Seventh Framework Programme

Call FP7-ICT-2009-6

Project: 247708 - SUDPLAN

Project full title:

Sustainable Urban Development Planner
for Climate Change Adaptation

Companion Report - for Project Deliverable D3.2.3

Product Implementation V3

Due date of deliverable: 30/06/2012
Actual submission date: 28/06/2012

Document Control Page

Title	Product Implementation V3 – Companion Report	
Creator	CIS	
Editor	Sascha Schlobinski	
Description	Brief report to accompany the software (Product Implementation) V3	
Publisher	SUDPLAN Consortium	
Contributors	All partners	
Type	Text	
Format	MS Word	
Language	EN-GB	
Creation date	15/05/2012	
Version number	1.0	
Version date	28/06/2012	
Last modified by	SMHI LGi	
Rights	Copyright “SUDPLAN Consortium”. During the drafting process, access is generally limited to the Consortium.	
Audience	☐ internal ☒ public ☐ restricted, access granted to:	
Review status	☐ Draft ☒ WP Manager accepted ☒ Co-ordinator accepted	Where applicable: ☐ Accepted by the GA ☐ Accepted by the GA as public document
Action requested	☐ to be revised by Partners involved in the preparation of the Project Deliverable ☐ to be revised by all SUDPLAN Partners ☐ for approval of the WP Manager ☐ for approval of the Quality Manager ☐ for approval of the Project Co-ordinator ☐ for approval of the General Assembly	
Requested deadline	30/06/2012	

Revision history

Version	Date	Modified by	Comments
0.1	20/12/2011	SSc	Draft Structure
1.0	27/06/2012	SSc	Integrated contributions of MSc, JWe, PKu DSt produced final draft
1.0	28/06/2012	LGi	Co-ordination approval

Table of Contents

Figures

Tables

1. Management Summary

This document has been produced by the consortium of the European Project FP7-247708 Sustainable Urban Development Planner for Climate Change Adaptation (SUDPLAN). It is the companion report to the third deliverable of *T3.3 Product Implementation* which represents the software developed on the basis of the results of *T3.2 Product Prototyping* and *T3.1 Requirements Specification*. It is an update of the first and second report (D3.2.1, D3.2.2) and provides an overview of the distinct building blocks of the SUDPLAN platform and briefly explains their architecture and reports on the developments performed during the second product implementation phase.

1.1. Purpose of this Document

This document is a brief report to accompany the software developed in WP3 (Scenario Management System) in the final year of the SUDPLAN project bundled into *D3.2.3 Product Implementation V3*. The software described here constitutes the components that make up one of the core results of SUDPLAN - the Scenario Management System. In contrast to *D3.3.3 Scenario Management System V3* the developed software components are not presented as an integrated solution but rather as individual building blocks. The results of *D3.2.3* combined with the results of *D3.3.3* are the basis for the third and final validation cycle performed by the pilot applications.

1.2. Intended Audience

This document targets all SUDPLAN partners.

1.3. Summary and Structure of the Document

This document is the companion report to the software deliverable *D3.2.3 Product Implementation V3* and is divided into three main parts.

The first part is an introductory chapter that explains the role of the partners involved in the work performed, gives an overview on the overall development process, and explains the basic purpose of the Scenario Management System. Furthermore, it highlights the relationship of the software deliverable to other tasks and deliverables of WP3 and gives a rationale for the choices of technologies and products used to implement the Scenario Management System Building Blocks.

The second part of the report gives a brief outlook on the overall architecture of the integrated SMS presented in *D3.3.3 Scenario Management System V3*.

The third part of this document provides detailed information on the following for each distinct Building Block of the Scenario Management System:

- The general purpose of the components of the Building Blocks and their relationships to the Scenario Management System requirements specified in *D3.1.2 Requirement Specification V2*.

- The software and tools (such as APIs, products) used for the implementation of the components, including a description of their architecture and provided functionalities.

- The incremental results of the third product implementation phase including an overview of realised functionalities.

- A summary of the key features of the current release and references to relevant software repositories necessary to access the results of the development effort of the final project year.

The document closes with conclusions that can be used to gain an overview without reading the entire document.

