS standards and a short introduction of object oriented SW engineering principles is given in the Theoretical Approach.

The Project Description includes a “running example”, introducing the computational perspective as brief and abstract as possible. Realization details concerning the UML model or Web Processing Service implementation are put into the Annexes.

1.6.2 Familiarity with Subject

This Thesis should be accessible to anyone with a working knowledge of Internet technology and GIS.

A deeper knowledge on spatio-temporal concepts and moving objects analysis is helpful but not required. Further prerequisites are the familiarity

with geodata and geoservices interoperability and the standardization of geo web services as defined by the Open Geospatial Consortium.

A background on web application development using the deegree framework is useful but not needed.

Brief introductions to the major concepts and methods are provided in chapter 3.

1.7 Thesis Structure

The main part of this Thesis starts with an Introduction, summarizing the author's intention for this work. The actual doing is described in the chapters 3 and and 4 to 6, respectively. The former is describing the theoretical approach, the latter the actual project. A summary, discussion and outlook conclude the paper in chapter 7.

In chapter 3.1.2, Figures from (Güting & Schneider, 2005) are used to illustrate moving objects concepts. Graphics are used to depict the deegree WPS configuration and the applied methods. Throughout chapter 4, UML diagrams are included to visualize the model from different perspectives. Screenshots are applied to illustrate screen contents.

2 Literature Survey

There are two subject areas which are relevant for this Thesis: the first one deals with concepts for spatio-temporal data in general and modelling, design and querying of moving objects in particular. The second one addresses the development of geoprocessing services providing GIS operations as web services, embedded in the OGC service framework.

The challenge of exploring the various arising geographical databases opens up the field of geographic knowledge discovery which put forth techniques for spatial data mining to generate new knowledge (Miler, Han, &Miller, 2001).

In (Roddick & Lees, 2001), a contribution in (Miler, Han, &Miller, 2001), five main types of rules to inspect spatio-temporal datasets for trends and coincidental behaviour are identified. An approach for modelling spatio-temporal data, the dimensioning-up of the spatial dimension to include time, disregards mismatches in the spatial and temporal dimensions concerning properties like linearity, directionality, granularity and scale. Because these properties deal only with limited aspects of time, process aspects of time are more important in data mining. Process time is, according to the author, essentially spatial in character and therefore, in contrast to concepts considering time as uni-directional and linear, able to capture temporal change. The process of interest in the context of this thesis is the movement of objects:

(Güting & Schneider, 2005) present concepts for spatio-temporal databases focusing on the representation of moving entities in databases and asking queries about such movements. To support moving objects in databases, standard database technology needs to be extended: the data model has to provide the respective data structures and SQL-like query language extensions for analysing them. The authors' present an abstract model for querying histories of movement or evolutions of spatial objects over time. They derive an implementable, discrete model and describe the implementation for database systems. Additionally to the history perspective on moving objects, this book looks also at continuously information which is related to the current position and near future movements.

(Parent et al., 2006) emphasize the crucial need of conceptual design methods for developing spatio-temporal databases and applications, which is currently not very well included in geographical information systems and moving objects databases. The authors' approach for spatio-temporal data modelling and manipulation, known as MADS, is introduced beside alternative proposals. MADS provides constructs for modelling data structures, spatial features, temporal features and multi representation features.

In (Kiehle et al., 2007) and (Kiehle et al., 2006a), the additional benefit of

geoprocessing services in comparison to currently available distributed spatial web services is emphasized: "Today SDIs main focus lies on the provision of geospatial data in the form of distributed spatial web services, the retrieval through catalogues, and visualization in the form of Web Map Services (WMS)."(Kiehle et al., 2007, p. 819) Taking SDIs one step further means "[...] providing a method to process geodata in an Open Geospatial Consortium (OGC) compliant way into information." (Kiehle et al., 2007, p. 819) The authors' introduce the OGC WPS draft specification, which is meanwhile an official OGC standard, and present two case studies utilizing WPS. Additionally the authors' examine the application of service chaining to orchestrate single services into complex processing units for WPS in particular and OWS in general.

The paper from (Heier & Kiehle, 2006) details an applied case study involving property information provided by a spatial buffer web service which is compliant to the recently defined Web Processing Service Specification defined by OGC and implemented based on the Open Source Spatial Data Infrastructure framework deegree.

To generate information from the huge amount of distributed available geodata, the various sources of distributed geodata and non-spatial data have to be integrated first. In the past, monolithic GIS were used and the process of information generation was done locally on these large-scale systems. To overcome the aforementioned shortfall, the concept of SDIs is employed based on geo web services according to standards published by the OGC. Applications incorporating the systems business logic can be built on top of web services. Users in turn access the business logic using clients. From an architectural point of view, a SDI can therefore be considered as a multi-tier system consisting of a data tier, a business logic tier and a presentation tier. The generation of information, also called geoprocessing, is a task of the business logic component that utilize a Web Processing Service (WPS) for this job (Kiehle, 2006).

The design and implementation of a geoprocessing service for spatial aggregation based on the WPS specification are presented in (Stollberg et al., 2007). GML was chosen as format for spatial input and output data. The prototype was realized as an extension of existing Open Source Software: for the WPS implementation, the Java based 52° North WPS framework was used.

In (Kiehle & Heier, 2005), the need for a standardized web-based geodata processing service (WPS) against the background of valuable service chains in SDIs is postulated. The generation of added value from geodata through data improvement over the internet within the scope of a SDI is not possible without high-order processing operations. In this paper, experiences made during an Interoperability Experiment (IE) for the definition of a WPS were presented.

The technical implementation of a geoprocessing component is described in (Kiehle & Heier, 2004). The business logic tier consists of a webserver and a Java servlet container, the geoprocessing service itself is realized in

Java. In contrast to the business logic component, described in (Kiehle, 2006), no external WPS is called for information generation. The geoprocessing service is implemented as an extension of the deegree WFS and uses the Java Topology Suite (JTS) for geodata processing. The approach of calling an external WPS in contrast provides several advantages:

- a service "is easily transferable to any computing platform, thus it is independent of the platform and implementation language."(Kiehle, 2006, p. 1750)
- the "output of service metadata as standardized XML [...] makes the service usable in highly complex service chains, in order to generate information out of data." (Kiehle, 2006, p. 1750)

Therefore, this approach of data processing without using a WPS is not further considered in this Thesis. The same applies to the Web Spatial Analysis Service (WSAS), described in (Straub et al., 2004). Meanwhile, an OGC standardized web service for geoprocessing, namely the WPS, is available which supersedes the WSAS. A service for web based intersection of distributed geodata can be realized by defining respective processes for a WPS.

Improvement of effectiveness and efficiency of web services for geoprocessing by considering performance issues as described in (Kiehle et al., 2006b) are not considered in detail in this Thesis. This is a possible topic for continuation of this work.

3 Hypothesis

3.1 Theoretical Approach

3.1.1 Spatio-Temporal Concepts

Spatio-temporal concepts have to deal with space and time. Approaches accommodating well known spatial concepts to include time fail, "[...] differences between the spatial and temporal dimensions cannot be overstated, even when examining apparently static phenomena"(Roddick & Lees, 2001, p. 4): time is considered unidirectional and linear, space bi-directional and non-linear. When both continuous phenomena are encoded with discrete numbers, different granularity is selected, so "[...] simple dimensioning-up strategies work poorly"(Roddick & Lees, 2001, p. 4).

A current approach for spatio-temporal data mining and knowledge discovery is the application of spatio-temporal rules on respective data. (Roddick & Lees, 2001) identified five main rule types:

- Spatio- Temporal Associations: The occurrence of a characteristic X is accompanied by the occurrence of a characteristic Y
- Spatio-Temporal Generalisation: Using concept hierarchies to aggregate data. Spatial-data-dominant generalisation (first ascending spatial hierarchies and then generalising attributes data by region) and non spatial-data-dominant generalisation (first ascending the a-spatial attribute hierarchies) can be distinguished.
- Spatio-Temporal Clustering: Characteristic features of objects in a spatio-temporal region or the spatio-temporal characteristics of a set of objects are sought
- Evolution Rules: Describes the manner in which spatial entities change over time, requires predicates[1] because of the big number of possible rules.

A list of possible spatio-temporal predicates, also presented in (Roddick & Lees, 2001):

- follows: One cluster of objects traces the same (or similar) spatial route as another cluster at a later time.
- coincides: One cluster of objects traces the same (or similar) spatial path whenever a second cluster undergoes specified activity.
- parallels: One cluster of objects traces the same (or a similar) spatial pattern but offset in space.

1 predicates are operators returning a boolean value, serving as filter conditions

- mutates: One cluster of objects transforms itself into a second cluster.

3.1.2 Moving Objects Databases

"Moving objects are basically geometries changing over time [...]"(Güting & Schneider, 2005, p. 1). Moving points are entities, for which "only the time-dependent position in space is relevant, not the extent"(Güting & Schneider, 2005, p. xvii). Regarding a moving point, its position is varying over time. On the other hand, moving entities with an extent can be characterised as moving regions, able to change its position as well as its extent and shape over time. Regardless of the geometry, spatio-temporal databases have to deal with entities changing continuously.

Time can be represented either in a discrete or in a continuous model. The latter is " [...] more appropriate for dealing with moving objects"(Güting & Schneider, 2005, p. 11). The definition of time data types depends on the used model. In a continuous model, they are defined as follows:

- *instant: a* point on the time line
- *period:* an anchored (absolute) interval on the time line
- *periods* or temporal element: a set of disjoint anchored intervals on the time line
- *interval:* a directed, unanchored (relative) duration of time – a time interval of known length with unspecified start and end instants

Concerning the semantics of the time domain, a temporal database is dealing with the valid time as well as with the transaction time. The valid time is the real world time when an event occurs while the transaction time refers to the recording time in the database.

There are two major perspectives on moving objects in databases:

- spatio-temporal data perspective: looking at histories of movements, asking queries for the past, analysing the continuous change of spatial data over time
- location management perspective: maintaining continuously information about the current position and near future positions, that is dealing with frequently changing location information; ask queries about the current and near future positions

Querying histories of movement or evolutions of spatial objects over time requires a spatio-temporal model providing respective data types and operations as well as spatio-temporal predicates.

A natural approach to spatio-temporal modelling is the enhancement of temporal databases by spatial data types. This approach works for some categories of spatio-temporal data but not for moving objects. Therefore, the alternative approach of extending the strategy used in spatial databases to offer abstract data types with suitable operations is more applicable for describing continuous movement. In the following, such an ab-

stract model, as described in (Güting & Schneider, 2005), is outlined.

Spatio-temporal data types, embedded in a 3D space (2D + time), e.g. are:

- *moving point* (*mpoint*)
- *moving region* (*mregion*).

All types, whose values are functions from time (regarding the valid time dimension), are temporal types and called “moving”.

Suitable operations e.g. are*:*

- *intersection*: returns the part of a moving point whenever it lies inside a moving region -> *mpoint*
- *distance*: returns the distance between two moving points which is a real valued function of time -> *mreal*
- *speed:* returns the speed of a moving point -> *mreal*
- *trajectory*: projects a moving point into the plane (range) -> *line*
- *deftime*: returns the set of time intervals when a moving point is defined -> *periods*
- *length*: returns the length of a line value -> *real*
- *min*: yields the minimal value assumed over time by a moving real-> *real.*

Additionally to the above listed main types, related data types are needed, including

- base types: *int*, *bool, real, string*
- spatial types: *point, points, line*, *region*
- time types: *instant*, periods = set of time intervals = range[1](instant)*.*

For all base and spatial types, there are corresponding time-dependent (temporal or “moving”) types[2].

As already mentioned, spatio-temporal data “live” in a 3D space which is the cross-product of the spatial and the time domain. Hence, for moving spatial types there are respective projections to domain (periods) and range (2D plane).

1 type constructor, applicable to base and time types, yields to a set of respecting types
2 obtained through application of type constructor “moving”

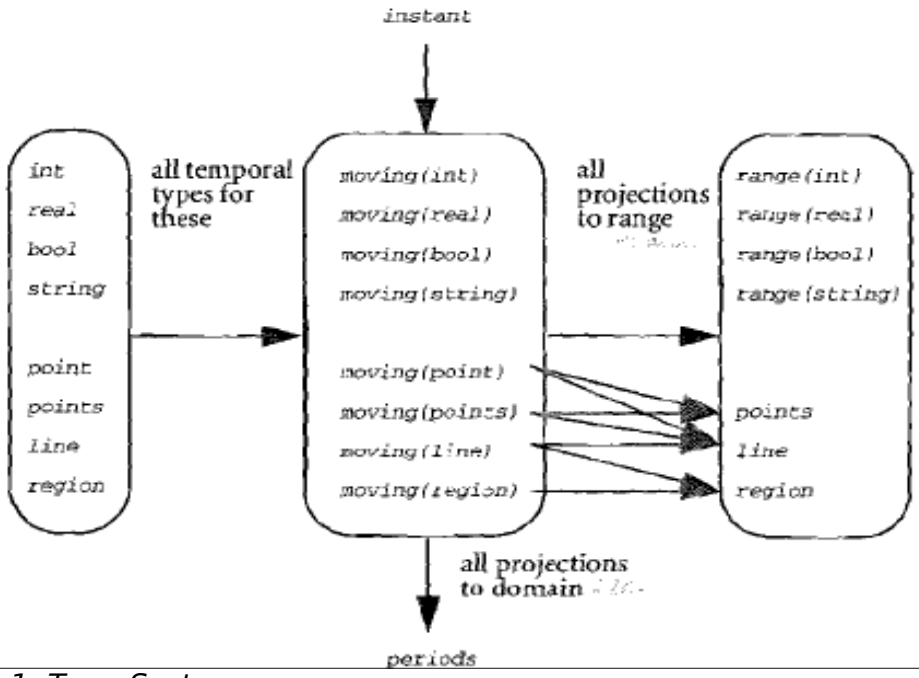


Figure 3.1: Type System

To develop a complete, consistent and non-proliferating set of spatio-temporal operations, the author's classify operations into

- operations on temporal types
- operations on non temporal types
- operations on sets of objects.

Additionally, types are classified into point types, representing single values, and point set types, representing sets of values. This approach yields more generic operations, in some cases specific considerations are however more appropriate (indicated through an alias in square brackets).

First, operations on non temporal types are designed and then extended to the temporal variants which is called temporal“lifting”.

Operations on non temporal types are grouped into six classes:

- predicates
- set operations
- aggregation
- numeric
- distance and direction
- base type specific.

Regarding predicates, relationships exists between two points, two point sets and between a point and a point set. Predicates are based on set theory, order relationships and topology.

	Sets	Order (1D Spaces)	Topology
point versus point	$u = v,\ u \neq v$	$u < v,\ u \leq v,\ u \geq v,\ u > v$	
point set versus point set	$U = V,\ U \neq V$ $U \cap V \neq \emptyset$ (**intersects**) $U \subseteq V$ (**inside**)	$\forall u \in U, \forall v \in V : u \leq v$ (**before**)	$\partial U \cap \partial V \neq \emptyset$ (**touches**) $\partial U \cap V^\circ \neq \emptyset$ (**attached**) $U^\circ \cap V^\circ \neq \emptyset$ (**overlaps**)
point versus point set	$u \in V$ (**inside**)	$\forall u \in U : u \leq v$ (**before**) $\forall v \in V : u \leq v$	$u \in \partial V$ (**on_border**) $u \in V^\circ$ (**in_interior**)

Figure 3.2: Predicates

Signatures of set operations are available for the relationships point versus point, point versus point set, point set versus point set in 1D and 2D spaces and finally for specialized operations.

	Operation	Signature		
1.	**intersection, minus**	$\pi \times \pi$	$\rightarrow \pi$	
2.	**intersection**	$\pi \otimes \sigma$	$\rightarrow \pi$	
	minus	$\pi \times \sigma$	$\rightarrow \pi$	
		$\sigma \times \pi$	$\rightarrow \sigma$	
	union	$\pi \otimes \sigma$	$\rightarrow \sigma$	
3.	**intersection, minus, union**	$\sigma \times \sigma$	$\rightarrow \sigma$	[1D]
4.	**intersection**	$\sigma_1 \times \sigma_2$	$\rightarrow min(\sigma_1, \sigma_2)$	[2D]
	minus	$\sigma_1 \times \sigma_2$	$\rightarrow \sigma_1$	[2D]
	union	$\sigma \times \sigma$	$\rightarrow \sigma$	[2D]
5.	**crossings**	*line* × *line*	→ *points*	
	touch_points	*region* ⊗ *line*	→ *points*	
		region × *region*	→ *points*	
	common_border	*region* ⊗ *line*	→ *line*	
		region × *region*	→ *line*	

Figure 3.3: Set Operations

Aggregation operations are used to reduce point sets to points.

Operation	Signature		Semantics
min, max	$\sigma \rightarrow \pi$	[1D]	$min(\rho(U)), max(\rho(U))$
avg	$\sigma \rightarrow \pi$	[1Dnum]	
avg[center]	$\sigma \rightarrow \pi$	[2D]	
single	$\sigma \rightarrow \pi$		if $\exists\, u$: $U = \{u\}$ then u else $\perp$
min[start], max[end]	*periods* → *instant*		

Figure 3.4: Aggregate Operations

Numeric operations compute numeric properties from point sets.

Operation	Signature		
no_components	σ	→ *int*	
size	σ	→ *real*	[cont]
perimeter	*region*	→ *real*	
size[**duration**]	*periods*	→ *real*	
size[**length**]	*line*	→ *real*	
size[**area**]	*region*	→ *real*	

Figure 3.5: Numeric Operations

For all continuous types, distance measurement operations exists. The angle of the line from a starting to an end point is measured by the direction operation.

Operation	Signature	
distance	π × π	→ *real* [cont]
	π ⊗ σ	→ *real* [cont]
	σ × σ	→ *real* [cont]
direction	*point* × *point*	→ *real*

Figure 3.6: Distance And Direction Operation

Operations on temporal types are grouped into five classes:

- projections to domain and range
- interaction with domain and range
- rate of change
- lifting
- when

For projections of temporal values, operations are offered to return the corresponding parts of the projections: periods for temporal functions, spatial point set types for 2D data types.

Operation	Signature		
deftime	*moving*(α)	→ *periods*	
rangevalues	*moving*(α)	→ *range*(α)	[1D]
locations	*moving*(*point*)	→ *points*	
	moving(*points*)	→ *points*	
trajectory	*moving*(*point*)	→ *line*	
	moving(*points*)	→ *line*	
routes	*moving*(*line*)	→ *line*	
traversed	*moving*(*line*)	→ *region*	
	moving(*region*)	→ *region*	
inst	*intime*(α)	→ *instant*	
val	*intime*(α)	→ α	

Figure 3.7: Projection Operations

A relationship between moving types and values, in either their domain or range, can be retrieved through respective interaction operations.

Operation	Signature		
atinstant	*moving*(α) × *instant*	→ *intime*(α)	
atperiods	*moving*(α) × *periods*	→ *moving*(α)	
initial, final	*moving*(α)	→ *intime*(α)	
present	*moving*(α) × *instant*	→ *bool*	
	moving(α) × *periods*	→ *bool*	
at	*moving*(α) × β	→ *moving*(α)	[1D]
	moving(α) × β	→ *moving*(*min*(α, β))	[2D]
atmin, atmax	*moving*(α)	→ *moving*(α)	[1D]
passes	*moving*(α) × β	→ *bool*	

Figure 3.8: Interaction With Values In Domain And Range

The rate of change of time-dependent values leads to four moving point specific derivative operations.

Operation	Signature		Semantics
derivative	*mreal*	→ *mreal*	μ' where $\mu'(t) = \lim_{\delta \to 0}(\mu(t+\delta) - \mu(t))/\delta$
speed	*mpoint*	→ *mreal*	μ' where $\mu'(t) = \lim_{\delta \to 0} f_{\mathbf{distance}}(\mu(t+\delta), \mu(t))/\delta$
mdirection	*mpoint*	→ *mreal*	μ' where $\mu'(t) = \lim_{\delta \to 0} f_{\mathbf{direction}}(\mu(t+\delta), \mu(t))/\delta$
turn	*mpoint*	→ *mreal*	μ' where $\mu'(t) = \lim_{\delta \to 0}(f_{\mathbf{mdirection}}(\mu(t+\delta)) - f_{\mathbf{mdirection}}(\mu(t)))/\delta$
velocity	*mpoint*	→ *mpoint*	μ' where $\mu'(t) = \lim_{\delta \to 0}(\mu(t+\delta) - \mu(t))/\delta$

Figure 3.9: Derivative Operations

In lifting operations, any argument of a operation on non temporal types

can be replaced by the respective temporal type and the result is also a corresponding temporal type.

The when operation allows to restrict a temporal value (or a moving objects) to the times when an arbitrary condition is fulfilled. Hence, the signature of the when operation shows two arguments, a temporal value and a predicate that is an operator on a non temporal type.

Operations on sets of objects are needed to manipulate sets of objects instead of atomic data types. The author's define for this purpose an operation, called decompose which makes the components of point set types accessible.

In the next step, discrete models can be derived form the abstract model building the basis for respective implementations. In a discrete model, counterparts of abstract types are designed. Discrete temporal types are high-level definitions of data structures, still programming language independent but based on the standard data types as known from programming languages.

A more discrete look on predicates for moving spatial objects, restricted to spatio-temporal predicates between two moving points, two moving regions and a moving point and a moving region, reveals the importance of topological predicates. Topological relationships describe the "[...] relative positions of spatial objects to each other without considering metric objects" (Güting & Schneider, 2005, p. 151). These topological relationships alter when temporal changes of spatial objects occur, that is if one or both objects move or reshape, topological transitions happen. The subject of spatio-temporal predicates therefore is to examine topological predicates over time.

Applying the concept of temporal lifting as well to topological predicates, a moving Boolean will be the result, which yields to undefined, whenever a point or region is undefined. In other words: only on the intersection of the domains of two spatio-temporal objects, the result is always true or false. This violates the essence of predicates, stating that a predicate is a function to Boolean which is either true or false. Therefore, a basic spatio-temporal predicate can be considered as a temporally lifted spatial predicate yielding a temporal/ moving Boolean, which is aggregated by determining whether that temporal Boolean was sometimes or always true. In contrast to a lifted predicate, a spatio-temporal predicate cannot be undefined, undefined values will be aggregated into Booleans.

Temporal aggregation is realized by defining two aggregate operators or quantifier:

- existential quantification: strict quantifier which aggregates undefined to false while ranging over the whole time interval
- universal quantification: aggregates only over a restricted time interval, more or less strict depending on the range of quantification

For universal temporal aggregation of Boolean values, there are several

definitions about the time interval over which the universal quantification can range, yielding finally to four possible quantifiers.

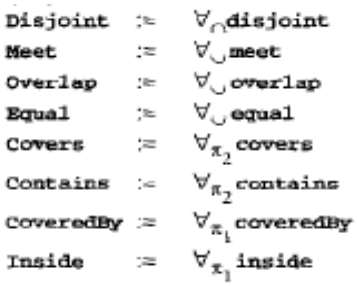


Figure 3.10: Basic Spatio-Temporal Predicates

After the basic spatio-temporal predicates are defined, developments, representing the change of spatial situations over time, have to be specified. Observations, made at this, reveal, that first, predicate constrictions have to be defined, and second, predicates need to be distinguished between instant predicates and period predicates. A concise syntax for developments denotes temporal compositions of spatial and spatio-temporal predicates by a sequence of predicates linked together by an infix operator. Therefore, following three operations were defined, with spatial predicate (p) and spatio-temporal predicates (P,Q):

- p *then* P
- P *until* p
- P *until* p *then* Q

Beside temporal composition, other logical connectives that can be defined to combine predicates, are described in an algebra with spatio-temporal predicates as objects and combinators as operations:

- temporal alternative
- predicate negation
- object reflection
- predicate reflection
- derived combinators

For all cases, the point/point, point/region and region/region case, a large number of distinct temporal evolutions of topological relationships can be obtained from a development graph. For each alternative, a own spatio-temporal predicate could be defined. To assemble this new complex or compound predicates, the eight basic spatio-temporal predicates are used.

3.1.3 Spatio-Temporal Conceptual Data Modelling

A conceptual model is free from implementation-based limitations, "[...] it

enables a direct mapping between the perceived real world and its representation with the concepts of the model" (Parent et al., 2006, p. 19). The MADS approach addresses conceptual data modelling and manipulation, for space and time dimension, including tools for this generic approach. Regarding space modelling, concepts for describing the discrete and continuous view are important. Concerning time modelling, from an application point of view, the valid time is the interesting one. Based on the underlying principle of orthogonality, each modelling dimension can be described separately.

Commonly, "[...] spatio-temporal is used in a loose sense, to refer to anything that deals with either space, or time, or both" (Parent et al., 2006, p. 60). Objects that are subjects to change over time in the sense of getting different values throughout their evolution, are called time-varying phenomena. MADS defines spatio-temporal an "[...] object type that either has both a spatial and a temporal extent, separately, or has a time-varying spatial extent" (Parent et al., 2006, p. 67). If the life cycle of an object, the history of his evolution, is of interest for an application, has to be decided at the level of each single object type. "A data model allowing to keep the history of spatial features supports spatio-temporal phenomena such as moving points and moving or deforming lines and surfaces." (Parent et al., 2006, p. 60)

MADS supports basic spatial and temporal data types and types for homogeneous or heterogeneous collections, organized into a generalization hierarchy. Both, hierarchies as well as spatio-temporal data types for moving objects, come along with associated operations and predicates. The definition of operators is based on the following objectives:

- Minimal set of operators: set of basic operators, can be combined to more complex operations
- Orthogonality: same operator applies for all dimensions, space and time and varying types
- Similarity: operators are similar in all dimensions
- Commonality: all operators share the same need to denote what is being manipulated

MADS extends the concept of time-varying or moving data types and operations, introduced in (Güting & Schneider, 2005), to the space dimension by supporting additional space-varying data types and associated operations.

3.1.4 OpenGIS Standards and Specifications

As declared on the OGC website[1], the Open Geospatial Consortium is a non-profit, international, voluntary consensus organization that is leading the development of standards for geospatial and location based services. To enable GIS interoperability, the technical documents of the OGC, avail-

1 http://www.opengeospatial.org

able to everyone at no cost, detail interfaces of software components. Therefore, independently developed components that implement the OpenGIS specifications plug and play immediately.

The conceptual foundation for the Implementation Specifications, which describe interfaces at the implementation level of detail, is provided by the Abstract Specifications. The OpenGIS Reference Model (ORM) provides an architecture framework for the ongoing work of the OGC.

In the ORM, the OGC vision is described as follows: the "OGC vision is a world in which everyone benefits from geographic information and services made available across any network, application, or platform." (OGC Reference Model OGC 03-040, p. VII) The derived mission of the OGC therefore is to "deliver spatial interface and encoding specifications that are openly and publicly available for global use" (OGC Reference Model OGC 03-040, p. VII). Accordingly, the purpose and scope of the ORM, which updates and replaces parts of the 1998 OpenGIS Guide, is to describe the OGC requirements baseline for geospatial interoperability and to provide a common architecture for the “Geo-Web”. For this purpose, ORM applies the Reference Model for Open Distributed Processing (RM-ODP), an international standard for architecting open, distributed processing systems. To address different aspects of a system, five viewpoints are defined to describe different perspectives:

- Enterprise Viewpoint: focuses on purpose, scope and policies for a system
- Information Viewpoint: focuses on semantics of information and information processing
- Computational Viewpoint: focuses on component and interface details without regard to distribution
- Engineering Viewpoint: focuses on the mechanisms and functions required to support distributed interaction between objects in a system
- Technology Viewpoint: focuses on the choice of technology

In the information viewpoint, the feature concept is introduced, which is a central term in the “OpenGIS world”: "A feature is an abstraction of a real world phenomenon. A geographic feature is a feature associated with a location relative to the Earth." (OGC Reference Model OGC 03-040, p. 7) Because geographic information is subjectively perceived and its content is depending on the needs of particular applications, the OpenGIS framework provides conceptual schemas to define abstract feature types and the possibility for domain experts to develop their own Application Schemas[1], fulfilling the special needs of this particular information community[2].

1 "An application schema provides the formal description of the data structure and content required by one or more information communities. An application schema is a set of conceptual schema for data required by one or more applications." OGC Reference Model OGC 03-040

2 An information community "[...] is a collection of people [...] who, at least part of the time, share a common digital geographic information language and common spatial

Metadata, data about data, are crucial thus enabling data discovery, retrieval and reuse. Three different kinds of metadata can be distinguished:

- dataset metadata
- service metadata, e.g. the capabilities document as the result of invoking the GetCapabilities operation, which is common to all OWS, as described later on in the section about the OGC Web Service Common Specification
- registry information model

The computational viewpoint provides key definitions for the service framework:

- a service is a "collection of operations, accessible through an interface, that allows a user to evoke a behavior of value to the user [ISO – 19119]" (OGC Reference Model OGC 03-040, p. 2)
- an interface is a "named set of operations that characterize the behavior of an entity" (OGC Reference Model OGC 03-040, p. 2)
- an operation is a "specification of an interaction that can be requested from an object to effect behavior [ISO 19119]" (OGC Reference Model OGC 03-040, p. 2)

The technology viewpoint is concerned with the infrastructure of a Distributed Computing Platform (DCP). The OGC has defined a number of encodings, based upon XML, to transfer data packages as messages between application clients and services, and between services. The language for describing and encoding geospatial information is GML. The interfaces of the OGC defined Web Services (OWS) have explicit bindings forHTTP. For invoking operations of a service, there are the two HTTP methods, GET and POST. Thus, the online resource for each operation, supported by a service instance, is a HTTP URL. Only the parameters, comprising the service request itself, are mandated by the OGC Web Service Common Specification.

The OGC Web Service Common Specification specifies many of the aspects that are common to all OWS interface Implementation Specifications like Web Map Service (WMS) or Web Feature Service (WFS) and especially Web Processing Service (WPS). These common aspects include operation request[1] and response[2] contents, their parameters[3] and encodings.

The mandatory GetCapabilities operation, provided by each OWS, allows any client[4] to retrieve metadata about the capabilities provided by any

---

feature definitions. This implies a common worldview as well as common abstractions, feature representations, and metadata." OGC Reference Model OGC 03-040

1 "invocation of an operation by a client" Open GIS Specification 06-121r3

2 "result of an operation, returned from a server to a client" Open GIS Specification 06-121r3

3 "variable whose name and value are included in an operation request or response" Open GIS Specification 06-121r3

4 "software component that can invoke an operation from a server" Open GIS Specification 06-121r3

server[1] that implements an OWS interface Implementation Specification. A service metadata document, which makes an OWS server partially self-describing, is returned as response to the requesting client. A GetCapabilities request can be KVP encoded (mandatory) or XML encoded (optional). If an error encounters during servicing this request, an exception report message shall be returned, otherwise a service metadata document, encoded in XML, shall be the normal response.

All other operations, except GetCapabilities, shall include the following request parameters:

- service: service type identifier, e.g. WPS
- request: operation name, e.g. DescribeProcess
- version: specification version for operation, e.g. 1.0.0

The OGC Web Service Common Specification defines two methods of encoding OWS operation requests: one uses XML as encoding language and the other uses keyword-value pairs (KVP) to encode the various parameters. HTTP supports two request methods: GET and POST. When the HTTP GET method is used, the request shall be composed as follows, using the KVP encoding: URL prefix, ending with ?, followed by key-value pairs for (at least) the mandatory parameters, where multiple key-value pairs were separated by an ampersand. An URL, intended for HTTP POST requests, is a complete URL without additional parameters. This is the URL to which clients transmit request parameters, using XML encoding, in the body of the POST message.

A HTTP response, normally a XML document, shall be accompanied by the appropriate MIME type.

Finally, the OGC Web Service Common Specification provides some guidance for editors of OWS Implementation Specifications in the form of best practices.

The WPS Implementation Specification, which specifies the interface to a Web Processing Service, is compliant to the two aforementioned Specifications, the ORM and the OGC Web Service Common Specification.

The standardized WPS interface facilitates the publishing of geospatial processes[2] and the discovery of and binding to those processes by clients. A WPS is able to offer any sort of GIS functionality, from simple calculations to complicated computation models. The WPS therefore follows the strategy of a service-driven generation of information by processing geodata, able to fulfil real geoprocessing tasks (a user-driven generation of information would be the opposite approach, were the user has to locate relevant raw data, assemble them over the internet e.g. using WFS, process the data with local client functionality and then visualize the result).

1 "a particular instance of a service [ISO 19119 edited]" Open GIS Specification 06-121r3
2 "model or calculation that is made available at a service instance" OGC Implementation Specification OGC 05-007r7

The required data, either vector or raster data, can be delivered across a network or available at the server. The WPS interface standardizes the way, how processes and their input[1] and output[2] data are described, how a client can request a process execution and how the output is handled. The actual data processing, executed by the process, is encapsulated by the generic interface that can be used to describe and web-enable any sort of geospatial process.

The latest version of the WPS Implementation Specification is 1.0.0, for prototyping, the predecessor version 0.4.0 (OpenGIS Discussion Paper) was used, because the Open Source framework deegree is implementing this version of the standard.

The WPS interface specifies three mandatory operations:

- GetCapabilities: allows a client to request and receive service metadata (capabilities) documents and additionally provides the names and general descriptions of each of the processes, offered by a WPS instance (HTTP GET method using KVP encoding is mandatory)
- DescribeProcess: allows a client to request and receive detailed information about the processes that can be run on the service instance, including the required inputs, their allowable formats and the outputs that can be produced. A complex data type data structure offers the capability to either encode the payload directly in the execute request or by referencing a remote location on the web (HTTP GET method using KVP encoding is mandatory)
- Execute: allows a client to run a specific process, implemented by the WPS, using provided input parameter values and returning the produced outputs. Optionally, allows the monitoring of the progress of a process execution via status messages. The result of a process execution can be returned directly to the client or, optionally, can be stored at a web-accessible location which is referenced in the execute response document (HTTP POST method using XML encoding is mandatory)

A WPS instance can offer one or more processes which are not coupled to the data they operate on. WPS processes can be chained with other services of the same kind (other WPS) or of different nature (e.g. WCS or WFS) and therefore map complex workflows (also called WSO).

Data input required and data produced by a WPS process can include data exchange standards such as GML.

The Geography Markup Language, also known as ISO 19136, is an XML grammar written in XML Schema for the modelling, transport and storage of geographic information. GML provides objects like features, coordinate reference systems, geometry, topology, time and units of measure, organized in 29 so called GML Core Schemas, building the basis for the definition

1 "data provided to a process" OGC Implementation Specification OGC 05-007r7
2 "result returned by a process" OGC Implementation Specification OGC 05-007r7

of own Application Schemas.

In the Dynamic Feature Schema, a number of types and relationships are defined to represent the time-varying properties of geographic features. A dynamic feature extends a geographic feature about dynamic properties. A TimeSlice is an abstract GML object that encapsulates updates of dynamic properties. The MovingObjectStatus element is one example of how the abstract TimeSlice element may be extended.

The latest version of the GML Implementation Specification is 3.2.1, for developing the GML Data Model (see chapter 4.5.2), the predecessor version 3.1.1 was used in order to reuse available types for moving objects modelling such as a track type. In version 3.2.1, the MovingObjectStatus was deprecated as a normative schema component and used instead just informatively as an example for the use of dynamic features. The description of moving objects in GML as one kind of features with properties that vary over time, differs from the ISO 191411 concept. While the MovingObjectStatus in GML describes the location and other properties of the moving object at certain time stamps, ISO 19141 describes location and direction as a function of time.

3.1.5 Principles Of Object Oriented Software Engineering

The main goals of the MDA[1] approach are

- portability
- interoperability
- re-usability.

These goals can be accomplished through architectural separation of concerns - building models to abstract the complexity of SW systems with different views on the system:

- Computation Independent Model (CIM): contains no details about the structure of the system, also called Domain Model
- Platform Independent Model (PIM): describes the general structure and flow of the system without details about concrete platforms
- Platform Specific Model (PSM): extends the platform independent part about details for a concrete platform

Using model transformation, one model can be converted into another model of the system. Similar to the CASE approach, in the final step, source code is generated from the PSM.

The object technology has the following strengths:

- provides a single paradigm, a language used by users, analysts, designers and implementers
- facilitates architectural and code re-use

1 MDA concepts as described in Lohmar, 2006

- models more closely reflect the real world
- provides stability
- is adaptive to change.

The four basic principles of object orientation[1] (OO) are:

- Abstraction: concentrating on the essential characteristics, ignoring less important
- Encapsulation: hide implementation from clients
- Modularity: breaking up something complex into manageable pieces
- Hierarchy: ranking or ordering of abstractions into a tree-like structure

An object represents a real-world entity with attributes, representing its state, and operations, representing its behaviour. Each object has a unique identity and a well defined boundary.

A class is a description of a set of objects that share the same attributes, operations, relationships and semantics. An object is an instance of a class. An attribute is a named property of a class that describes a range of values that instances of the property may hold. An operation is the implementation of a service that can be requested from any object of the class to affect behaviour.

Polymorphism is the ability to hide many different implementations behind a single interface. Interfaces formalize polymorphism and support “plug-and-play” architectures. An interface is a collection of operations that are used to specify a service of a class or a component.

An association shows relationships between classes. Multiplicity is the number of instances one class relates to one instance of another class. An aggregation is a special form of association that models a whole-part relationship between an aggregate (the whole) and its parts. A dependency is a relationship between two model elements where a change in one may cause a change in the other. Generalisation is a relationship among classes where one class shares the structure and / or its behaviour of one or more classes. It defines a hierarchy of abstractions where a subclass inherits from one or more super classes. Realisation is a semantic relationship between two classifiers whereas one classifier serves as the contract that the other classifier agrees to carry out.

The following section gives a short overview about the key concepts of Object Oriented Analysis (OOA) and Object Oriented Design (OOD). The purposes of OOA and OOD are to

- transform the requirements into a system design
- evolve a robust architecture for the system
- adapt the design to match the implementation environment, design-

1 OO, OOA / OOD as described in IBM Rational University, 2003

ing it for performance

Input artefacts are the Use Case model and supplementary specification from the requirements discipline. The result of analysis and design is a design model, an architecture represented by a number of architectural views which are abstractions or simplifications of the entire design.

The goal of analysis is to understand the problem and to begin to develop a visual model independent of implementation and technology concerns. Analysis focuses on translating the functional requirements into software concepts. A goal of design is to refine the model in terms of adapting it to the implementation and the deployment environment and considering performance aspects.

Although Use Cases are not part of traditional object orientation, they are a recommended method for organizing requirements. A Use Case realization describes how a particular Use Case is realized in terms of collaborating objects, ties together the Use Cases from the Use Case model with the classes and relationships of the design model. Within the UML, a Use Case realization can be represented using a set of diagrams that model the context of the collaboration and the interactions of the collaborations.

The UML is a object oriented modelling language used to

- visualise
- specify
- construct
- document

the artefacts of SW systems. The UML is a de-facto standard in OOD, standardized by the OMG. The UML is only a language, process independent, but optimally used in a process that is Use Case driven, architecture centric, iterative and incremental.

The UML provides a graphical notation, concepts and semantics as well as respective rules to build a model, a simplification of reality, the blueprint of a system. Additionally, the UML provides a set of diagrams to visualize the model from different perspectives. UML 2.x includes thirteen diagrams that can be classified into three groups:

- behaviour diagrams: a type of diagram that depicts behavioural features of a system or business process. This includes activity, state machine, and Use Case diagrams as well as the four interaction diagrams.
- interaction diagrams: a subset of behaviour diagrams which emphasize object interactions. This includes communication, interaction overview, sequence, and timing diagrams.
- structure diagrams: a type of diagram that depicts the elements of a specification that are irrespective of time. This includes class, composite structure, component, deployment, object, and package diagrams.

Because it is not possible for one closed language to ever be sufficient to express all possible nuances of all models across all domains, the UML is opened-ended and provides an extensibility mechanism that includes

- stereotypes
- tagged values
- constraints

One remark concerning the OO principles encapsulation and modularity is important at this point: the difference between a package and a subsystem. A package is a general purpose mechanism for organizing elements into groups. A subsystem in contrast is a combination of a package (can contain other model elements) and a class (has behaviour). In UML, a subsystem can be represented using a stereotyped package.

The UML is used as modelling language during OOA / OOD in this work and therefore UML diagrams are used throughout this paper to illustrate model elements and parts of the SW design.

The continuous pursue of the object oriented paradigm results in the usage of Object Oriented Programming (OOP) for software implementation. OOP facilitates flexibility and re-usability of software code. Several programming languages support OOP, Java is the preferred language for prototype realization.

XML is an open standard edited by the W3C for creating custom markup languages. “Its primary purpose is to facilitate the sharing of structured data across different information systems, particularly via the Internet, and it is used both, to encode documents and to serialize data. It is classified as an extensible language because it allows its users to define their own elements”[1]. The XML specification defines a meta language which builds the base to define application specific languages.

GML is an OpenGIS Implementation Specification for a XML language to encode geographic features. GML, also named ISO 19136, is not only a XML dialect, it consists of 29 XML schemata. These so called core schemata of GML, also called grammars, are the base for defining one's one Application Schema, which is the goal of data modelling using GML. This Application Schema contains the object classes of the specific application domain.

A XML language is describe by a XML schema. A XML schema, short XSD, is a XML language itself and describes the language constructs, elements and attributes with their data types, of a concrete language.

3.2 Methods

The software development process is guided by the principles of MDA. According to this model-driven software development approach, a Computation Independent Model (CIM) is developed in a first step. This model will

1 From Wikipedia, the free encyclopedia

be enhanced to a so called Platform Independent Model (PIM), which is still independent from technology constraints, just considering functional aspects. Finally, a Platform Specific Model (PSM) will be generated, taking into account platform and / or programming language specifics.

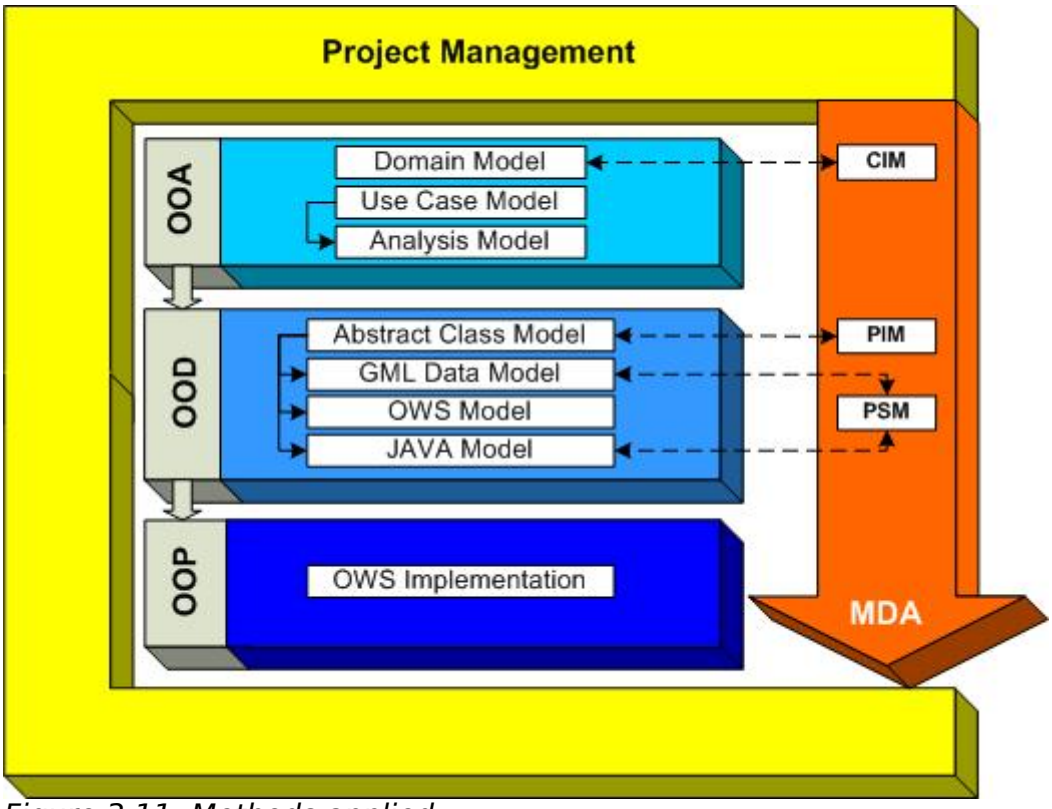


Figure 3.11: Methods applied

Assigned to object oriented SW development, the first action in the analysis phase is the creation of a domain model. This model describes objects of the business domain captured by the software system under development. Functional requirements to the system are defined according to the outside-in-method: Use Cases describe the system environment from the user perspective, the system itself is initially handled as a black box. The domain knowledge to model Use Cases for moving objects analysis are primarily taken from military surveillance projects where similar functional requirements are present. In the next development step, the Use Case realisation, a analysis model is derived from the Use Case model, identifying key abstractions that the system have to handle. Analysis classes represent an early conceptual model of objects from the problem domain. UML sequence diagrams are used to find these objects by illustrating the dynamic flow of each Use Case.

In the design phase, the analysis model is refined leading to an abstract class model, showing complete designed classes and their relationships to each other, pictured in UML class and package diagrams. Concepts from spatio-temporal modelling as defined in the MADS[1] approach as well as the abstract model of MOD[2] ran into the SW architecture, defined and im-

1 Parent et al., 2006
2 Güting & Schneider, 2005

proved iteratively. To decouple the behaviour and signature (operations) from its implementation (methods) and to allow multiple and various implementations of the business logic, advantage of the OO interface paradigm is taken.

Based on the abstract class model, a discrete data model, showing how the persistence layer is implemented in GML, is designed. The abstract interface is adapted to the principles of OpenGIS Web Services (OWS) to be in line with the respective OGC standards: Abstract Specifications in general and the WPS Implementation Specification in particular, which describes an interface according to the OpenGIS Reference Model (ORM). Finally, a prototypical implementation of the service interface using the OO programming language Java is described in the Java model. Once more, UML class and package diagrams, enhanced with appropriate stereotypes, are used to display the respective models.

The engineering approach, as described preceding, as well as all the additional tasks, necessary to accomplish the “Master Thesis project”, are embedded into respective project management methods to plan the activities in an adequate order, define pre- and post-conditions and control the whole progress.

3.3 Tools

- Citavi: bibliography
- FreeMind, EverNote: brainstorming
- OpenOffice Writer: text processing
- Microsoft Visio: graphical illustrations
- Enterprise Architect: software engineering
- XML Spy: XML encoding
- Eclipse: prototype implementation
- Microsoft Project: project management
- Garmin MapSource: test data generation

4 Project Description

In the following subchapters, the software architecture and a prototypical realization are described in more detail.

The architecture model, outlined in chapter 4.1 to 4.5, is introduced by a number of different views. A architectural view is a simplified description or abstraction of a system from a particular perspective, covering particular concerns and omitting entities that are not relevant to this perspective. Views in this sense are “slices” of the whole model. Each of these views, and the UML notation, used to represent them, will be discussed in the subsequent chapters.