1.4. Abbreviations and Acronyms

Acronym	Description
ABF	Administrators' Best Friend
ADC	Architecture and Data Committee (GEOSS)
Airviro	Air quality management system consisting of databases, dispersion models and utilities to facilitate data collection, emission inventories etc; see http://www.smhi.se/airviro
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CLI	Command-Line Interface
CS	Common Services
CTM	Chemistry Transport Model
DEM	Digital Elevation Model
DG-INFSO	Directorate General for Information Society (EC)
DoW	SUDPLAN Description of Work
DSS	Decision Support Systems
EC	European Commission
ECMWF	The European Centre for Medium-Range Weather Forecasts (also co-ordinating FP7-SPACE project MACC)
EDSS	Environmental DSS
EO	Earth Observation
ESA	European Space Agency
ESDI	European Spatial Data Infrastructure
EU	European Union
FP6/7	6th/7th Framework Programme (EC)
FRQ	Frequency of Occurrence
GeoJSON	Geo JavaScript Object Notation
GEOSS	Global Earth Observation System of Systems

Acronym	Description
GeoTIFF	Geo Tagged Image File Format
GIS	Geographic Information System
GMES	Global Monitoring for Environment and Security
GUI	Graphical User Interface
ICT	Information and Communication Technologies
ID	Identifier
IDF	Intensity Duration Frequency, curves or tables showing the statistical relationship between rain intensity and duration
INSPIRE	Infrastructure for Spatial Information in Europe
INTAMAP	INTeroperability and Automated MAPping (FP6 integrated project)
IO	Input & Output
ISO	International Standardization Organisation
IST	Information Society Technology
JRC	Joint Research Centre (EC)
KML	Keyhole Markup Language
MGRS	Military Grid Reference System
NASA	National Aeronautics and Space Administration
O&M	Observation and Measurements (OGC specification draft)
OASIS	1) Organization for the Advancement of Structured Information Standards 2) Open Advanced System for Disaster and Emergency Management (FP6 project)
OGC	Open Geospatial Consortium
OMG	Object Management Group
ORCHESTRA	Open Architecture and Spatial Data Infrastructure for Risk Management (FP6 integrated project)
SANY	SANY Sensors Anywhere (FP6 integrated project)
SAS	Sensor Alert Service (OGC specification draft)
SDI	Spatial Data Infrastructure
SDK	Software Development Kit
SEIS	Shared Environmental Information System
SensorML	Sensor Model Language
SISE	Single Information Space in Europe for the Environment
SMS	SUDPLAN Scenario Management System
SOS	OGC Sensor Observation Service
SPI	Service Provider Interface
SPS	OGC Sensor Planning Service
UTM	Universal Transverse Mercator
WFS	OGC Web Feature Service
WMS	OGC Web Map Service

2. Introduction

The main objective of WP3 work is to develop an '*easy-to-use web-based planning, prediction, decision support and training tool, for the use in an urban context, based on a what-if scenario execution environment*' (DoW).

The development work performed to provide the software was divided according to the WP3 partner's field of development expertise.

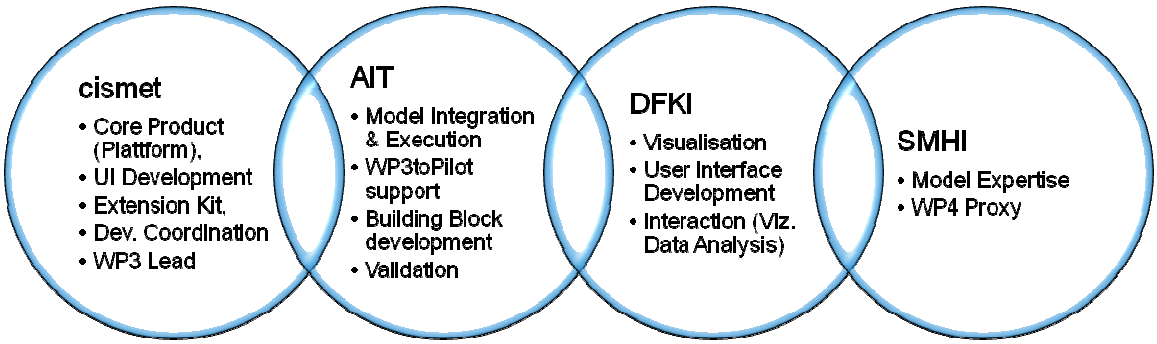

Figure 1: WP3 Partner Roles

The current implementation results which will be integrated into the Scenario Management System are classified as basic scenario management core functionality, model integration, and visualisation. With respect to the WP3 partner's responsibilities, the Scenario Management System can be divided into the three distinct Building Blocks **SMS Framework**, **Model as a Service Integration,** and **Advanced Visualisation.** These Building Blocks are further explained in Chapter *4 Scenario Management System Building Blocks*.