In chapter 4.6, an exemplary implementation of a chosen subset of the modelled functionality is presented. It is about a geoprocessing task to check whether a moving object is crossing a particular area of interest.

This process serves as a “running example” throughout the whole Project Description chapter. The whole UML model with all its diagrams and comprehensive documentation is available in Annex A.

4.1 Domain Model (CIM)

A domain model, often part of a business domain model, captures a description of what the software knows about the domain and the objects it contains. It is used to define a common, consistent vocabulary across a project. The domain model is a CIM because it focuses on the environment of the system, the details of the structure and processing of the system are suppressed at this business level.

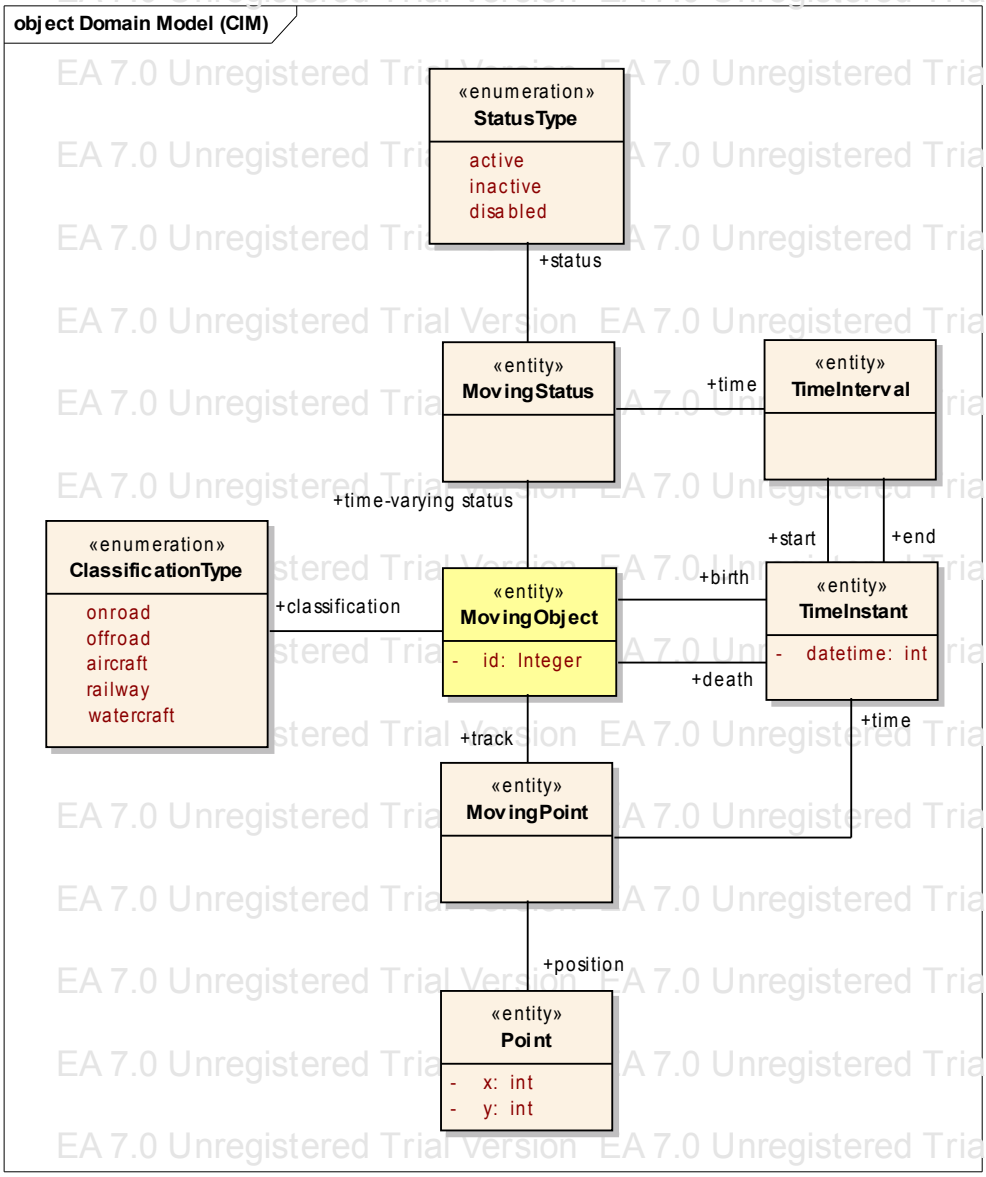


Figure 4.1: Domain Model

Description of the entities, depicted in the UML class diagram:

Entity	Description
ClassificationType	Possible values a classification type can adopt
MovingObject	Object, changing its position over time
MovingPoint	If the moving object has no spatial extension, its track consists of a number of points
MovingStatus	Status of the moving object which is also time-varying
Point	Position of a moving point expressed using coordinates
StatusType	Possible values a status type can adopt
TimeInstant	Point in time for which a moving point is valid respectively birth and death of a moving object indicating its lifetime. Time instants also indicate the start and end of a time interval
TimeInterval	Timespan for which a moving status is valid

Table 1: Domain Model

4.2 Non-Functional Requirements

In contrast to functional requirements, defining what a system shall do, a non-functional requirement defines which attributes a system shall have. Such attributes are normally classified into the following categories:

- reliability
- look and feel
- usability
- performance and efficiency
- sustainability
- portability
- security aspects

The following enumeration lists general, non-functional requirements on web services:

- Services shall be coarse-grained
- Services shall be repeatable
- Services shall be atomar
- Services shall be stateless
- Interfaces shall be independent from implementation technology

The following enumeration lists non-functional requirements on OWS, derived from the ORM:

- OWS shall be agile to be able to adapt to changing business rules and operational requirements
- OWS shall support the easy and seamless introduction of new technologies and the evolution of existing ones
- OWS shall provide for robustness and consistent error handling and recovery to support mission-critical systems development
- OWS shall accommodate authentication, security and privacy features and support asset protection
- OWS shall be platform independent concerning DCP, hardware, OS, programming language and encodings
- OWS shall support implementations of N-tiered, component architectures
- OWS shall support standard interfaces and metadata while accommodating the use of other complementary standards and specifications in environments where OpenGIS specifications are implemented
- OWS shall support interoperability by specifying interface definitions, service descriptions and protocols for software collaboration and negotiation
- OWS shall accommodate independently developed implementations of a service and many independently provided instantiations of different types of services
- OWS shall accommodate a wide range of data policies
- OWS shall be vendor and data neutral
- OWS shall be data content format independent

4.3 Use Case Model

A Use Case model describes functional requirements in terms of Use Cases. Use Cases describe the intended functionality of the system from a users perspective. The system's environment is modelled using actors.

The Use Case model, depicted in Figure 4.2, shows an actor called “Analyst” which is able to perform the Use Case “Detect AOI Crossing Objects”.

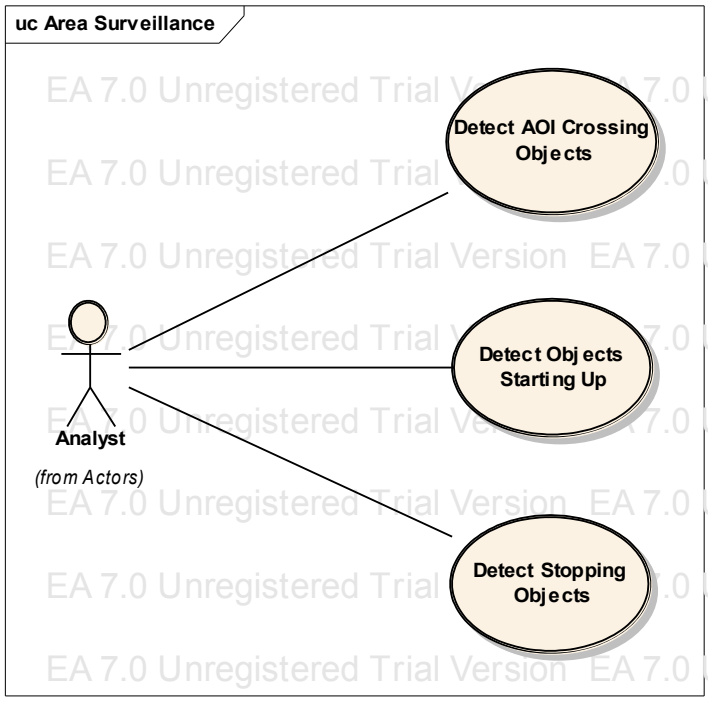


Figure 4.2: Use Case Model

4.4 Analysis Model

During Use Case analysis, initial classes of the system, so called analysis classes, that perform a Use Case's flow of events, were identified for each Use Case.

4.4.1 Use Case Realization

One Use Case realization per Use Case, describing how analysis classes collaborate to perform a Use Case, was developed utilizing an interaction diagram, or more precisely, a sequence diagram.

The technique for finding analysis classes uses three different perspectives of the system to drive identification of candidate classes:

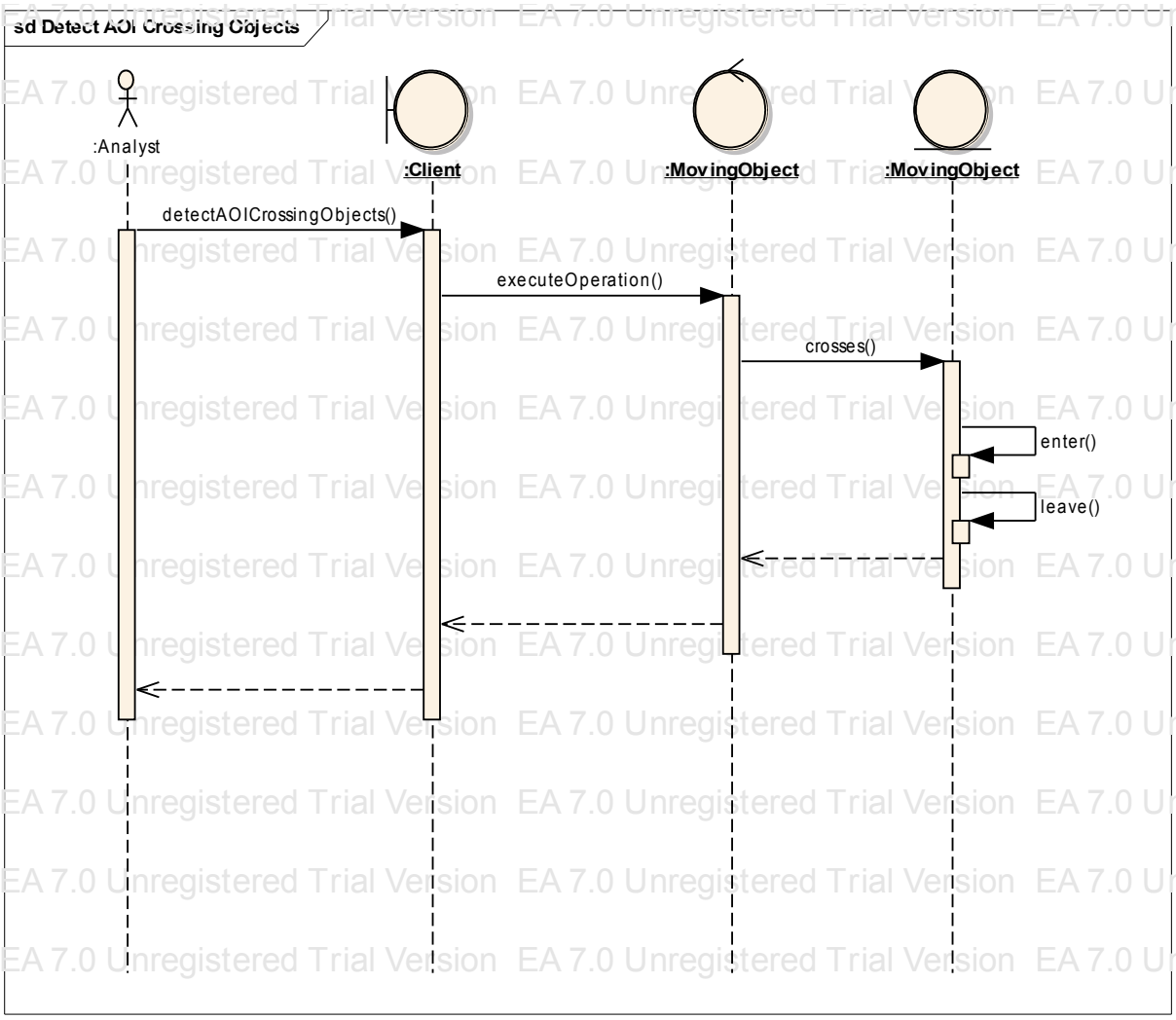


Figure 4.3: Use Case Realization

- the boundary between the system and its actors, represented by a boundary class
- the information the system uses, represented by a entity class
- the control logic of the system, represented by a control class

Stereotypes are used to represent these perspectives: in Figure 4.3, a “Client” is representing the boundary between the “Analyst” and the system, the system information is represented by the “MovingObject” entity and a “MovingObject” controller in between is used to decouple the systems interface from its business logic.

The following information flow takes place: the “Analyst” invokes an operation to detect AOI crossing objects at the “Client” which forwards this request to the business logic. This entity checks whether the respective object first enters and then leaves the specified AOI and the result is returned back to the “Analyst”.

4.4.2 View of Participating Classes

In the View of Participating Classes (VOPC) class diagram, the analysis

classes with their responsibilities, attributes and relationships to each other, are illustrated.

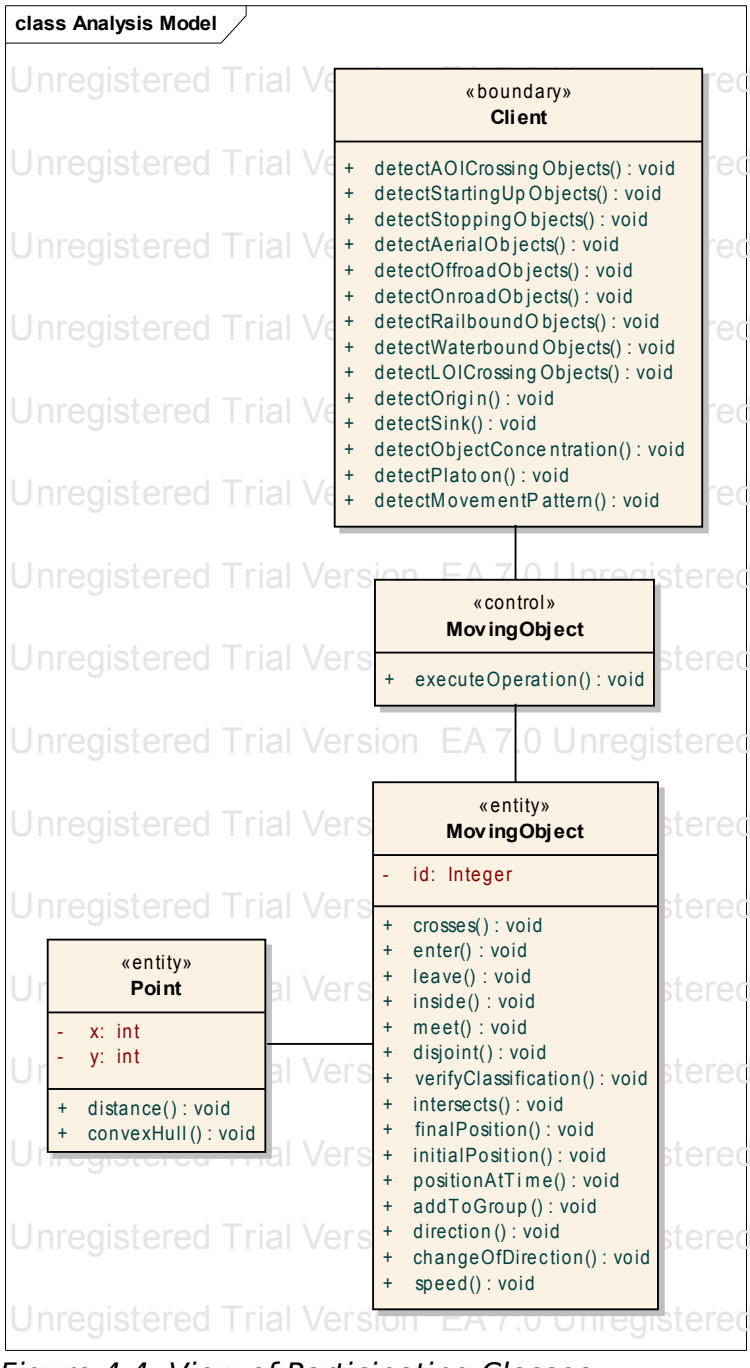


Figure 4.4: View of Participating Classes

Documentation of the classes, depicted in the UML class diagram:

Class	Stereotype	Description
Client	boundary	User interface, presentation layer
MovingObject	control	Controller, decouples user interface from business logic
MovingObject	entity	Object, changing its position over time
Point	entity	Position of a moving point, expressed using coordinates

Table 2: View of Participating Classes

4.5 Design Model

The analysis classes, identified during Use Case analysis, are refined into design elements. During identifying design elements, the decision is made, which analysis classes are really classes, which are subsystems and have to be further decomposed, which are existing components where only an interface have to be declared and which are user interfaces.

4.5.1 Abstract Class Model (PIM)

The Platform Independent Model is focusing on functional aspects.

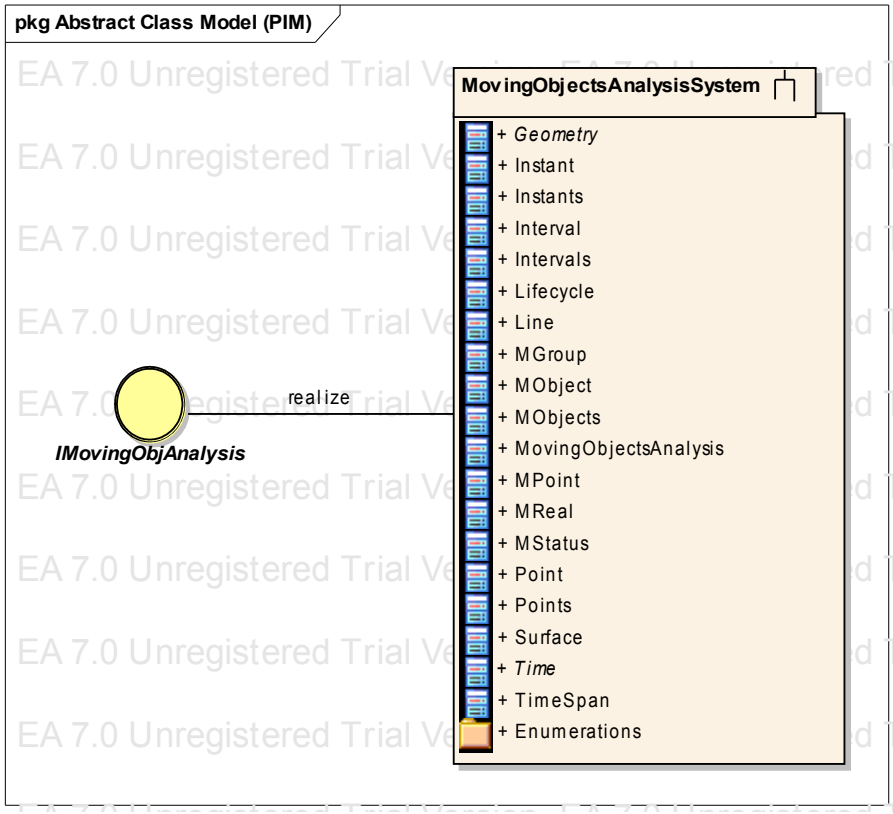
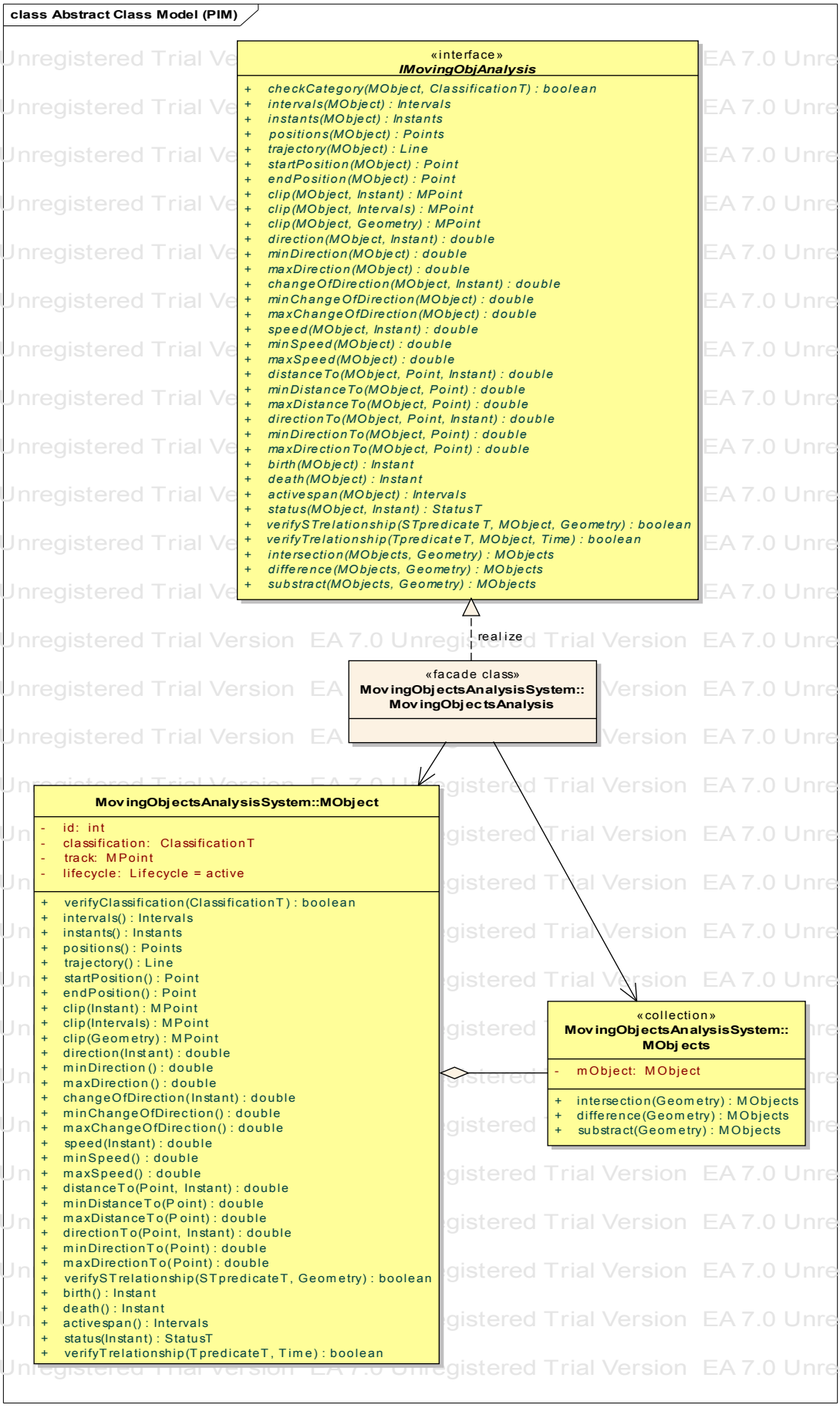


Figure 4.5: Moving Objects Analysis Subsystem & Interface

During transition from analysis model to design model, from “problem domain” to “solution domain”, the entity classes from analysis Model were transferred into a subsystem with an appropriate interface. The interface

defines the subsystems behaviour by offering a set of operations. The implementation, fulfilling these operations, is hidden behind the interface, thus a client using the interface is not affected from changes in the implementation (principle of encapsulation).

Figure 4.6 shows the set of operations, offered by the “IMovingObjAnalysis” interface and, although not in the main focus of this work, a rough idea about how an implementation behind the interface could look like. For this purpose, a structural design pattern, a façade, is applied. The façade is a class with selected operations, comprising a frequently needed subset of the subsystems functionality, delegating the functionality to other classes of the subsystem. The classes behind the façade, implementing the interface of the subsystem, are MObject, a spatio-temporal object type and MObjects, a collection containing multiple such objects.


class Abstract Class Model (PIM)
«interface»
IMovingObjAnalysis
+ checkCategory(MObject, ClassificationT) : boolean
+ intervals(MObject) : Intervals
+ instants(MObject) : Instants
+ positions(MObject) : Points
+ trajectory(MObject) : Line
+ startPosition(MObject) : Point
+ endPosition(MObject) : Point
+ clip(MObject, Instant) : MPoint
+ clip(MObject, Intervals) : MPoint
+ clip(MObject, Geometry) : MPoint
+ direction(MObject, Instant) : double
+ minDirection(MObject) : double
+ maxDirection(MObject) : double
+ changeOfDirection(MObject, Instant) : double
+ minChangeOfDirection(MObject) : double
+ maxChangeOfDirection(MObject) : double
+ speed(MObject, Instant) : double
+ minSpeed(MObject) : double
+ maxSpeed(MObject) : double
+ distanceTo(MObject, Point, Instant) : double
+ minDistanceTo(MObject, Point) : double
+ maxDistanceTo(MObject, Point) : double
+ directionTo(MObject, Point, Instant) : double
+ minDirectionTo(MObject, Point) : double
+ maxDirectionTo(MObject, Point) : double
+ birth(MObject) : Instant
+ death(MObject) : Instant
+ activespan(MObject) : Intervals
+ status(MObject, Instant) : StatusT
+ verifySTrelationship(STpredicateT, MObject, Geometry) : boolean
+ verifyTrelationship(TpredicateT, MObject, Time) : boolean
+ intersection(MObjects, Geometry) : MObjects
+ difference(MObjects, Geometry) : MObjects
+ substract(MObjects, Geometry) : MObjects
realize
«facade class»
MovingObjectsAnalysisSystem::
MovingObjectsAnalysis
MovingObjectsAnalysisSystem::MObject
- id: int
- classification: ClassificationT
- track: MPoint
- lifecycle: Lifecycle = active
+ verifyClassification(ClassificationT) : boolean
+ intervals() : Intervals
+ instants() : Instants
+ positions() : Points
+ trajectory() : Line
+ startPosition() : Point
+ endPosition() : Point
+ clip(Instant) : MPoint
+ clip(Intervals) : MPoint
+ clip(Geometry) : MPoint
+ direction(Instant) : double
+ minDirection() : double
+ maxDirection() : double
+ changeOfDirection(Instant) : double
+ minChangeOfDirection() : double
+ maxChangeOfDirection() : double
+ speed(Instant) : double
+ minSpeed() : double
+ maxSpeed() : double
+ distanceTo(Point, Instant) : double
+ minDistanceTo(Point) : double
+ maxDistanceTo(Point) : double
+ directionTo(Point, Instant) : double
+ minDirectionTo(Point) : double
+ maxDirectionTo(Point) : double
+ verifySTrelationship(STpredicateT, Geometry) : boolean
+ birth() : Instant
+ death() : Instant
+ activespan() : Intervals
+ status(Instant) : StatusT
+ verifyTrelationship(TpredicateT, Time) : boolean
«collection»
MovingObjectsAnalysisSystem::
MObjects
- mObject: MObject
+ intersection(Geometry) : MObjects
+ difference(Geometry) : MObjects
+ substract(Geometry) : MObjects

4.5.2 GML Data Model (PSM)

A PSM is focusing on technical aspects. In case of the GML data model, the focus is on data definition for XML documents using XML Schema. A moving object can be modelled as a discrete geographic feature that can handle all the information about a real world entity, including updates and history. Its characteristics are recorded as feature attributes. Multiple moving objects can be accumulated in a feature collection.

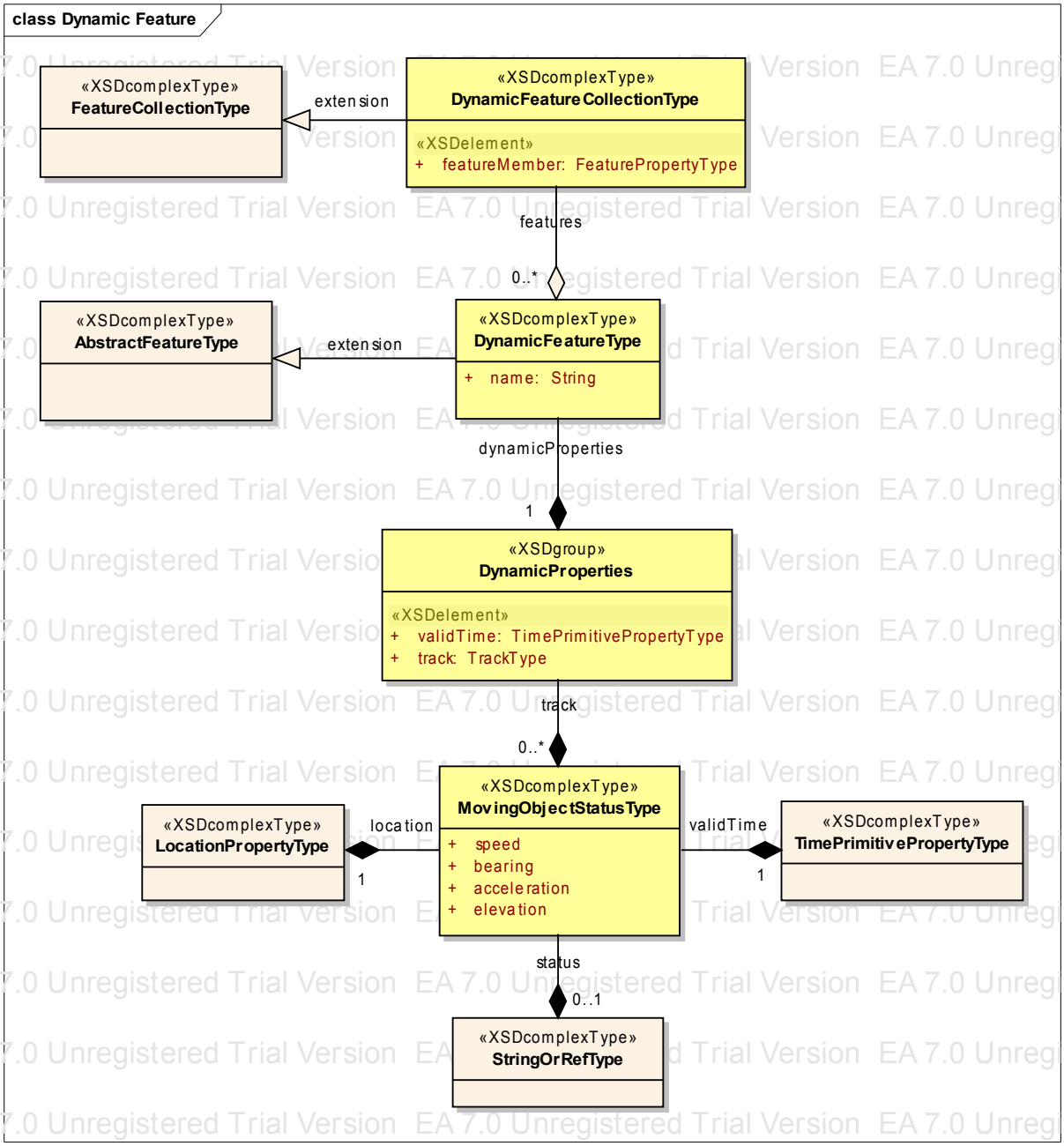


Figure 4.7: Dynamic Feature GML Core Schema

To model geographic features in XML, the GML Implementation Specification provides 29 GML Core Schemas. The Dynamic Feature Schema (see Figure 4.7) is appropriate to define spatio-temporal features. The content model of a dynamicFeatureType extends the standard AbstractFeature-

Type with a dynamicProperties model group that contains a history element. In GML version 3.1.1, a MovingObjectStatusType is used to encapsulate various dynamic properties of moving objects. It's one example of how an AbstractTimeSliceType, encapsulating the time-varying properties of a dynamic feature, may be extended.

An Application Schema, defining the data structure of a moving object that changes its position dynamically over time, was defined based on the Dynamic Feature Core Schema (see Figure 4.8). The MovingObjectType extends the DynamicFeatureType with a sequence of staticProperties, comprising the moving objects lifecycle (date of birth and date of death), an identifier and a classification element. Additionally, the Application Schema restricts the time property of the track element to a time instant and the location property to point-shaped position. Finally, dedicated enumerations for the dynamic status property and the static classification property are defined. The XSD file itself is put in Annex B.

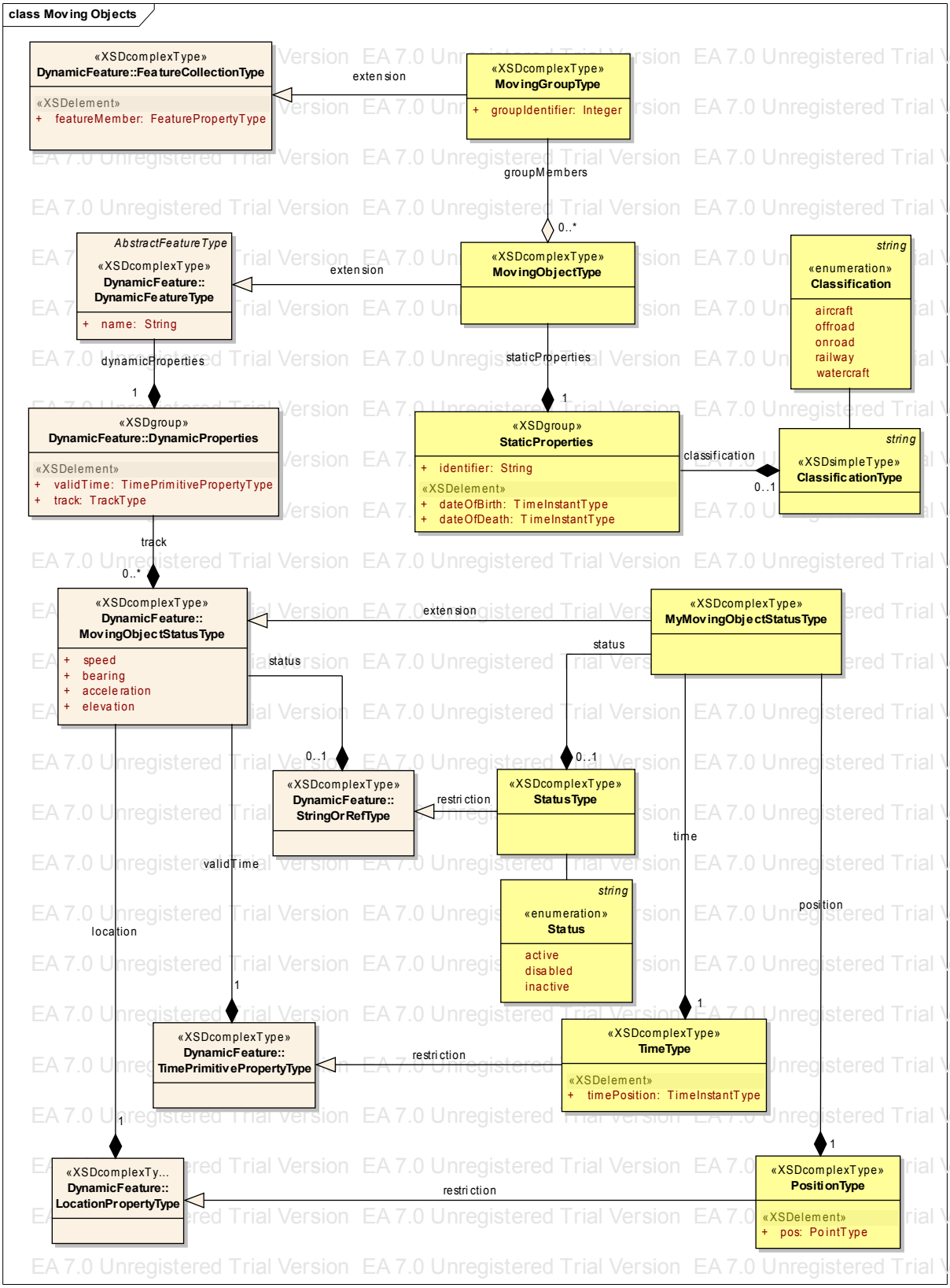


Figure 4.8: Moving Objects Application Schema

4.5.3 OWS Model

The OWS model defines an OpenGIS compliant interface. For each geoprocessing functionality, offered by the "IMovingObjAnalysis" interface, a WPS

process has to be developed, exemplary done for the crosses process as illustrated in Figure 4.9.

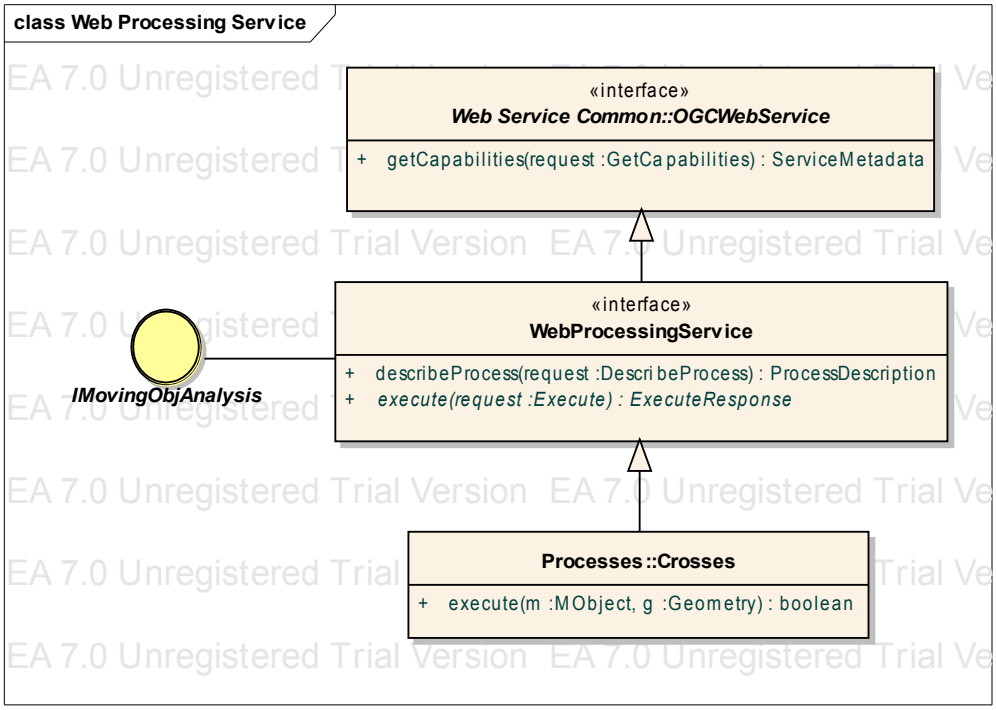


Figure 4.9: Web Processing Service

This means, first of all, the interface has to offer the mandatory getCapabilities() operation which has to be provided by each OpenGIS Web Service. This operation provides service metadata and, if describing the capabilities of a WPS, as in this case, additionally the offered processes.

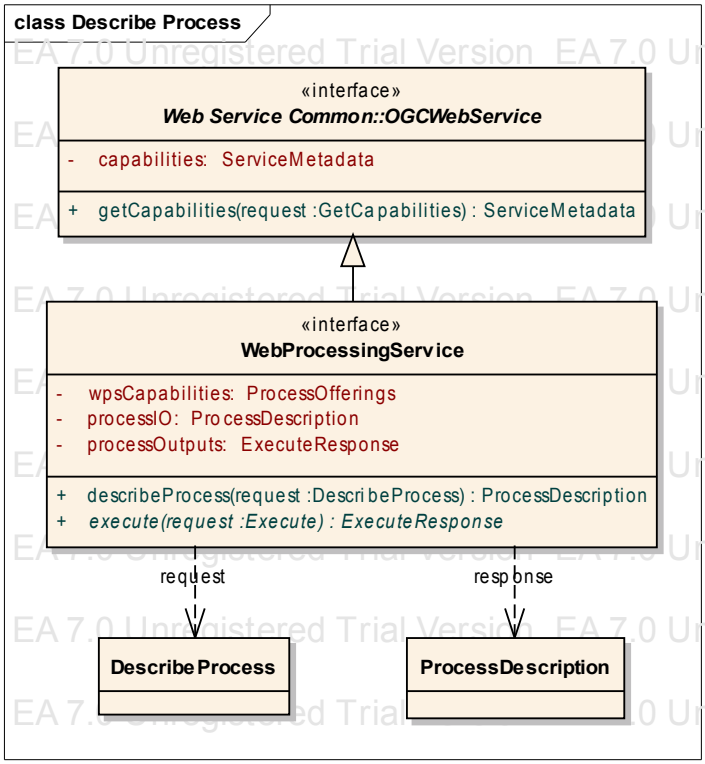


Figure 4.10: Describe Process Request & Response

The remaining two operations are specified by the Web Processing Service Implementation Specification, which defines the interface to a WPS. The abstract execute() operation has to be redefined by a specific process which is performing the specific geoprocessing task.

The describeProcess() operation therefore provides detailed information about such a process, including the necessary input parameter to invoke its execution.

4.6 Prototypical Web Service Implementation

A PSM is focusing on technical aspects. In case of the prototype, the focus is on service implementation using the programming language Java.

The realized Web Processing Service shall only demonstrate the feasibility of the designed concept in principle and therefore implicates some limitations:

- only the crosses process is exemplary implemented
- no real processing algorithm is accomplished which is analysing the input data and generating a respective processing result – instead, just “true” is returned which applies to the used test data

4.6.1 Architectural Approach

Multi-tier architecture:

- presentation tier: information presentation, human interface
- business logic tier: data access, data processing / information generation, result forwarding to presentation tier
- data tier: provision of data, web services as data provider

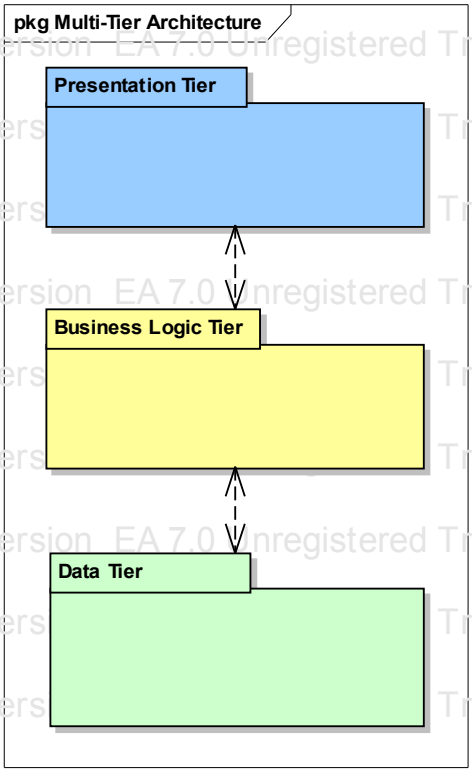


Figure 4.11: Multi-Tier Architecture

The prototype comprises only the central business logic tier, implemented as a WPS. The required data are already locally available, so no geodata access service (like a WFS) is required. The WPS GetCapabilities and DescribeProcess requests, using the HTTP GET method and KVP encoding, are carried out directly as URL in a web browser. For invoking the Execute request, using the HTTP POST method and XML encoding, a “generic OGC Web Service client” (see Figure 4.18) was used. The processing result can be analysed on basis of the XML encoded execute response document and therefore no client application has to be developed.

The WPS is realized as a web application. In order to process movement data into valuable information, the web application requests the input data from the user and returns the processing result, which is either true or false, as response. In an operational Use Case, which is going beyond the scope of the prototype realized in this work, a more sophisticated web application would encapsulate the WPS: after receiving all necessary input data, the web application would call the WPS and apply the crosses process for each moving object. Finally, the application would return the moving objects which are crossing the AOI.

The business logic of movement analysis is encapsulated in the WPS pro-

cess, the web application is responsible for interaction with the user. The WPS process for generating this value-added information and the web application for communication with the client are implemented using the Open Source SDI framework deegree.

4.6.2 Java Framework Deegree

Deegree[1] is a free and Open Source Software (OSS) framework based on the OO programming language Java. It is a comprehensive implementation of OGC and ISO standards. Version 2.1 includes, besides well-established OWS like WMS or WFS, an implementation of the WPS 0.4.0 specification:

- current deegree version 2.3: pre-version
- used deegree version 2.1: latest stable version, implements GML version 3.1 and WPS version 0.4.0

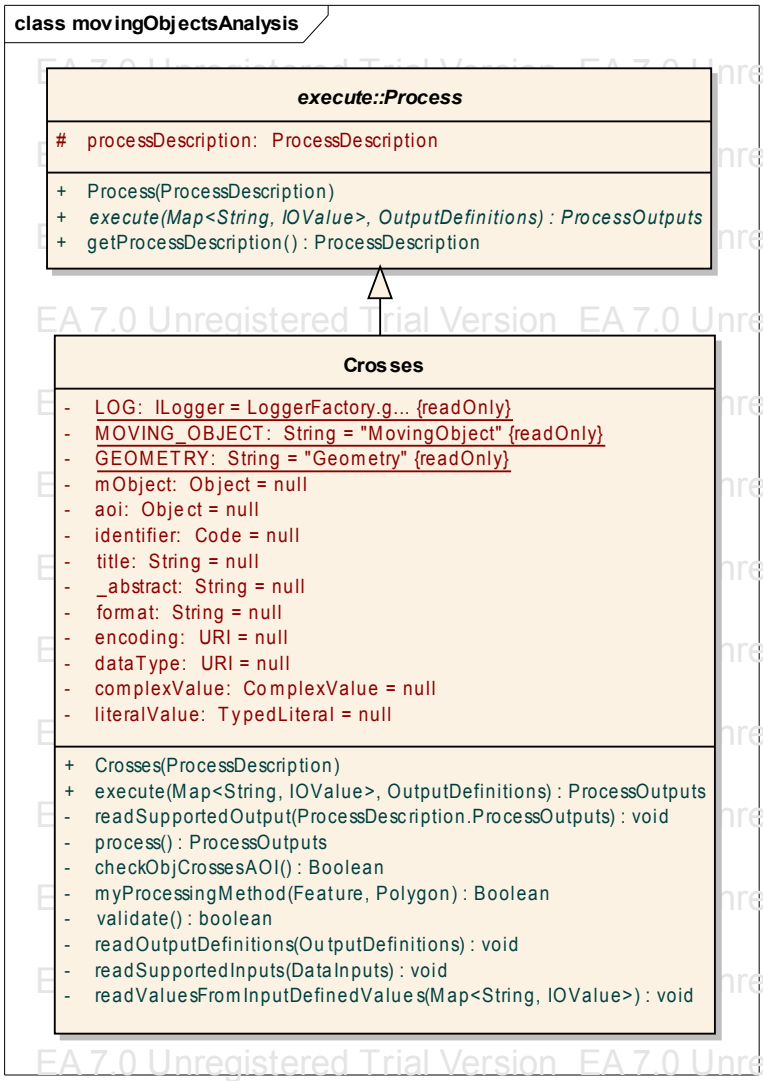


Figure 4.12: Crosses Process

1 According to the mail announcement of Dr. Markus Lupp from 03.01.2008, a Project Steering Committee (PSC) for the deegree project was formed which aspires the application of deegree to become an official OSGeo project

"The deegree WPS allows the definition of custom processes by implementing the relevant business logic and configuration of input-/output parameters through a well-defined XML document."(Kiehle et al., 2007, p. 823) The deegree WPS reference implementation, also described in (Kiehle et al., 2007) and (Heier & Kiehle, 2006), includes a buffer process as an example of such a custom process. Accordingly, a crosses process was implemented extending the number of deegree WPS processes (see Figure 4.12 for the respective UML class).

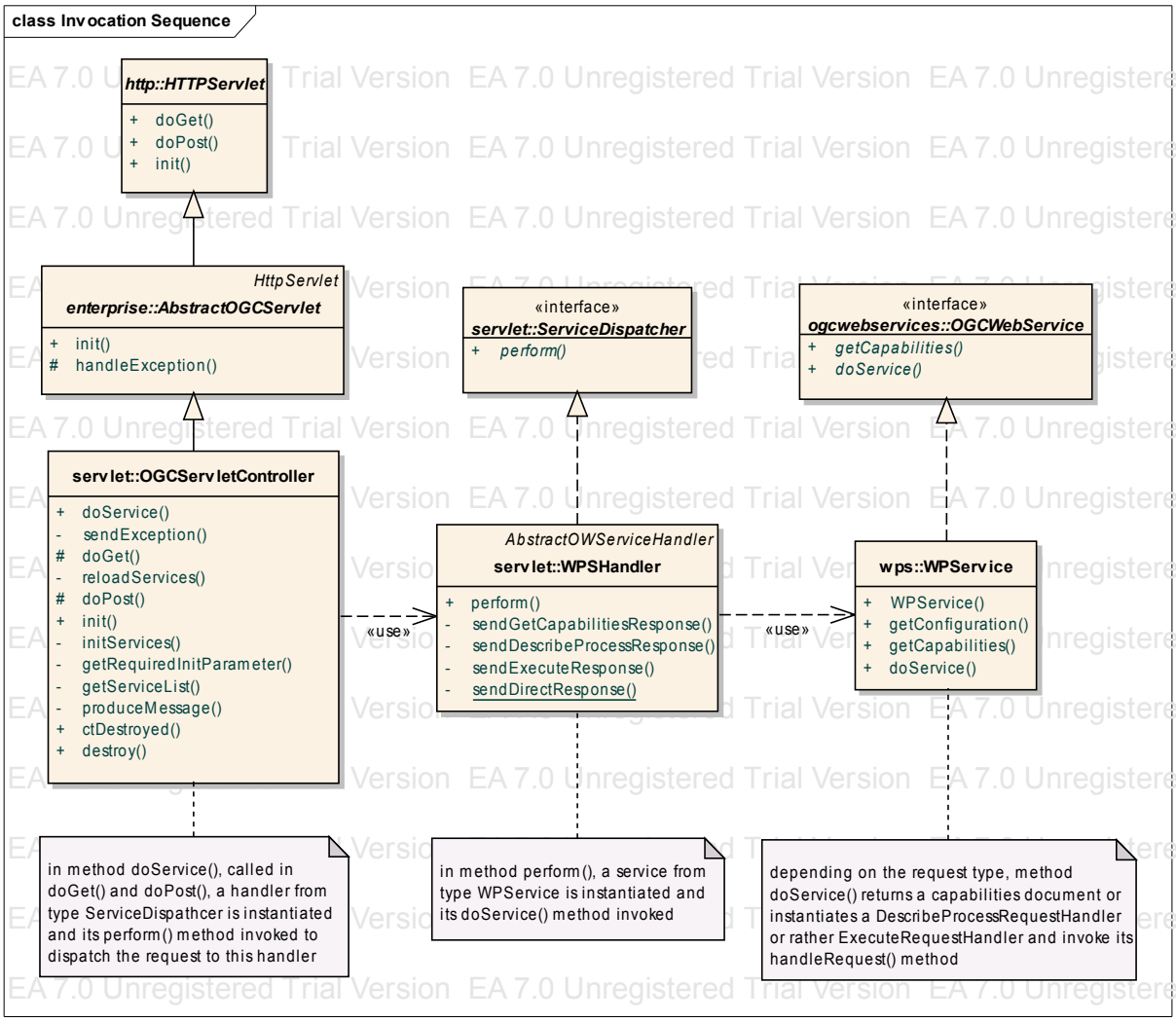


Figure 4.13: Deegree Framework For Building Web Applications

"All infrastructure-specific tasks like error handling, process registration, and advertisement of metadata in OGC-compliant format as well as subordinate tasks like logging and access constraints are handled by the framework."(Kiehle et al., 2007, p. 823) The deegree framework provides the respective classes to handle HTTP requests and forward a client query to the Web Processing Service, as illustrated in Figure 4.13. The same applies to the reverse way concerning the WPS response.

In short, the following steps are necessary to define a custom WPS process using the deegree framework:

- Definition of a XML configuration document which serves as a tem-

plate for the DescribeProcess response (process description document) and as configuration file for the WPS inside the deegree framework. In order to serve as a deegree configuration file, the ordinary XML document is extended by special deegree parameter tags. Such a tag is used to refer to the name of the process implementation class which is called deegree:responsibleClass

- Implementation of the processing logic by subclassing the abstract super class *org.deegree.ogcwebservices.wps.execute.Process.java*

A detailed description of implementing and setting up a custom WPS is given in the following chapter and outlined in Figure 4.14.

4.6.3 Implementation, Configuration and Deployment

As already mentioned in the chapter before, a Java class realizing a custom process by subclassing the abstract super class Process.java has to be implemented. The source code of the Crosses.java class is put in Annex D.

This process class, responsible for handling the incoming requests, has to be referred in the process description document. This XML document describes the custom process, especially the input parameters for its execution. Another response document, the capabilities document, needs to be adapted to describe common service metadata e.g. the service provider. Finally, the execute request document, including the input data (further described in chapter 4.6.4) for process execution, has to be encoded in XML (all XML files were put in Annex C).

Figure 4.14 illustrates graphically the configuration items of a WPS as well as their relationships to each other: the red boxes show the aforementioned XML documents and Java class that need to be developed. The capabilities document refers to a directory where the process description document is stored. This in turn holds the name of the process class. The process class handles the request invoked through the execute request document. This again contains, amongst others, input data from a moving object data type, described through the respective GML Application Schema. The entry point in this sequence, however, is the deployment descriptor, which refers to the capabilities document. The web application and its runtime environment is described in the following paragraphs.

In order to get a web application running, first of all, its runtime environment has to be set up:

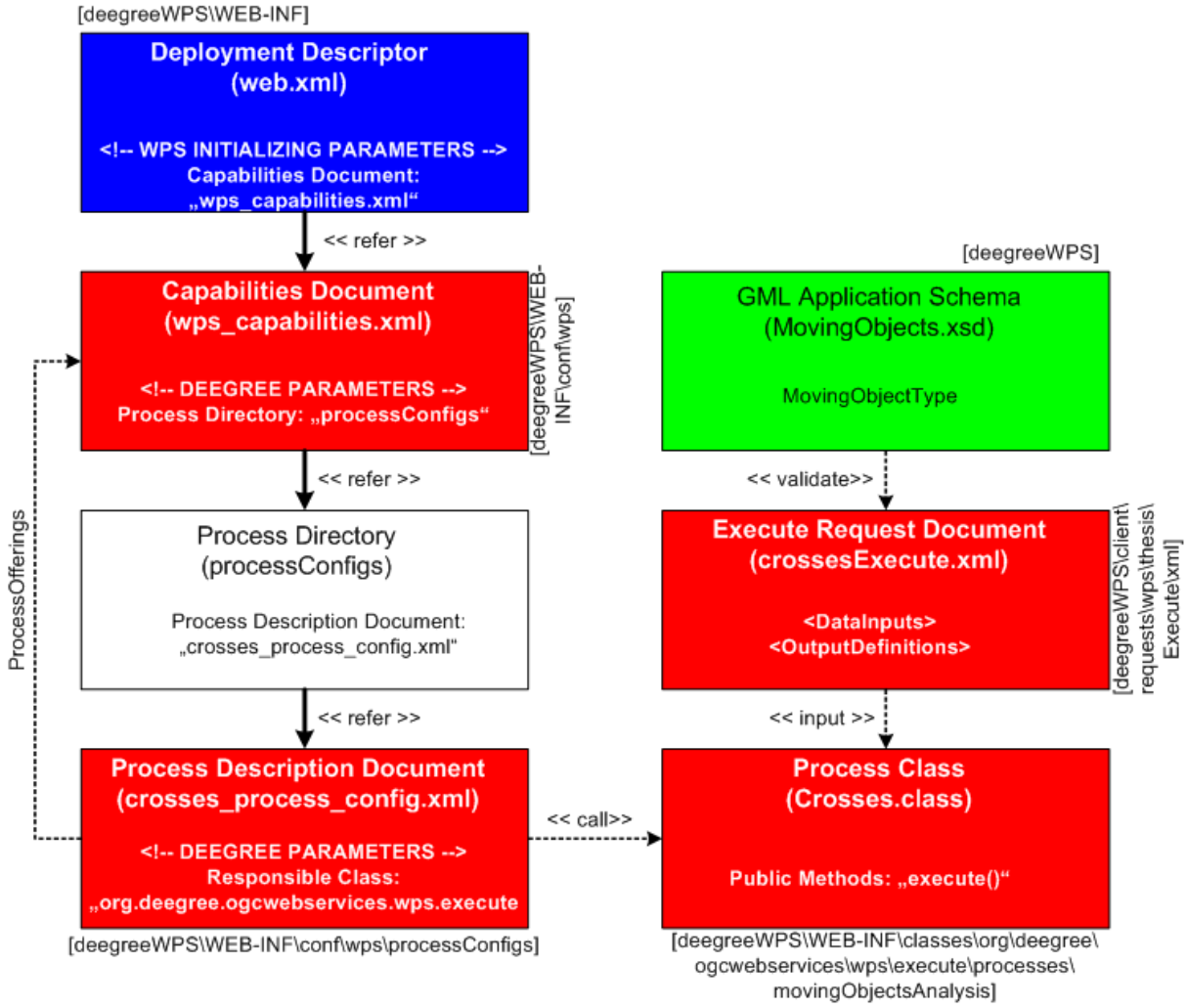


Figure 4.14: Deegree WPS Configuration

- webserver: Apache HTTP webserver (see remarks concerning development & test environment)
- Java servlet container: Apache Tomcat

A web application, in its unpacked form, is defined as a hierarchy of directories and files in a standard layout. The top-level directory of a web application, a subdirectory of Tomcat's webapps directory which is also called the context path of the application, contains files that comprise the application's user interface. The WEB-INF subdirectory contains the web applications deployment descriptor, a XML file named web.xml, a classes subdirectory containing Java class files, required for the application (servlet and non-servlet classes, that are not combined into JAR files) and a lib subdirectory containing JAR files. The classes in the WEB-INF/classes/ directory as well as all classes in JAR files, found in the WEB-INF/lib/ directory, are made visible to other classes within the web application. If the web application gets assigned the context path “deegreeWPS”, then a request URI *http://localhost:8080/deegreeWPS/services?* will activate the servlet mapped to the URL pattern /services.

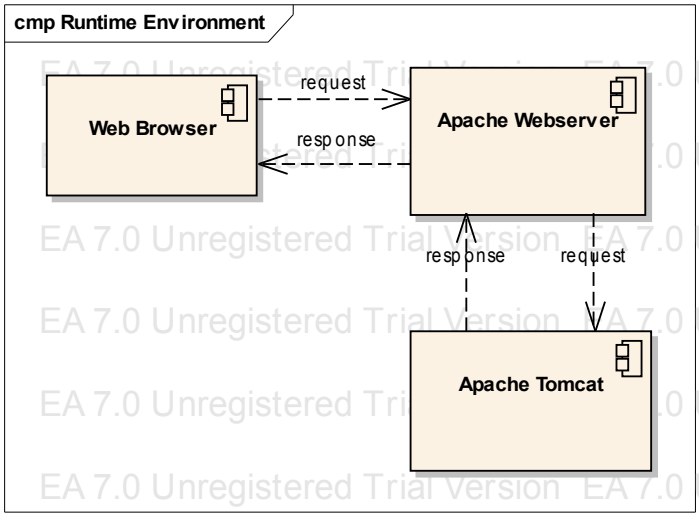


Figure 4.15: Runtime Environment

The webserver receives the requests from a web browser, forwards them to the Java servlet container for further processing and finally provides the processing result as response.

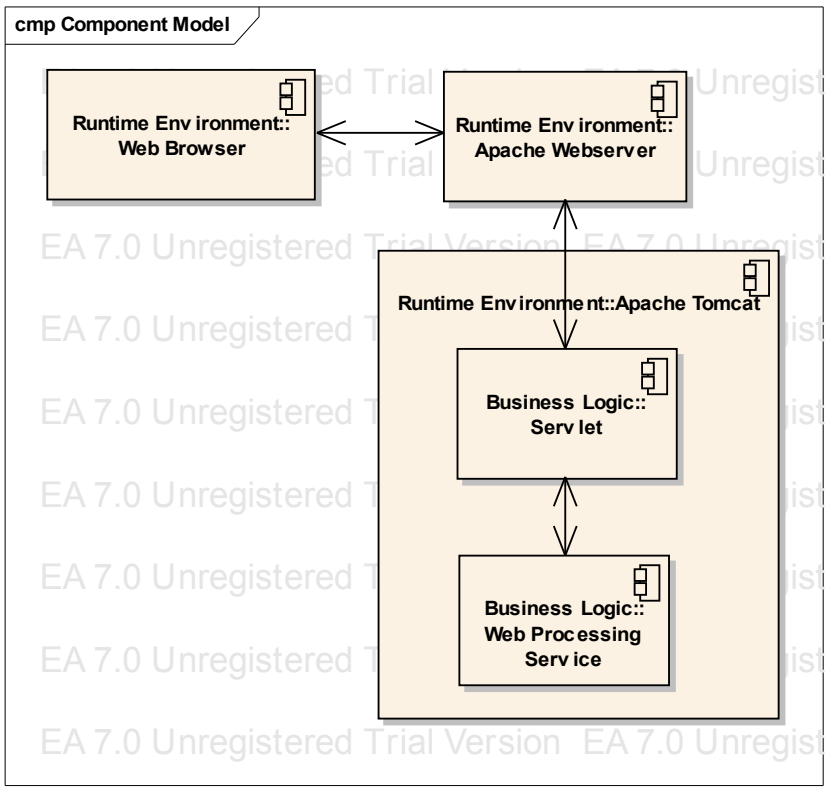


Figure 4.16: Web Application Component View

The Java servlet container provides an environment to execute Java code on a webserver. The servlet container is a running Java program where servlets have to register themselves. After dynamic response generation, the servlet container returns the processing result to the webserver.

The Java servlet technology makes web applications possible. Servlets are not compiled Java programs with a main method, they are rather single

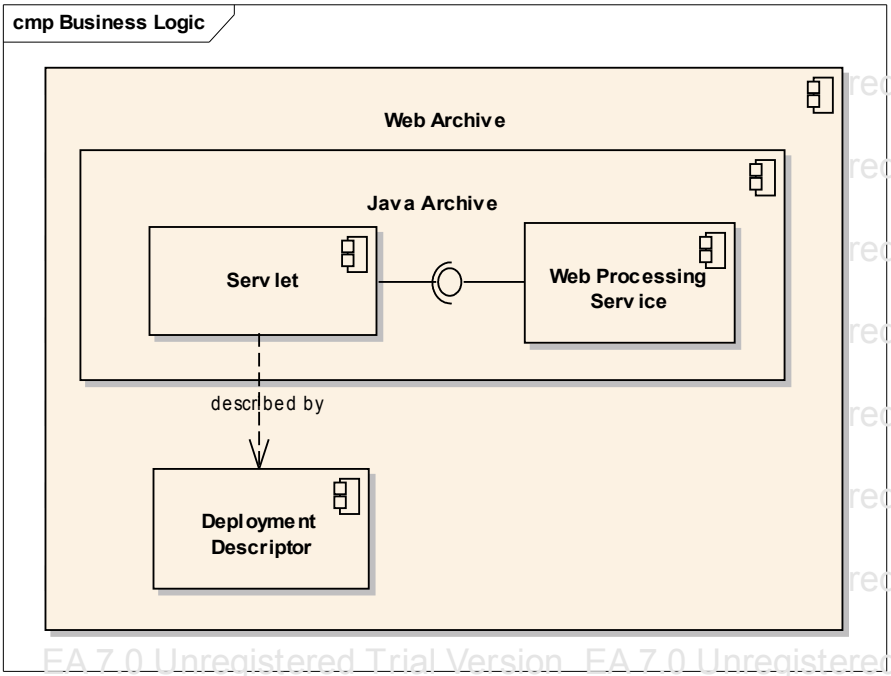


Figure 4.17: Web Application Business Logic

classes able to process HTTP methods (at least GET and POST), instantiated and activated by the servlet container on demand. The compiled servlet together with a deployment descriptor (see Figure 4.12), a XML file holding meta information about the servlet, and further classes, implementing the business logic “behind” the servlet (in this case performing the process algorithm by utilizing a WPS), can be packed into a single archive-file. Such a Web ARchive (WAR file) presents a web application in a packed form and is used to distribute a web application to be installed.

A WAR file for the buffer WPS, including the deegree infrastructure to handle HTTP requests (OGC servlet and further classes for building web applications as illustrated in Figure 4.13), is provided by Dr. Christian Kiehle on the lat/lon GmbH website for download[1].

For the crosses WPS web application, the downloaded and unpacked web archive from Kiehle was used to create the respective deegree configuration files. InFigure 4.14, the path for each file, relative to the applications context path, is given in squared brackets.

In order to formulate the WPS execution request using the HTTP POST method, the generic OGC web service client from deegree was used. A request URI referring to *deegreeWPS/client/client.html* will retrieve the start page of the generic client from the respective directory.

1 URL: http://web.lat-lon.de/~kiehle/WPS.zip

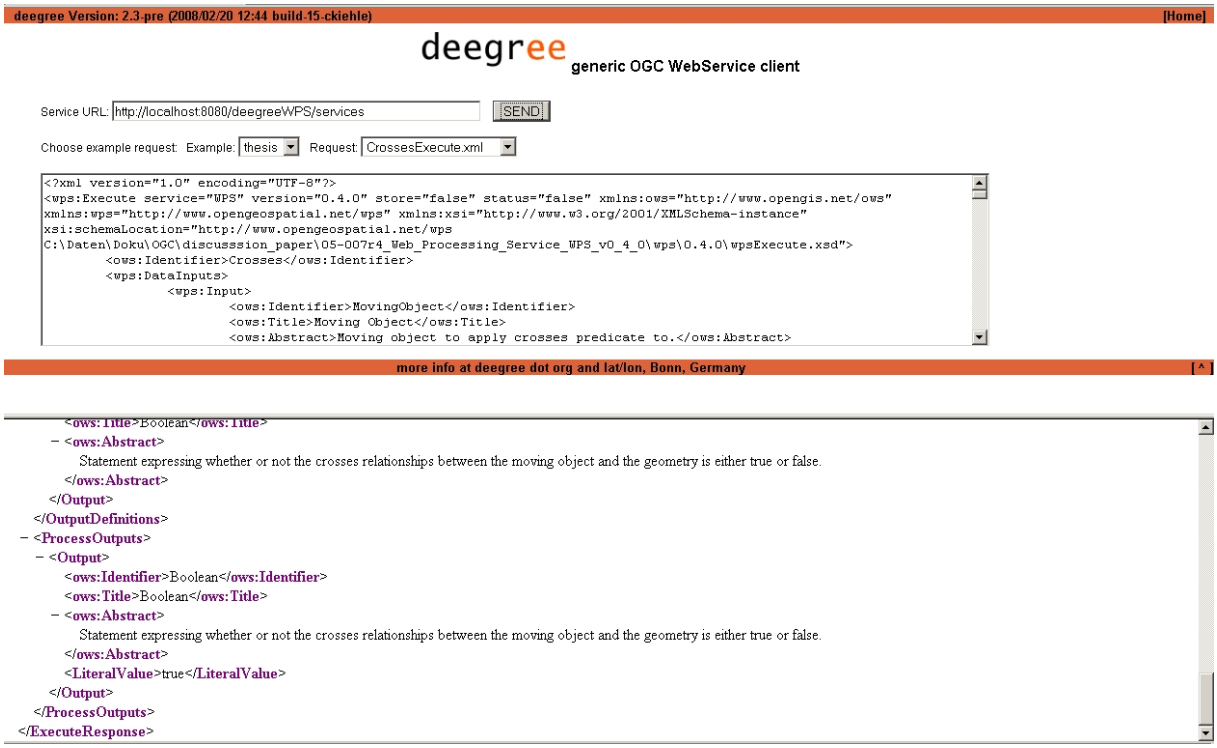


Figure 4.18: Generic OGC Web Service Client

To avoid various exceptions while running the web application, the following “best practices” should be ensured, considering common installation issues as well as customization to the deegree framework:

- avoid blanks in the installation path of Apache Tomcat
- increase the Java Virtual Machine's memory: *set JAVA_OPTS=-Xmx768*m in catalina.bat
- in the process description document, set the tag *<wps:SupportedUOMs/>* in the ProcessOutputs section
- in the execute request document:
  - set *xmlns:ows="http://www.opengis.net/ows"* in the XML head
  - use a web accessible URL in the *schema* attribute of the ComplexValue tag when describing DataInputs
  - set *schema* and *uom* attributes of Output tag when describing OutputDefinitions

On a development & test environment, webserver and web browser can be located on the same machine. In a production environment, the web browser would be deployed on a client and the webserver, including the servlet container, would be deployed on a server. Another simplification is to use the HTTP server from Apache Tomcat instead of installing a standalone Apache HTTP webserver – this is sufficient for a development & test environment, in a production environment an Apache HTTP webserver would be utilized.

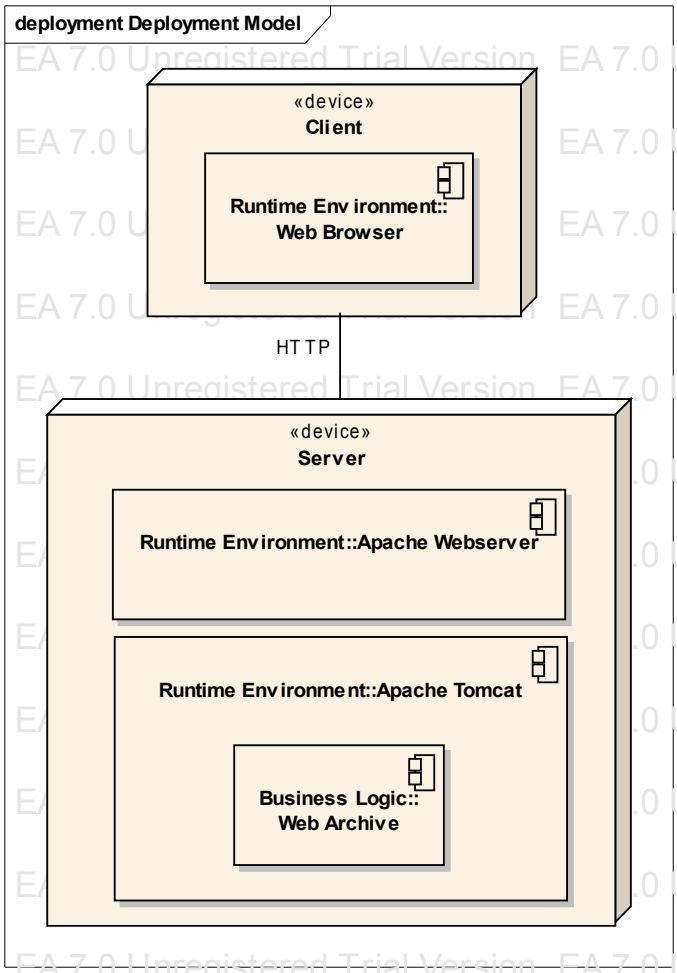


Figure 4.19: Web Application Deployment View

4.6.4 Test Area And Data

4.6.4.1 Case Study

A vehicle is driving from the city of Meersburg to the city of Constance. This means, the vehicle has to overcome the Lake Constance which separates the journey's starting point from the destination. This can be done either by driving around the lake or by crossing the obstacle using a car-ferry. The vehicle in this case study, illustrated as yellow trajectory, is crossing the lake using a car-ferry.

The goal is to find out which alternative the vehicle was going to choose utilizing a WPS.

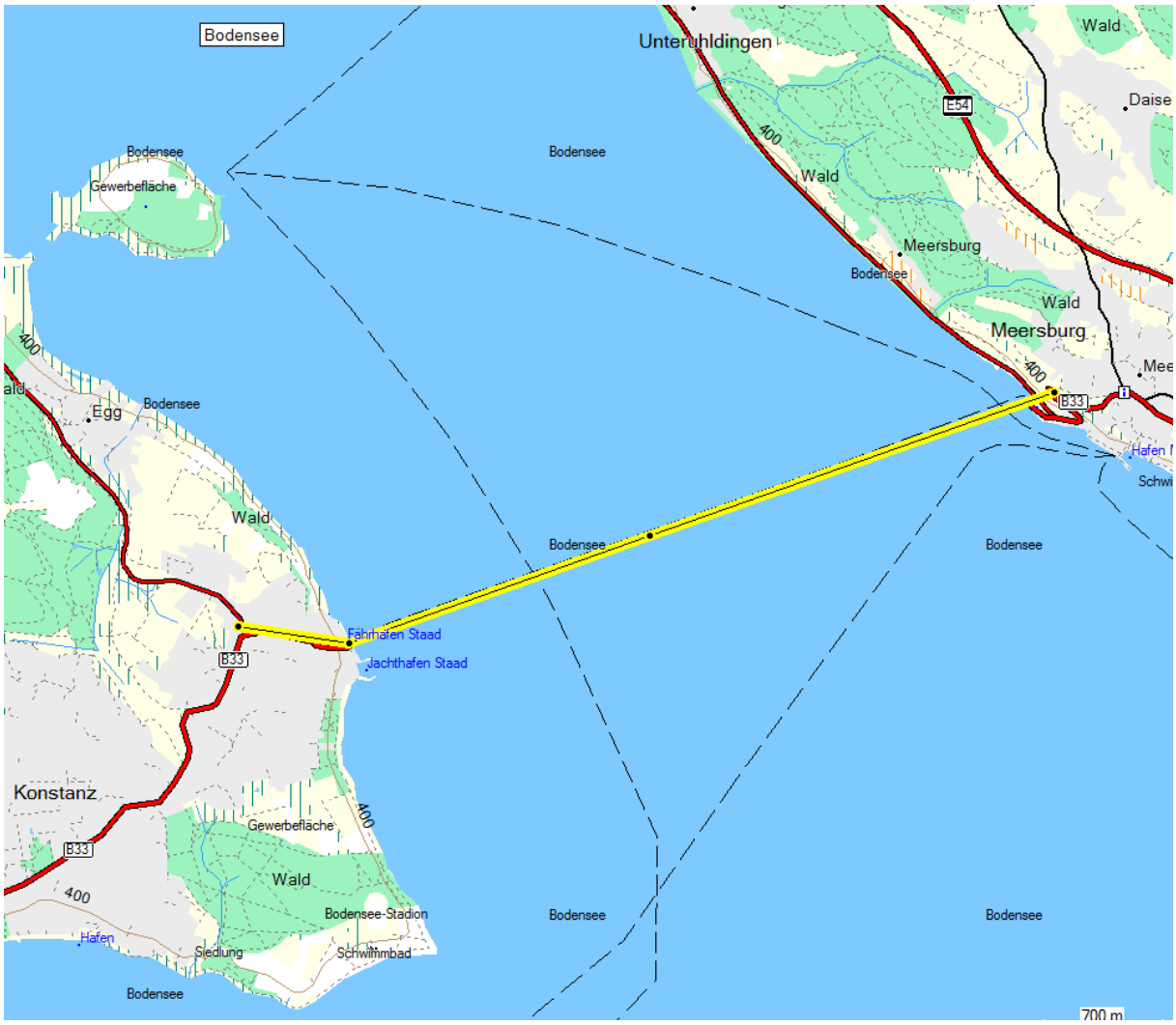


Figure 4.20: Case Study

4.6.4.2 Outline Of Spatio-Temporal Problem

The approach is to check the validity of a particular development, moving object crosses AOI, which means the change of topological relationships between a moving objects' geometry and a polygon geometry over the moving objects' lifetime. For this purpose, a spatio-temporal crosses predicate is applied.

The following input data sets, GML[1] encoded, were required:

- moving object: Garmin's MapSource tool was used to create a track. The single track points were defined on top of a digital road map and stored in the GPX format. Finally, the coordinates of the track points were carried over in a GML file to define a moving object geometry.
- polygon: a topographic vector map was used to generalize the approximate shape of the Lake Constance and the resulting coordinates were carried over in a GML file to define a polygon geometry.

1 The GML files itself are outlined in Annex C

5 Results

This chapter provides an overview of the achieved results with regard to the expectations, expressed in the questions outlined in chapter 1.4.

Is it possible to apply methods from project management and software engineering to elaborate a geospatial processing problem?

First of all, a project plan was created to define the necessary development activities, to set up a time schedule including milestones and to control the project's progress. A model driven SW development approach according to the principles of MDA led to a comprehensive visual UML model incorporating all the analysis concepts and the software design developed during the SW engineering process. Due to the fact that the UML is process independent, applicable across all domains and additionally opened-ended for extensions, this modelling language was used to design the geoprocessing functionality in a Use Case driven, architecture centric, iterative and incremental approach.

Is it possible to take over concepts from conceptual spatio-temporal modelling and Moving Objects Databases in the software design?

The developed design model incorporates a Platform Independent Model (PIM), called abstract class model, that defines an interface for moving objects analysis. The operations, offered by this interface, were carried over from spatio-temporal modelling, as describe in the MADS approach, and from Moving Objects Databases, respectively.

Is it possible to define processes for moving object analysis based on the OGC WPS Implementation Specification?

In the OWS model, own WPS processes were defined to describe and web-enable functionality for moving objects analysis. The WPS interface standardizes the way, how processes and their input and output data are described and how a client can request a process execution. The actual data processing is encapsulated by the generic WPS interface and executed by the custom processes. Because WPS processes are not coupled to the data they operate on, they can be reused in any context or, due to their atomic character, orchestrated into service chains or rather integrated with other OpenGIS Web Services to form higher-level services.

Is it possible to use the GML Dynamic Feature Core Schema to model a moving object Application Schema?

The Dynamic Feature Schema defines a number of types and relationships to represent time-varying properties of geographic features. A dynamic feature therefore extends a geographic feature about dynamic properties. A TimeSlice element is an abstract GML object that encapsulates updates of dynamic properties. A MovingObjectStatus extends the abstract

TimeSlice to represent a track. The developed Application Schema put some restrictions on the track type allowing only a time instant type as valid time, a point type as location and a custom enumeration as status. Finally, a group element was defined in the Application Schema representing a moving object's static properties.

Is it possible to utilize the deegree SW framework to implement a prototypical WPS to proof the defined concept?

The deegree framework handles all infrastructure-specific tasks like error handling, logging and process registration. The deegree WPS allows the definition of custom processes. A crosses process, implementing the processing logic for a moving objects analysis functionality, was implemented by subclassing the abstract process super class. A respective XML configuration document for the WPS inside the framework was defined. In a special deegree parameter tag, an ordinary XML process description document, describing especially the input parameters to execute the process, was extended to refer to the crosses class. Another response document, the capabilities document, was adapted describing common service metadata e.g. the service provider of the crosses WPS. Finally, the execute request document, including the input data for the process execution, was encoded in XML.

6 Analysis of Results

In general, the achieved results met the expectations existing in the run-up to this Thesis.

The chosen model driven development approach was optimally adapted to analyse and design the SW architecture. A Use Case driven process yielded to initial functionality which was iteratively and incrementally enhanced

- considering spatio-temporal concepts
- be compliant with the OpenGIS Web Processing Service Implementation Specification
- containing implementation and platform specific details.

The project plan, only marginally updated during the project phase, was a crucial measure in order to finalize the work in time.

Data types, operations and predicates from conceptual spatio-temporal modelling, as addressed in the MADS approach, could be integrated in the Platform Independent Model without much effort because both models are free from implementation specific details. The same is valid for an abstract Moving Objects Database model, discrete data structures and SQL-like queries however had to be translated from the “relational database world” into the “object oriented world”.

Because a WPS is able to offer all sorts of GIS functionality, it can be also applied to moving objects analysis. The WPS, as described in the respective OGC Implementation Specification, offers a generic interface that can be used to describe and web-enable any geospatial process. A custom process is executing the actual data processing while the standardized WPS interface specifies how input and output data are described and the service request is handled.

The GML Dynamic Feature Schema can be utilized for any kind of time-varying features. The Core Schema offers an abstract element that can be extended to represent the dynamic properties of a moving object. In GML version 3.2.1, a respective track element is included as a normative schema component and therefore has not to be developed in the Application Schema. In the latest version, this element was deprecated and used instead as an informatively example. In order to keep the effort for the Application Schema as low as possible, GML version 3.2.1 was applied.

The deegree SW framework is an OWS reference implementation including also Java classes to realize an OpenGIS compliant WPS. The latest stable deegree version, however, supports not the latest WPS version 1.0.0, instead the predecessor version 0.4.0. The capabilities documents therefore have to be encoded according to the schema definitions of this prior ver-

sion. The deegree WPS allows the definition of custom processes by subclassing an abstract super class. This can be done according to the buffer process, which is part of the framework. The XML documents have to be enhanced with the necessary deegree parameters in order to get the service to run. For this purpose, the respective buffer files can be adapted to the own process' needs. In order to have all properties of a moving object available in the course of implementing the crosses process business logic, a respective data type should be defined in the deegree framework.

7 Summary, Discussion, Outlook

7.1 Summary

This chapter is summarizing what the author has done during this work.

First of all, a literature survey was conducted in order to get a picture of current developments in spatio-temporal modelling and web-enabled, standardized geoprocessing research.

The question about a proper methodical approach for software engineering and project management led to a UML model and a project plan.

In the early software design, concepts from conceptual spatio-temporal modelling and Moving Objects Databases were applied and a platform independent design model was developed. In order to offer this functionality through an interoperable geoprocessing service, a WPS process, describing and web-enabling functionality for moving objects analysis, was modelled. The WPS Implementation Specification requires GML encoding for the process input data: therefore, an Application Schema, based on the Dynamic Feature Core Schema, defining the data structure of a moving object, was developed.

In order to demonstrate the feasibility of the designed concept, a web application, based on the deegree SW framework, was prototypically implemented. The crosses process was coded as a Java class, the deegree framework configured and the runtime environment established. Finally, the accurate request-/ response behaviour for all three mandatory WPS interface operations could be demonstrated.

7.2 Discussion

This Thesis connects concepts that are innately discrete: analysis of moving objects and OpenGIS Web Services. By identifying moving objects as spatio-temporal entities with a time-varying position and moving objects analysis as a geoprocessing task, the OGC standards framework contains the proper measures to implement a standardized, distributed processing service:

- the OWS Implementation Specification for a Web Processing Service interface
- the Geography Markup Language with the Dynamic Feature Schema

The professional literature is using different terms in order to denominate spatio-temporal data types. While (Güting & Schneider, 2005) use the term temporal respectively moving spatial type, (Parent et al., 2006) use

the term time-varying spatial type. Caution is advised as well when using the terms time and temporal data type respectively.

The deegree framework provides the infrastructure to implement OpenGIS Web Services in general and custom WPS processes in particular. Because it follows the OO principles of abstraction, encapsulation, modularity and hierarchy, comprehensive web applications can be build very fast. The developer can concentrate on the business logic and don't have to care about underlying infrastructure-specific tasks that are handled by the framework.

In order to configure the deegree WPS, deegree specific tags are required in the capabilities and process description document. Because these deegree parameters are not part of the standard XML Schema, a schema validation yields to an incorrect result. The author's recommendation therefore is to first develop pure standard conform XML documents, validate them against the respective schema definitions and finally insert deegree specific parameters. Additionally, the deegree WPS implementation requires some definitions, necessary in order to avoid exceptions at runtime, that are not mandatory to implement concerning the implementation specification. The author suggests to revise these parts of the deegree WPS in an upcoming release.

As already denoted, there are some pitfalls to go around in order to get the final web application running. This refers to installation and configuration as well as the WPS itself. Some of them are listed as “best practices” in chapter 4.6.3. The author recommends to set up a WPS Implementation Guide closing implementation specific gaps essential for developers to implement the standard in a common and interoperable way. The deegree specific parameters as well as “best practices” could be part of such an Implementation Guide, which is one of the three pillars, besides the Implementation Specification itself and a prototype, interoperability projects are based on.

7.3 Outlook

7.3.1 Generalisation of Proposed Solution

This Thesis focused on spatio-temporal processing functionality for moving points: in a more general approach, all kinds of time-varying objects, also spatio-temporal object types with a time-varying spatial extent, moving regions, have to be considered.

Another restriction was given by considering only Use Cases from a single business domain. In order to come closer to a complete set of operations and data types for spatio-temporal analysis and therefore to get something like a “Spatio-Temporal Algebra”, other business domains and respective Use Cases have to be considered. Possible business domains, dealing with similar topics, could be public transportation (traffic telematics), fleet management and Supply Chain Event Management (SCEM).

7.3.2 Proposed Steps For Continuation of Work

After finishing this work, there are some topics remaining for future activities.

Instead returning a static defined processing result, the implemented crosses process could be extended about real processing algorithms. For example, a Moving Objects Database could be applied in order to perform process execution.

Furthermore, other processes could be implemented and optionally called by a more sophisticated web application encapsulating the WPS. In order to realize complex processing workflows, web services could be orchestrated to service chains. A WFS could be employed as geodata access service, providing the necessary input data for a WPS that delivers the processing result to a WMS to prepare a map based visualisation. An orchestration engine as central service chaining unit and a rule engine to enforce specific workflow rules, organised in a Web Service Orchestration (WSO[1]) framework, could enhance an ordinary OWS based web application with additional functionality and flexibility. Such a web application would require a more sophisticated client application with a map front end as well in order to be able to display a map based processing result.

Another possibility to continue this work could be the consideration of performance aspects. This includes questions like process distribution (number of processes for WPS, number of WPS per server), application of grid computing and other performance issues like caching, data granularity or asynchronous communication as described in (Kiehle et al., 2006b).

Also an important topic are current investigations into the semantic geospatial web. Semantic interoperability is a crucial extension of syntactic interoperability on the way to intelligent web services.

A more general future task is to support current SDI developments and other GIS interoperability initiatives (like GIPSIE[2] or Digital Earth) to promote the development of OpenGIS in general and geoprocessing services in particular.

1 Kiehle et al., 2007
2 GIS Interoperability Project Stimulation the Industry in Europe

Bibliography

[1] Güting, Ralf Hartmut; Schneider, Markus. *Moving objects databases,* 1st ed.; Data Management Systems; Morgan Kaufmann; Elsevier Morgan Kaufmann: Amsterdam, 2005, Vol. 1.

[2] Heier, Christian; Kiehle, Christian. Automatisierte Liegenschaftsauskunft mittels OGC Web Processing Service. *GIS - Zeitschrift für Geoinformatik* **2006**, *7/2006* (7) 12–16.

[3] *Mastering Object-Oriented Analysis and Design with UML,* IBM Rational University, 2003.

[4] Kiehle, Christian. Business logic for geoprocessing of distributed geodata. *Computers and Geosciences* **2006**, *32* (10) 1746–1757.

[5] Kiehle, Christian; Heier, Christian. Entwicklung eines Geoprocessing Moduls zur Informationsgewinnung aus distributiven Geodaten. *GIS - Zeitschrift für Geoinformatik* **2004**, *6/2004* (6) 26–31.

[6] Kiehle, Christian; Heier, Christian. Standardisierte Geodatenverarbeitung im Internet - der OGC Web Processing Service. *GIS - Zeitschrift für Geoinformatik* **2005**, *6/2005* (6) 39–43.

[7] Kiehle, Christian; Heier, Christian; Greve, Klaus. Standardized Geoprocessing. Taking Spatial Data Infrastructures one Step Furtherhttp://www.agile2006.hu/papers/a273.pdf.

[8] Kiehle, Christian; Heier, Christian; Greve, Klaus. Requirements for Next Generation Spatial Data Infrastructures – Standardized Web Based Geoprocessing and Web Service Orchestration. *Transactions in GIS* **2007** (11(6)) 819–834.

[9] Kiehle, Christian; Scholten, Marius; Klamma, Ralf. Evaluating Performance in Spatial Data Infrastructures for Geoprocessing. *IEEE Internet Computing* **2006**, *10* (5) 34–41.

[10] Lohmar, Frank. *MDA: QVT - Der neue OMG Standard.,* Borland.

[11] Miler, H. J., Han, J., Miller, H. J., Eds.: *Geographic Data Mining and Knowledge Discovery,* 1st ed.; Research monographs in geographic information systems; Taylor & Francis: London, 2001.

[12] OGC Implementation Specification; OGC 05-007r7. *Web Processing Service;* Open Geospatial Consortium Inc., 08.06.2007.

[13] Open GIS Specification; 06-121r3. *OGC Web Service Common Specification;* Open Geospatial Consortium Inc., 09.02.2007.

[14] OGC Reference Model; OGC 03-040. *OGC Reference Model;* Open

Geospatial Consortium Inc., 16.09.2003.

[15] Parent, Christine; Spaccapietra, Stefano; Zimanyi, Esteban; Zimányi, Esteban. *Conceptual modeling for traditional and spatio-temporal applications: The MADS approach,* 1st ed.; Springer: Berlin, 2006.

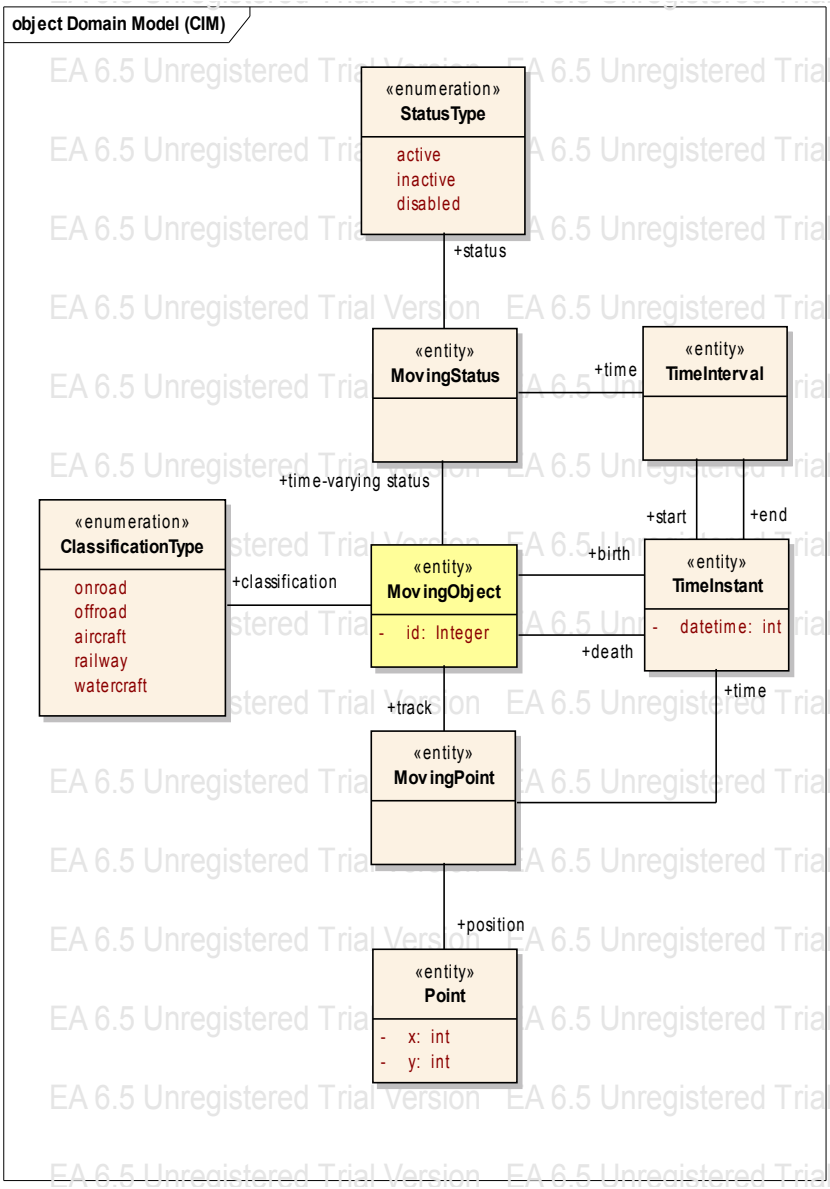
[16] Roddick, John F.; Lees, Brian G. Paradigms for spatial and spatio-temporal data mining. In *Geographic Data Mining and Knowledge Discovery.* Miler, H. J., Han, J., Miller, H. J., Eds. Research monographs in geographic information systems Taylor & Francis: London, 2001.

[17] Stollberg, Beate; Lutz, Michael; Ostländer, Nicole; Bernard, Lars. Geoprozessierung in Geodateninfrastrukturen - Aufgaben für die nächste Generation. *GIS - Zeitschrift für Geoinformatik* **2007**, *4/2007* (4) 22–27.

[18] Straub, Florian; Donaubauer, Andreas; Löwis of Menar, Olaf von. Webbasierte Verschneidung verteilter Geodaten für die forstliche Standortserkundung. *GeoBIT* **2004**, *11/2004* (11) 30–31.

UML Model

Domain Model



Domain Model (CIM)
Package: Business Domain Model

Domain Model (CIM)::ClassificationType
Class: Possible values a classification type can adopt.

Domain Model (CIM)::ClassificationType Attributes

Attribute	Type
onroad	private : *int*
offroad	private : *int*
aircraft	private : *int*
railway	private : *int*
watercraft	private : *int*

Domain Model (CIM)::MovingObject

Class: Object, changing its position over time

Domain Model (CIM)::MovingObject Attributes

Attribute	Type
id	private : *Integer*

Domain Model (CIM)::MovingObject Methods

Method	Type
crosses ()	public: *void*
enter ()	public: *void*
leave ()	public: *void*
inside ()	public: *void*
meet ()	public: *void*
disjoint ()	public: *void*
verifyClassification ()	public: *void*
intersects ()	public: *void*
finalPosition ()	public: *void*
initialPosition ()	public: *void*
positionAtTime ()	public: *void*
addToGroup ()	public: *void*
direction ()	public: *void*
changeOfDirection ()	public: *void*
speed ()	public: *void*

Domain Model (CIM)::MovingPoint

Class: If the moving object has no spatial extension, its track consists of a number of points.

Domain Model (CIM)::MovingStatus

Class: Status of the moving object which is also time-varying.

Domain Model (CIM)::Point

Class: Position of a moving point expressed using coordinates

Domain Model (CIM)::Point Attributes

Attribute	Type
x	private : *int*
y	private : *int*

Domain Model (CIM)::Point Methods

Method	Type
distance ()	public: *void*

convexHull ()	public: *void*

Domain Model (CIM)::StatusType

Class: Possible values a status type can adopt.

Domain Model (CIM)::StatusType Attributes

Attribute	Type
active	private : *int*
inactive	private : *int*
disabled	private : *int*

Domain Model (CIM)::TimeInstant

Class: Point in time for which a moving point is valid respectively birth and death of a moving object indicating its lifetime. Time instants also indicate the start and end of a time interval.

Domain Model (CIM)::TimeInstant Attributes

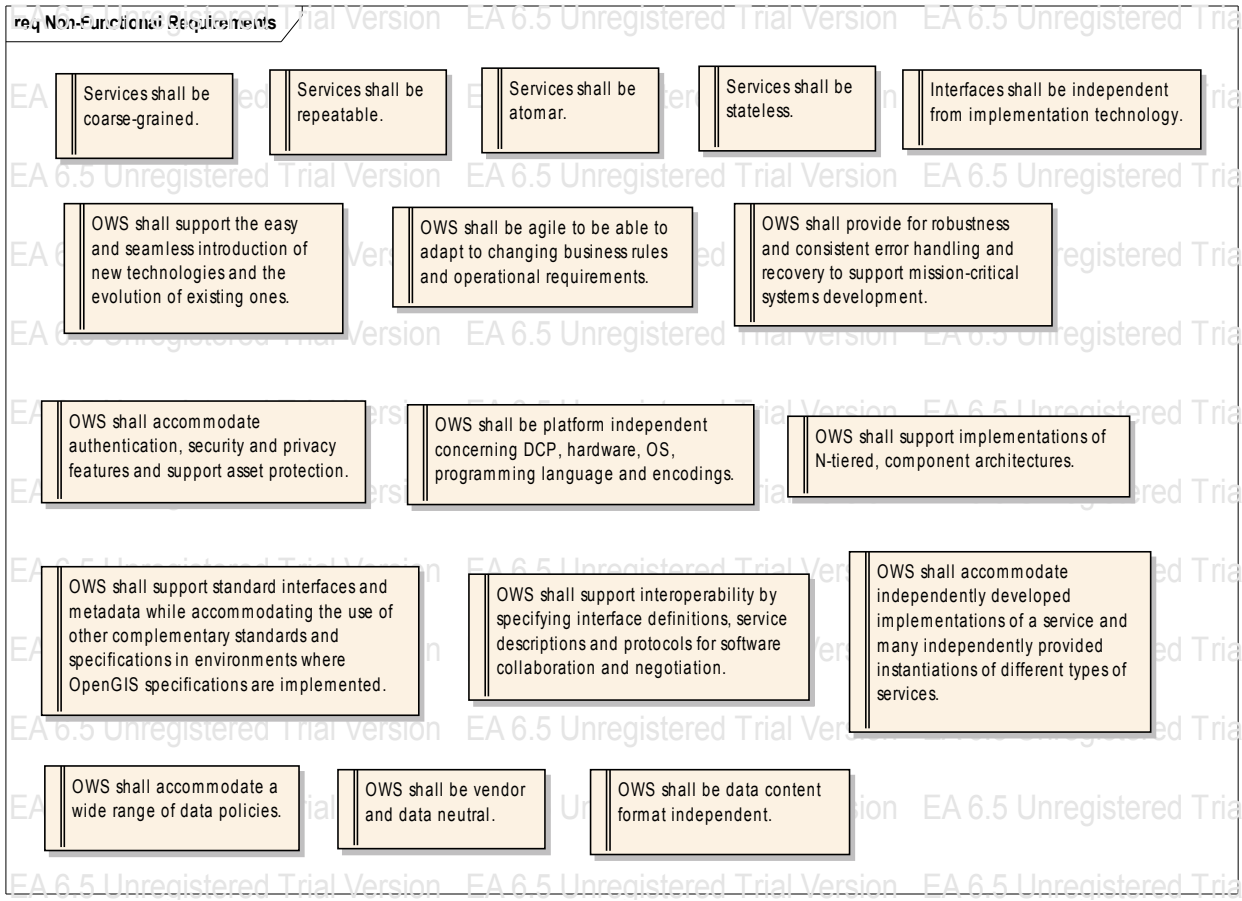
Attribute	Type
datetime	private : *int*

Domain Model (CIM)::TimeInterval

Class: Timespan for which a moving status is valid.

Requirements Model

Non-Functional Requirements



Non-Functional Requirements

Package: Non-functional requirements to services in general and OWS in particular

Interfaces shall be independent from implementation technology.

OWS shall accommodate a wide range of data policies.

OWS shall accommodate authentication, security and privacy features and support asset protection.

OWS shall accommodate independently developed implementations of a service and many independently provided instantiations of different types of services.

OWS shall be agile to be able to adapt to changing business rules and operational requirements.

OWS shall be data content format independent.

OWS shall be platform independent concerning DCP, hardware, OS, programming language and encodings.

OWS shall be vendor and data neutral.

OWS shall provide for robustness and consistent error handling and recovery to support mission-critical systems development.

OWS shall support implementations of N-tiered, component architectures.

OWS shall support interoperability by specifying interface definitions, service descriptions and protocols for software collaboration and negotiation.

OWS shall support standard interfaces and metadata while accommodating the use of other complementary standards and specifications in environments where OpenGIS specifications are implemented.

OWS shall support the easy and seamless introduction of new technologies and the evolution of existing ones.

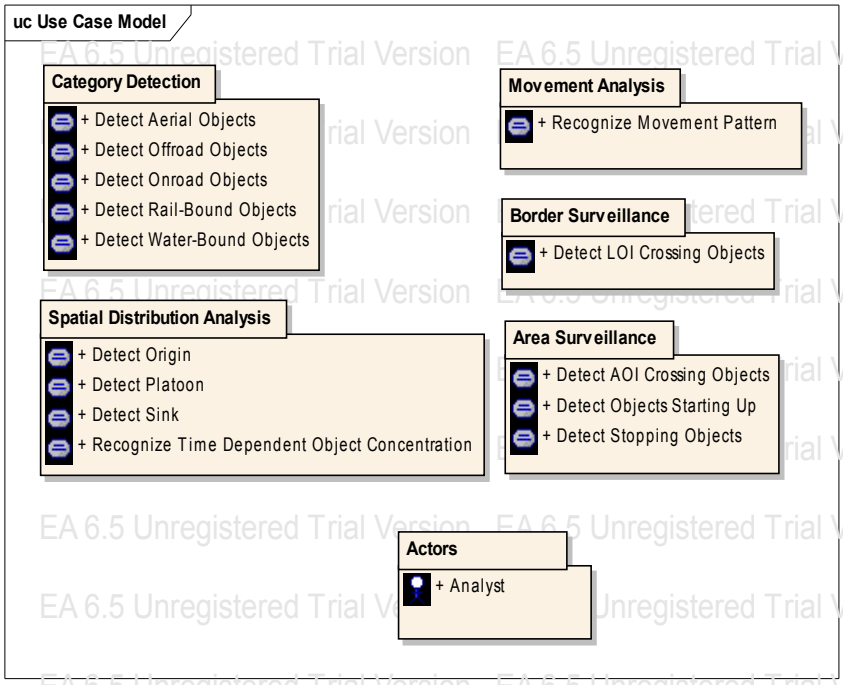
Services shall be atomar.

Services shall be coarse-grained.

Services shall be repeatable.

Services shall be stateless.

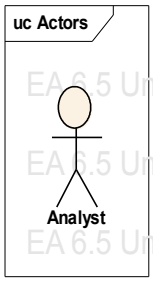
Use Case Model



Use Case Model

Package: Functional Requirements, expressed as Use Cases

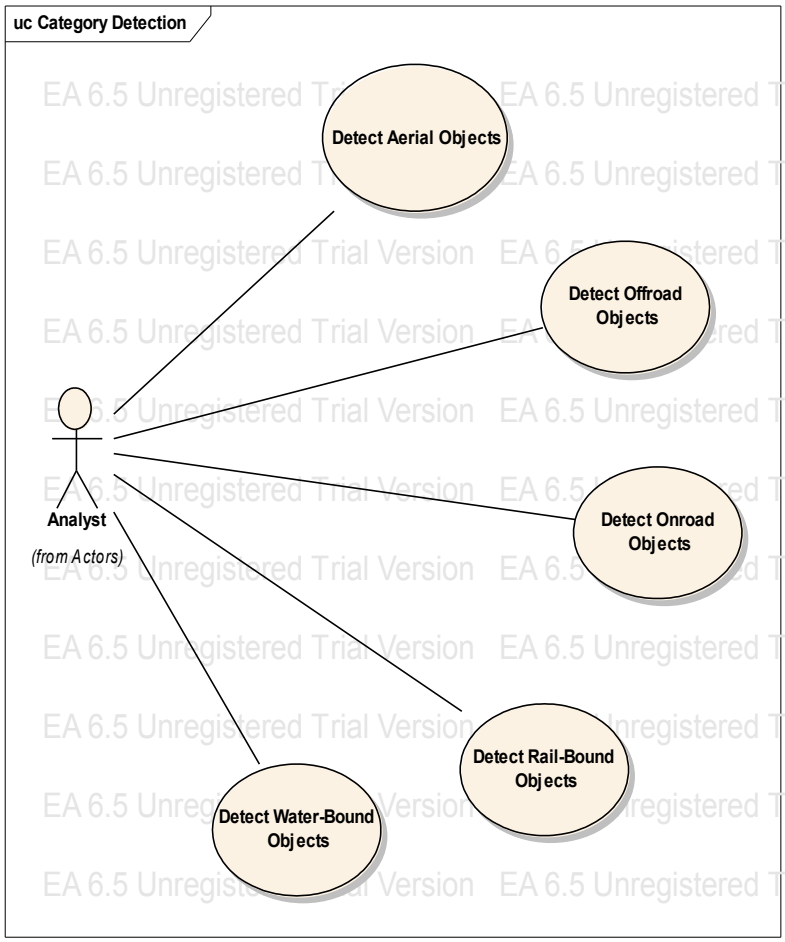
Actors



Actors

Package: External environment, user how interacts with the system

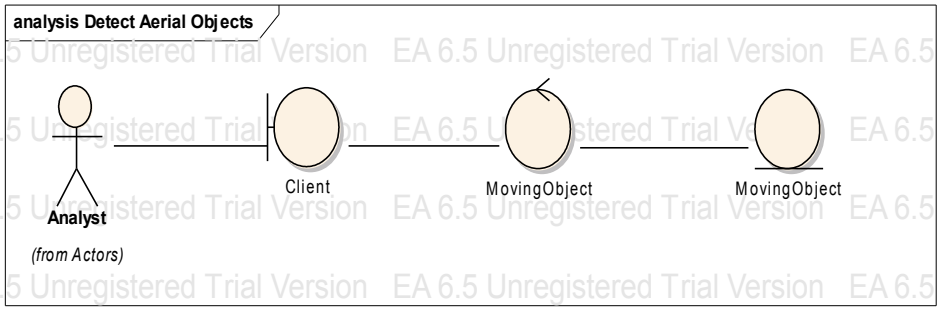
Category Detection

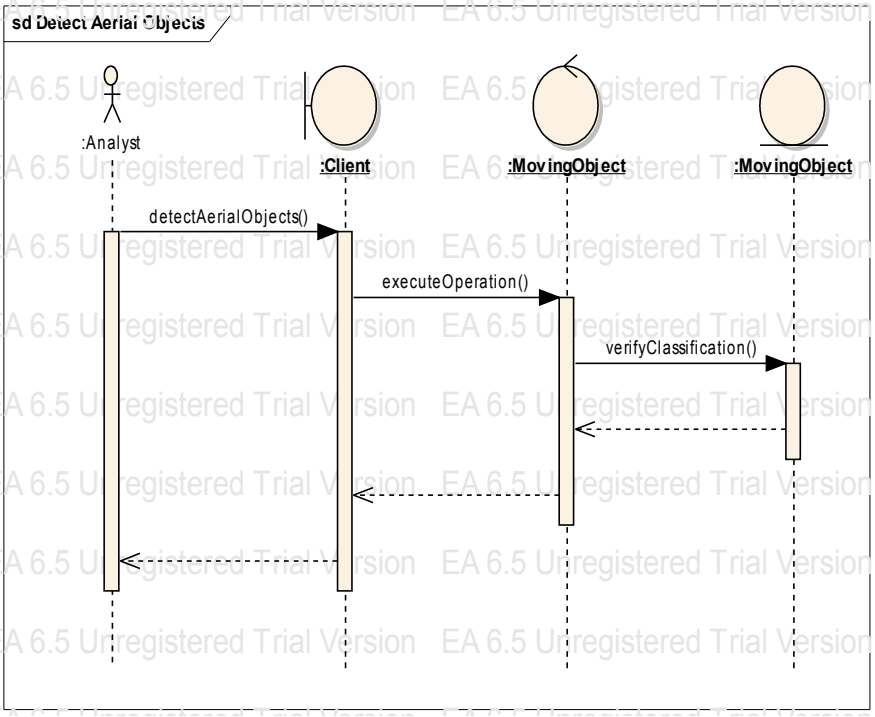


Category Detection

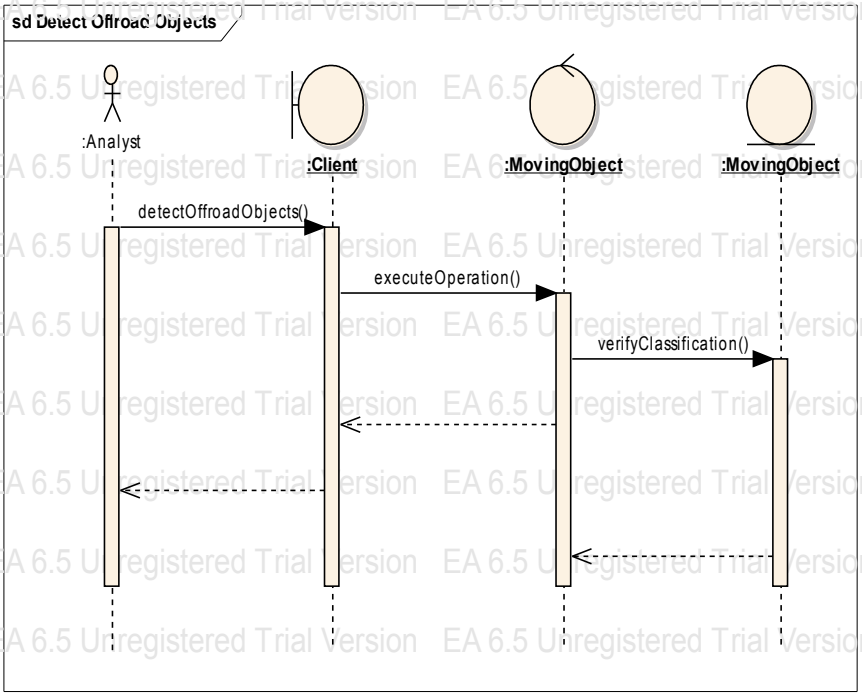
Package: Use Case package

Detect Aerial Objects

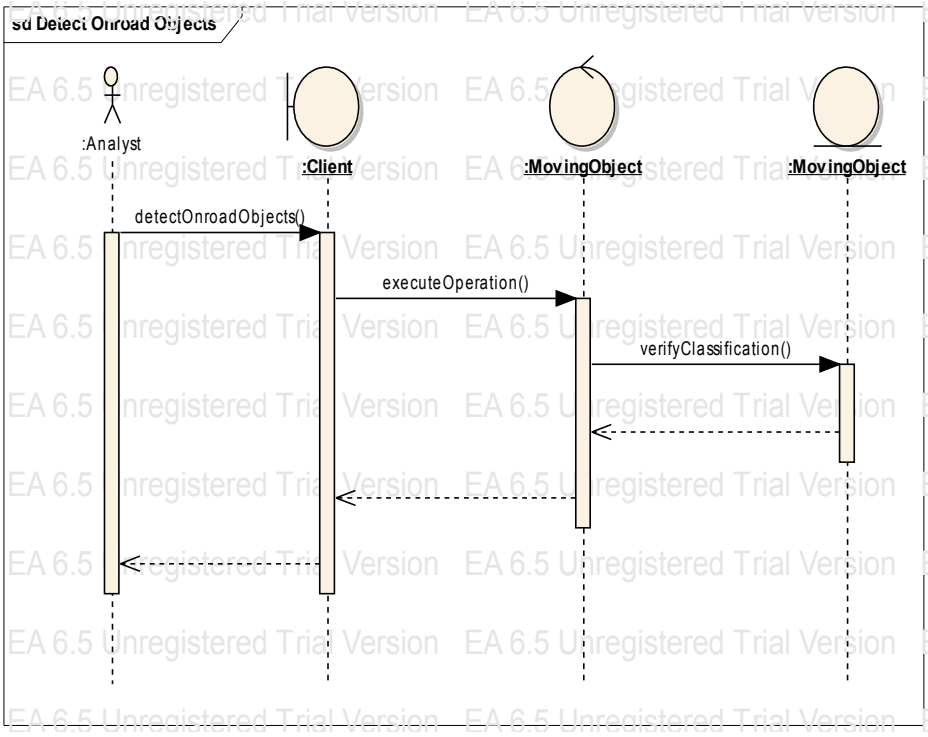




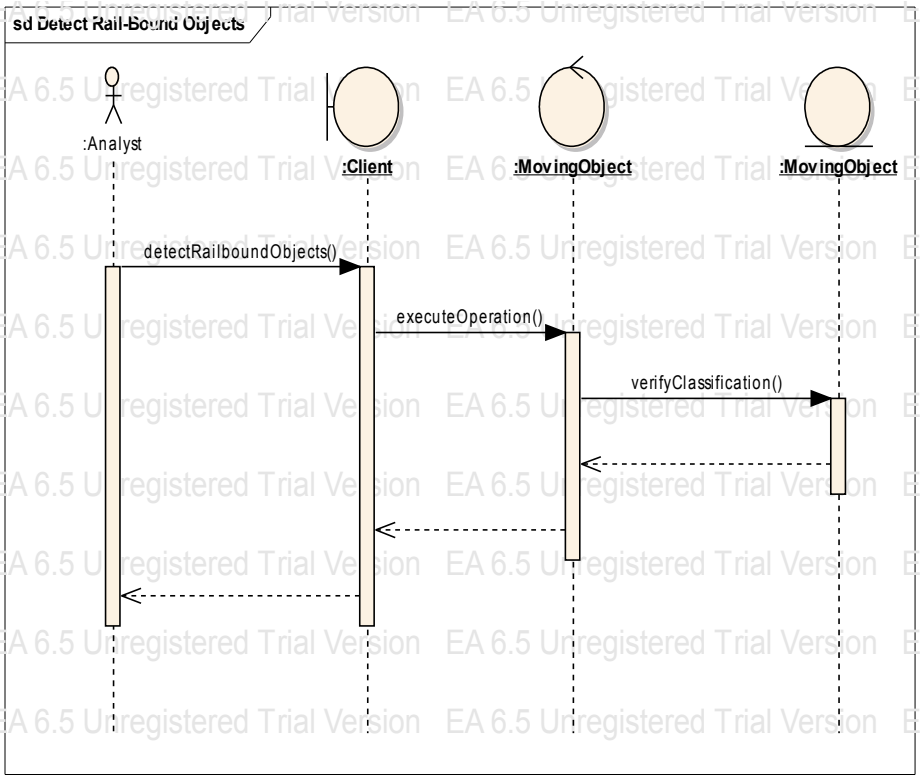
Detect Offroad Objects



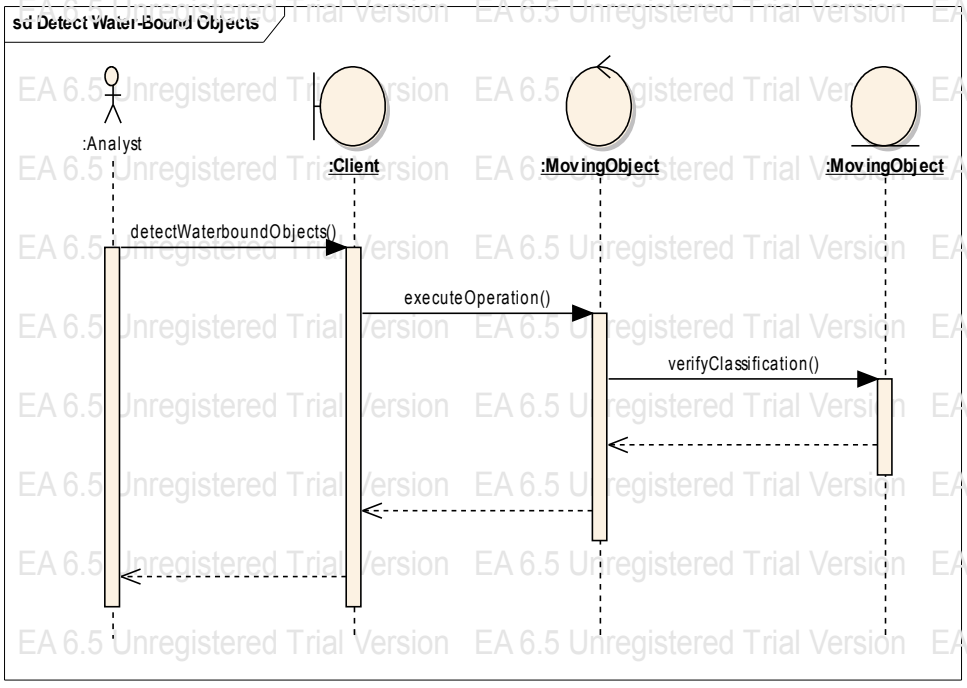
Detect Onroad Objects



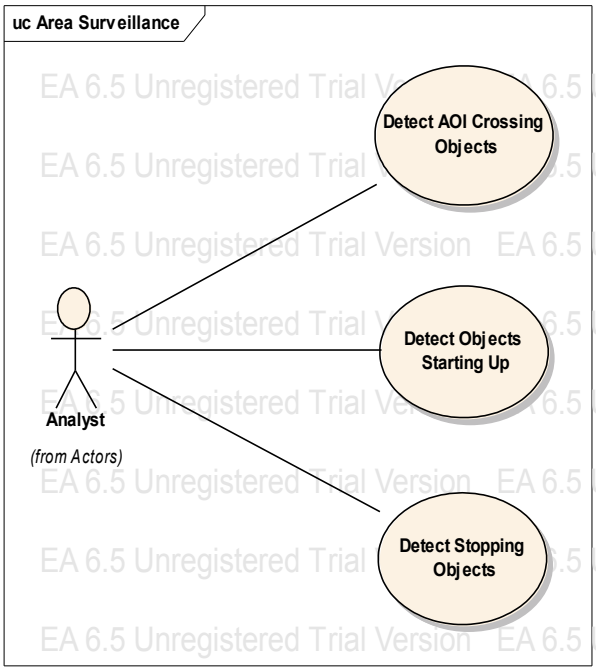
Detect Rail-Bound Objects



Detect Water-Bound Objects



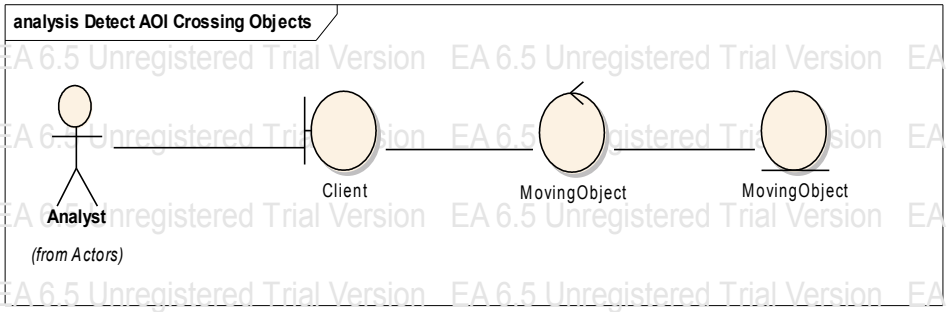
Area Surveillance

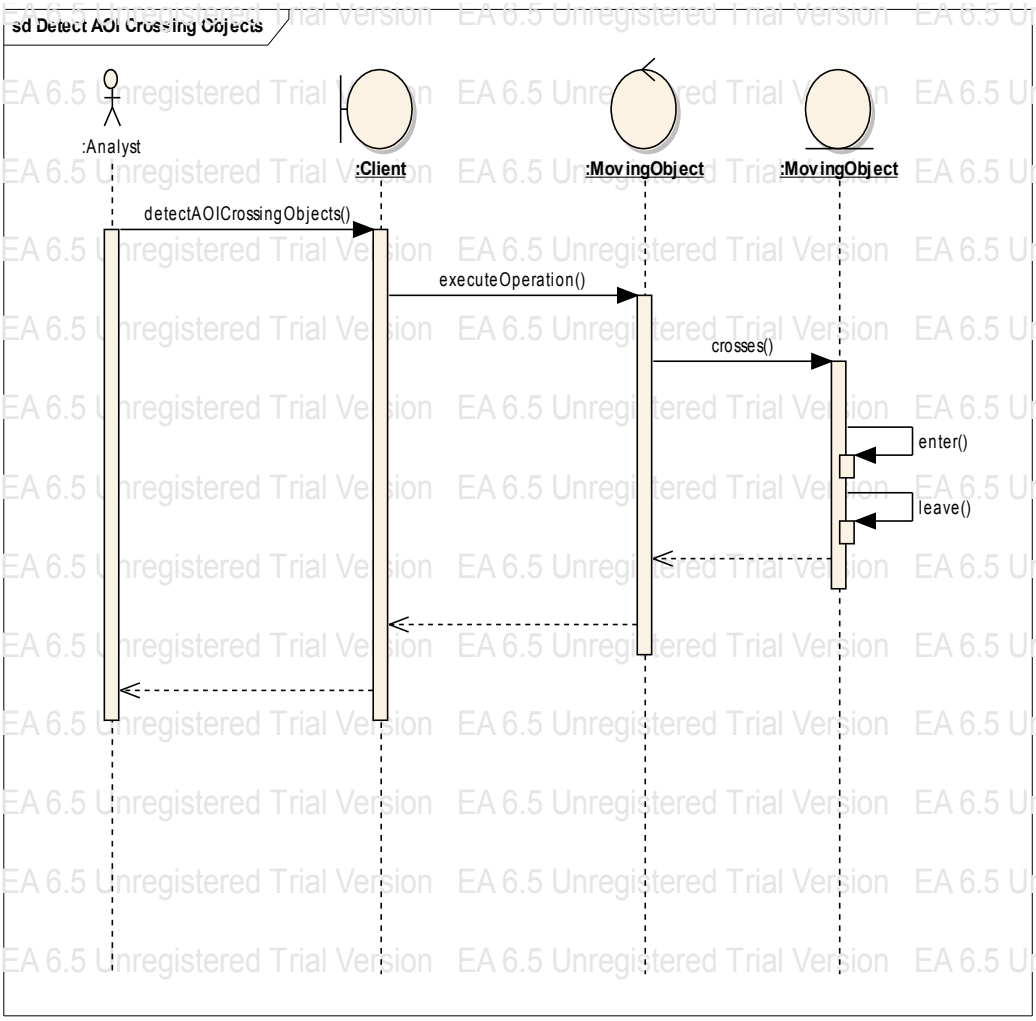


Area Surveillance

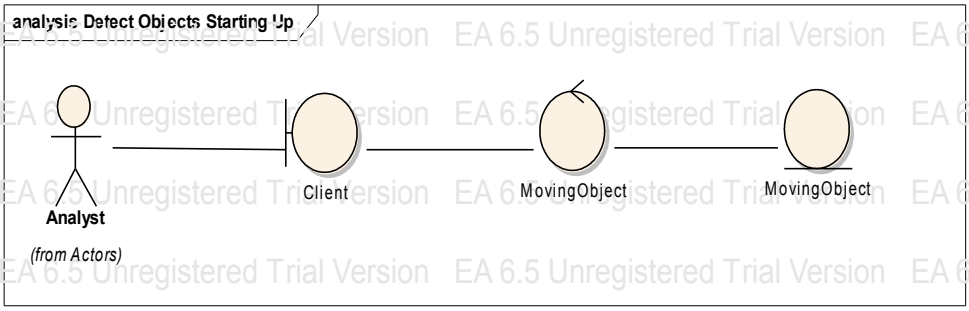
Package: Use Case package

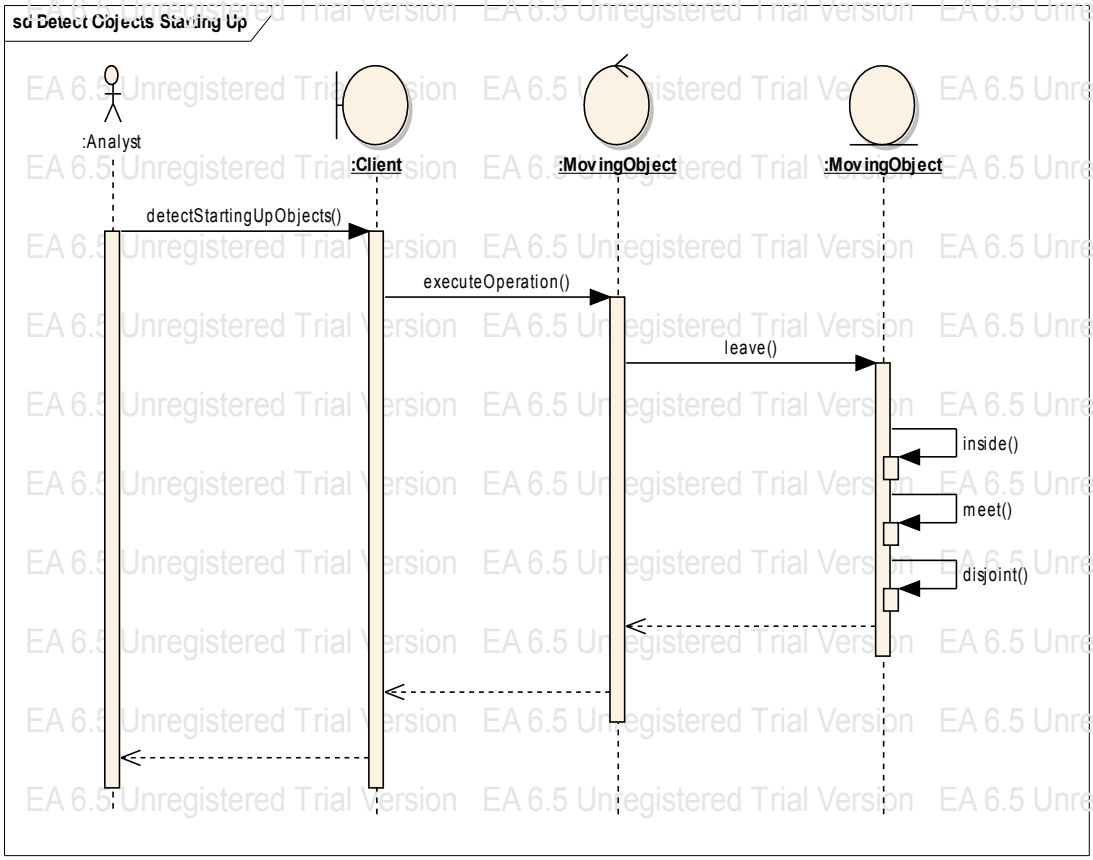
Detect AOI Crossing Objects



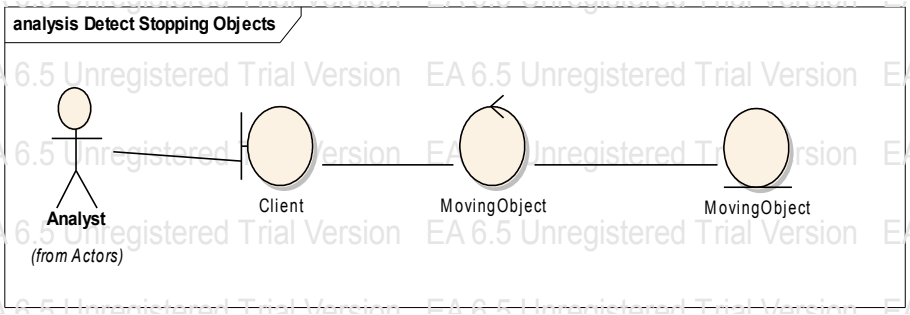


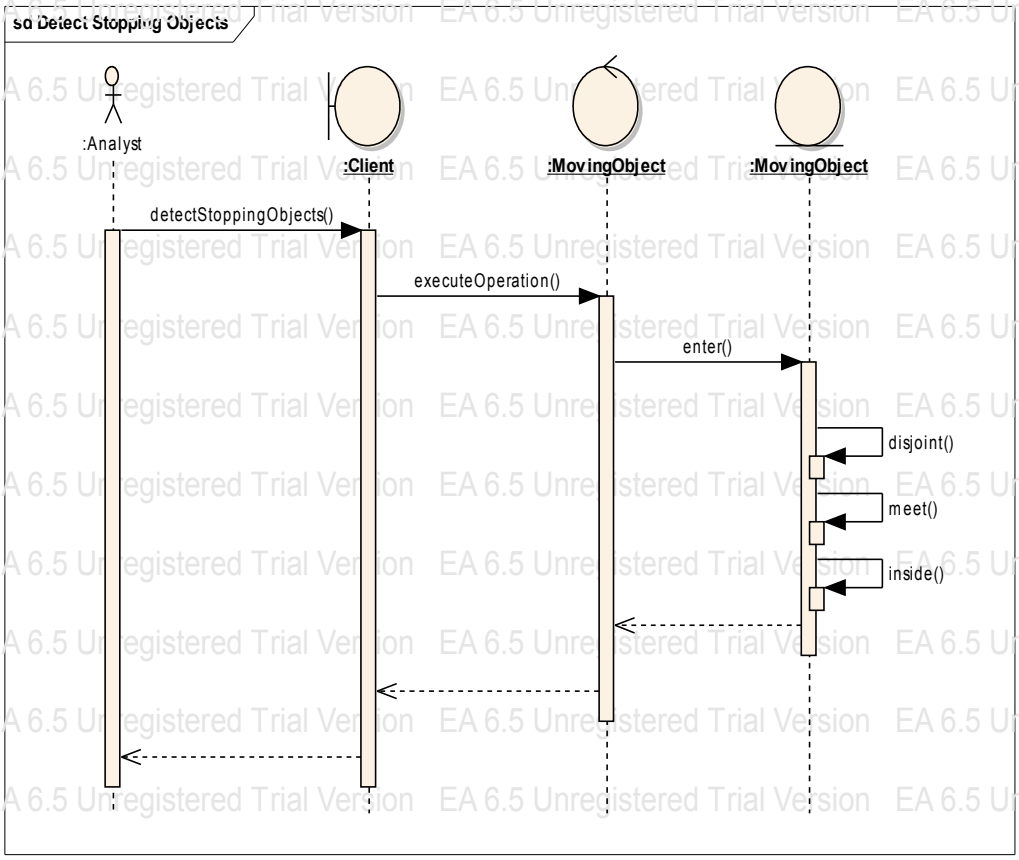
Detect Objects Starting Up



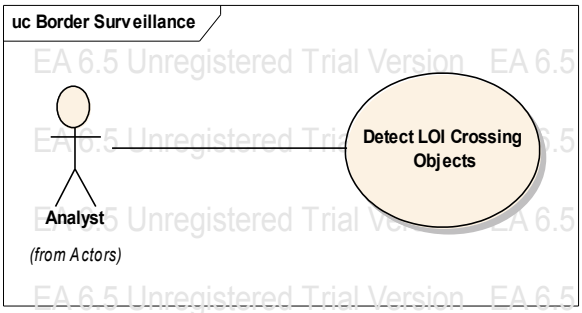


Detect Stopping Objects





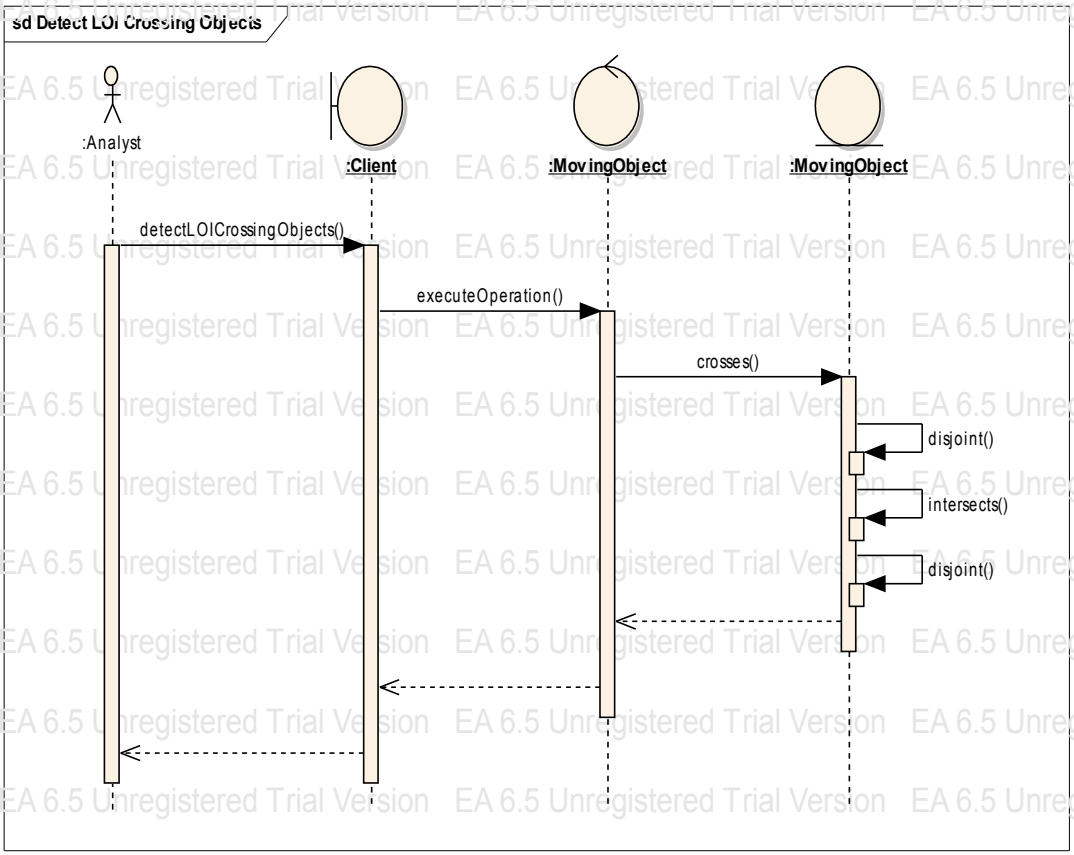
Border Surveillance



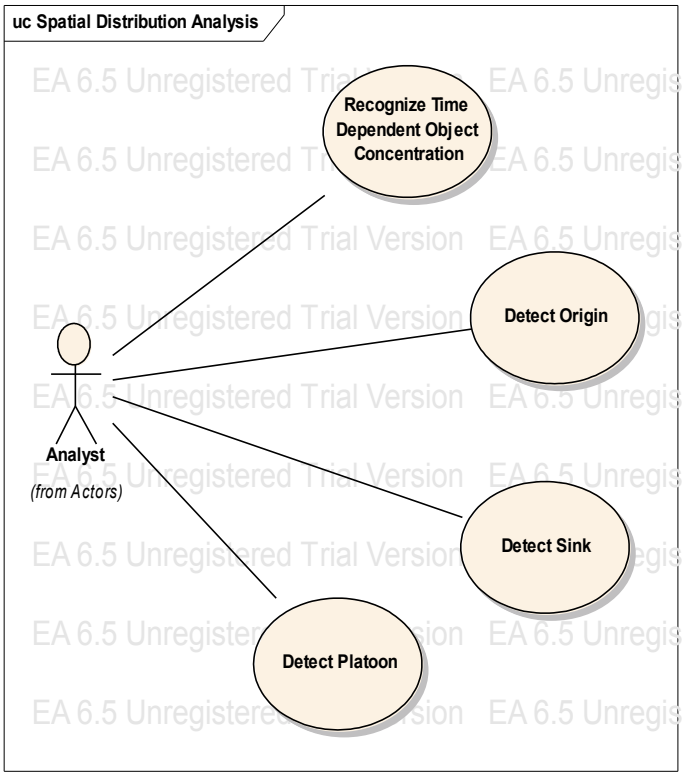
Border Surveillance

Package: Use Case package

Detect LOI Crossing Objects



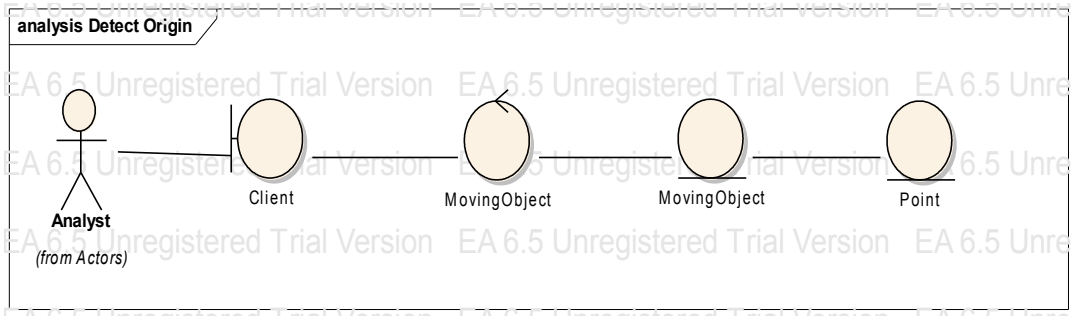
Spatial Distribution Analysis

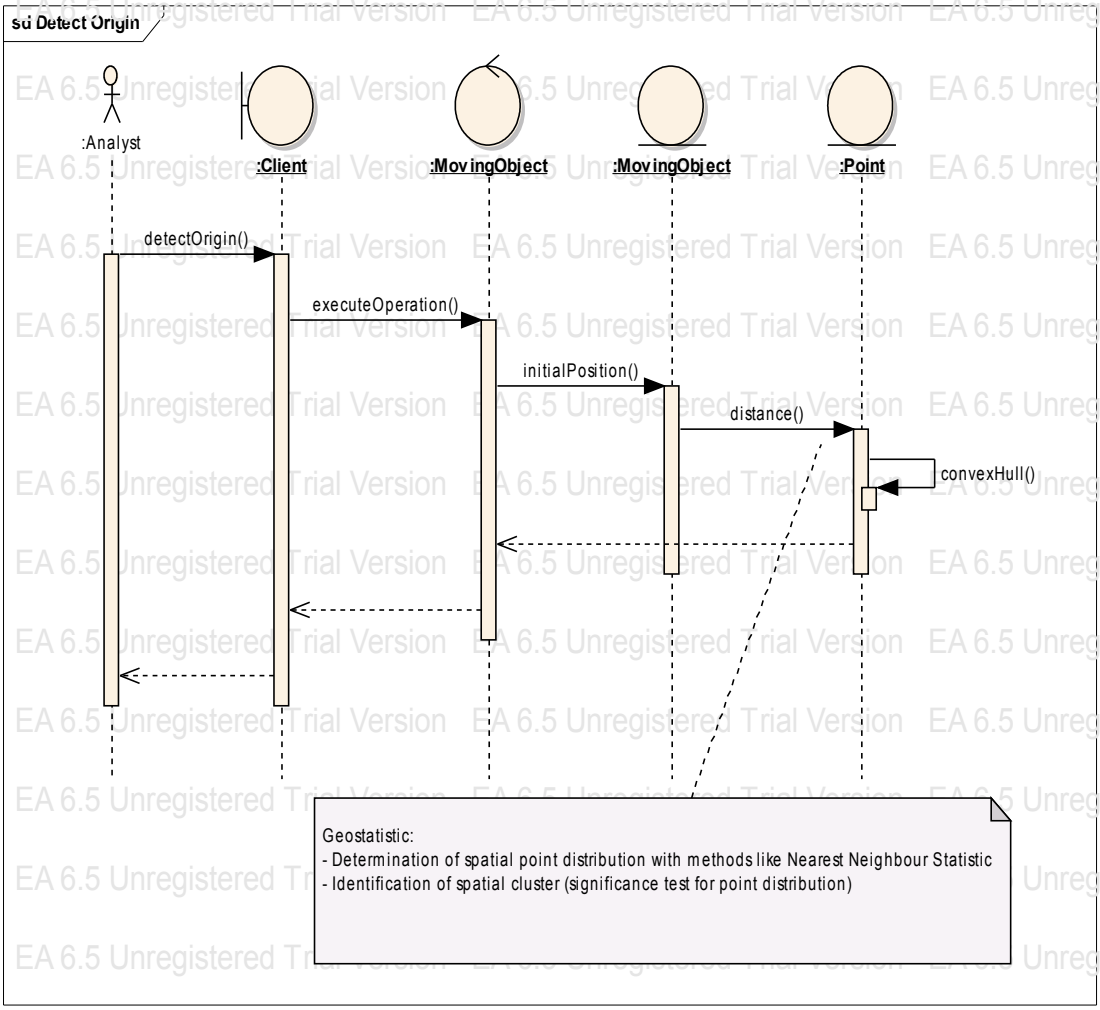


Spatial Distribution Analysis

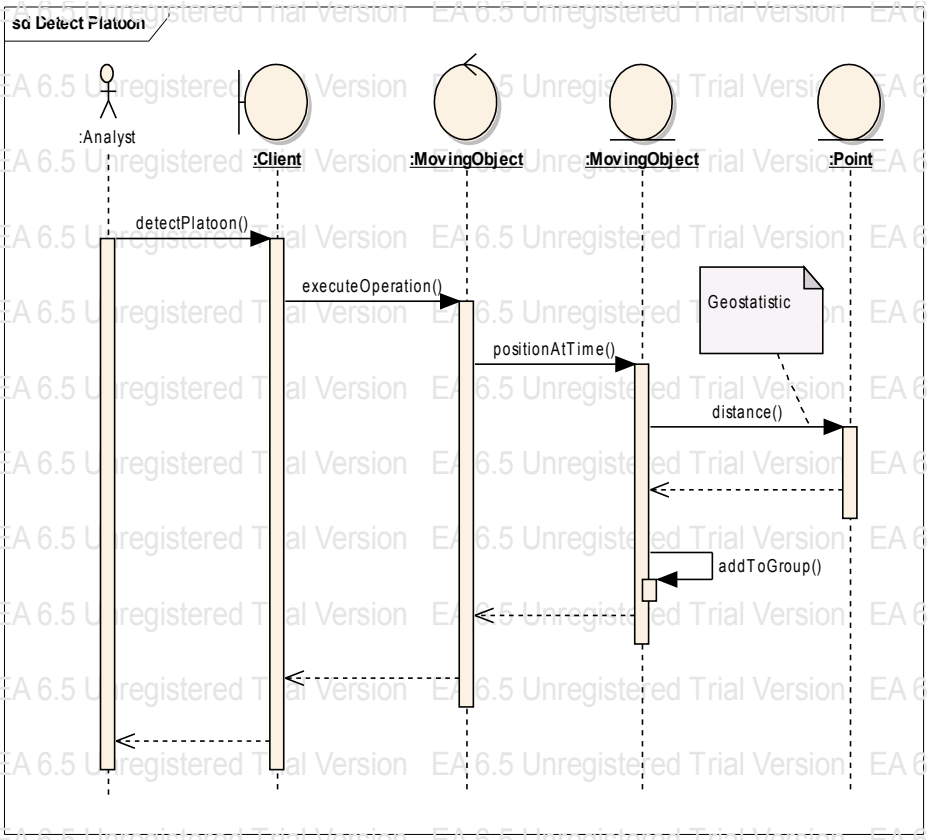
Package: Use Case package, usage of geostatistic, Point Pattern Analysis

Detect Origin

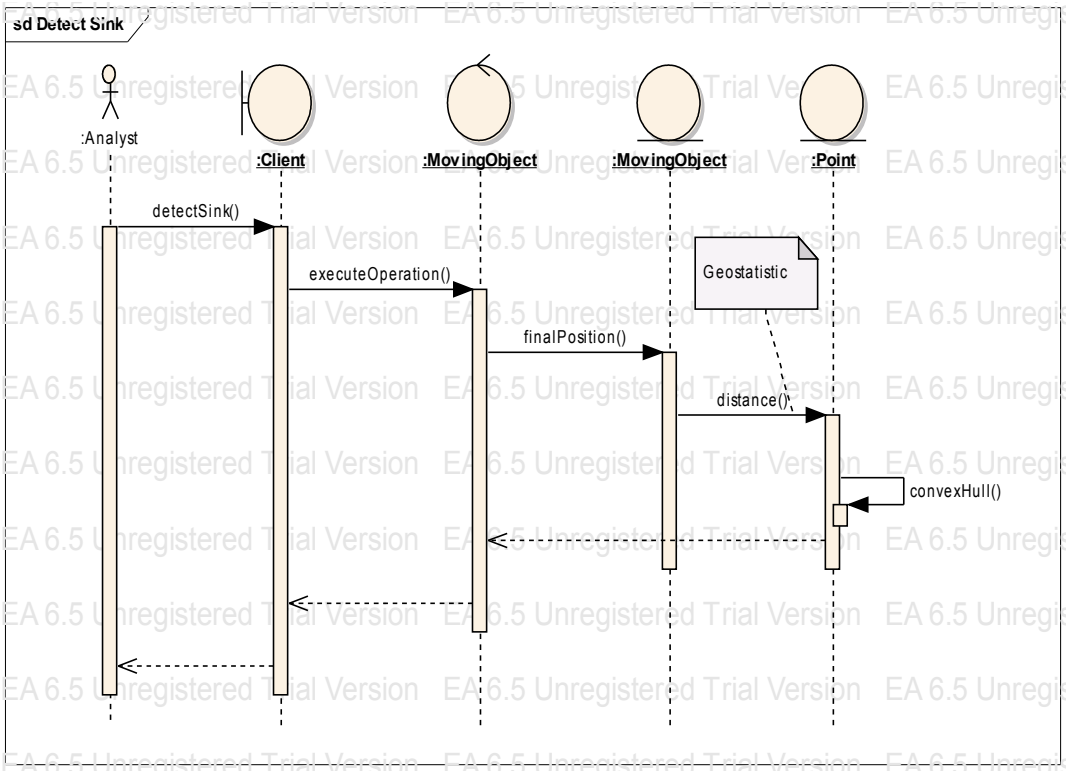




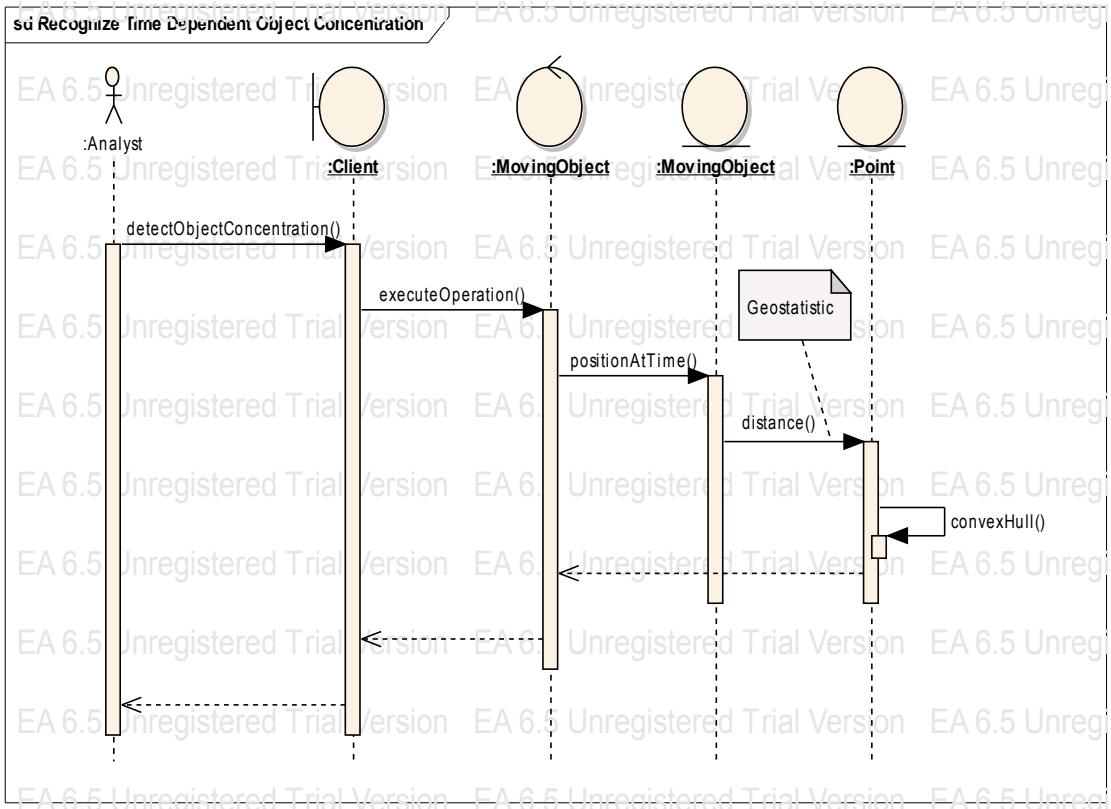
Detect Platoon



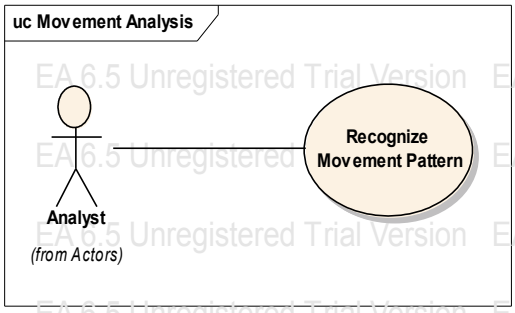
Detect Sink



Recognize Time Dependent Object Concentration



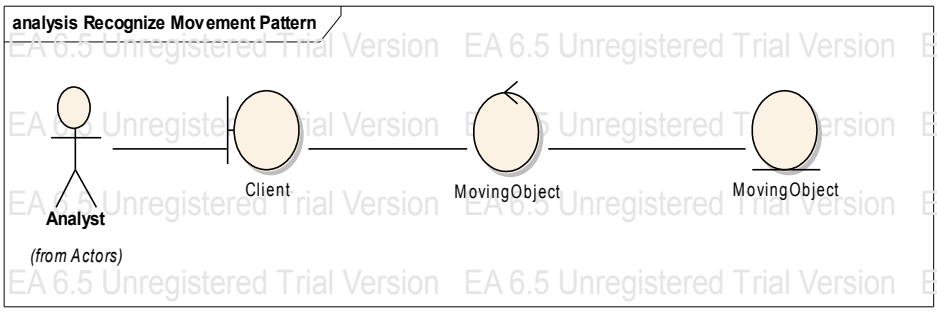
Movement Analysis

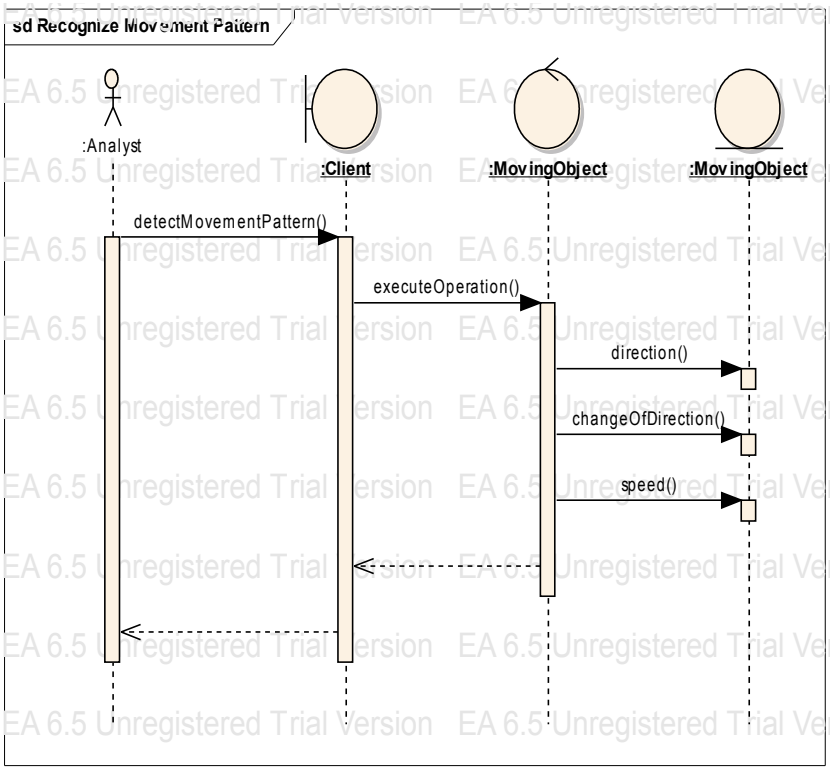


Movement Analysis

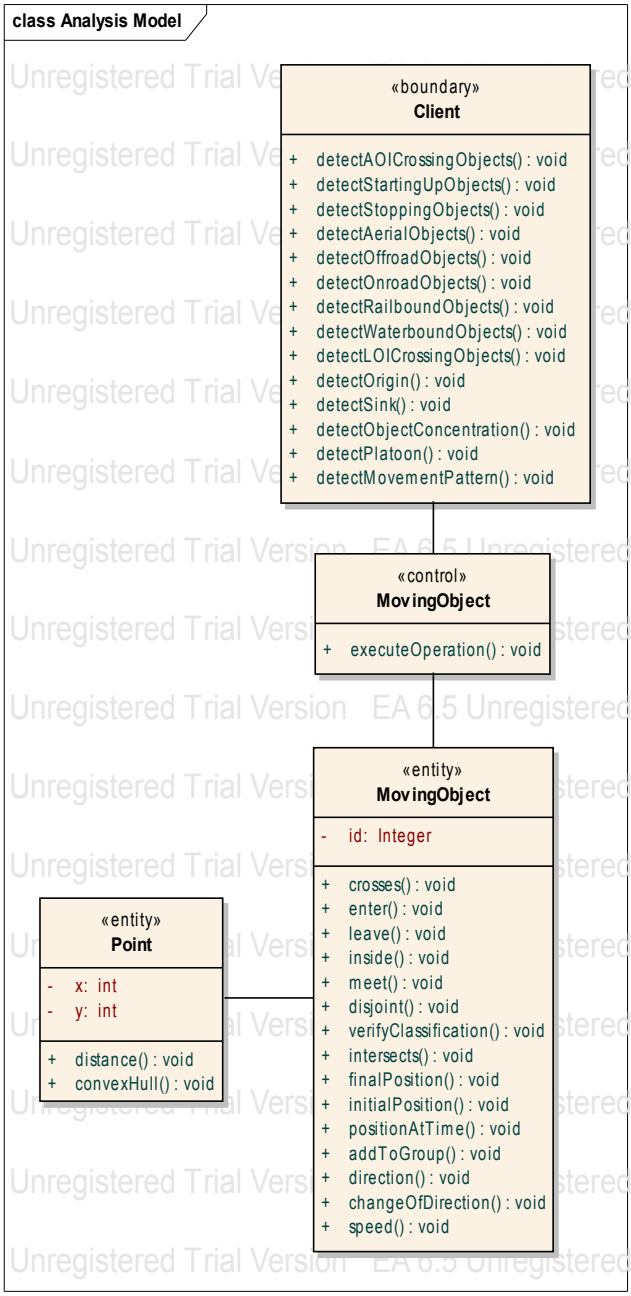
Package: Use Case package

Recognize Movement Pattern





Analysis Model



Analysis Model::Client

Class: User Interface, presentation layer

Analysis Model::Client Methods

Method	Type
detectAOICrossingObjects ()	public: *void*
detectStartingUpObjects ()	public: *void*
detectStoppingObjects ()	public: *void*
detectAerialObjects ()	public: *void*
detectOffroadObjects ()	public: *void*

detectOnroadObjects ()	public: *void*
detectRailboundObjects ()	public: *void*
detectWaterboundObjects ()	public: *void*
detectLOICrossingObjects ()	public: *void*
detectOrigin ()	public: *void*
detectSink ()	public: *void*
detectObjectConcentration ()	public: *void*
detectPlatoon ()	public: *void*
detectMovementPattern ()	public: *void*

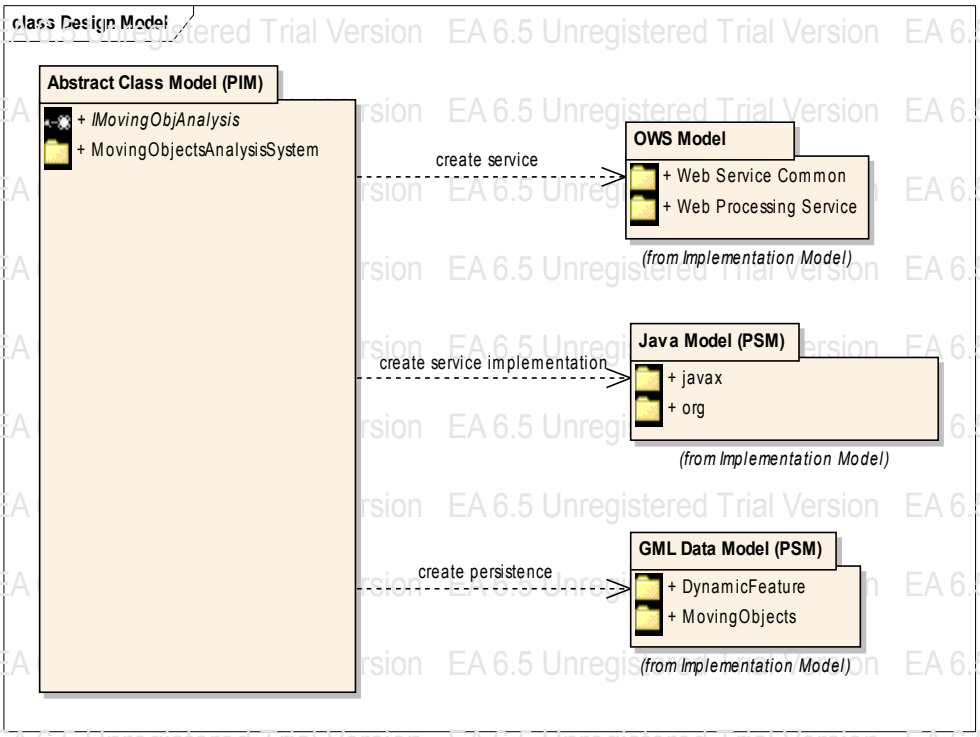
Analysis Model::MovingObject

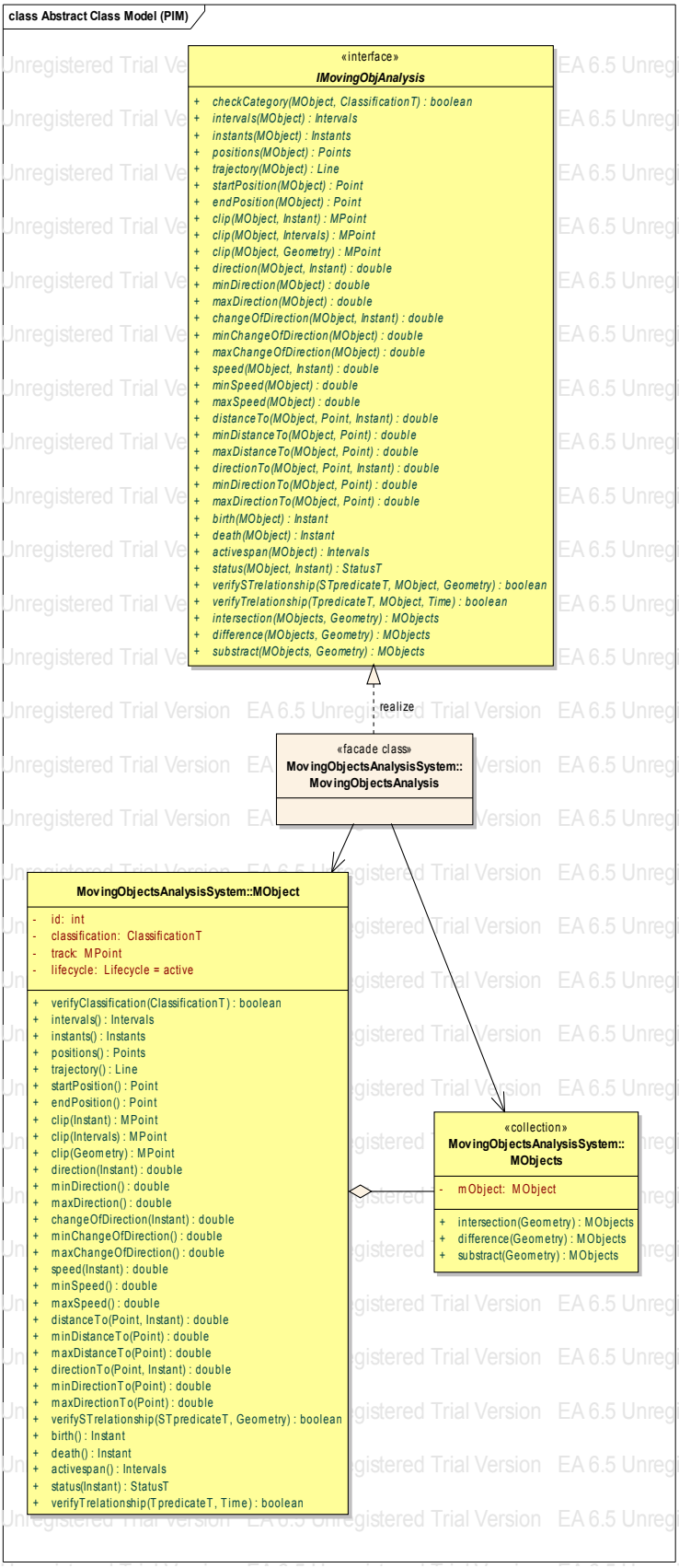
Class: Controller, decouples User Interface from business logic

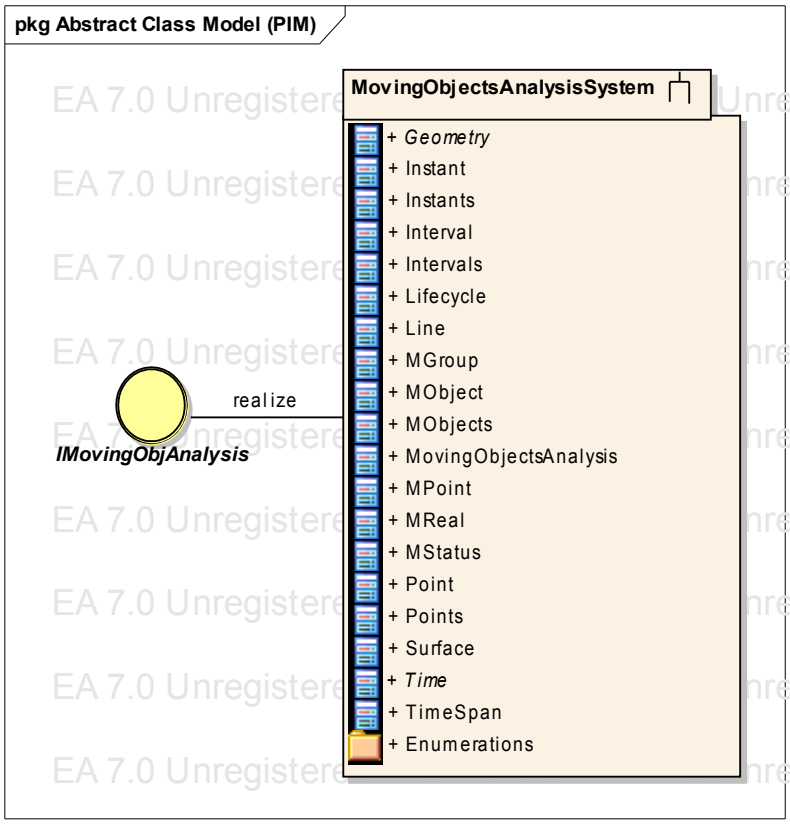
Analysis Model::MovingObject Methods

Method	Type
executeOperation ()	public: *void*

Abstract Class Model




class Abstract Class Model (PIM)
«interface»
IMovingObjAnalysis
+ checkCategory(MObject, ClassificationT) : boolean
+ intervals(MObject) : Intervals
+ instants(MObject) : Instants
+ positions(MObject) : Points
+ trajectory(MObject) : Line
+ startPosition(MObject) : Point
+ endPosition(MObject) : Point
+ clip(MObject, Instant) : MPoint
+ clip(MObject, Intervals) : MPoint
+ clip(MObject, Geometry) : MPoint
+ direction(MObject, Instant) : double
+ minDirection(MObject) : double
+ maxDirection(MObject) : double
+ changeOfDirection(MObject, Instant) : double
+ minChangeOfDirection(MObject) : double
+ maxChangeOfDirection(MObject) : double
+ speed(MObject, Instant) : double
+ minSpeed(MObject) : double
+ maxSpeed(MObject) : double
+ distanceTo(MObject, Point, Instant) : double
+ minDistanceTo(MObject, Point) : double
+ maxDistanceTo(MObject, Point) : double
+ directionTo(MObject, Point, Instant) : double
+ minDirectionTo(MObject, Point) : double
+ maxDirectionTo(MObject, Point) : double
+ birth(MObject) : Instant
+ death(MObject) : Instant
+ activespan(MObject) : Intervals
+ status(MObject, Instant) : StatusT
+ verifySTrelationship(STpredicateT, MObject, Geometry) : boolean
+ verifyTrelationship(TpredicateT, MObject, Time) : boolean
+ intersection(MObjects, Geometry) : MObjects
+ difference(MObjects, Geometry) : MObjects
+ substract(MObjects, Geometry) : MObjects
realize
«facade class»
MovingObjectsAnalysisSystem::
MovingObjectsAnalysis
MovingObjectsAnalysisSystem::MObject
- id: int
- classification: ClassificationT
- track: MPoint
- lifecycle: Lifecycle = active
+ verifyClassification(ClassificationT) : boolean
+ intervals() : Intervals
+ instants() : Instants
+ positions() : Points
+ trajectory() : Line
+ startPosition() : Point
+ endPosition() : Point
+ clip(Instant) : MPoint
+ clip(Intervals) : MPoint
+ clip(Geometry) : MPoint
+ direction(Instant) : double
+ minDirection() : double
+ maxDirection() : double
+ changeOfDirection(Instant) : double
+ minChangeOfDirection() : double
+ maxChangeOfDirection() : double
+ speed(Instant) : double
+ minSpeed() : double
+ maxSpeed() : double
+ distanceTo(Point, Instant) : double
+ minDistanceTo(Point) : double
+ maxDistanceTo(Point) : double
+ directionTo(Point, Instant) : double
+ minDirectionTo(Point) : double
+ maxDirectionTo(Point) : double
+ verifySTrelationship(STpredicateT, Geometry) : boolean
+ birth() : Instant
+ death() : Instant
+ activespan() : Intervals
+ status(Instant) : StatusT
+ verifyTrelationship(TpredicateT, Time) : boolean
«collection»
MovingObjectsAnalysisSystem::
MObjects
- mObject: MObject
+ intersection(Geometry) : MObjects
+ difference(Geometry) : MObjects
+ substract(Geometry) : MObjects



Abstract Class Model (PIM)::IMovingObjAnalysis

Interface: services, offered by the MovingObjectsAnalysisSystem

Abstract Class Model (PIM)::IMovingObjAnalysis Interfaces

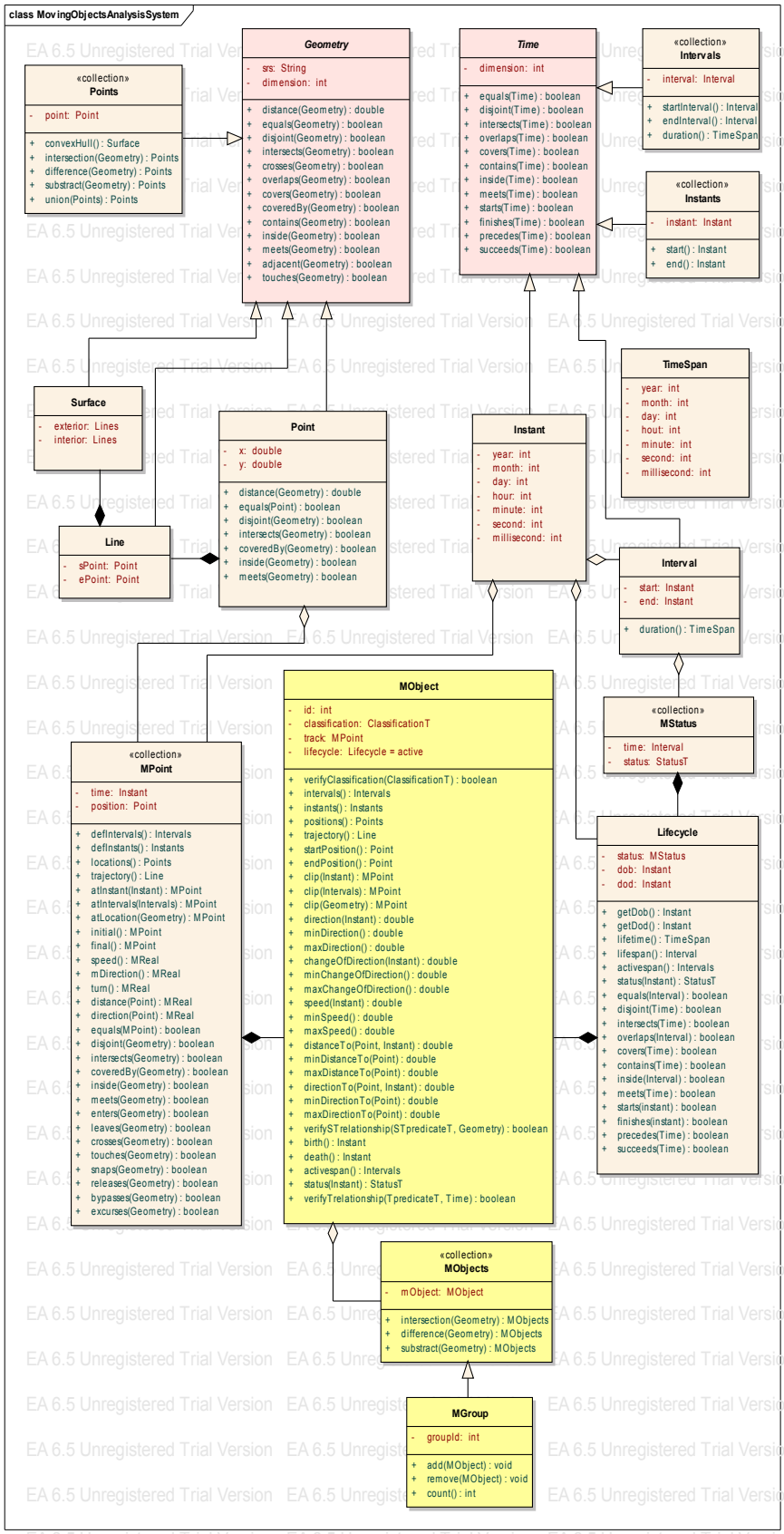
Method	Type	Notes
checkCategory (*MObject, ClassificationT*)	public: *boolean*	param: m [MObject - in] param: c [ClassificationT - in] Testing whether the moving object complies with the specified classification.
intervals (*MObject*)	public: *Intervals*	param: m [MObject - in] Returns the set of time intervals between the single track points.
instants (*MObject*)	public: *Instants*	param: m [MObject - in] Returns the set of instants for which a track point is defined.
positions (*MObject*)	public: *Points*	param: m [MObject - in] Returns the set of track points for which an instant is defined when the moving object is projected into the plane.
trajectory (*MObject*)	public: *Line*	param: m [MObject - in] Returns the line when the moving object is projected into the plane.
startPosition (*MObject*)	public: *Point*	param: m [MObject - in] Returns the first track point of the moving object's lifespan.

endPosition (*MObject*)	public: *Point*	param: m [MObject - in] Returns the last track point of the moving object's lifespan.
clip (*MObject, Instant*)	public: *MPoint*	param: m [MObject - in] param: i [Instant - in] Returns the point/instant pair that is defined for the particular instant i.
clip (*MObject, Intervals*)	public: *MPoint*	param: m [MObject - in] param: i [Intervals - in] Returns the portion of the track that is defined for the particular intervals i.
clip (*MObject, Geometry*)	public: *MPoint*	param: m [MObject - in] param: g [Geometry - in] Returns the portion of the track that is defined for the Geometry g.
direction (*MObject, Instant*)	public: *double*	param: m [MObject - in] param: i [Instant - in] Returns the direction of the moving object at instant i, the angle between the x-axis and a tangent to the trajectory of the moving object.
minDirection (*MObject*)	public: *double*	param: m [MObject - in] Returns the minimum direction of the moving object during its lifespan.
maxDirection (*MObject*)	public: *double*	param: m [MObject - in] Returns the maximum direction of the moving object during its lifespan.
changeOfDirection (*MObject, Instant*)	public: *double*	param: m [MObject - in] param: i [Instant - in] Returns the change of direction of the moving object at instant i.
minChangeOfDirection (*MObject*)	public: *double*	param: m [MObject - in] Returns the minimum change of direction of the moving object during its lifespan.
maxChangeOfDirection (*MObject*)	public: *double*	param: m [MObject - in] Returns the maximum change of direction of the moving object during its lifespan.
speed (*MObject, Instant*)	public: *double*	param: m [MObject - in] param: i [Instant - in] Returns the speed of the moving object at instant i.
minSpeed (*MObject*)	public: *double*	param: m [MObject - in] Returns the minimum speed of the moving object during its lifespan.
maxSpeed (*MObject*)	public: *double*	param: m [MObject - in] Returns the maximum speed of the moving object during its lifespan.
distanceTo (*MObject,*	public: *double*	param: m [MObject - in]

Point, Instant)		param: p [Point - in] param: i [Instant - in] Returns the distance between the moving object and p at instant i.
minDistanceTo (*MObject, Point*)	public: *double*	param: m [MObject - in] param: p [Point - in] Returns the minimum distance between the moving object and p during the moving objects' lifespan.
maxDistanceTo (*MObject, Point*)	public: *double*	param: m [MObject - in] param: p [Point - in] Returns the maximum distance between the moving object and p during the moving objects' lifespan.
directionTo (*MObject, Point, Instant*)	public: *double*	param: m [MObject - in] param: p [Point - in] param: i [Instant - in] Returns the direction (angle of the line from the moving object to the second point, measured in degrees, relative to a horizontal line) between the moving point and p at instant i.
minDirectionTo (*MObject, Point*)	public: *double*	param: m [MObject - in] param: p [Point - in] Returns the minimum direction between the moving point and p during the moving objects' lifespan.
maxDirectionTo (*MObject, Point*)	public: *double*	param: m [MObject - in] param: p [Point - in] Returns the maximum direction between the moving point and p during the moving objects' lifespan.
birth (*MObject*)	public: *Instant*	param: m [MObject - in] Returns the instant at which the moving objects becomes active for the first time.
death (*MObject*)	public: *Instant*	param: m [MObject - in] Returns the instant just before the moving object enters the disabled status.
activespan (*MObject*)	public: *Intervals*	param: m [MObject - in] Returns the intervals in which the moving object is active.
status (*MObject, Instant*)	public: *StatusT*	param: m [MObject - in] param: i [Instant - in] Returning the status of the moving object at a particular instant i.
verifySTrelationship (*STpredicateT, MObject, Geometry*)	public: *boolean*	param: stp [STpredicateT - in] param: m [MObject - in] param: g [Geometry - in] Testing whether the specified spatio-temporal (ST) topological relationship applies to the time-varying position of the moving object and the specified geometry g.

verifyTrelationship (*TpredicateT, MObject, Time*)	public: *boolean*	param: stp [TpredicateT - in] param: m [MObject - in] param: t [Time - in] Testing whether the specified temporal (T) topological relationship applies to the lifecycle of the moving object and the specified time t.
intersection (*MObjects, Geometry*)	public: *MObjects*	param: ms [MObjects - in] param: g [Geometry - in] Returning the spatial intersection of this MObjects with the parameter g, which can either be a Line, a Surface, a Points or another MObjects geometry. The result type is the minimum of the two types in assumed dimensional order: points < line < surface. [testing intersects predicate for each mObject]
difference (*MObjects, Geometry*)	public: *MObjects*	param: ms [MObjects - in] param: g [Geometry - in] Returning the spatial difference of this MObjects with the parameter g. The result type is the minimum of the two types in assumed dimensional order: points < line < surface. [testing disjoint predicate for each mObject]
substract (*MObjects, Geometry*)	public: *MObjects*	param: ms [MObjects - in] param: g [Geometry - in] Remove all values from this MObjects that are also in parameter g. The result type is also a MObjects since substracting a lower-dimensional value returns the MObjects unchanged and subtracting a higher-dimensional value does not increase the dimension of the MObjects. [testing intersects predicate for each mObject]

MovingObjectsAnalysisSystem



MovingObjectsAnalysisSystem

Package
Implements: IMovingObjAnalysis. : business logic component

MovingObjectsAnalysisSystem::Geometry

Class: abstract spatial object type, corresponds to GM_Object in ISO 19107 "Geographic Information - Spatial schema"

MovingObjectsAnalysisSystem::Geometry Attributes

Attribute	Type	Notes
srs	private : *String*	Referenzsystem
dimension	private : *int*	0 = Point, 1 = Line oder 2 = Surface

MovingObjectsAnalysisSystem::Geometry Methods

Method	Type	Notes
distance (*Geometry*)	public: *double*	param: g [Geometry - in] Returning the shortest distance between any two points of this Geometry value and the parameter g.
equals (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this Geometry is equal to the parameter g which is of the same dimension.
disjoint (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this Geometry and the parameter g do not intersect.
intersects (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this Geometry and the parameter g share at least one point.
crosses (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this Geometry intersects the parameter g and the dimension of their intersection is less than the dimension of the geometries.
overlaps (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this Geometry intersects the parameter g which is of the same dimension and the dimension of their intersection is equal to the dimension of both geometries.
covers (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this Geometry contains every point of the parameter g.
coveredBy (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether parameter g contains every point of this Geometry.
contains (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether the interior of this Geometry contains every point of the interior of the parameter g.

inside (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether every point of the interior of this Geometry is located in the interior of the parameter g.
meets (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this Geometry nad the parameter g intersect in a point while their interiors are disjoint.
adjacent (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this Geometry and the parameter g intersect in a line while their interiors are disjoint.
touches (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this Geometry either meets or is adjacent to the parameter g.

MovingObjectsAnalysisSystem::Instant

Class

Extends: Time. : primitive/simple anchored time/temporal type

MovingObjectsAnalysisSystem::Instant Attributes

Attribute	Type
year	private : *int*
month	private : *int*
day	private : *int*
hour	private : *int*
minute	private : *int*
second	private : *int*
millisecond	private : *int*

MovingObjectsAnalysisSystem::Instants

Class

Extends: Time. : Instant Set, collection of instants.

MovingObjectsAnalysisSystem::Instants Attributes

Attribute	Type
instant	private : *Instant*

MovingObjectsAnalysisSystem::Instants Methods

Method	Type	Notes
start ()	public: *Instant*	Returns the earliest instant in time.
end ()	public: *Instant*	Returns the latest instant in time.

MovingObjectsAnalysisSystem::Interval

Class

Extends: Time. : primitive/simple anchored time/temporal type

MovingObjectsAnalysisSystem::Interval Attributes

Attribute	Type
start	private : *Instant*
end	private : *Instant*

MovingObjectsAnalysisSystem::Interval Methods

Method	Type	Notes
duration ()	public: *TimeSpan*	Returns the duration of time from the start to the end of the interval.

MovingObjectsAnalysisSystem::Intervals

Class
Extends: Time. : Interval Set, collection of intervals. There is a constraint that the intervals are disjoint (not overlapping).

MovingObjectsAnalysisSystem::Intervals Attributes

Attribute	Type
interval	private : *Interval*

MovingObjectsAnalysisSystem::Intervals Methods

Method	Type	Notes
startInterval ()	public: *Interval*	Returns the earliest interval, the interval which starts and ends first.
endInterval ()	public: *Interval*	Returns the latest interval, the interval which starts and ends latest.
duration ()	public: *TimeSpan*	Returns the duration of the intervals, from start of the first to the end of the last interval.

MovingObjectsAnalysisSystem::Lifecycle

Class: type used for capturing lifecycle information, is a parameterized complex type composed of a time-varying attribute status and two temporal attributes dob and dod (which enclose the lifecycle interval).
Operations in a formal language, based on an algebraic approach.

MovingObjectsAnalysisSystem::Lifecycle Attributes

Attribute	Type	Notes
status	private : *MStatus*	time-varying attribute, type of temporal extent: interval
dob	private : *Instant*	date of birth
dod	private : *Instant*	date of death

MovingObjectsAnalysisSystem::Lifecycle Methods

Method	Type	Notes
getDob ()	public: *Instant*	Returns the instant at which this lifecycle becomes active for the first time.
getDod ()	public: *Instant*	Returns the instant just before this lifecycle enters the disabled status.
lifetime ()	public: *TimeSpan*	Returns the duration between dob and dod as a duration of time that is not linked to the timeline.
lifespan ()	public: *Interval*	Returns the interval between the instants in

		which this lifecycle becomes first active and becomes disabled (between dob and dod).
activespan ()	public: *Intervals*	Returns the intervals in which this lifecycle is active.
status (*Instant*)	public: *StatusT*	param: i [Instant - in] Returning the status of this lifecycle at a particular instant.
equals (*Interval*)	public: *boolean*	param: i [Interval - in] Testing whether this lifecycle is equal to the parameter i.
disjoint (*Time*)	public: *boolean*	param: t [Time - in] Testing whether this lifecycle and the parameter t do not intersect.
intersects (*Time*)	public: *boolean*	param: t [Time - in] Testing whether this lifecycle and the parameter t have at least one instant in common.
overlaps (*Interval*)	public: *boolean*	param: i [Interval - in] Testing whether the interior of this lifecycle and the parameter i intersect, while the intersection of this lifecycle and i is not equal to one of them.
covers (*Time*)	public: *boolean*	param: t [Time - in] Testing whether this lifecycle contains every instant of the parameter t.
contains (*Time*)	public: *boolean*	param: t [Time - in] Testing whether any instant of the parameter t belongs to this lifecycle.
inside (*Interval*)	public: *boolean*	param: i [Interval - in] Testing whether any instant of this lifecycle belongs to the parameter i (predicate also called during).
meets (*Time*)	public: *boolean*	param: t [Time - in] Testing whether the last instant of this interval is equal to the first instant of the parameter t or vice-versa.
starts (*instant*)	public: *boolean*	param: i [instant - in] Testing whether the first instant of this lifecycle and the parameter i are the same.
finishes (*instant*)	public: *boolean*	param: i [instant - in] Testing whether the last instant of this lifecycle and the parameter i are the same.
precedes (*Time*)	public: *boolean*	param: t [Time - in] Testing whether the last instant of this lifecycle is smaller than the first instant of parameter t (predicate also called before).
succeeds (*Time*)	public: *boolean*	param: t [Time - in] Testing whether the first instant of this lifecycle is greater than the last instant of parameter t (predicate also called after).

MovingObjectsAnalysisSystem::Line

Class

Extends: Geometry. : primitive/simple geometry type

MovingObjectsAnalysisSystem::Line Attributes

Attribute	Type	Notes
sPoint	private : *Point*	Start point of the (oriented) line.
ePoint	private : *Point*	End point of the (oriented) line.

MovingObjectsAnalysisSystem::Line Methods

Method	Type	Notes
distance (*Geometry*)	public: *double*	param: g [Geometry - in] Returning the shortest distance between any two points of this Line value and the parameter g.
equals (*Line*)	public: *boolean*	param: l [Line - in] Testing whether this Line is equal to the parameter l.
disjoint (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this Line and the parameter g do not intersect.
intersects (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this Line and the parameter g share at least one point.
crosses (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this Line intersects the parameter g and the dimension of their intersection is less than the dimension of the geometries.
overlaps (*Line*)	public: *boolean*	param: l [Line - in] Testing whether this Line intersects the parameter l which is of the same dimension and the dimension of their intersection is equal to the dimension of both geometries.
covers (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this Line contains every point of the parameter g.
coveredBy (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether parameter g contains every point of this Line.
contains (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether the interior of this Line contains every point of the interior of the parameter g.
inside (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether every point of the interior of this Line is located in the interior of the parameter g.

meets (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this Line and the parameter g intersect in a point while their interiors are disjoint.
adjacent (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this Line and the parameter g intersect in a line while their interiors are disjoint.
touches (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this Line either meets or is adjacent to the parameter g.
intersection (*Geometry*)	public: *Geometry*	param: g [Geometry - in] Returning the spatial intersection () of this Line with the parameter g. The result type is the minimum of the two types in assumed dimensional order: points < line < surface.
difference (*Geometry*)	public: *Geometry*	param: g [Geometry - in] Returning the spatial difference of this Line with the parameter g. The result type is the minimum of the two types in assumed dimensional order: points < line < surface.
substract (*Geometry*)	public: *Line*	param: g [Geometry - in] Remove all values from this Line that are also in parameter g. The result type is also a line since substracting a lower-dimensional value returns the line unchanged and subtracting a higher-dimensional value does not increase the dimension of the line.
union (*Line*)	public: *Line*	param: l [Line - in] Returning the spatial union of this Line with the parameter l, which can only be a Line geometry.

MovingObjectsAnalysisSystem::MGroup

Class
Extends: MObjects. :

MovingObjectsAnalysisSystem::MGroup Attributes

Attribute	Type
groupId	private : *int*

MovingObjectsAnalysisSystem::MGroup Methods

Method	Type	Notes
add (*MObject*)	public: *void*	param: o [MObject - in] Adds a new moving object to the group.
remove (*MObject*)	public: *void*	param: o [MObject - in] Removes a moving object from the group.
count ()	public: *int*	Counts the number of moving objects belonging to the group.

MovingObjectsAnalysisSystem::MObject

Class: Moving object: spatial and temporal object type, object with spatial extent/ geometry and lifecycle. Additionally, spatial attribute is time-varying.
Operations in a user-oriented language, based on a textual language.

MovingObjectsAnalysisSystem::MObject Attributes

Attribute	Type	Notes
id	private : *int*	thematic attribute [static]
classification	private : *ClassificationT*	thematic attribute [static]
track	private : *MPoint*	temporal/moving (def. from MOD) or time-varying (def. from MADS) spatial attribute. Type of temporal extent: instant, set composed of (time, value)-couples. Illustrates the time-varying positions of the moving object. [time-varying]
lifecycle	private : *Lifecycle*	predefined attribute which keeps track of the status of instances: complex type, consisting of a [time-varying] attribute status and two temporal attributes dob and dod. Initial Value: active;

MovingObjectsAnalysisSystem::MObject Methods

Method	Type	Notes
verifyClassification (*ClassificationT*)	public: *boolean*	param: c [ClassificationT - in] Testing whether the moving object complies with the specified classification.
intervals ()	public: *Intervals*	Returns the set of time intervals between the single track points.
instants ()	public: *Instants*	Returns the set of instants for which a track point is defined.
positions ()	public: *Points*	Returns the set of track points for which an instant is defined when the moving object is projected into the plane.
trajectory ()	public: *Line*	Returns the line when the moving object is projected into the plane.
startPosition ()	public: *Point*	Returns the first track point of the moving object's lifespan.
endPosition ()	public: *Point*	Returns the last track point of the moving object's lifespan.
clip (*Instant*)	public: *MPoint*	param: i [Instant - in] Returns the point/instant pair that is defined for the particular instant i.
clip (*Intervals*)	public: *MPoint*	param: i [Intervals - in] Returns the portion of the track that is defined for the particular intervals i.
clip (*Geometry*)	public: *MPoint*	param: g [Geometry - in] Returns the portion of the track that is defined for the Geometry g.
direction (*Instant*)	public: *double*	param: i [Instant - in] Returns the direction of the moving object at instant i, the angle between the x-axis and a tangent to the trajectory of the moving object.
minDirection ()	public: *double*	Returns the minimum direction of the moving

		object during its lifespan.
maxDirection ()	public: *double*	Returns the maximum direction of the moving object during its lifespan.
changeOfDirection (*Instant*)	public: *double*	param: i [Instant - in] Returns the change of direction of the moving object at instant i.
minChangeOfDirection ()	public: *double*	Returns the minimum change of direction of the moving object during its lifespan.
maxChangeOfDirection ()	public: *double*	Returns the maximum change of direction of the moving object during its lifespan.
speed (*Instant*)	public: *double*	param: i [Instant - in] Returns the speed of the moving object at instant i.
minSpeed ()	public: *double*	Returns the minimum speed of the moving object during its lifespan.
maxSpeed ()	public: *double*	Returns the maximum speed of the moving object during its lifespan.
distanceTo (*Point, Instant*)	public: *double*	param: p [Point - in] param: i [Instant - in] Returns the distance between the moving object and p at instant i.
minDistanceTo (*Point*)	public: *double*	param: p [Point - in] Returns the minimum distance between the moving object and p during the moving objects' lifespan.
maxDistanceTo (*Point*)	public: *double*	param: p [Point - in] Returns the maximum distance between the moving object and p during the moving objects' lifespan.
directionTo (*Point, Instant*)	public: *double*	param: p [Point - in] param: i [Instant - in] Returns the direction (angle of the line from the moving object to the second point, measured in degrees, relative to a horizontal line) between the moving point and p at instant i.
minDirectionTo (*Point*)	public: *double*	param: p [Point - in] Returns the minimum direction between the moving point and p during the moving objects' lifespan.
maxDirectionTo (*Point*)	public: *double*	param: p [Point - in] Returns the maximum direction between the moving point and p during the moving objects' lifespan.
verifySTrelationship (*STpredicateT, Geometry*)	public: *boolean*	param: stp [STpredicateT - in] param: g [Geometry - in] Testing whether the specified spatio-temporal (ST) topological relationship applies to the time-varying position of the moving object and the specified geometry g.
birth ()	public: *Instant*	Returns the instant at which the moving objects becomes active for the first time.
death ()	public: *Instant*	Returns the instant just before the moving object

		enters the disabled status.
activespan ()	public: *Intervals*	Returns the intervals in which the moving object is active.
status (*Instant*)	public: *StatusT*	param: i [Instant - in] Returning the status of the moving object at a particular instant i.
verifyTrelationship (*TpredicateT, Time*)	public: *boolean*	param: stp [TpredicateT - in] param: t [Time - in] Testing whether the specified temporal (T) topological relationship applies to the lifecycle of the moving object and the specified time t.

MovingObjectsAnalysisSystem::MObjects

Class: collection of moving objects
Operations in a user-oriented language, based on a textual language.

MovingObjectsAnalysisSystem::MObjects Attributes

Attribute	Type
mObject	private : *MObject*

MovingObjectsAnalysisSystem::MObjects Methods

Method	Type	Notes
intersection (*Geometry*)	public: *MObjects*	param: g [Geometry - in] Returning the spatial intersection of this MObjects with the parameter g, which can either be a Line, a Surface, a Points or another MObjects geometry. The result type is the minimum of the two types in assumed dimensional order: points < line < surface. [testing intersects predicate for each mObject]
difference (*Geometry*)	public: *MObjects*	param: g [Geometry - in] Returning the spatial difference of this MObjects with the parameter g. The result type is the minimum of the two types in assumed dimensional order: points < line < surface. [testing disjoint predicate for each mObject]
substract (*Geometry*)	public: *MObjects*	param: g [Geometry - in] Remove all values from this MObjects that are also in parameter g. The result type is also a MObjects since substracting a lower-dimensional value returns the MObjects unchanged and subtracting a higher-dimensional value does not increase the dimension of the MObjects. [testing intersects predicate for each mObject]

MovingObjectsAnalysisSystem::MovingObjectsAnalysis

Class
Implements: IMovingObjAnalysis. : class with selected operations, comprising a frequently needed subset of the subsystems functionality, delegating the functionality to other classes of the subsystem

MovingObjectsAnalysisSystem::MPoint

Class: Infinite Set (collection of elements) of (instant,point)-pairs.
Operations in a formal language, based on an algebraic approach.

MovingObjectsAnalysisSystem::MPoint Attributes

Attribute	Type
time	private : *Instant*
position	private : *Point*

MovingObjectsAnalysisSystem::MPoint Methods

Method	Type	Notes
defIntervals ()	public: *Intervals*	Returns the temporal extent on which the function is defined: this is the set of time intervals when a temporal function is defined (also called defTime).
defInstants ()	public: *Instants*	Returns the temporal extent on which the function is defined: this is the set of instants for which a Point is defined (also called defTime).
locations ()	public: *Points*	Returns the spatial extent on which the function is defined: the set of points for which an instant is defined when the moving point is projected into the plane (also called defSpace).
trajectory ()	public: *Line*	Returns the line when the moving point is projected into the plane.
atInstant (*Instant*)	public: *MPoint*	param: i [Instant - in] Returns the (instant,point)-pair that is defined for the particular instant i.
atIntervals (*Intervals*)	public: *MPoint*	param: i [Intervals - in] Returns the portion of the function that is defined for the particular intervals i.
atLocation (*Geometry*)	public: *MPoint*	param: g [Geometry - in] Returns the portion of the function that is defined for the spatial extend g.
initial ()	public: *MPoint*	Returns the (instant,point)-pair for the first instant of the definition time.
final ()	public: *MPoint*	Returns the (instant,point)-pair for the last instant of the definition time.
speed ()	public: *MReal*	Returns a moving real type indicating the speed of a moving point as a function over time (rate of change/derivative).
mDirection ()	public: *MReal*	Returns the direction of movement, the angle between the x-axis and a tangent to the trajectory of the moving point (rate of change/derivative).
turn ()	public: *MReal*	Returns the change of direction at all times (rate of change/derivative).
distance (*Point*)	public: *MReal*	param: p [Point - in] Returns the time-varying distance between a moving point and p.
direction (*Point*)	public: *MReal*	param: p [Point - in] Returns the time-varying direction (angle of the line from the first to the second point, measured in degrees, relative to a horizontal line) between a moving point and p.
equals (*MPoint*)	public: *boolean*	param: m [MPoint - in]

		Testing whether this MPoint is equal to the parameter m. Preferred temporal aggregation: universal quantifier ranging over the whole lifetime which has to be the same for both objects (requires identical domains).
disjoint (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this MPoint and the parameter g do not intersect. Preferred temporal aggregation: universal quantifier ranging only over the two objects' common lifetime.
intersects (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this MPoint and the parameter g share at least one point. Preferred temporal aggregation: universal quantifier ranging only over the two objects' common lifetime.
coveredBy (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether parameter g contains this MPoint. Preferred temporal aggregation: universal quantifier ranging over the whole lifetime of this MPoint.
inside (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this MPoint is located in the interior of the parameter g. Preferred temporal aggregation: universal quantifier ranging over the whole lifetime of this MPoint.
meets (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this MPoint and the parameter g intersect in a point while their interiors are disjoint. Preferred temporal aggregation: universal quantifier ranging over the whole lifetime which has to be de same for both objects (requires identical domain).
enters (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this MPoint is disjoint of the parameter g until they meet and then is inside of the parameter g. Development, combines basic spatio-temporal predicates to a ordered sequence. Therefore, predicates are constricted to respective intervals and classified into instant predicates (lowercase) and period predicates (capital letter): Disjoint > meet > Inside.
leaves (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this MPoint is inside of the parameter g until they meet and then is disjoint of the parameter g (is the opposite of enters).

		Development, combines basic spatio-temporal predicates to a ordered sequence. Therefore, predicates are constricted to respective intervals and classified into instant predicates (lowercase) and period predicates (capital letter): Inside > meet > Disjoint
crosses (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this MPoint first enters and then leaves the parameter g. Development, combines basic spatio-temporal predicates to a ordered sequence. Therefore, predicates are constricted to respective intervals and classified into instant predicates (lowercase) and period predicates (capital letter): Disjoint > meet > Inside > meet > Disjoint.
touches (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this MPoint is disjoint of the parameter g until they meet and then is disjoint again. Development, combines basic spatio-temporal predicates to a ordered sequence. Therefore, predicates are constricted to respective intervals and classified into instant predicates (lowercase) and period predicates (capital letter): Disjoint > meet > Disjoint.
snaps (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this MPoint is disjoint of the parameter g and then meets parameter g. Development, combines basic spatio-temporal predicates to a ordered sequence. Therefore, predicates are constricted to respective intervals and classified into instant predicates (lowercase) and period predicates (capital letter): Disjoint > Meet.
releases (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this MPoint meets the parameter g and then is disjoint of parameter g (is the opposite of snaps). Development, combines basic spatio-temporal predicates to a ordered sequence. Therefore, predicates are constricted to respective intervals and classified into instant predicates (lowercase) and period predicates (capital letter): Meet > Disjoint.
bypasses (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this MPoint first snaps and then releases the parameter g. Development, combines basic spatio-temporal predicates to a ordered sequence. Therefore, predicates are constricted to respective intervals and classified into instant predicates (lowercase) and period predicates (capital letter): Disjoint > Meet > Disjoint.
excurses (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this MPoint meets the parameter g, then they are disjoint and then they meet

		again. Development, combines basic spatio-temporal predicates to a ordered sequence. Therefore, predicates are constricted to respective intervals and classified into instant predicates (lowercase) and period predicates (capital letter): Meet > Disjoint > Meet.

MovingObjectsAnalysisSystem::MReal

Class: Infinite Set (collection of elements) of (instant,value)-pairs.

MovingObjectsAnalysisSystem::MReal Attributes

Attribute	Type
instant	private : *Instant*
value	private : *double*

MovingObjectsAnalysisSystem::MReal Methods

Method	Type	Notes
atMin ()	public: *MReal*	Returns the (instant,value)-pairs for the minimum value of the value range.
atMax ()	public: *MReal*	Returns the (instant,value)-pairs for the maximum value of the value range.
at (*double*)	public: *MReal*	param: v [double - in] Returns the portion of the function that is defined for the range of v -> restricts the range of the function.

MovingObjectsAnalysisSystem::MStatus

Class: Infinite Set (collection of elements) of (interval,status)-pairs.

MovingObjectsAnalysisSystem::MStatus Attributes

Attribute	Type
time	private : *Interval*
status	private : *StatusT*

MovingObjectsAnalysisSystem::Point

Class
Extends: Geometry. : primitive/simple geometry type

MovingObjectsAnalysisSystem::Point Attributes

Attribute	Type
x	private : *double*
y	private : *double*

MovingObjectsAnalysisSystem::Point Methods

Method	Type	Notes
distance (*Geometry*)	public: *double*	param: g [Geometry - in] Returning the shortest distance between any two

		points of this Point value and the parameter g.
equals (*Point*)	public: *boolean*	param: p [Point - in] Testing whether this Point is equal to the parameter p.
disjoint (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this Point and the parameter g do not intersect.
intersects (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this Point and the parameter g share at least one point.
coveredBy (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether parameter g contains this Point.
inside (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this Point is located in the interior of the parameter g.
meets (*Geometry*)	public: *boolean*	param: g [Geometry - in] Testing whether this Point and the parameter g intersect in a point while their interiors are disjoint (g either Line or Surface).

MovingObjectsAnalysisSystem::Points

Class

Extends: Geometry. : Point Set, collection of points.

MovingObjectsAnalysisSystem::Points Attributes

Attribute	Type
point	private : *Point*

MovingObjectsAnalysisSystem::Points Methods

Method	Type	Notes
convexHull ()	public: *Surface*	Returning the convex hull of this Points.
intersection (*Geometry*)	public: *Points*	param: g [Geometry - in] Returning the spatial intersection of this Points with the parameter g, which can either be a Line, a Surface or another Points geometry. The result type is the minimum of the two types in assumed dimensional order: points < line < surface.
difference (*Geometry*)	public: *Points*	param: g [Geometry - in] Returning the spatial difference of this Points with the parameter g. The result type is the minimum of the two types in assumed dimensional order: points < line < surface.
substract (*Geometry*)	public: *Points*	param: g [Geometry - in] Remove all values from this Points that are also in parameter g. The result type is also a Points since substracting a lower-dimensional value returns the Points unchanged and subtracting a higher-dimensional value does not increase the dimension of the Points.
union (*Points*)	public: *Points*	param: p [Points - in]

		Returning the spatial union of this Points with the parameter p, which can only be a Points geometry.

MovingObjectsAnalysisSystem::Surface

Class
Extends: Geometry. : primitive/simple geometry type

MovingObjectsAnalysisSystem::Surface Attributes

Attribute	Type	Notes
exterior	private : *Lines*	Exterior boundary of the surface.
interior	private : *Lines*	List of interior boundaries of the surface.

MovingObjectsAnalysisSystem::Time

Class: abstract temporal object type, corresponds to TM_Object in ISO 19108 "Geographic Information - Temporal schema". Is the root of the hierarchy defining anchored temporal types.

MovingObjectsAnalysisSystem::Time Attributes

Attribute	Type	Notes
dimension	private : *int*	0 = instant, 1 = interval

MovingObjectsAnalysisSystem::Time Methods

Method	Type	Notes
equals (*Time*)	public: *boolean*	param: t [Time - in] Testing whether this Time is equal to the parameter t.
disjoint (*Time*)	public: *boolean*	param: t [Time - in] Testing whether this Time and the parameter t do not intersect.
intersects (*Time*)	public: *boolean*	param: t [Time - in] Testing whether this Time and the parameter t have at least one instant in common.
overlaps (*Time*)	public: *boolean*	param: t [Time - in] Testing whether the interior of this Time and the parameter t intersect, while the intersection of this Time and t is not equal to one of them.
covers (*Time*)	public: *boolean*	param: t [Time - in] Testing whether this Time contains every instant of the parameter t.
contains (*Time*)	public: *boolean*	param: t [Time - in] Testing whether any instant of the parameter t belongs to this Time.
inside (*Time*)	public: *boolean*	param: t [Time - in] Testing whether any instant of this Time belongs to the parameter t.
meets (*Time*)	public: *boolean*	param: t [Time - in]

		Testing whether the last instant of this Time is equal to the first instant of the parameter t or vice-versa.
starts (*Time*)	public: *boolean*	param: t [Time - in] Testing whether the first instant of this Time and the parameter t are the same.
finishes (*Time*)	public: *boolean*	param: t [Time - in] Testing whether the last instant of this Time and the parameter t are the same.
precedes (*Time*)	public: *boolean*	param: t [Time - in] Testing whether the last instant of this time is smaller than the first instant of parameter t.
succeeds (*Time*)	public: *boolean*	param: t [Time - in] Testing whether the first instant of this Time is greater than the last instant of parameter t.

MovingObjectsAnalysisSystem::TimeSpan

Class: primitive/simple unanchored time/temporal type, duration of time that is not linked to the timeline.

MovingObjectsAnalysisSystem::TimeSpan Attributes

Attribute	Type
year	private : *int*
month	private : *int*
day	private : *int*
hout	private : *int*
minute	private : *int*
second	private : *int*
millisecond	private : *int*

Enumerations::ClassificationT

Class:

Enumerations::ClassificationT Attributes

Attribute	Type	Notes
onroad «enum»	public : *int*	landgestütztes onroad Fahrzeug wie Auto, LKW, usw.
offroad «enum»	public : *int*	landgestütztes offroad Fahrzeug wie Geländewagen, Kettenfahrzeug usw.
aircraft «enum»	public : *int*	Luftfahrzeug
railway «enum»	public : *int*	Schienenfahrzeug
watercraft «enum»	public : *int*	Wasserfahrzeug

Enumerations::StatusT

Class: Defines possible status values and transition rules which are restricting status changes. The value "scheduled" is N/A for this context. Temporal extend, associated to the active status: interval.
Valid transitions:
active <-> suspended -> disabled
| /\
|_____|

Enumerations::StatusT Attributes

Attribute	Type	Notes
active «enum»	public : *int*	active
suspended «enum»	public : *int*	temporarily inactive
disabled «enum»	public : *int*	expired, never becomes active again

Enumerations::STpredicateT

Class:

Enumerations::STpredicateT Attributes

Attribute	Type
equals «enum»	public : *int*
disjoint «enum»	public : *int*
intersects «enum»	public : *int*
coveredBy «enum»	public : *int*
inside «enum»	public : *int*
meets «enum»	public : *int*
enters «enum»	public : *int*
leaves «enum»	public : *int*
crosses «enum»	public : *int*
touches «enum»	public : *int*
snaps «enum»	public : *int*
releases «enum»	public : *int*
bypasses «enum»	public : *int*
excurses «enum»	public : *int*

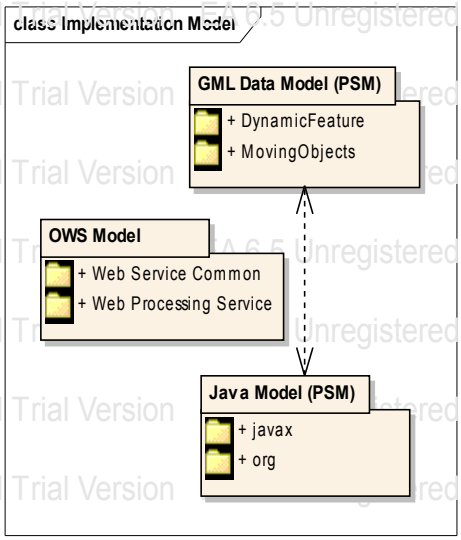
Enumerations::TpredicateT

Class:

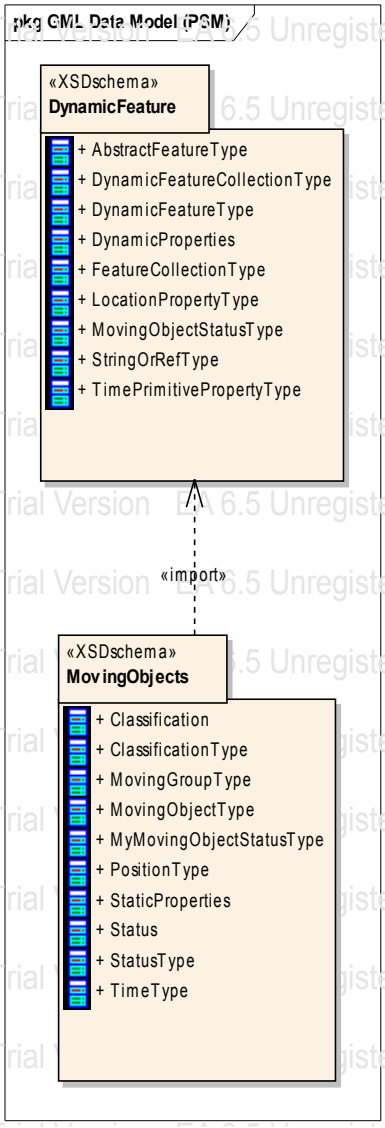
Enumerations::TpredicateT Attributes

Attribute	Type
equals «enum»	public : *int*
disjoint «enum»	public : *int*
intersects «enum»	public : *int*
overlaps «enum»	public : *int*
covers «enum»	public : *int*
contains «enum»	public : *int*
inside «enum»	public : *int*
meets «enum»	public : *int*
starts «enum»	public : *int*
finishes «enum»	public : *int*
precedes «enum»	public : *int*
succeeds «enum»	public : *int*

Implementation Model



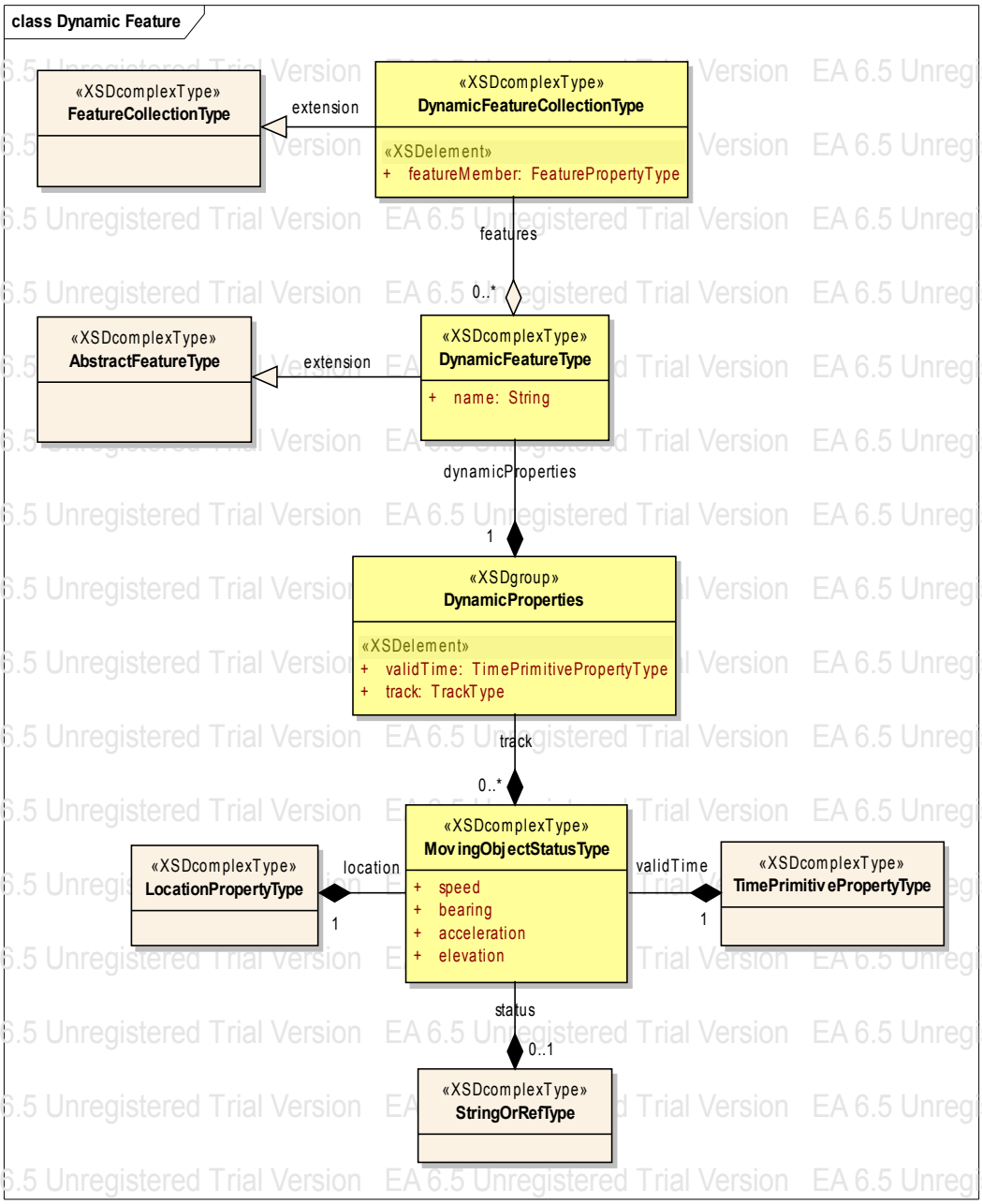
GML Data Model



GML Data Model (PSM)

Package: definition of a XML structure for spatio-temporal data types, based on GML core schemas

DynamicFeature



DynamicFeature

Package: GML Core Schema

DynamicFeature::AbstractFeatureType

Class: Provides a set of common properties for defining concrete feature types with location and time reference.

DynamicFeature::DynamicFeatureCollectionType

Class

Extends: FeatureCollectionType. : A collection of dynamic features that may possess a history and/or a timestamp.

DynamicFeature::DynamicFeatureCollectionType Attributes

Attribute	Type
featureMember «XSDelement»	public : *FeaturePropertyType*

DynamicFeature::DynamicFeatureType

Class
Extends: AbstractFeatureType. : A dynamic feature may possess a history and/or a timestamp.

DynamicFeature::DynamicFeatureType Attributes

Attribute	Type
name	public :*String*

DynamicFeature::DynamicProperties

Class: Time-varying properties of dynamic features.

DynamicFeature::DynamicProperties Attributes

Attribute	Type
validTime «XSDelement»	public : *TimePrimitivePropertyType*
track «XSDelement»	public : *TrackType*

DynamicFeature::FeatureCollectionType

Class: Contains zero or more features.

DynamicFeature::FeatureCollectionType Attributes

Attribute	Type
featureMember «XSDelement»	public : *FeaturePropertyType*

DynamicFeature::LocationPropertyType

Class: Generalised location property

DynamicFeature::MovingObjectStatusType

Class: Encapsulates various dynamic properties of moving objects.

DynamicFeature::MovingObjectStatusType Attributes

Attribute	Type
speed	public :
bearing	public :
acceleration	public :
elevation	public :

DynamicFeature::StringOrRefType

Class: Text property type, is of string type, so the text can be included inline but also referenced remotely via xlinks

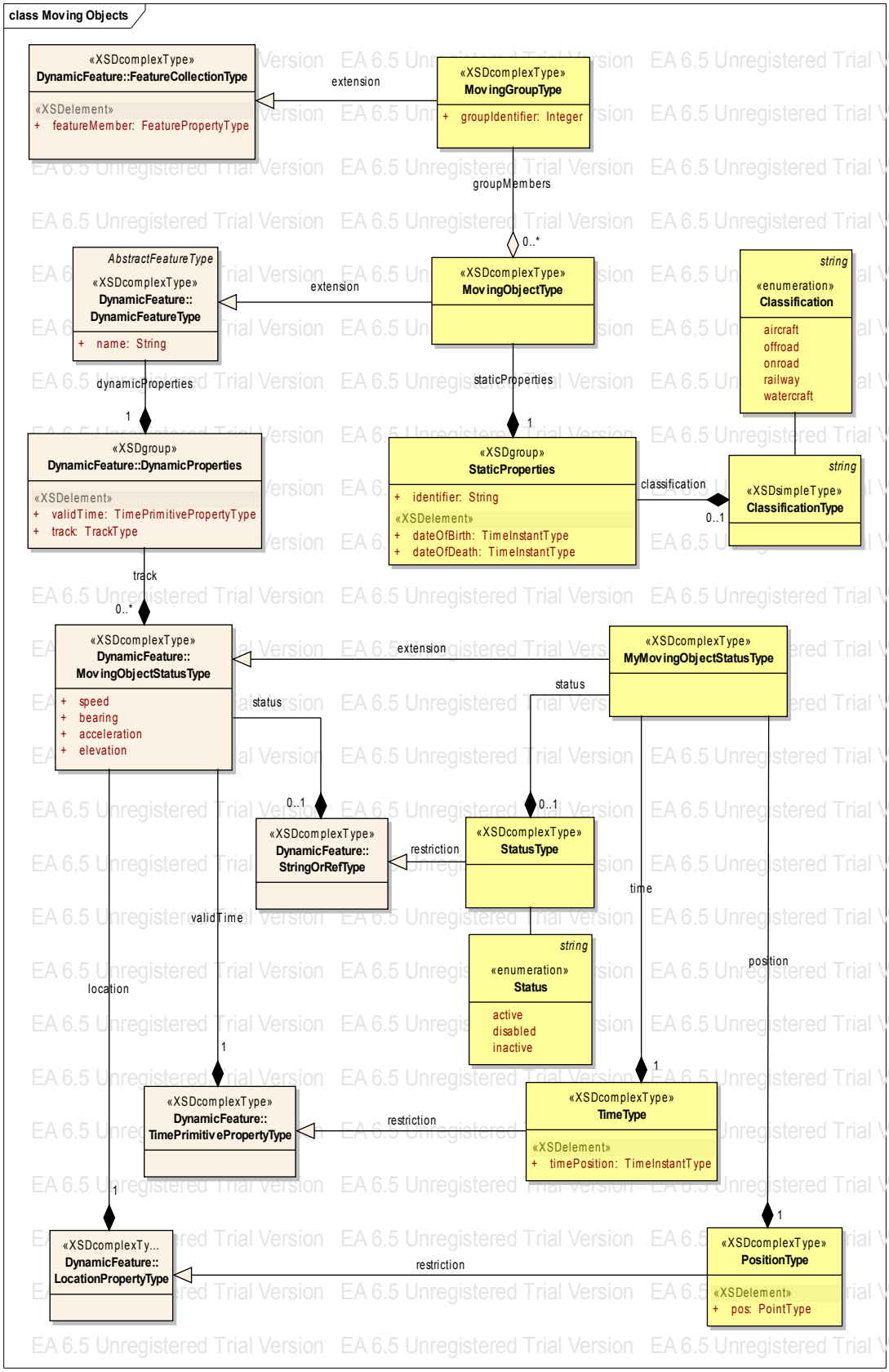
DynamicFeature::TimePrimitivePropertyType

Class: Generalised temporal property

DynamicFeature::TimePrimitivePropertyType Attributes

Attribute	Type
validTime «XSDelement»	public : *TimePrimitivePropertyType*
location «XSDelement»	public :

MovingObjects



MovingObjects

Package: GML Application Schema

MovingObjects::Classification

Class

Implements: string. : List of values to classify a moving object.

MovingObjects::Classification Attributes

Attribute	Type
aircraft	Public :
offroad	Public :
onroad	Public :
railway	Public :
watercraft	Public :

MovingObjects::ClassificationType

Class

Implements: string. : Category a moving object belongs to.

MovingObjects::MovingGroupType

Class

Extends: FeatureCollectionType. : Group a moving object is assigned to.

MovingObjects::MovingGroupType Attributes

Attribute	Type
groupIdentifier	public :*Integer*

MovingObjects::MovingObjectType

Class

Extends: DynamicFeatureType. : Object changing its position over time.

MovingObjects::MyMovingObjectStatusType

Class

Extends: MovingObjectStatusType. : Status of a moving objects which changes depending on the object's movement.

MovingObjects::PositionType

Class

Extends: LocationPropertyType. : Location of a moving object without spatial extent (point-shaped).

MovingObjects::PositionType Attributes

Attribute	Type
pos «XSDelement»	public : *PointType*

MovingObjects::StaticProperties

Class: Time invariant properties of a moving object.

MovingObjects::StaticProperties Attributes

Attribute	Type
identifier	public : *String*
dateOfBirth «XSDelement»	public : *TimeInstantType*
dateOfDeath «XSDelement»	public : *TimeInstantType*

MovingObjects::Status

Class
Implements: string. : List of values to describe the status of a moving object.

MovingObjects::Status Attributes

Attribute	Type
active	public :
disabled	public :
inactive	public :

MovingObjects::StatusType

Class
Extends: StringOrRefType. : Status of a moving object.

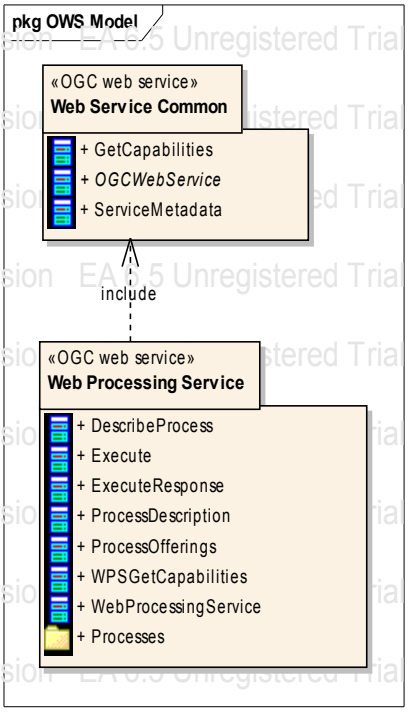
MovingObjects::TimeType

Class
Extends: TimePrimitivePropertyType. : Time stamp of a moving object as an instant in time.

MovingObjects::TimeType Attributes

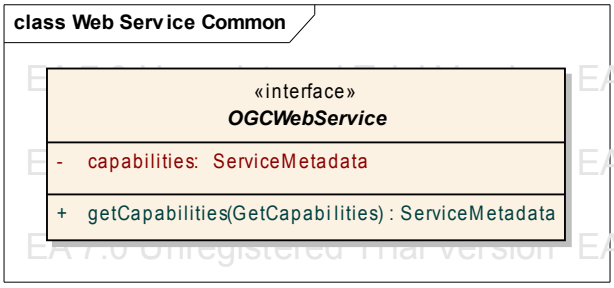
Attribute	Type
timePosition «XSDelement»	public : *TimeInstantType*

OWS Model



OWS Model

***Package**: interface definition of a OpenGIS Web Service (OWS) Web Service Common*



Web Service Common::GetCapabilities

***Class**:* GetCapabilities request handler of an ordinary OWS

Web Service Common::OGCWebService

***Class**:* OpenGIS Web Service interface

Web Service Common::OGCWebService Attributes

Attribute	Type	Notes
capabilities	private : *ServiceMetadata*	Contains metadata about the service provider and, for WPS, additionally metadata about available processes.

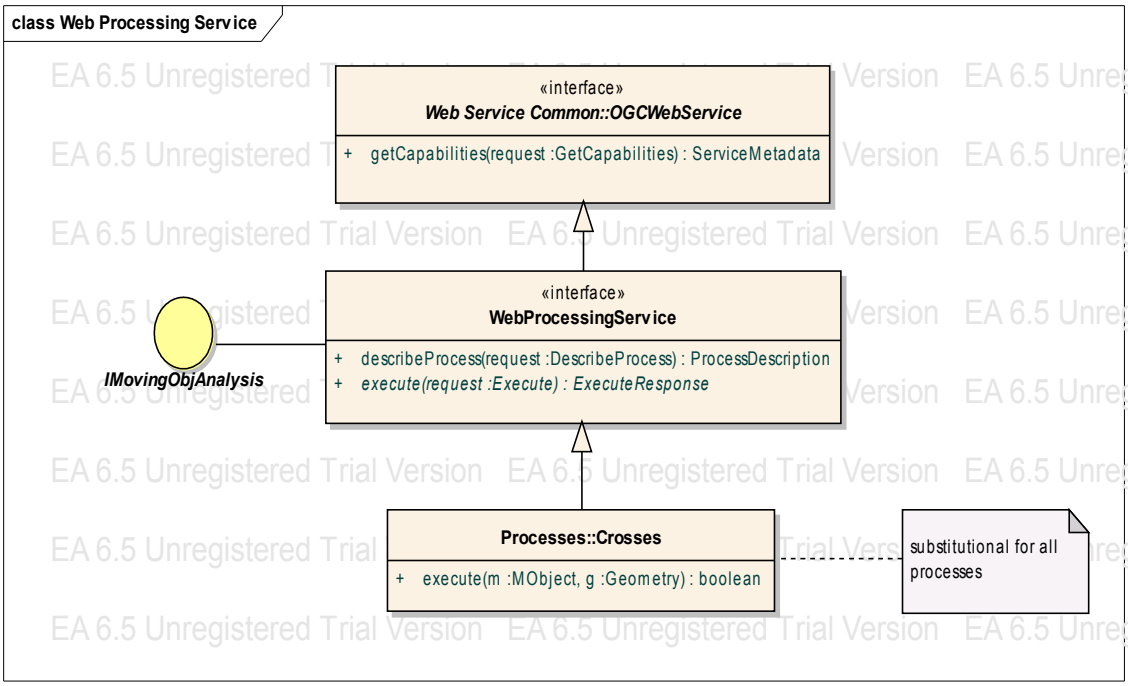
Web Service Common::OGCWebService Methods

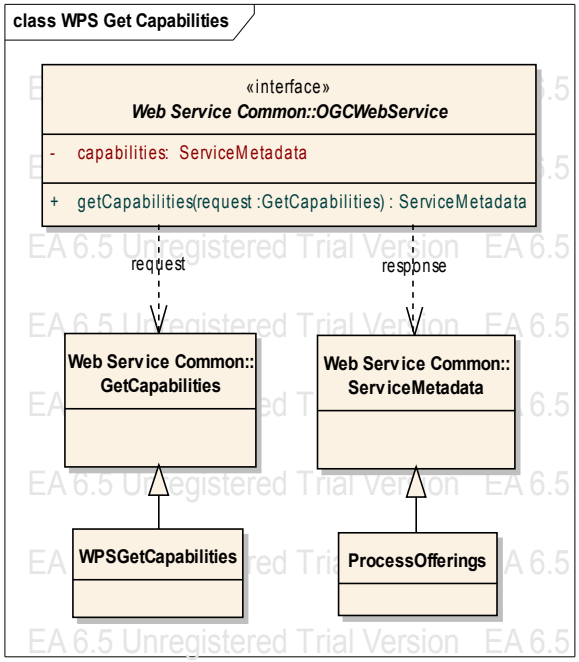
Method	Type	Notes
getCapabilities (*GetCapabilities*)	public: *ServiceMetadata*	param: request [GetCapabilities - in] Retrieval of service metadata. Returns capabilities document that includes metadata about the service provider and, for WPS, additionally metadata about available processes.

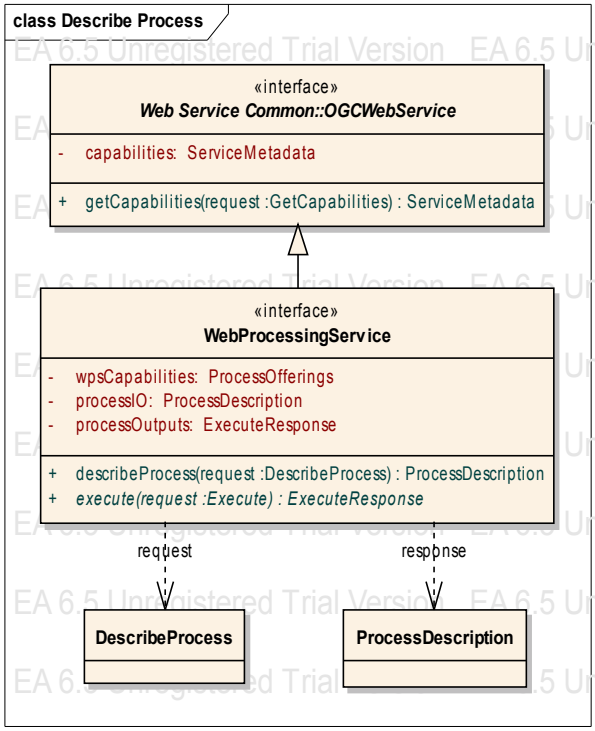
Web Service Common::ServiceMetadata

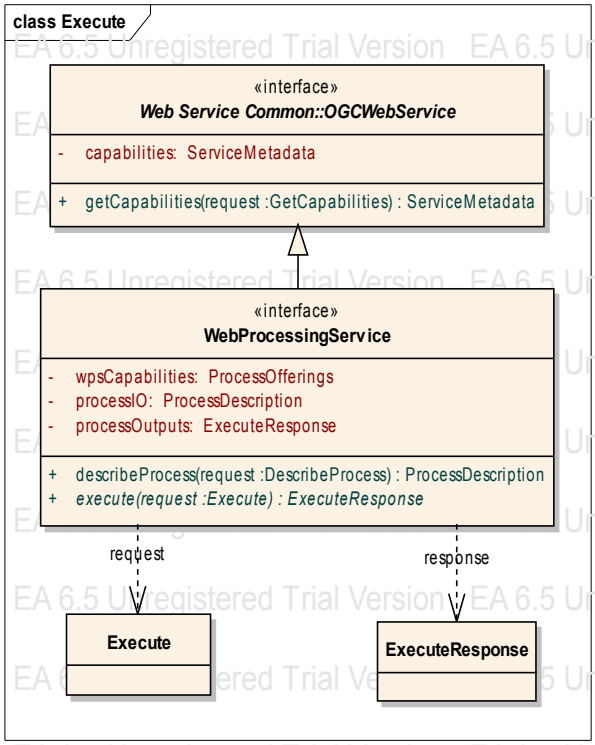
***Class**:* WPS capabilities document

Web Processing Service









Web Processing Service::DescribeProcess

Class: DescribeProcess request handler of a WPS

Web Processing Service::Execute

Class: Execute request handler of a WPS

Web Processing Service::ExecuteResponse

Class: Execute response document

Web Processing Service::ProcessDescription

Class: Process description document

Web Processing Service::ProcessOfferings

Class

Extends: ServiceMetadata. : List of processes offered by the WPS, belongs to the service metadata

Web Processing Service::WPSGetCapabilities

Class

Extends: GetCapabilities. : GetCapabilities request handler of a WPS

Web Processing Service::WebProcessingService

Class

Extends: OGCWebService. Implements: IMovingObjAnalysis. : OpenGIS WPS interface

Web Processing Service::WebProcessingService Attributes

Attribute	Type	Notes
wpsCapabilities	private : *ProcessOfferings*	
processIO	private : *ProcessDescription*	Contains a detailed description about input and output parameters of a specific process.
processOutputs	private : *ExecuteResponse*	

Web Processing Service::WebProcessingService Methods

Method	Type	Notes
describeProcess (*DescribeProcess*)	public: *ProcessDescription*	param: request [DescribeProcess - in] Retrieval of detailed description about input and output parameters of a specific process. Returns process description document. An input parameter may be defined either as a complex data type (e.g. GML), a literal data type (e.g. an integer value), or a bounding box data type. The complex data type data structure offers the capability to either encode the payload directly in the request or by referencing a remote location.
execute (*Execute*)	public abstract: *ExecuteResponse*	param: request [Execute - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes

Package: Contains atomic WPS processes for Moving Object Analysis tasks

Processes::Activespan

Class

Extends: WebProcessingService. : Returns the intervals in which the moving object is active.

Processes::Activespan Methods

Method	Type	Notes
execute (*MObject*)	public: *Intervals*	param: m [MObject - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::Birth

Class

Extends: WebProcessingService. : Returns the instant at which the moving objects becomes active for the first time.

Processes::Birth Methods

Method	Type	Notes
execute (*MObject*)	public: *Instant*	param: m [MObject - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::ChangeOfDirection

Class

Extends: WebProcessingService. : Returns the change of direction of the moving object at instant i.

Processes::ChangeOfDirection Methods

Method	Type	Notes
execute (*MObject, Instant*)	public: *double*	param: m [MObject - in] param: i [Instant - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::CheckCategory

Class

Extends: WebProcessingService. : Testing whether the moving object complies with the specified classification.

Processes::CheckCategory Methods

Method	Type	Notes
execute (*MObject, ClassificationT*)	public: *boolean*	param: m [MObject - in] param: c [ClassificationT - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::ClipGeometry

Class

Extends: WebProcessingService. : Returns the portion of the track that is defined for the Geometry g.

Processes::ClipGeometry Methods

Method	Type	Notes
execute (*MObject, Geometry*)	public: *MPoint*	param: m [MObject - in] param: g [Geometry - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::ClipInstant

Class

Extends: WebProcessingService. : Returns the point/instant pair that is defined for the particular instant i.

Processes::ClipInstant Methods

Method	Type	Notes
execute (*MObject, Instant*)	public: *MPoint*	param: m [MObject - in] param: i [Instant - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::ClipInterval

Class

Extends: WebProcessingService. : Returns the portion of the track that is defined for the particular intervals i.

Processes::ClipInterval Methods

Method	Type	Notes
execute (*MObject, Intervals*)	public: *MPoint*	param: m [MObject - in] param: i [Intervals - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::Death

Class

Extends: WebProcessingService. : Returns the instant just before the moving object enters the disabled status.

Processes::Death Methods

Method	Type	Notes
execute (*MObject*)	public: *Instant*	param: m [MObject - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::Direction

Class

Extends: WebProcessingService. : Returns the direction of the moving object at instant i, the angle between the x-axis and a tangent to the trajectory of the moving object.

Processes::Direction Methods

Method	Type	Notes
execute (*MObject, Instant*)	public: *double*	param: m [MObject - in] param: i [Instant - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::DirectionTo

Class

Extends: WebProcessingService. : Returns the maximum distance between the moving object and

p during the moving objects' lifespan.

Processes::DirectionTo Methods

Method	Type	Notes
execute (*MObject, Point, Instant*)	public: *double*	param: m [MObject - in] param: p [Point - in] param: i [Instant - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::DistanceTo

Class
Extends: WebProcessingService. : Returns the distance between the moving object and p at instant i.

Processes::DistanceTo Methods

Method	Type	Notes
execute (*MObject, Point, Instant*)	public: *double*	param: m [MObject - in] param: p [Point - in] param: i [Instant - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::EndPosition

Class
Extends: WebProcessingService. : Returns the last track point of the moving object's lifespan.

Processes::EndPosition Methods

Method	Type	Notes
execute (*MObject*)	public: *Point*	param: m [MObject - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::Finishes

Class
Extends: WebProcessingService. : Testing whether the last instant of the lifecycle of the moving

object and the specified time t are the same.

Processes::Finishes Methods

Method	Type	Notes
execute (*MObject, Time*)	public: *boolean*	param: m [MObject - in] param: t [Time - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::Instants

Class
Extends: WebProcessingService. : Returns the set of instants for which a track point is defined.

Processes::Instants Methods

Method	Type	Notes
execute (*MObject*)	public: *Instants*	param: m [MObject - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::Intervals

Class
Extends: WebProcessingService. : Returns the set of time intervals between the single track points.

Processes::Intervals Methods

Method	Type	Notes
execute (*MObject*)	public: *Intervals*	param: m [MObject - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::MaxChangeOfDirection

Class
Extends: WebProcessingService. : Returns the maximum change of direction of the moving object during its lifespan.

Processes::MaxChangeOfDirection Methods

Method	Type	Notes
execute (*MObject*)	public: *double*	param: m [MObject - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::MaxDirection

Class

Extends: WebProcessingService. : Returns the maximum direction of the moving object during its lifespan.

Processes::MaxDirection Methods

Method	Type	Notes
execute (*MObject*)	public: *double*	param: m [MObject - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::MaxDirectionTo

Class

Extends: WebProcessingService. : Returns the maximum direction between the moving point and p during the moving objects' lifespan.

Processes::MaxDirectionTo Methods

Method	Type	Notes
execute (*MObject, Point*)	public: *double*	param: m [MObject - in] param: p [Point - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::MaxDistanceTo

Class

Extends: WebProcessingService. : Returns the maximum distance between the moving object and p during the moving objects' lifespan.

Processes::MaxDistanceTo Methods

Method	Type	Notes
execute (*MObject, Point*)	public: *double*	param: m [MObject - in] param: p [Point - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::MaxSpeed

Class

Extends: WebProcessingService. : Returns the maximum speed of the moving object during its lifespan.

Processes::MaxSpeed Methods

Method	Type	Notes
execute (*MObject*)	public: *double*	param: m [MObject - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::MinChangeOfDirection

Class

Extends: WebProcessingService. : Returns the minimum change of direction of the moving object during its lifespan.

Processes::MinChangeOfDirection Methods

Method	Type	Notes
execute (*MObject*)	public: *double*	param: m [MObject - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::MinDirection

Class

Extends: WebProcessingService. : Returns the minimum direction of the moving object during its lifespan.

Processes::MinDirection Methods

Method	Type	Notes
execute (*MObject*)	public: *double*	param: m [MObject - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::MinDirectionTo

Class

Extends: WebProcessingService. : Returns the minimum direction between the moving point and p during the moving objects' lifespan.

Processes::MinDirectionTo Methods

Method	Type	Notes
execute (*MObject, Point*)	public: *double*	param: m [MObject - in] param: p [Point - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::MinDistanceTo

Class

Extends: WebProcessingService. : Returns the distance between the moving object and p at instant i.

Processes::MinDistanceTo Methods

Method	Type	Notes
execute (*MObject, Point*)	public: *double*	param: m [MObject - in] param: p [Point - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::MinSpeed

Class

Extends: WebProcessingService. : Returns the minimum speed of the moving object during its lifespan.

Processes::MinSpeed Methods

Method	Type	Notes
execute (*MObject*)	public: *double*	param: m [MObject - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::Positions

Class

Extends: WebProcessingService. : Returns the set of track points for which an instant is defined when the moving object is projected into the plane.

Processes::Positions Methods

Method	Type	Notes
execute (*MObject*)	public: *Points*	param: m [MObject - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::Precedes

Class

Extends: WebProcessingService. : Testing whether the ast instant of the lifecycle of the moving object is smaller than the first instant of the specified time t.

Processes::Precedes Methods

Method	Type	Notes
execute (*MObject, Time*)	public: *boolean*	param: m [MObject - in] param: t [Time - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::SpatioTemporalBypasses

Class

Extends: WebProcessingService. : Testing whether the time-varying position of the moving object first snaps and then releases the specified geometry g.

Processes::SpatioTemporalBypasses Methods

Method	Type	Notes
execute (*MObject, Geometry*)	public: *boolean*	param: m [MObject - in] param: g [Geometry - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::SpatioTemporalCoveredBy

Class

Extends: WebProcessingService. : Testing whether the specified geometry g contains the time-varying position of the moving object.

Processes::SpatioTemporalCoveredBy Methods

Method	Type	Notes
execute (*MObject, Geometry*)	public: *boolean*	param: m [MObject - in] param: g [Geometry - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::SpatioTemporalCrosses

Class

Extends: WebProcessingService. : Testing whether the time-varying position of the moving object first enters and then leaves the specified geometry g.

Processes::SpatioTemporalCrosses Methods

Method	Type	Notes
execute (*MObject, Geometry*)	public: *boolean*	param: m [MObject - in] param: g [Geometry - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::SpatioTemporalDifference

Class

Extends: WebProcessingService. : Returns the spatial difference of this MObjects with the parameter g. The result type is the minimum of the two types in assumed dimensional order: points < line < surface.

Processes::SpatioTemporalDifference Methods

Method	Type	Notes
execute (*MObjects, Geometry*)	public: *MObjects*	param: ms [MObjects - in] param: g [Geometry - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::SpatioTemporalDisjoint

Class

Extends: WebProcessingService. : Testing whether the time-varying position of the moving object and the specified geometry g do not intersect.

Processes::SpatioTemporalDisjoint Methods

Method	Type	Notes
execute (*MObject, Geometry*)	public: *boolean*	param: m [MObject - in] param: g [Geometry - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::SpatioTemporalEnters

Class

Extends: WebProcessingService. : Testing whether the time-varying position of the moving object is disjoint of the specified geometry g until they meet and then is inside of the specified geometry g.

Processes::SpatioTemporalEnters Methods

Method	Type	Notes
execute (*MObject, Geometry*)	public: *boolean*	param: m [MObject - in] param: g [Geometry - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::SpatioTemporalEquals

Class

Extends: WebProcessingService. : Testing whether the time-varying position of the moving object is equal to the specified geometry g.

Processes::SpatioTemporalEquals Methods

Method	Type	Notes
execute (*MObject, Geometry*)	public: *boolean*	param: m [MObject - in] param: g [Geometry - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::SpatioTemporalExcurses

Class

Extends: WebProcessingService. : Testing whether the time-varying position of the moving object meets the specified geometry g, then they are disjoint and then they meet again.

Processes::SpatioTemporalExcurses Methods

Method	Type	Notes
execute (*MObject, Geometry*)	public: *boolean*	param: m [MObject - in] param: g [Geometry - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::SpatioTemporalInside

Class

Extends: WebProcessingService. : Testing whether the time-varying position of the moving object is located in the interior of the specified geometry g.

Processes::SpatioTemporalInside Methods

Method	Type	Notes
execute (*MObject, Geometry*)	public: *boolean*	param: m [MObject - in] param: g [Geometry - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::SpatioTemporalIntersection

Class

Extends: WebProcessingService. : Returns the spatial intersection of this MObjects with the parameter g, which can either be a Line, a Surface, a Points or another MObjects geometry. The result type is the minimum of the two types in assumed dimensional order: points < line < surface.

Processes::SpatioTemporalIntersection Methods

Method	Type	Notes
execute (*MObjects, Geometry*)	public: *MObjects*	param: ms [MObjects - in] param: g [Geometry - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::SpatioTemporalIntersects

Class

Extends: WebProcessingService. : Testing whether the time-varying position of the moving object and the specified geometry g share at least one point.

Processes::SpatioTemporalIntersects Methods

Method	Type	Notes
execute (*MObject, Geometry*)	public: *boolean*	param: m [MObject - in] param: g [Geometry - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::SpatioTemporalLeaves

Class

Extends: WebProcessingService. : Testing whether the time-varying position of the moving object is disjoint of the specified geometry g until they meet and then is inside of the specified geometry g.

Processes::SpatioTemporalLeaves Methods

Method	Type	Notes
execute (*MObject, Geometry*)	public: *boolean*	param: m [MObject - in] param: g [Geometry - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::SpatioTemporalMeets

Class

Extends: WebProcessingService. : Testing whether the time-varying position of the moving object and the specified geometry g intersect in a point while their interiors are disjoint.

Processes::SpatioTemporalMeets Methods

Method	Type	Notes
execute (*MObject, Geometry*)	public: *boolean*	param: m [MObject - in] param: g [Geometry - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::SpatioTemporalReleases

Class

Extends: WebProcessingService. : Testing whether the time-varying position of the moving object meets the specified geometry g and then is disjoint of the specified geometry g (is the opposite of snaps).

Processes::SpatioTemporalReleases Methods

Method	Type	Notes
execute (*MObject, Geometry*)	public: *boolean*	param: m [MObject - in] param: g [Geometry - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::SpatioTemporalSnaps

Class

Extends: WebProcessingService. : Testing whether the time-varying position of the moving object is disjoint of the specified geometry g and then meets the specified geometry g.

Processes::SpatioTemporalSnaps Methods

Method	Type	Notes
execute (*MObject, Geometry*)	public: *boolean*	param: m [MObject - in] param: g [Geometry - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::SpatioTemporalSubstract

Class

Extends: WebProcessingService. : Remove all values from this MObjects that are also in parameter g. The result type is also a MObjects since substracting a lower-dimensional value returns the MObjects unchanged and subtracting a higher-dimensional value does not increase the dimension of the MObjects.

Processes::SpatioTemporalSubstract Methods

Method	Type	Notes
execute (*MObjects, Geometry*)	public: *MObjects*	param: ms [MObjects - in] param: g [Geometry - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::SpatioTemporalTouches

Class

Extends: WebProcessingService. : Testing whether the time-varying position of the moving object is disjoint of the specified geometry g until they meet and then is disjoint again.

Processes::SpatioTemporalTouches Methods

Method	Type	Notes
execute (*MObject, Geometry*)	public: *boolean*	param: m [MObject - in] param: g [Geometry - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::Speed

Class

Extends: WebProcessingService. : Returns the speed of the moving object at instant i.

Processes::Speed Methods

Method	Type	Notes
execute (*MObject, Instant*)	public: *double*	param: m [MObject - in] param: i [Instant - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::StartPosition

Class
Extends: WebProcessingService. : Returns the first track point of the moving object's lifespan.

Processes::StartPosition Methods

Method	Type	Notes
execute (*MObject*)	public: *Point*	param: m [MObject - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::Starts

Class
Extends: WebProcessingService. : Testing whether the first instant of the lifecycle of the moving object and the specified time t are the same.

Processes::Starts Methods

Method	Type	Notes
execute (*MObject, Time*)	public: *boolean*	param: m [MObject - in] param: t [Time - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::Status

Class
Extends: WebProcessingService. : Returning the status of the moving object at a particular instant i.

Processes::Status Methods

Method	Type	Notes
execute (*MObject, Instant*)	public: *StatusT*	param: m [MObject - in] param: i [Instant - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::Succeeds

Class

Extends: WebProcessingService. : Testing whether the first instant of the lifecycle of the moving object is greater than the last instant of the specified time t.

Processes::Succeeds Methods

Method	Type	Notes
execute (*MObject, Time*)	public: *boolean*	param: m [MObject - in] param: t [Time - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::TemporalContains

Class

Extends: WebProcessingService. : Testing whether any instant of the specified time t belongs to the lifecycle of the moving object.

Processes::TemporalContains Methods

Method	Type	Notes
execute (*MObject, Time*)	public: *boolean*	param: m [MObject - in] param: t [Time - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::TemporalCovers

Class

Extends: WebProcessingService. : Testing whether the lifecycle of the moving object contains every instant of the specified time t.

Processes::TemporalCovers Methods

Method	Type	Notes
execute (*MObject, Time*)	public: *boolean*	param: m [MObject - in] param: t [Time - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::TemporalDisjoint

Class

Extends: WebProcessingService. : Testing whether the lifecycle of the moving object and the specified time t do not intersect.

Processes::TemporalDisjoint Methods

Method	Type	Notes
execute (*MObject, Time*)	public: *boolean*	param: m [MObject - in] param: t [Time - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::TemporalEquals

Class

Extends: WebProcessingService. : Testing whether the lifecycle of the moving object is equal to the specified time t.

Processes::TemporalEquals Methods

Method	Type	Notes
execute (*MObject, Time*)	public: *boolean*	param: m [MObject - in] param: t [Time - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::TemporalInside

Class

Extends: WebProcessingService. : Testing whether any instant of the lifecycle of the moving object belongs to the specified time t.

Processes::TemporalInside Methods

Method	Type	Notes
execute (*MObject, Time*)	public: *ExecuteResponse*	param: m [MObject - in] param: t [Time - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::TemporalIntersects

Class

Extends: WebProcessingService. : Testing whether the lifecycle of the moving object and the specified time t have at least one instant in common.

Processes::TemporalIntersects Methods

Method	Type	Notes
execute (*MObject, Time*)	public: *boolean*	param: m [MObject - in] param: t [Time - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::TemporalMeets

Class

Extends: WebProcessingService. : Testing whether the last instant of the lifecycle of the moving object is equal to the first instant of the specified time t or vice-versa.

Processes::TemporalMeets Methods

Method	Type	Notes
execute (*MObject, Time*)	public: *boolean*	param: m [MObject - in] param: t [Time - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::TemporalOverlaps

Class

Extends: WebProcessingService. : Testing whether the interior of the lifecycle of the moving object and the specified time t intersect, while the intersection of the lifecycle of the moving object and t is not equal to one of them.

Processes::TemporalOverlaps Methods

Method	Type	Notes
execute (*MObject, Time*)	public: *boolean*	param: m [MObject - in] param: t [Time - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Processes::Trajectory

Class

Extends: WebProcessingService. : Returns the line when the moving object is projected into the plane.

Processes::Trajectory Methods

Method	Type	Notes
execute (*MObject*)	public: *Line*	param: m [MObject - in] Provides the underlying functionality of the service and allows monitoring the progress of the process execution via status messages. Returns process result. The result of a process can be returned directly to the client or it can be stored at a web-accessible location which is referenced in the execute response document.

Java Model

This class diagram shows class dependencies and calls enabling a HTTP request to be forwarded from a client to the WPS process.

Java Model (PSM)

Package: prototypical implementation

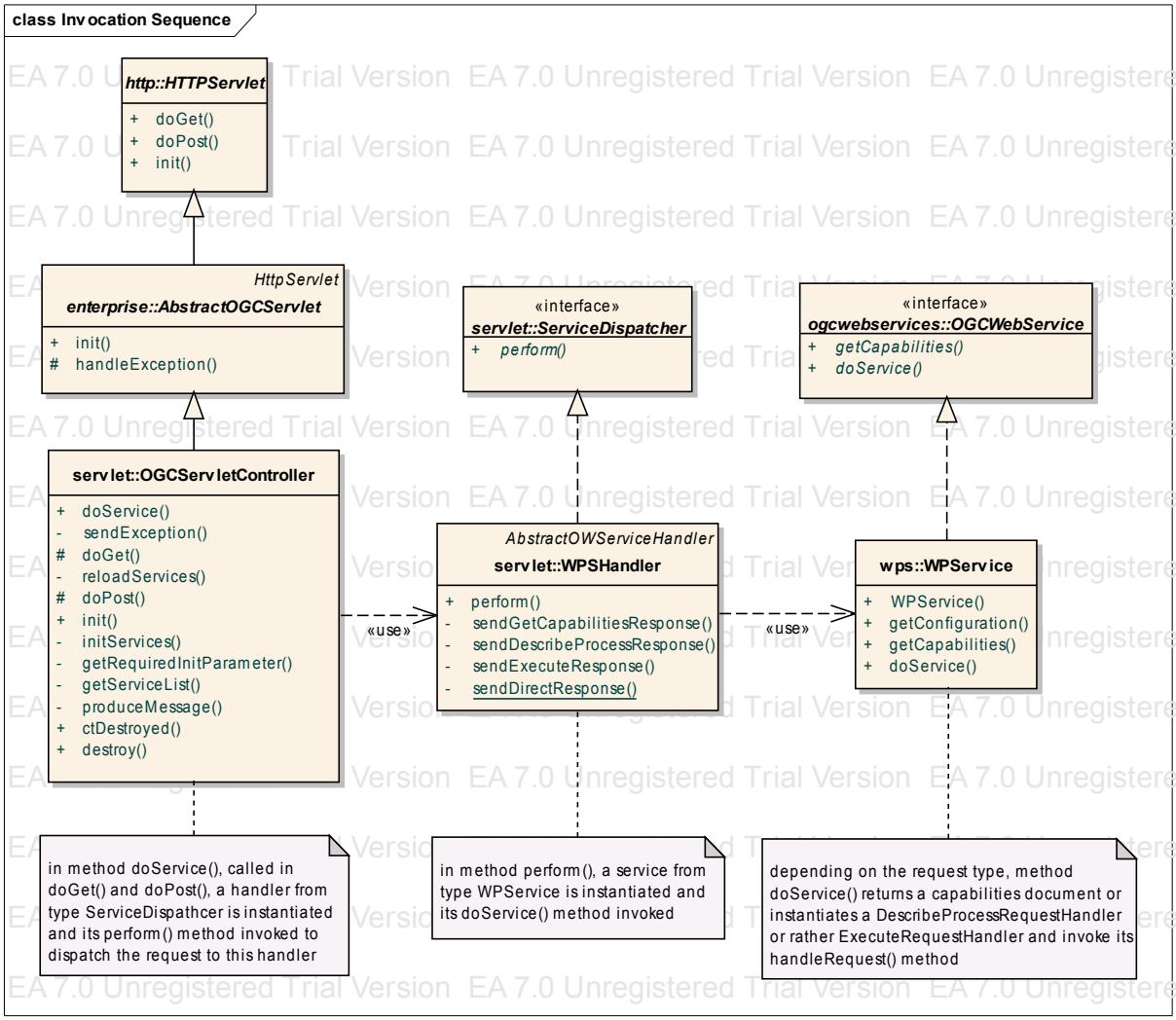
Note: in method doService(), called in doGet() and doPost(), a handler from type ServiceDispathcer is instantiated and its perform() method invoked to dispatch the request to this handler

Note: in method perform(), a service from type WPService is instantiated and its doService() method invoked

Note: depending on the request type, method doService() returns a capabilities document or instantiates a DescribeProcessRequestHandler or rather ExecuteRequestHandler and invoke its handleRequest() method

javax

Package: Root element of javax package hierarchy



servlet

Package: Servlet package

http

Package: Servlet binding to HTTP protocol

http::HTTPServlet

Class:

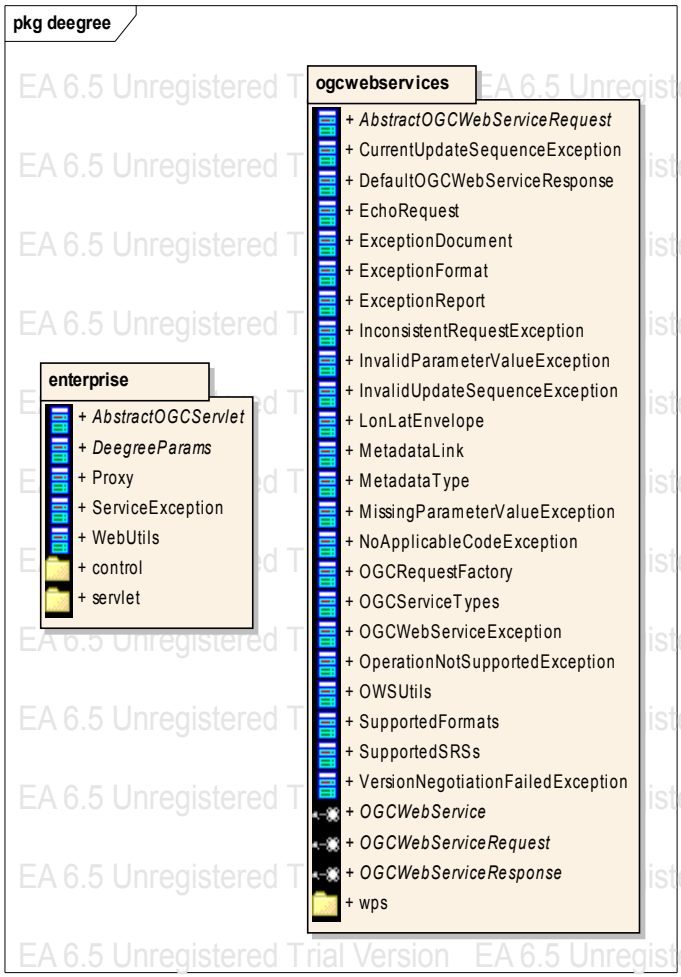
http::HTTPServlet Methods

Method	Type
doGet ()	public: *void*
doPost ()	public: *void*
init ()	public: *void*

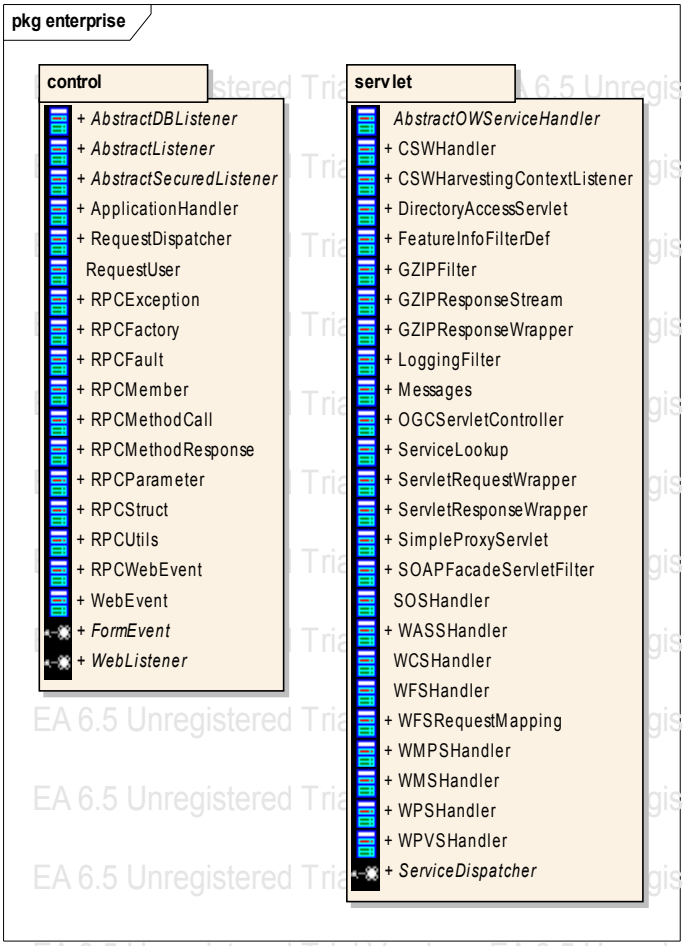
org

Package: Root element of deegree package hierarchy

deegree


pkg deegree
ogcwebservices
+ AbstractOGCWebServiceRequest
+ CurrentUpdateSequenceException
+ DefaultOGCWebServiceResponse
+ EchoRequest
+ ExceptionDocument
+ ExceptionFormat
+ ExceptionReport
+ InconsistentRequestException
+ InvalidParameterValueException
+ InvalidUpdateSequenceException
+ LonLatEnvelope
+ MetadataLink
+ MetadataType
+ MissingParameterValueException
+ NoApplicableCodeException
+ OGCRequestFactory
+ OGCServiceTypes
+ OGCWebServiceException
+ OperationNotSupportedException
+ OWSUtils
+ SupportedFormats
+ SupportedSRSs
+ VersionNegotiationFailedException
+ OGCWebService
+ OGCWebServiceRequest
+ OGCWebServiceResponse
+ wps
enterprise
+ AbstractOGCServlet
+ DeegreeParams
+ Proxy
+ ServiceException
+ WebUtils
+ control
+ servlet
EA 6.5 Unregistered Trial Version

enterprise



Package: Deegree apps, for building web applications

enterprise::AbstractOGCServlet

Class

Extends: HTTPServlet. Implements: HttpServlet. : Abstract servlet that serves as an OCC-compliant HTTP-frontend to any OGC-WebService (WFS, WMS, ...).
@todo refactoring required, move to package servlet

enterprise::AbstractOGCServlet Methods

Method	Type	Notes
init (*ServletConfig*)	public: *void*	param: servletConfig [ServletConfig - in] servlet configuration Called by the servlet container to indicate that the servlet is being placed into service. Sets the debug level according to the debug parameter defined in the ServletEngine's environment. <p> <p>
handleException (*String, Exception, HttpServletResponse*)	protected: *void*	param: msg [String - in] param: ex [Exception - in] param: response [HttpServletResponse - in] handles fatal errors by creating a OGC exception

		XML and sending it back to the client @deprecated

control

Package: not further described here

servlet

servlet::AbstractOWServiceHandler

Class

Implements: ServiceDispatcher. : This class provides methods that are common to all services that comply to the OWS Common Implementation Specification 0.3.0.
 At the moment, the only implemented functionality allows the sending of exception reports to the client, but in the future this may be extended by providing a method that sends responses to GetCapabilities requests.

@since 2.0

servlet::AbstractOWServiceHandler Attributes

Attribute	Type	Notes
RESPONSE_TYPE	private const static : *String*	private static final String RESPONSE_TYPE = "application/vnd.ogc.se_xml"; Initial Value: "text/xml";
LOG	private static : *ILogger*	Initial Value: LoggerFactory.getLogger(AbstractOWServiceHandler.class);

servlet::AbstractOWServiceHandler Methods

Method	Type	Notes
sendException (*HttpServletResponse, OGCWebServiceException*)	public: *void*	param: httpResponse [HttpServletResponse - in] param: serviceException [OGCWebServiceException - in] serviceException Sends an exception report to the client. The exception report complies to the OWS Common Implementation Specification 0.3.0.
sendException (*HttpServletResponse, Exception*)	public: *void*	param: httpResponse [HttpServletResponse - in] param: serviceException [Exception - in] serviceException Sends an exception report to the client. The exception report complies to the OWS Common Implementation Specification 0.3.0.

servlet::OGCServletController

Class

Extends: AbstractOGCServlet. : An <code>OGCServletController</code> handles all incoming requests. The controller for all OGC service requests. Dispatcher to specific handler for WMS, WFS and other.
@see Front controller

servlet::OGCServletController Attributes

Attribute	Type	Notes
address	public static : *String*	address is the url of the client which requests. Initial Value: null;
serialVersionUID	private const static	Initial Value: -4461759017823581221L;

	: *long*	
LOG	private const static : *ILogger*	Initial Value: LoggerFactory.getLogger(OGCServletController.class);
SERVICE	private const static : *String*	Initial Value: "services";
HANDLER_CLASS	private const static : *String*	Initial Value: ".handler";
HANDLER_CONF	private const static : *String*	Initial Value: ".config";
SERVICE_FACTORIES _MAPPINGS	private const static : *Map<Class, String>*	Initial Value: new HashMap<Class, String>();
ERR_MSG	private const static : *String*	Initial Value: "Can't set configuration for {0}";

servlet::OGCServletController Methods

Method	Type	Notes
doService (*HttpServletRequest, HttpServletResponse*)	public: *void*	param: request [HttpServletRequest - in] param: response [HttpServletResponse - in] @TODO refactor and optimize code for initializing handler
sendException (*HttpServletResponse, OGCWebServiceException, HttpServletRequest*)	private: *void*	param: response [HttpServletResponse - in] param: e [OGCWebServiceException - in] e param: request [HttpServletRequest - in] Sends the passed <tt>OGCWebServiceException</tt> to the calling client.
doGet (*HttpServletRequest, HttpServletResponse*)	protected: *void*	param: request [HttpServletRequest - in] param: response [HttpServletResponse - in] @see javax.servlet.http.HttpServlet#doGet(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)
reloadServices (*HttpServletRequest, HttpServletResponse*)	private: *void*	param: request [HttpServletRequest - in] param: response [HttpServletResponse - in]
doPost (*HttpServletRequest, HttpServletResponse*)	protected: *void*	param: request [HttpServletRequest - in] param: response [HttpServletResponse - in] (non-Javadoc) @see javax.servlet.http.HttpServlet#doPost(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)
init ()	public: *void*	@see javax.servlet.GenericServlet#init()
initServices (*ServletContext*)	private: *void*	param: context [ServletContext - in]
getRequiredInitParameter (*String*)	private: *String*	param: name [String - in]

getServiceList ()	private: *String*	@return the services, separated by ","
produceMessage (*String, Object[]*)	private: *String*	param: pattern [String - in] param: args [Object[] - in] Formats the provided string and the args array into a String using MessageFormat. @return the message to present the client.
ctDestroyed ()	public: *void*	@see javax.servlet.ServletContextListener#contextDestroyed(javax.servlet.ServletContextEvent)
destroy ()	public: *void*	

servlet::WPSHandler

Class
Extends: AbstractOWServiceHandler. Implements: ServiceDispatcher. : WPSHandler.java
Created on 08.03.2006. 17:01:31h
@since 2.0

servlet::WPSHandler Attributes

Attribute	Type	Notes
LOG	private const static : *ILogger*	Initial Value: LoggerFactory.getLogger(WPSHandler.class);

servlet::WPSHandler Methods

Method	Type	Notes
perform (*OGCWebServiceRequest, HttpServletResponse*)	public: *void*	param: request [OGCWebServiceRequest - in] param: httpServletResponse [HttpServletResponse - in]
sendGetCapabilitiesResponse (*HttpServletResponse, WPSCapabilities*)	private: *void*	param: httpResponse [HttpServletResponse - in] param: capabilities [WPSCapabilities - in] Sends the response to a GetCapabilities request to the client.
sendDescribeProcessResponse (*HttpServletResponse, ProcessDescriptions*)	private: *void*	param: httpResponse [HttpServletResponse - in] param: processDescriptions [ProcessDescriptions - in] Sends the response to a DescribeProcess request to the client. @param capabilities
sendExecuteResponse (*HttpServletResponse, ExecuteResponse*)	private: *void*	param: httpResponse [HttpServletResponse - in] param: executeResponse [ExecuteResponse - in] Sends the response to an Execute request to the client. @param httpServletResponse @param request
sendDirectResponse (*HttpServletResponse, ComplexValue*)	private static: *void*	param: httpResponse [HttpServletResponse - in] param: complexValue [ComplexValue - in] complexValue Writes the passed <code>ComplexValue</code> to the <code>HTTPServletResponse</code>

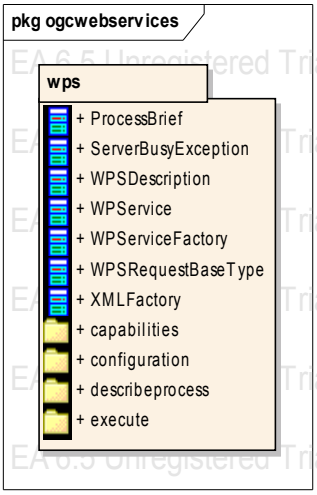
servlet::ServiceDispatcher

Interface: ServiceDispatcher
@since 2.0

servlet::ServiceDispatcher Interfaces

Method	Type	Notes
perform (*OGCWebServiceRequest, HttpServletResponse*)	public: *void*	param: request [OGCWebServiceRequest - in] param: response [HttpServletResponse - in]

ogcwebservices



ogcwebservices

Package: Deegree OWS framework, business logic

ogcwebservices::OGCWebService

Interface:

ogcwebservices::OGCWebService Interfaces

Method	Type	Notes
getCapabilities ()	public: *OGCCapabilities*	returns the capabilities of a OGC web service @return the capabilities of a OGC web service
doService (*OGCWebServiceRequest*)	public: *Object*	param: request [OGCWebServiceRequest - in] request (WMS, WCS, WFS, CSW, WFS-G, WMPS) to perform the implementation of this method performs the handling of the passed OGCWebServiceEvent directly and returns the result to the calling class/ method @return result of a service operation

ogcwebservices::OGCWebServiceRequest

Interface: This is the base interface for all request on OGC Web Services (OWS). Each class that capsulates a request against an OWS has to implements this interface.
@since 1.0

ogcwebservices::OGCWebServiceRequest Interfaces

Method	Type	Notes
getVendorSpecificParameters ()	public: *Map*	Finally, the requests allow for optional vendor-specific parameters (VSPs) that will enhance the results of a request. Typically, these are used for

		private testing of non-standard functionality prior to possible standardization. A generic client is not required or expected to make use of these VSPs. @return the vendor specificparameters
getVendorSpecificParameter (*String*)	public: *String*	param: name [String - in] the "key" of a vsp Finally, the requests allow for optional vendor-specific parameters (VSPs) that will enhance the results of a request. Typically, these are used for private testing of non-standard functionality prior to possible standardization. A generic client is not required or expected to make use of these VSPs. @return the value requested by the key
getId ()	public: *String*	@return the ID of a request
getVersion ()	public: *String*	@return the requested service version
getServiceName ()	public: *String*	@return the name of the service that is targeted by the request
getRequestParameter ()	public: *String*	@return the URI of a HTTP GET request. If the request doesn't support HTTP GET a <tt>WebServiceException</tt> will be thrown @deprecated should be replaced by a factory class TODO

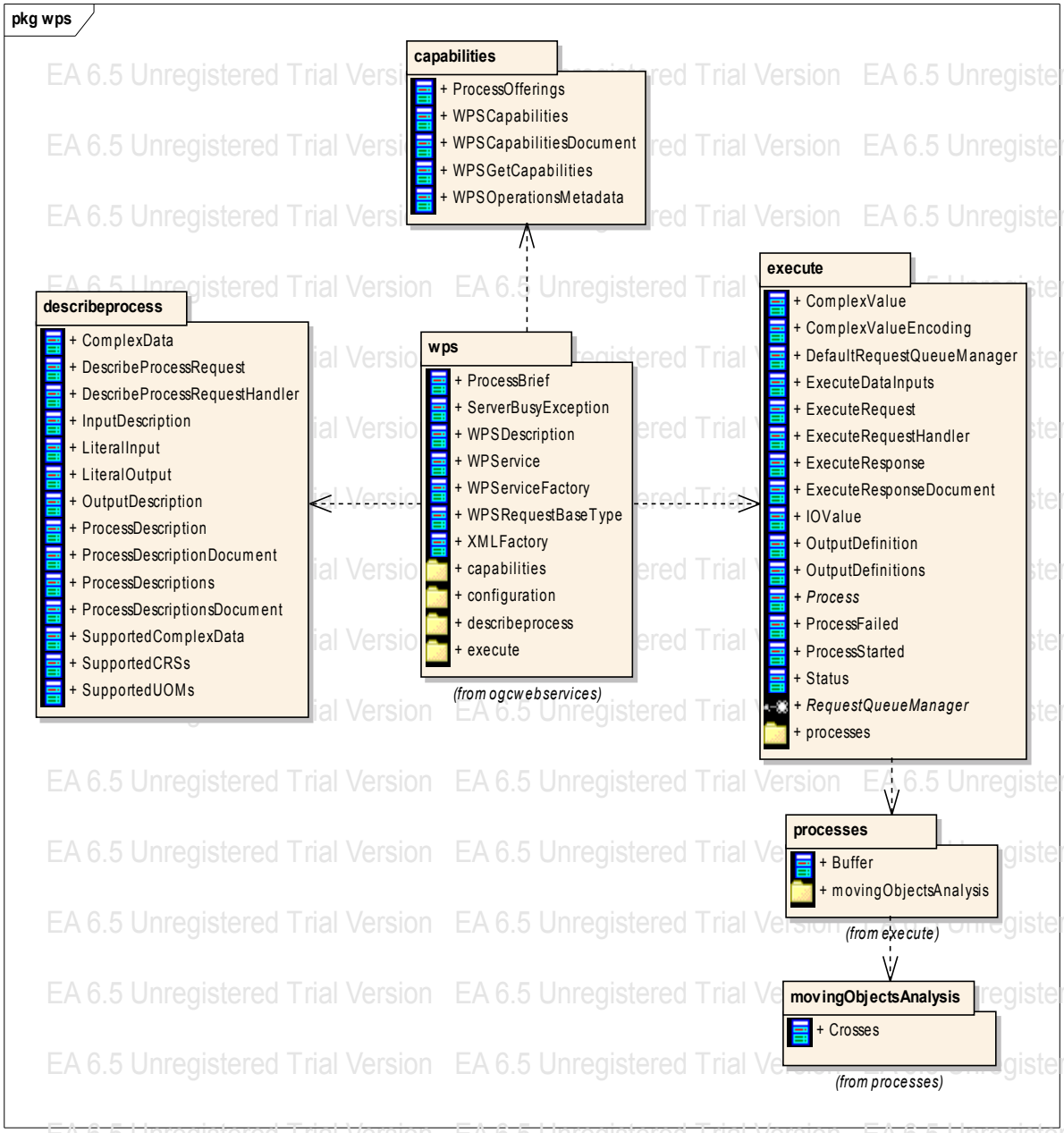
ogcwebservices::OGCWebServiceResponse

Interface: This is the base interface for all responses to OGC Web Services (OWS) requests. Each class that capsulates a response within an OWS has to implement this interface.

ogcwebservices::OGCWebServiceResponse Interfaces

Method	Type	Notes
getRequest ()	public: *OGCWebService Request*	returns the request that causes the response.
getException ()	public: *OGCWebService Exception*	returns an XML encoding of the exception that raised. If no exception raised <tt>null</tt> will be returned.

wps



wps

Package: Web Processing Service (WPS) package

wps::WPService

Class

Implements: OGCWebService. : WPService.java Created on 08.03.2006. 17:34:15h
@since 2.0

wps::WPService Attributes

Attribute	Type	Notes
LOG	private const static : *ILogger*	Initial Value: LoggerFactory.getLogger(WPService.class);
TP	private const static :	Initial Value: TriggerProvider.create(

	TriggerProvider	WPService.class);
configuration	private : *WPSConfiguration*	Initial Value: null;

wps::WPService Methods

Method	Type	Notes
WPService (*WPSConfiguration*)	public:	param: configuration [WPSConfiguration - in] configuration
getConfiguration ()	public: *WPSConfiguration*	@return Returns the configuration.
getCapabilities ()	public: *OGCCapabilities*	(non-Javadoc) @see org.deegree.ogcwebservices.OGCWebService#getCapabilities()
doService (*OGCWebServiceRequest*)	public: *Object*	param: request [OGCWebServiceRequest - in] (non-Javadoc) @see org.deegree.ogcwebservices.OGCWebService#doService(org.deegree.ogcwebservices.OGCWebServiceRequest)

capabilities

***Package*: not further described in this document**

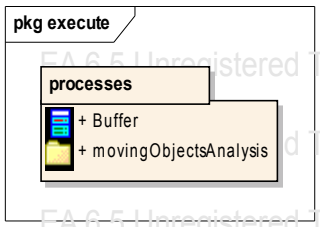
configuration

***Package*: not further described in this document**

describeprocess

***Package*: not further described in this document**

execute



execute::Process

***Class*:** Process.java Created on 11.03.2006. 18:32:39h

execute::Process Attributes

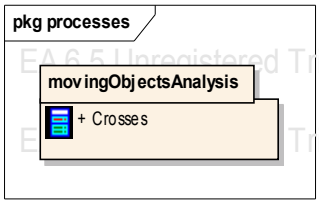
Attribute	Type
processDescription	protected : *ProcessDescription*

execute::Process Methods

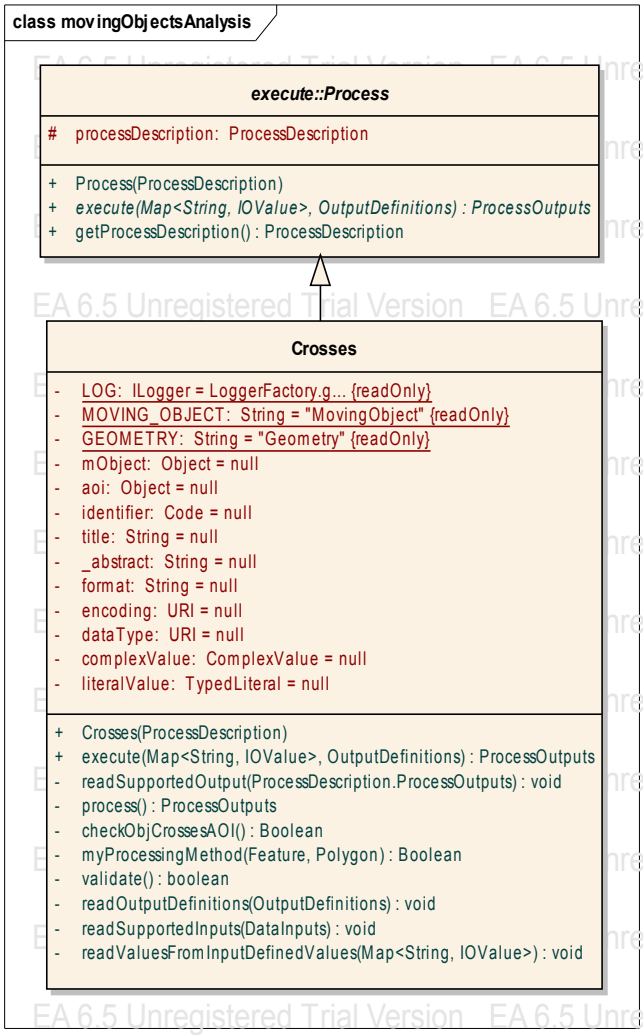
Method	Type	Notes
Process (*ProcessDescription*)	public:	param: processDescription [ProcessDescription - in] processDescription
execute (*Map<String, IOValue>,*	public abstract: *ProcessOutputs*	param: inputs [Map<String, IOValue> - in] param: outputDefinitions [OutputDefinitions - in

OutputDefinitions)		] @return
getProcessDescription ()	public: *ProcessDescription*	@return processDescription

processes



movingObjectsAnalysis



movingObjectsAnalysis::Crosses

Class

Extends: Process. : Crosses.java Created on 20.03.2008 This class describes an exemplary Process Implementation. The corresponding configuration document is '<root>\WEB-INF\conf\wps\processConfigs.xml'. Process configuration is described further inside the configuration document. The process implementor has to ensure, that the process implemented extends the

abstract super class Process. This example process IS NOT intended to describe a best practice approach. In some cases simplifying assumptions have been made for sake of simplicity.

movingObjectsAnalysis::Crosses Attributes

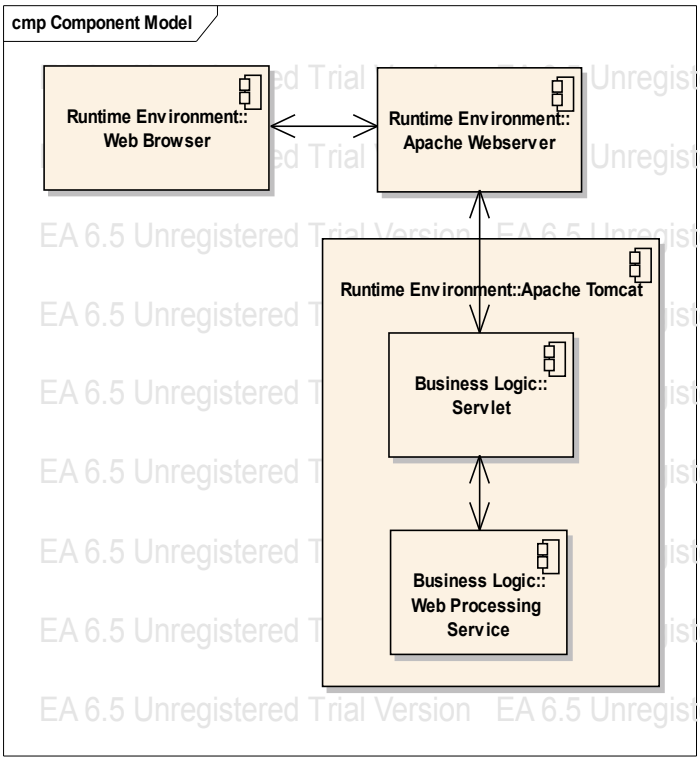
Attribute	Type	Notes
LOG	private const static : *ILogger*	define an ILogger for this class Initial Value: LoggerFactory.getLogger(Crosses.class);
MOVING_OBJECT	private const static : *String*	The provided crosses implementation is just a dummy returning hard coded results. In a operational implementation the processing should be done by a underlying Moving Objects Database. Crosses is a spatio-temporal predicate which is used to verify the relationships between a moving object and a geometry which can either be true or false. The crosses predicate takes two inputs: 1.) The moving object to apply the crosses predicate to 2.) The Geometry to to be tested whether the moving object first enters and then leaves it (polygon) The <wps:DataInputs> section defines two elements: MovingObject (mandatory) and Geometry (mandatory). Initial Value: "MovingObject";
GEOMETRY	private const static : *String*	Initial Value: "Geometry";
mObject	private : *Object*	object and aoi represent the <wps:ComplexData/> elements in the DataInputs section. This sample process is feeded with a movingObject and a polygon. Initial Value: null;
aoi	private : *Object*	Initial Value: null;
identifier	private : *Code*	Initial Value: null;
title	private : *String*	Initial Value: null;
_abstract	private : *String*	Initial Value: null;
format	private : *String*	Initial Value: null;
encoding	private : *URI*	Initial Value: null;
dataType	private : *URI*	values for ProcessOutput, will be filled dynamically Initial Value: null;
complexValue	private : *ComplexValue*	Initial Value: null;
literalValue	private : *TypedLiteral*	Initial Value: null;

movingObjectsAnalysis::Crosses Methods

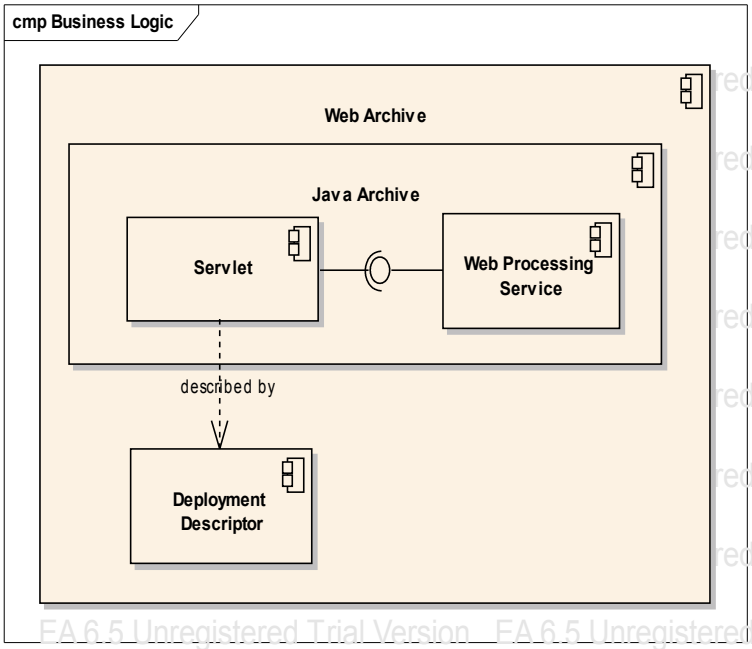
Method	Type	Notes
Crosses (*ProcessDescription*)	public:	param: processDescription [ProcessDescription - in] processDescription Constructor
execute (*Map<String, IOValue>, OutputDefinitions*)	public: *ProcessOutputs*	param: inputs [Map<String, IOValue> - in] param: outputDefinitions [OutputDefinitions - in] This is the central method for implementing a

		process. A <code>Map<String,IOValue></code> serves as an input object. Each String represents the key (e.g. MovingObject) which holds an IOValue as value (e.g. an object representing a complete <wps:Input> element with all corresponding sub-elements). The process implementation is responsible for retrieving all specified values according to the process configuration document. The method returns a <code>ProcessOutputs</code> object, which encapsulates the result of the process's operation.
readSupportedOutput (*ProcessDescription.ProcessOutputs*)	private: *void*	param: configuredProcessOutput [ProcessDescription.ProcessOutputs - in] configuredProcessOutput Private method that reads configured output for process output
process ()	private: *ProcessOutputs*	Private method for creating the processOutputs data structure and initiating processing @return boolean value
checkObjCrossesAOI ()	private: *Boolean*	Private method for reading the input data and calling the actual crosses process @return boolean value
myProcessingMethod (*Feature, Polygon*)	private: *Boolean*	param: movingObj [Feature - in] dynamicFeature to apply crosses predicate to param: polygon [Polygon - in] geometry to be tested whether movingObj first enters and then leaves it Private method for implementing the actual crosses process. This is just a dummy process were no real algorithms were applied. According to the moving objects ID, the crosses predicate is set either to true (for MObject_2) or false (for MObject_1) respectively. @return boolean value
validate ()	private: *boolean*	Private method for validating provided input parameters against configured input parameters. Actually, this is a very sophisticated implementation. @return
readOutputDefinitions (*OutputDefinitions*)	private: *void*	param: outputDefinitions [OutputDefinitions - in] outputDefinitions FIXME Assumes (simplified for the actual process) that only one output is defined. Private method that reads the output definitions into local variables.
readSupportedInputs (*DataInputs*)	private: *void*	param: configuredDataInputs [DataInputs - in] configuredDataInputs Private method that reads configured data inputs for validation
readValuesFromInputDefinedValues (*Map<String, IOValue>*)	private: *void*	param: inputs [Map<String, IOValue> - in] Private method for assigning input values to local variables

Component Model



Business Logic



Business Logic::Web Archive

Component: Web application, packed into a WAR file

Business Logic::Deployment Descriptor

Component: Servlet metadata, stored in the web.xml file

Business Logic::Java Archive

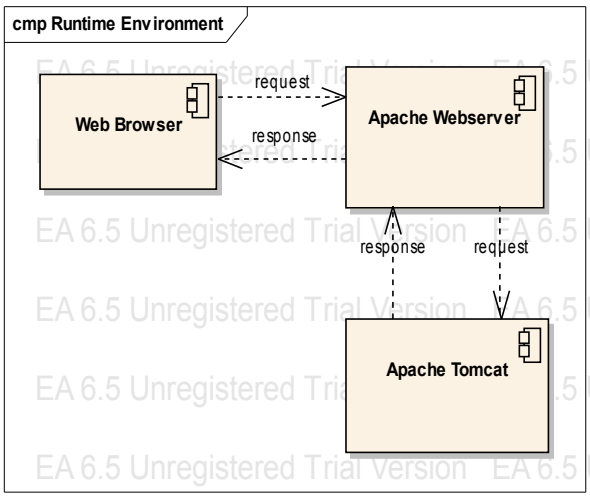
Component: Business logic and classes for building web applications, packed into a JAR file

Business Logic::Servlet

Component: Java Servlet from the deegree OWS Java framework

Business Logic::Web Processing Service

Component: ***WPS and Process from the deegree OWS Java framework Runtime Environment***
defines how the system is structured



Runtime Environment::Apache Tomcat

Component: Java Servlet Container

Runtime Environment::Apache Webserver

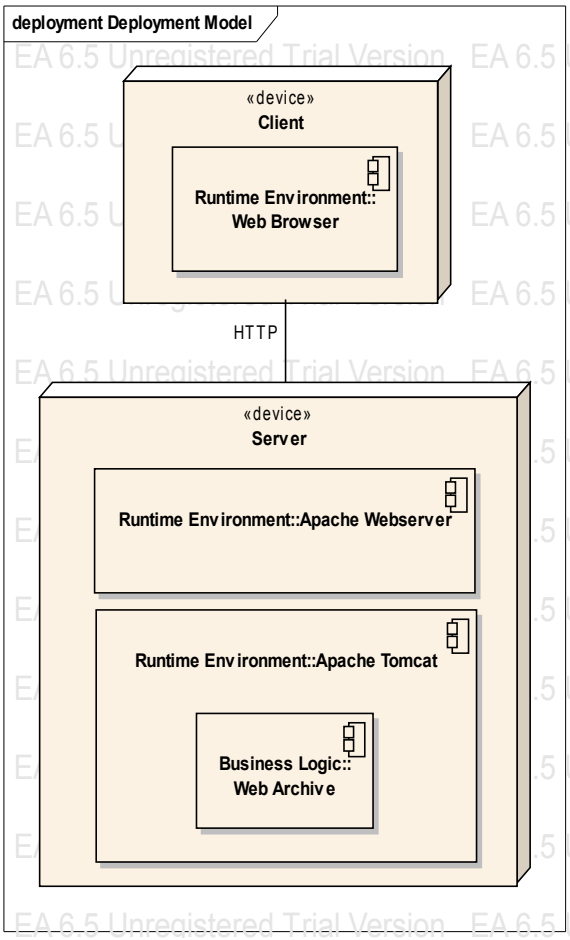
Component:

Runtime Environment::Web Browser

Component: dsfsdfsdfdsf

Deployment Model

Describes how and where system components are deployed onto physical nodes



XML Schema

Moving Objects GML Application Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2008 (http://www.altova.com) by Tobias Fleischmann (UNIGIS) -->
<xsd:schema xmlns:mObj="http://localhost:8080/deegreeWPS" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:gml="http://www.opengis.net/gml" targetNamespace="http://localhost:8080/deegreeWPS"
elementFormDefault="qualified" attributeFormDefault="unqualified">
      <xsd:import namespace="http://www.opengis.net/gml"
schemaLocation="C:\Daten\XSD\SCHEMAS_OPENGIS_NET\gml\3.1.1\base\gml.xsd"/>
      <xsd:simpleType name="ClassificationType">
            <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="onroad"/>
                  <xsd:enumeration value="offroad"/>
                  <xsd:enumeration value="aircraft"/>
                  <xsd:enumeration value="watercraft"/>
                  <xsd:enumeration value="railway"/>
            </xsd:restriction>
      </xsd:simpleType>
      <xsd:complexType name="StatusType">
            <xsd:simpleContent>
                  <xsd:restriction base="gml:StringOrRefType">
                        <xsd:enumeration value="active"/>
                        <xsd:enumeration value="inactive"/>
                        <xsd:enumeration value="disabled"/>
                  </xsd:restriction>
            </xsd:simpleContent>
      </xsd:complexType>
      <xsd:complexType name="PositionType">
            <xsd:complexContent>
                  <xsd:restriction base="gml:LocationPropertyType">
```

```
                  <xsd:sequence minOccurs="0">
                        <xsd:choice>
                              <xsd:element ref="gml:Point"/>
                              <xsd:element ref="gml:Null"/>
                        </xsd:choice>
                  </xsd:sequence>
            </xsd:restriction>
      </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="TimeType">
      <xsd:complexContent>
            <xsd:restriction base="gml:TimePrimitivePropertyType">
                  <xsd:sequence minOccurs="0">
                        <xsd:element ref="gml:TimeInstant"/>
                  </xsd:sequence>
            </xsd:restriction>
      </xsd:complexContent>
</xsd:complexType>
<xsd:group name="StaticProperties">
      <xsd:sequence>
            <xsd:element ref="mObj:Identifier"/>
            <xsd:element ref="mObj:DateOfBirth"/>
            <xsd:element ref="mObj:DateOfDeath"/>
            <xsd:element ref="mObj:Classification" minOccurs="0"/>
      </xsd:sequence>
</xsd:group>
<xsd:complexType name="MyMovingObjectStatusType">
      <xsd:complexContent>
            <xsd:extension base="gml:MovingObjectStatusType"/>
      </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="MovingObjectType">
      <xsd:complexContent>
            <xsd:extension base="gml:DynamicFeatureType">
                  <xsd:group ref="mObj:StaticProperties"/>
            </xsd:extension>
      </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="MovingGroupType">
      <xsd:complexContent>
            <xsd:extension base="gml:FeatureCollectionType">
```

```
                        <xsd:sequence>
                              <xsd:element name="groupIdentifier" type="xsd:integer"/>
                        </xsd:sequence>
                  </xsd:extension>
            </xsd:complexContent>
      </xsd:complexType>
      <xsd:element name="Identifier" type="xsd:string"/>
      <xsd:element name="DateOfBirth" type="gml:TimeInstantType"/>
      <xsd:element name="DateOfDeath" type="gml:TimeInstantType"/>
      <xsd:element name="Classification" type="mObj:ClassificationType"/>
      <xsd:element name="Status" type="mObj:StatusType" substitutionGroup="gml:status"/>
      <xsd:element name="Position" type="mObj:PositionType" substitutionGroup="gml:location"/>
      <xsd:element name="Time" type="mObj:TimeType" substitutionGroup="gml:validTime"/>
      <xsd:element name="MyMovingObjectStatus" type="mObj:MyMovingObjectStatusType"
substitutionGroup="gml:MovingObjectStatus"/>
      <xsd:element name="MovingObject" type="mObj:MovingObjectType" substitutionGroup="gml:_Feature"/>
      <xsd:element name="MovingObjectCollection" type="gml:FeatureCollectionType"
substitutionGroup="gml:_FeatureCollection"/>
      <xsd:element name="MovingGroup" type="mObj:MovingGroupType" substitutionGroup="gml:_FeatureCollection"/>
</xsd:schema>
```

XML Encoding

Deployment Descriptor

```
<?xml version="1.0"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-
app_2_3.dtd">
<web-app>
  <display-name>deegree 2.0</display-name>
  <description>deegree 2.0 OWS</description>
  <servlet>
    <servlet-name>owservice</servlet-name>
    <servlet-class>org.deegree.enterprise.servlet.OGCServletController</servlet-class>

    <init-param>
      <param-name>services</param-name>
      <param-value>wps</param-value>
      <description>
        list of supported services, e.g.: wfs,wms (comma separated), will be replaced by ant
      </description>
    </init-param>

    <!-- WMS INITIALIZING PARAMETERS -->
    <init-param>
      <param-name>wms.handler</param-name>
      <param-value>org.deegree.enterprise.servlet.WMSHandler</param-value>
    </init-param>
    <init-param>
      <param-name>wms.config</param-name>
      <param-value>WEB-INF/conf/wms/wms_1_3_0_reference_implementation_configuration.xml</param-value>
      <!-- Replace with the following line for a 1.1.1 reference implementation. -->
      <!--     <param-value>WEB-INF/conf/wms/wms_1_1_1_reference_implementation_configuration.xml</param-value>-->
    </init-param>
```

```
<!-- WFS INITIALIZING PARAMETERS -->
<init-param>
  <param-name>wfs.handler</param-name>
  <param-value>org.deegree.enterprise.servlet.WFSHandler</param-value>
</init-param>
<init-param>
  <param-name>wfs.config</param-name>
  <param-value>WEB-INF/conf/wfs/example_wfs_configuration.xml</param-value>
</init-param>

<!-- WPS INITIALIZING PARAMETERS -->
<init-param>
  <param-name>wps.handler</param-name>
  <param-value>org.deegree.enterprise.servlet.WPSHandler</param-value>
</init-param>
<init-param>
  <param-name>wps.config</param-name>
  <param-value>WEB-INF/conf/wps/wps_capabilities.xml</param-value>
</init-param>

<!-- WSS INITIALIZING PARAMETERS -->
<init-param>
  <param-name>wss.handler</param-name>
  <param-value>org.deegree.enterprise.servlet.WASSHandler</param-value>
</init-param>
<init-param>
  <param-name>wss.config</param-name>
  <param-value>WEB-INF/conf/wass/wss/example_wss_capabilities.xml</param-value>
</init-param>

<!-- WAS INITIALIZING PARAMETERS -->
<init-param>
  <param-name>was.handler</param-name>
  <param-value>org.deegree.enterprise.servlet.WASSHandler</param-value>
</init-param>
<init-param>
  <param-name>was.config</param-name>
  <param-value>WEB-INF/conf/wass/was/example_was_capabilities.xml</param-value>
</init-param>
```

```
<!-- WCS INITIALIZING PARAMETERS -->
<init-param>
  <param-name>wcs.handler</param-name>
  <param-value>org.deegree.enterprise.servlet.WCSHandler</param-value>
</init-param>
<init-param>
  <param-name>wcs.config</param-name>
  <param-value>WEB-INF/conf/wcs/example_wcs_capabilities.xml</param-value>
</init-param>

<!-- CSW INITIALIZING PARAMETERS -->
<init-param>
  <param-name>csw.handler</param-name>
  <param-value>org.deegree.enterprise.servlet.CSWHandler</param-value>
</init-param>
<init-param>
  <param-name>csw.config</param-name>
  <param-value>WEB-INF/conf/csw/example_csw_configuration.xml</param-value>
</init-param>

<!-- SOS INITIALIZING PARAMETERS -->
<init-param>
  <param-name>sos.handler</param-name>
  <param-value>org.deegree.enterprise.servlet.SOSHandler</param-value>
</init-param>
<init-param>
  <param-name>sos.config</param-name>
  <param-value>WEB-INF/conf/sos/example_sos_capabilities.xml</param-value>
</init-param>

<!-- WPVS INITIALIZING PARAMETERS -->
<init-param>
  <param-name>wpvs.handler</param-name>
  <param-value>org.deegree.enterprise.servlet.WPVSHandler</param-value>
</init-param>
<init-param>
  <param-name>wpvs.config</param-name>
  <param-value>WEB-INF/conf/wpvs/wpvs_configuration.xml</param-value>
</init-param>
```

```
        <!-- WPVS-client INITIALIZING PARAMETERS -->
    <init-param>
      <param-name>wpvc.handler</param-name>
      <param-value>org.deegree.enterprise.servlet.WPVSHandler</param-value>
    </init-param>
    <init-param>
      <param-name>wpvc.config</param-name>
      <param-value>WEB-INF/conf/wpvs/wpvs_configuration.xml</param-value>
    </init-param>

    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>owservice</servlet-name>
    <url-pattern>/services</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>/index.jsp</welcome-file>
  </welcome-file-list>

  <error-page>
    <error-code>500</error-code>
    <location>/error.jsp</location>
  </error-page>

  <error-page>
    <exception-type>org.deegree.ogcwebservices.OGCWebServiceException</exception-type>
    <location>/error.jsp</location>
  </error-page>

</web-app>
```

Capabilities Document

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This file is part of the deegree wps implementation. It represents an exemplary configuration document for a
wps capabilities interface. -->
<!-- Only wps specific elements will be explained below. The implementation is based on version 0.4.0 of the OGC Web
```

```
Processing Service Specification. -->
<Capabilities xmlns:xlink="http://www.w3.org/1999/xlink" version="0.4.0" xmlns:wps="http://www.opengis.net/wps"
xmlns:ows="http://www.opengis.net/ows" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wps ..\wpsGetCapabilities.xsd"
xmlns:deegree="http://www.deegree.org/wps">
      <!-- deegreeParams are for internal configuration of the deegree framework-->
      <deegree:deegreeParams>
            <deegree:DefaultOnlineResource xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="simple"
xlink:href="http://localhost:8080/wps_deegree/wps"/>
            <deegree:CacheSize>100</deegree:CacheSize>
            <deegree:RequestTimeLimit>35</deegree:RequestTimeLimit>
            <!-- The ProcessDirectoryList will be scanned recursively for processes which will be automatically
registered to the wps server. The path is relative to the file you are currently reading.-->
            <deegree:ProcessDirectoryList>
                  <deegree:ProcessDirectory>processConfigs</deegree:ProcessDirectory>
            </deegree:ProcessDirectoryList>
            <!-- The RequestQueueManager is responsible for storing requests. This feature will be essential at the
time the server supports storage of output.
                        Any RequestQueueManager implementation shall implement the
org.deegree.ogcwebservices.wps.execute.RequestQueueManager interface.
                        The current DefaultRequestQueueManager simply stores requests to a map and retrieves them
afterwards from the map. -->
            <deegree:RequestQueueManager>

<deegree:ResponsibleClass>org.deegree.ogcwebservices.wps.execute.DefaultRequestQueueManager</deegree:ResponsibleClas
s>
            </deegree:RequestQueueManager>
      </deegree:deegreeParams>
      <!-- The remainder of this file is a standard capabilities description. -->
      <ows:ServiceIdentification>
            <ows:Title>deegree WPS Server</ows:Title>
            <ows:Abstract>deegree compliant WPS Server hosted by Tobias Fleischmann</ows:Abstract>
            <ows:Keywords>
                  <ows:Keyword>WPS</ows:Keyword>
                  <ows:Keyword>geospatial</ows:Keyword>
                  <ows:Keyword>geoprocessing</ows:Keyword>
            </ows:Keywords>
            <ows:ServiceType>WPS</ows:ServiceType>
            <ows:ServiceTypeVersion>0.2.0</ows:ServiceTypeVersion>
            <ows:ServiceTypeVersion>0.1.0</ows:ServiceTypeVersion>
```

```
            <ows:Fees>NONE</ows:Fees>
            <ows:AccessConstraints>NONE</ows:AccessConstraints>
      </ows:ServiceIdentification>
      <ows:ServiceProvider>
            <ows:ProviderName>Tobias Fleischmann</ows:ProviderName>
            <ows:ProviderSite xlink:href="http://www.unigis.ac.at/"/>
            <ows:ServiceContact>
                  <ows:IndividualName>Tobias Fleischmann</ows:IndividualName>
                  <ows:PositionName>UNIGIS MSc student</ows:PositionName>
                  <ows:ContactInfo>
                        <ows:Address>
                              <ows:DeliveryPoint>Richard-Wagner-Strasse 9</ows:DeliveryPoint>
                              <ows:City>Oberteuringen</ows:City>
                              <ows:AdministrativeArea>BW</ows:AdministrativeArea>
                              <ows:PostalCode>88094</ows:PostalCode>
                              <ows:Country>Germany</ows:Country>
                              <ows:ElectronicMailAddress>mt.u1249@web.de</ows:ElectronicMailAddress>
                        </ows:Address>
                  </ows:ContactInfo>
            </ows:ServiceContact>
      </ows:ServiceProvider>
      <ows:OperationsMetadata>
            <ows:Operation name="GetCapabilities">
                  <ows:DCP>
                        <ows:HTTP>
                              <ows:Get xlink:href="http://localhost:8080/deegreeWPS/services?"/>
                        </ows:HTTP>
                  </ows:DCP>
            </ows:Operation>
            <ows:Operation name="DescribeProcess">
                  <ows:DCP>
                        <ows:HTTP>
                              <ows:Get xlink:href="http://localhost:8080/deegreeWPS/services?"/>
                              <!-- <ows:Post xlink:href="http://localhost:8080/deegreeWPS/services"/> -->
                        </ows:HTTP>
                  </ows:DCP>
            </ows:Operation>
            <ows:Operation name="Execute">
                  <ows:DCP>
                        <ows:HTTP>
                              <!-- <ows:Get xlink:href="http://localhost:8080/deegreeWPS/services?"/> -->
```

```
                        <ows:Post xlink:href="http://localhost:8080/deegreeWPS/services"/>
                    </ows:HTTP>
                </ows:DCP>
            </ows:Operation>
        </ows:OperationsMetadata>
        <!-- The server will generate a dynamic process offerings section out of the process descriptions. -->
</Capabilities>
```

Process Description Document

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This file is part of the deegree wps implementation. It represents an exemplary process configuration document
for a buffer process. -->
<!-- Only wps specific elements will be explained below. The implementation is based on version 0.4.0 of the OGC Web
Processing Service Specification. -->
<!-- the root element defines the optional parameters store and status, which are currently not supported. -->
<wps:ProcessDescriptions xmlns:wps="http://www.opengeospatial.net/wps" xmlns:ows="http://www.opengis.net/ows"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:deegree="http://www.deegree.org/wps"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wps
..\wpsDescribeProcess.xsd">
<wps:ProcessDescription processVersion="1" storeSupported="false" statusSupported="false">
        <!-- deegreeParams are for internal configuration of the deegree framework-->
        <deegree:deegreeParams>
            <!-- The responsibleClass represents the actual processImplementing class, identified by it's fully
qualified classname. Each process shall only define one responsible class.
                Although it is of course possible to set up a detailed sub-package structure below the responsible
class. -->

<deegree:responsibleClass>org.deegree.ogcwebservices.wps.execute.processes.movingObjectsAnalysis.Crosses</deegree:re
sponsibleClass>
        </deegree:deegreeParams>
        <ows:Identifier>Crosses</ows:Identifier>
        <ows:Title>Crosses Predicate</ows:Title>
        <ows:Abstract>Testing whether the time-varying position of a moving object first enters and then leaves the
specified geometry. Accepts a GML geometry and provides a boolean output.</ows:Abstract>
        <ows:Metadata xlink:title="spatio-temporal predicate"/>
        <ows:Metadata xlink:title="moving object"/>
        <ows:Metadata xlink:title="geometry"/>
```

```
      <ows:Metadata xlink:title="GML"/>
      <!-- BEGIN OF DATAINPUTS -->
      <!-- the DataInputs section describes the process inputs for the implementation. -->
      <wps:DataInputs>
            <!-- An Input element refers to a single input -->
            <wps:Input>
                  <ows:Identifier>MovingObject</ows:Identifier>
                  <ows:Title>Moving Object</ows:Title>
                  <ows:Abstract>Moving object to apply crosses predicate to.</ows:Abstract>
                  <wps:ComplexData>
                        <wps:SupportedComplexData>
                              <wps:Format>text/xml</wps:Format>
                              <wps:Encoding>UTF-8</wps:Encoding>
                              <wps:Schema>"http://localhost:8080/deegreeWPS/MovingObjects.xsd"</wps:Schema>
                        </wps:SupportedComplexData>
                  </wps:ComplexData>
                  <wps:MinimumOccurs>1</wps:MinimumOccurs>
            </wps:Input>
            <wps:Input>
                  <ows:Identifier>Geometry</ows:Identifier>
                  <ows:Title>Geometry</ows:Title>
                  <ows:Abstract>Geometry to be tested whether moving object first enters and then leaves
it.</ows:Abstract>
                  <wps:ComplexData>
                        <wps:SupportedComplexData>
                              <wps:Format>text/xml</wps:Format>
                              <wps:Encoding>UTF-8</wps:Encoding>
                              <wps:Schema>http://schemas.opengis.net/gml/3.0.0/base/gml.xsd</wps:Schema>
                        </wps:SupportedComplexData>
                  </wps:ComplexData>
                  <wps:MinimumOccurs>1</wps:MinimumOccurs>
            </wps:Input>
      </wps:DataInputs>
      <!-- END OF DATAINPUTS -->
      <!-- BEGIN OF PROCESSOUTPUTS-->
      <wps:ProcessOutputs>
            <!-- A process may define several outputs-->
            <wps:Output>
                  <ows:Identifier>Boolean</ows:Identifier>
                  <ows:Title>Boolean</ows:Title>
                  <ows:Abstract>Statement expressing whether or not the crosses relationships between the moving
```

```
object and the geometry is either true or false.</ows:Abstract>
                  <wps:LiteralOutput>
                        <ows:DataType ows:reference="urn:ogc:def:dataType:OGC:1.1:boolean">boolean</ows:DataType>
                        <wps:SupportedUOMs/>
                  </wps:LiteralOutput>
                  </wps:Output>
            </wps:ProcessOutputs>
      <!-- END OF PROCESSOUTPUTS-->
</wps:ProcessDescription>
</wps:ProcessDescriptions>
```

Execute Request Document

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute service="WPS" version="0.4.0" store="false" status="false" xmlns:ows="http://www.opengis.net/ows"
xmlns:wps="http://www.opengeospatial.net/wps" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengeospatial.net/wps C:\Daten\Doku\OGC\discusssion_paper\05-
007r4_Web_Processing_Service_WPS_v0_4_0\wps\0.4.0\wpsExecute.xsd">
      <ows:Identifier>Crosses</ows:Identifier>
      <wps:DataInputs>
            <wps:Input>
                  <ows:Identifier>MovingObject</ows:Identifier>
                  <ows:Title>Moving Object</ows:Title>
                  <ows:Abstract>Moving object to apply crosses predicate to.</ows:Abstract>
                  <wps:ComplexValue format="text/xml" encoding="UTF-8"
schema="http://localhost:8080/deegreeWPS/MovingObjects.xsd">
                        <mObj:MovingObject xmlns:mObj="http://localhost:8080/deegreeWPS"
xmlns:gml="http://www.opengis.net/gml">
                              <gml:description>Vehicle moving from Meersburg to Constance by crossing the Lake
Constance using a car ferry</gml:description>
                              <gml:srsName>EPSG:4326</gml:srsName>
                              <gml:track>
                                    <mObj:MyMovingObjectStatus>
                                          <gml:description>TrkPt_1: Meersburg</gml:description>
                                          <mObj:Time>
                                                <gml:TimeInstant>
                                                      <gml:timePosition>2008-03-02T10:30:00Z</gml:timePosition>
                                                </gml:TimeInstant>
```

```
            </mObj:Time>
            <mObj:Position>
                  <gml:Point>
                        <gml:pos srsDimension="2">47.6954467 9.2663681</gml:pos>
                  </gml:Point>
            </mObj:Position>
            <mObj:Status>active</mObj:Status>
      </mObj:MyMovingObjectStatus>
      <mObj:MyMovingObjectStatus>
            <gml:description>TrkPt_2: halfway ferry across the lake</gml:description>
            <mObj:Time>
                  <gml:TimeInstant>
                        <gml:timePosition>2008-03-02T10:40:00Z</gml:timePosition>
                  </gml:TimeInstant>
            </mObj:Time>
            <mObj:Position>
                  <gml:Point>
                        <gml:pos srsDimension="2">47.6879009 9.2346861</gml:pos>
                  </gml:Point>
            </mObj:Position>
            <mObj:Status>active</mObj:Status>
      </mObj:MyMovingObjectStatus>
      <mObj:MyMovingObjectStatus>
            <gml:description>TrkPt_3: Konstanz Staad</gml:description>
            <mObj:Time>
                  <gml:TimeInstant>
                        <gml:timePosition>2008-03-02T10:50:00Z</gml:timePosition>
                  </gml:TimeInstant>
            </mObj:Time>
            <mObj:Position>
                  <gml:Point>
                        <gml:pos srsDimension="2">47.6830500 9.2026031</gml:pos>
                  </gml:Point>
            </mObj:Position>
            <mObj:Status>active</mObj:Status>
      </mObj:MyMovingObjectStatus>
      <mObj:MyMovingObjectStatus>
            <gml:description>TrkPt_4: Konstanz Rheinbruecke</gml:description>
            <mObj:Time>
                  <gml:TimeInstant>
                        <gml:timePosition>2008-03-02T10:58:00Z</gml:timePosition>
```

```
                                    </gml:TimeInstant>
                                </mObj:Time>
                                <mObj:Position>
                                    <gml:Point>
                                        <gml:pos srsDimension="2">47.6668804 9.1789418</gml:pos>
                                    </gml:Point>
                                </mObj:Position>
                                <mObj:Status>active</mObj:Status>
                            </mObj:MyMovingObjectStatus>
                        </gml:track>
                        <mObj:Identifier>MObj_2</mObj:Identifier>
                        <mObj:DateOfBirth>
                            <gml:timePosition>2008-03-02T10:30:00Z</gml:timePosition>
                        </mObj:DateOfBirth>
                        <mObj:DateOfDeath>
                            <gml:timePosition>2008-03-02T10:58:00Z</gml:timePosition>
                        </mObj:DateOfDeath>
                        <mObj:Classification>onroad</mObj:Classification>
                    </mObj:MovingObject>
                </wps:ComplexValue>
            </wps:Input>
            <wps:Input>
                <ows:Identifier>Geometry</ows:Identifier>
                <ows:Title>Geometry</ows:Title>
                <ows:Abstract>Geometry to be tested whether moving object first enters and then leaves
it.</ows:Abstract>
                <wps:ComplexValue format="text/xml" encoding="UTF-8"
schema="http://schemas.opengis.net/gml/3.0.0/base/gml.xsd">
                    <gml:Polygon xmlns:gml="http://www.opengis.net/gml" srsName="EPSG:4326">
                        <gml:exterior>
                            <gml:LinearRing>
                                <gml:pos srsDimension="2">47.8057600 9.0250785</gml:pos>
                                <gml:pos srsDimension="2">47.8189652 9.0577710</gml:pos>
                                <gml:pos srsDimension="2">47.7397338 9.2361782</gml:pos>
                                <gml:pos srsDimension="2">47.7139522 9.2361782</gml:pos>
                                <gml:pos srsDimension="2">47.7139522 9.2371123</gml:pos>
                                <gml:pos srsDimension="2">47.6623889 9.3520028</gml:pos>
                                <gml:pos srsDimension="2">47.6749653 9.4089809</gml:pos>
                                <gml:pos srsDimension="2">47.6472972 9.5070582</gml:pos>
                                <gml:pos srsDimension="2">47.5982492 9.5444209</gml:pos>
```

```
                                        <gml:pos srsDimension="2">47.5221619 9.7513172</gml:pos>
                                        <gml:pos srsDimension="2">47.5039261 9.7307677</gml:pos>
                                        <gml:pos srsDimension="2">47.4775156 9.4851076</gml:pos>
                                        <gml:pos srsDimension="2">47.5536029 9.3720852</gml:pos>
                                        <gml:pos srsDimension="2">47.5705811 9.3870303</gml:pos>
                                        <gml:pos srsDimension="2">47.6680483 9.1423042</gml:pos>
                                        <gml:pos srsDimension="2">47.6686771 9.1423042</gml:pos>
                                        <gml:pos srsDimension="2">47.6743365 9.0134027</gml:pos>
                                        <gml:pos srsDimension="2">47.6435243 8.9227980</gml:pos>
                                        <gml:pos srsDimension="2">47.6435243 8.8816990</gml:pos>
                                        <gml:pos srsDimension="2">47.6441531 8.8816990</gml:pos>
                                        <gml:pos srsDimension="2">47.6611312 8.8788967</gml:pos>
                                        <gml:pos srsDimension="2">47.6605024 8.9358750</gml:pos>
                                        <gml:pos srsDimension="2">47.6969740 9.0031279</gml:pos>
                                        <gml:pos srsDimension="2">47.7284150 8.9358750</gml:pos>
                                        <gml:pos srsDimension="2">47.7460220 8.9610949</gml:pos>
                                        <gml:pos srsDimension="2">47.7064063 9.0956008</gml:pos>
                                        <gml:pos srsDimension="2">47.6718212 9.1425045</gml:pos>
                                        <gml:pos srsDimension="2">47.6629279 9.2061546</gml:pos>
                                        <gml:pos srsDimension="2">47.6685872 9.2169631</gml:pos>
                                        <gml:pos srsDimension="2">47.6898774 9.2061546</gml:pos>
                                        <gml:pos srsDimension="2">47.7079335 9.1757307</gml:pos>
                                        <gml:pos srsDimension="2">47.7346134 9.1725281</gml:pos>
                                        <gml:pos srsDimension="2">47.8057600 9.0250785</gml:pos>
                                    </gml:LinearRing>
                                </gml:exterior>
                            </gml:Polygon>
                        </wps:ComplexValue>
                </wps:Input>
        </wps:DataInputs>
        <wps:OutputDefinitions>
                <wps:Output format="text/xml" encoding="UTF-8"
schema="http://schemas.opengis.net/gml/3.0.0/base/basicTypes.xsd" uom="urn:ogc:def:dataType:OGC:1.1:boolean">
                        <ows:Identifier>Boolean</ows:Identifier>
                        <ows:Title>Boolean</ows:Title>
                        <ows:Abstract>Statement expressing whether or not the crosses relationships between the moving
object and the geometry is either true or false.</ows:Abstract>
                </wps:Output>
        </wps:OutputDefinitions>
</wps:Execute>
```

Test Data (GML)

```
<?xml version="1.0" encoding="UTF-8"?>
<gml:Polygon xmlns:gml="http://www.opengis.net/gml" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/gml C:\Daten\XSD\SCHEMAS_OPENGIS_NET\gml\3.1.1\base\geometryBasic2d.xsd"
srsName="EPSG:4326">
      <gml:exterior>
            <gml:LinearRing>
                  <gml:pos srsDimension="2">47.8057600 9.0250785</gml:pos>
                  <gml:pos srsDimension="2">47.8189652 9.0577710</gml:pos>
                  <gml:pos srsDimension="2">47.7397338 9.2361782</gml:pos>
                  <gml:pos srsDimension="2">47.7139522 9.2361782</gml:pos>
                  <gml:pos srsDimension="2">47.7139522 9.2371123</gml:pos>
                  <gml:pos srsDimension="2">47.6623889 9.3520028</gml:pos>
                  <gml:pos srsDimension="2">47.6749653 9.4089809</gml:pos>
                  <gml:pos srsDimension="2">47.6472972 9.5070582</gml:pos>
                  <gml:pos srsDimension="2">47.5982492 9.5444209</gml:pos>
                  <gml:pos srsDimension="2">47.5221619 9.7513172</gml:pos>
                  <gml:pos srsDimension="2">47.5039261 9.7307677</gml:pos>
                  <gml:pos srsDimension="2">47.4775156 9.4851076</gml:pos>
                  <gml:pos srsDimension="2">47.5536029 9.3720852</gml:pos>
                  <gml:pos srsDimension="2">47.5705811 9.3870303</gml:pos>
                  <gml:pos srsDimension="2">47.6680483 9.1423042</gml:pos>
                  <gml:pos srsDimension="2">47.6686771 9.1423042</gml:pos>
                  <gml:pos srsDimension="2">47.6743365 9.0134027</gml:pos>
                  <gml:pos srsDimension="2">47.6435243 8.9227980</gml:pos>
                  <gml:pos srsDimension="2">47.6435243 8.8816990</gml:pos>
                  <gml:pos srsDimension="2">47.6441531 8.8816990</gml:pos>
                  <gml:pos srsDimension="2">47.6611312 8.8788967</gml:pos>
                  <gml:pos srsDimension="2">47.6605024 8.9358750</gml:pos>
                  <gml:pos srsDimension="2">47.6969740 9.0031279</gml:pos>
                  <gml:pos srsDimension="2">47.7284150 8.9358750</gml:pos>
                  <gml:pos srsDimension="2">47.7460220 8.9610949</gml:pos>
                  <gml:pos srsDimension="2">47.7064063 9.0956008</gml:pos>
                  <gml:pos srsDimension="2">47.6718212 9.1425045</gml:pos>
                  <gml:pos srsDimension="2">47.6629279 9.2061546</gml:pos>
                  <gml:pos srsDimension="2">47.6685872 9.2169631</gml:pos>
                  <gml:pos srsDimension="2">47.6898774 9.2061546</gml:pos>
```

```
                    <gml:pos srsDimension="2">47.7079335 9.1757307</gml:pos>
                    <gml:pos srsDimension="2">47.7346134 9.1725281</gml:pos>
                    <gml:pos srsDimension="2">47.8057600 9.0250785</gml:pos>
              </gml:LinearRing>
       </gml:exterior>
</gml:Polygon>


<?xml version="1.0" encoding="UTF-8"?>
<mObj:MovingObject xmlns:mObj="file:///C:/Daten/XSD" xmlns:gml="http://www.opengis.net/gml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="file:///C:/Daten/XSD
file:///C:/Daten/XSD/MovingObjects.xsd">
       <gml:description>Vehicle moving from Meersburg to Constance by crossing the Lake Constance using a car-
ferry</gml:description>
       <gml:name>MObject_2</gml:name>
       <gml:srsName>EPSG:4326</gml:srsName>
       <gml:track>
              <mObj:MyMovingObjectStatus>
                    <gml:description>TrkPt_1: Meersburg</gml:description>
                    <mObj:Time>
                          <gml:TimeInstant>
                                <gml:timePosition>2008-03-02T10:30:00Z</gml:timePosition>
                          </gml:TimeInstant>
                    </mObj:Time>
                    <mObj:Position>
                          <gml:Point>
                                <gml:pos srsDimension="2">47.6954467 9.2663681</gml:pos>
                          </gml:Point>
                    </mObj:Position>
                    <mObj:Status>active</mObj:Status>
              </mObj:MyMovingObjectStatus>
              <mObj:MyMovingObjectStatus>
                    <gml:description>TrkPt_2: halfway ferry across the lake</gml:description>
                    <mObj:Time>
                          <gml:TimeInstant>
                                <gml:timePosition>2008-03-02T10:40:00Z</gml:timePosition>
                          </gml:TimeInstant>
                    </mObj:Time>
                    <mObj:Position>
                          <gml:Point>
                                <gml:pos srsDimension="2">47.6879009 9.2346861</gml:pos>
                          </gml:Point>
                    </mObj:Position>
                    <mObj:Status>active</mObj:Status>
```

```
            </mObj:MyMovingObjectStatus>
            <mObj:MyMovingObjectStatus>
                  <gml:description>TrkPt_3: ferry port Staad</gml:description>
                  <mObj:Time>
                        <gml:TimeInstant>
                              <gml:timePosition>2008-03-02T10:50:00Z</gml:timePosition>
                        </gml:TimeInstant>
                  </mObj:Time>
                  <mObj:Position>
                        <gml:Point>
                              <gml:pos srsDimension="2">47.6822627 9.2111293</gml:pos>
                        </gml:Point>
                  </mObj:Position>
                  <mObj:Status>active</mObj:Status>
            </mObj:MyMovingObjectStatus>
            <mObj:MyMovingObjectStatus>
                  <gml:description>TrkPt_4: Constance</gml:description>
                  <mObj:Time>
                        <gml:TimeInstant>
                              <gml:timePosition>2008-03-02T10:58:00Z</gml:timePosition>
                        </gml:TimeInstant>
                  </mObj:Time>
                  <mObj:Position>
                        <gml:Point>
                              <gml:pos srsDimension="2">47.6831430 9.2024432</gml:pos>
                        </gml:Point>
                  </mObj:Position>
                  <mObj:Status>active</mObj:Status>
            </mObj:MyMovingObjectStatus>
      </gml:track>
      <mObj:Identifier>MObj_2</mObj:Identifier>
      <mObj:DateOfBirth>
            <gml:timePosition>2008-03-02T10:30:00Z</gml:timePosition>
      </mObj:DateOfBirth>
      <mObj:DateOfDeath>
            <gml:timePosition>2008-03-02T10:58:00Z</gml:timePosition>
      </mObj:DateOfDeath>
      <mObj:Classification>onroad</mObj:Classification>
</mObj:MovingObject>
```

Annex D

Java Source Code

Process Class

```
package org.deegree.ogcwebservices.wps.execute.processes.movingObjectsAnalysis;

import java.net.URI;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import org.deegree.datatypes.Code;
import org.deegree.datatypes.values.TypedLiteral;
import org.deegree.framework.log.ILogger;
import org.deegree.framework.log.LoggerFactory;
import org.deegree.model.feature.Feature;
import org.deegree.model.spatialschema.Polygon;
import org.deegree.ogcwebservices.MissingParameterValueException;
import org.deegree.ogcwebservices.OGCWebServiceException;
import org.deegree.ogcwebservices.wps.describeprocess.OutputDescription;
import org.deegree.ogcwebservices.wps.describeprocess.ProcessDescription;
import org.deegree.ogcwebservices.wps.describeprocess.ProcessDescription.DataInputs;
import org.deegree.ogcwebservices.wps.execute.ComplexValue;
import org.deegree.ogcwebservices.wps.execute.ExecuteResponse;
import org.deegree.ogcwebservices.wps.execute.IOValue;
import org.deegree.ogcwebservices.wps.execute.OutputDefinition;
import org.deegree.ogcwebservices.wps.execute.OutputDefinitions;
import org.deegree.ogcwebservices.wps.execute.Process;
```

```
import org.deegree.ogcwebservices.wps.execute.ExecuteResponse.ProcessOutputs;

/**
 * Crosses.java
 *
 * Created on 20.03.2008
 *
 * This class describes an exemplary Process Implementation. The corresponding configuration
 * document is '<root>\WEB-INF\conf\wps\processConfigs.xml'. Process configuration is described
 * further inside the configuration document.
 *
 * The process implementor has to ensure, that the process implemented extends the abstract super
 * class Process.
 *
 * This example process IS NOT intended to describe a best practice approach. In some cases
 * simplifying assumptions have been made for sake of simplicity.
 *
 * @author Tobias Fleischmann
 * @version 1.0.
 *
 */

public class Crosses extends Process {

      /**
    * Constructor
    * @param processDescription
    */
      public Crosses(ProcessDescription processDescription) {
            super(processDescription);
      }

      // define an ILogger for this class
    private static final ILogger LOG = LoggerFactory.getLogger( Crosses.class );

      /**
     * The provided crosses implementation is just a dummy returning hard coded results. In a
     * operational implementation the processing should be done by a underlying Moving Objects Database.
     * Crosses is a spatio-temporal predicate which is used to verify the relationships between a moving
```

```
     * object and a geometry which can either be true or false.
     * The crosses predicate takes two inputs:
     *
     * 1.) The moving object to apply the crosses predicate to
     * 2.) The Geometry to to be tested whether the moving object first enters and then leaves
     *     it (polygon)
     *
     * The <wps:DataInputs> section defines two elements: MovingObject (mandatory) and Geometry
     * (mandatory).
     */

      private static final String MOVING_OBJECT = "MovingObject";
      private static final String GEOMETRY = "Geometry";

      /**
     * object and aoi represent the <wps:ComplexData/> elements in the DataInputs section. This
     * sample process is feeded with a movingObject and a polygon.
     */

      private Object mObject = null;
      private Object aoi = null;

      private Code identifier = null;

    private String title = null;

    private String _abstract = null;

    @SuppressWarnings("unused")
    private String format = null;
    @SuppressWarnings("unused")
    private URI encoding = null;

    /**
       * values for ProcessOutput, will be filled dynamically
       */
    private URI dataType = null;

    private ComplexValue complexValue = null;

    private TypedLiteral literalValue = null;
```

```
  /**
  * This is the central method for implementing a process. A <code>Map<String,IOValue></code>
* serves as an input object. Each String represents the key (e.g. MovingObject) which holds
* an IOValue as value (e.g. an object representing a complete <wps:Input> element with all
* corresponding sub-elements). The process implementation is responsible for retrieving all
* specified values according to the process configuration document.
*
* The method returns a <code>ProcessOutputs</code> object, which encapsulates the result of
* the process's operation.
*
* @param inputs
* @param outputDefinitions
* @throws OGCWebServiceException
*/
  public ProcessOutputs execute(Map<String, IOValue> inputs, OutputDefinitions outputDefinitions)
                                    throws OGCWebServiceException {

        // delegate the read out of parameters to a private method
    readValuesFromInputDefinedValues( inputs );

    // prepare validation of data inputs against process description
    // get configured ( = supported) inputs from processDescription
    DataInputs configuredDataInputs = processDescription.getDataInputs();
    // delegate the read out of configured ( = supported ) inputs to a private method
    readSupportedInputs( configuredDataInputs );

    // prepare filling of ProcessOutputs data structure, more precise dataType of LiteralValue
    // get configured ( = supported ) output from processDescription
    ProcessDescription.ProcessOutputs configuredProcessOutput = processDescription.getProcessOutputs();
    // delegate the read out of configured ( = supported ) output to a private method
    readSupportedOutput( configuredProcessOutput );

    // delegate the read out of configured outputs to a private method
    readOutputDefinitions( outputDefinitions );

    // define a processOutputs object
    ProcessOutputs processOutputs = null;
```

```
        // validate, that data inputs correspond to process description
        boolean isValid = validate();

        // initiate processing if validation returns ok, otherwise throw exception
        if ( isValid ) {
            processOutputs = process();
        } else {
            throw new OGCWebServiceException( getClass().getName(), "The configuration is invalid:" +
                        " retry with data inputs according to process description" );
        }

        return processOutputs;
    }

    /**
    * Private method that reads configured output for process output
    *
    * @param configuredProcessOutput
    */
    private void readSupportedOutput( ProcessDescription.ProcessOutputs configuredProcessOutput) {

        // define an outputDescription object
        OutputDescription outputDescription = null;

        // get list of supported WPS output descriptions
        List<OutputDescription> outputDescriptions = configuredProcessOutput.getOutput();

        // get first and only list entry because this process has only one output
        Iterator<OutputDescription> outputDescriptionIterator = outputDescriptions.iterator();
        if ( outputDescriptionIterator.hasNext() )
              outputDescription = outputDescriptionIterator.next();

        // set member variable needed for process output
        this.dataType = outputDescription.getLiteralOutput().getDataType().getAbout();
    }

    /**
    * Private method for creating the processOutputs data structure and initiating processing
    *
    * @return boolean value
    * @throws OGCWebServiceException
```

```
*/
 private ProcessOutputs process()
                                    throws OGCWebServiceException {

       // create empty ProcessOutputs data structure
       ProcessOutputs processOutputs = new ExecuteResponse.ProcessOutputs();

       // initiate processing
   Boolean predicate = checkObjCrossesAOI();

   // cast explicitly to a String
   String value = predicate.toString();

   // fill ProcessOutputs data structure with processing result
   this.literalValue = new TypedLiteral(value, dataType);

   IOValue ioValue = new IOValue( this.identifier, this.title, this._abstract, null,
                                          this.complexValue, null, this.literalValue );
   List<IOValue> processOutputsList = new ArrayList<IOValue>( 1 );
   processOutputsList.add( ioValue );
   processOutputs.setOutputs( processOutputsList );

       return processOutputs;
 }

 /**
* Private method for reading the input data and calling the actual crosses process
*
* @return boolean value
* @throws OGCWebServiceException
*/
 private Boolean checkObjCrossesAOI()
                                        throws OGCWebServiceException {

       // initialize return value with true
       Boolean result = new Boolean ( true);

       // check type of input data
       if (!(mObject instanceof Feature))
```

```
                LOG.logError("can't cast moving object to feature -> returning result=false");
        if (!(aoi instanceof Polygon))
                LOG.logError("can't cast geometry to polygon -> returning result=false");

        if ( mObject instanceof Feature && aoi instanceof Polygon) {
                LOG.logInfo("successfully casted input data");
                // cast explicitly to a feature and a polygon respectively
                Feature mObj = (Feature) this.mObject;
                Polygon polygon = (Polygon) this.aoi;

                // call respective processing algorithm to perform analysis
                // e.g. a Moving Objects Database (MOD)
                result = myProcessingMethod( mObj, polygon);
        }
        return result;
 }

 /**
* Private method for implementing the actual crosses process
*
* @param movingObj: dynamic feature to apply crosses predicate to
* @param polygon: geometry to be tested whether movingObj first enters and then leaves it
* @return boolean value
* @throws OGCWebServiceException
*/
 private Boolean myProcessingMethod(Feature movingObj, Polygon polygon)
                                                throws OGCWebServiceException {

        // TODO: implement processing algorithm and return process result

        return true;
 }

 /**
* Private method for validating provided input parameters against configured input parameters.
* Actually, this is a very sophisticated implementation.
*
* @return
*/
 private boolean validate() {
```

```
        // dummy which replaces a real validation by a hard coded ok
        boolean isValid = true;

        return isValid;
 }

 /**
* FIXME Assumes (simplified for the actual process) that only one output is defined.
* Private method that reads the output definitions into local variables.
*
* @param outputDefinitions
*/
 private void readOutputDefinitions(OutputDefinitions outputDefinitions) {
        List<OutputDefinition> outputDefinitionList = outputDefinitions.getOutputDefinitions();
   Iterator<OutputDefinition> outputDefinitionListIterator = outputDefinitionList.iterator();
   while ( outputDefinitionListIterator.hasNext() ) {
        OutputDefinition outputDefinition = outputDefinitionListIterator.next();
        this._abstract = outputDefinition.getAbstract();
        this.title = outputDefinition.getTitle();
        this.identifier = outputDefinition.getIdentifier();
        this.format = outputDefinition.getFormat();
        this.encoding = outputDefinition.getEncoding();
   }
 }

 /**
* Private method that reads configured data inputs for validation
*
* @param configuredDataInputs
*/
 private void readSupportedInputs(DataInputs configuredDataInputs) {

        // TODO:
        // Get list of configured ( = supported ) WPSInputDescriptions from
        // configuredDataInputs, get inputDescription for each configured input and
        // write variables for each input separately
 }
```

```
      /**
     * Private method for assigning input values to local variables
     *
     * @param inputs
     * @throws OGCWebServiceException
     */
      private void readValuesFromInputDefinedValues(Map<String, IOValue> inputs)
                                          throws OGCWebServiceException {

            IOValue ioMovingObj = null;
            IOValue ioGeometryObj = null;

            // check for mandatory values
            if ((inputs.containsKey( MOVING_OBJECT )) && (inputs.containsKey( GEOMETRY ))){
                  ioMovingObj = inputs.get( MOVING_OBJECT );
              this.mObject = ioMovingObj.getComplexValue();
              ioGeometryObj = inputs.get( GEOMETRY );
              this.aoi = ioGeometryObj.getComplexValue();
            } else {
                throw new MissingParameterValueException( getClass().getName(), "A required Input Parameter is
missing." );
            }
      }

}
```