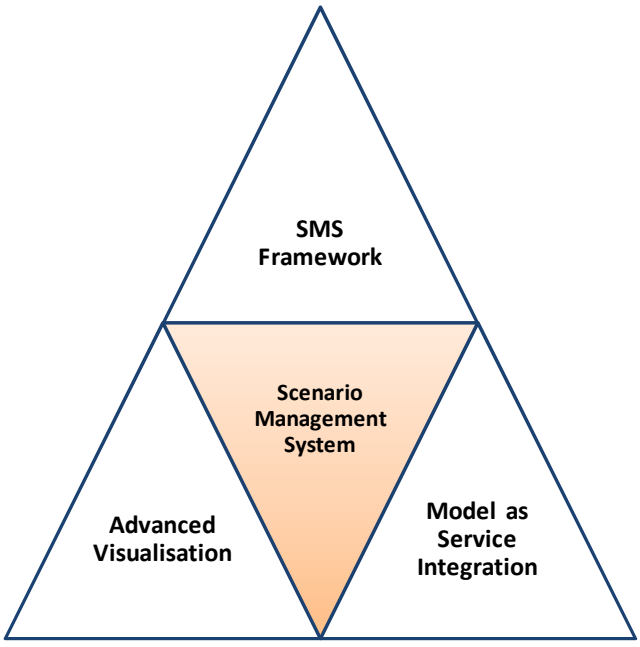

Figure 2: SMS Building Blocks

The general software development process in SUDPLAN follows a three iteration spiral development approach, in which we have now have reached the end of the third and final cycle.

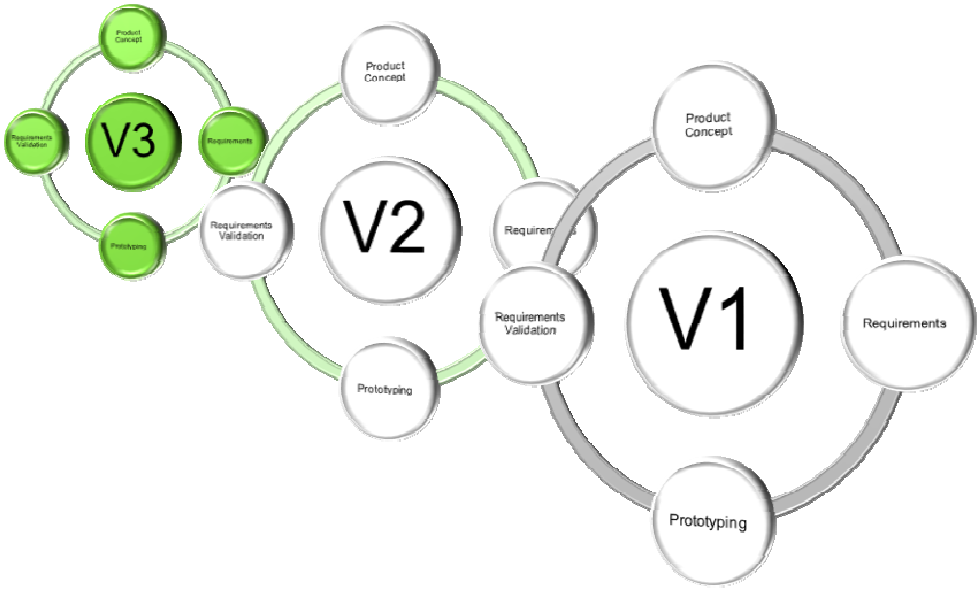

Figure 3: SUDPLAN Spiral Approach

The incremental results of the third cycle are given at the end of each Building Block chapter.

2.1. Tasks and Documents involved

As can be seen in *Figure 4: Dependencies in the Development* Process, the implementation of the SMS Building Blocks has been influenced by several activities in the overall project.

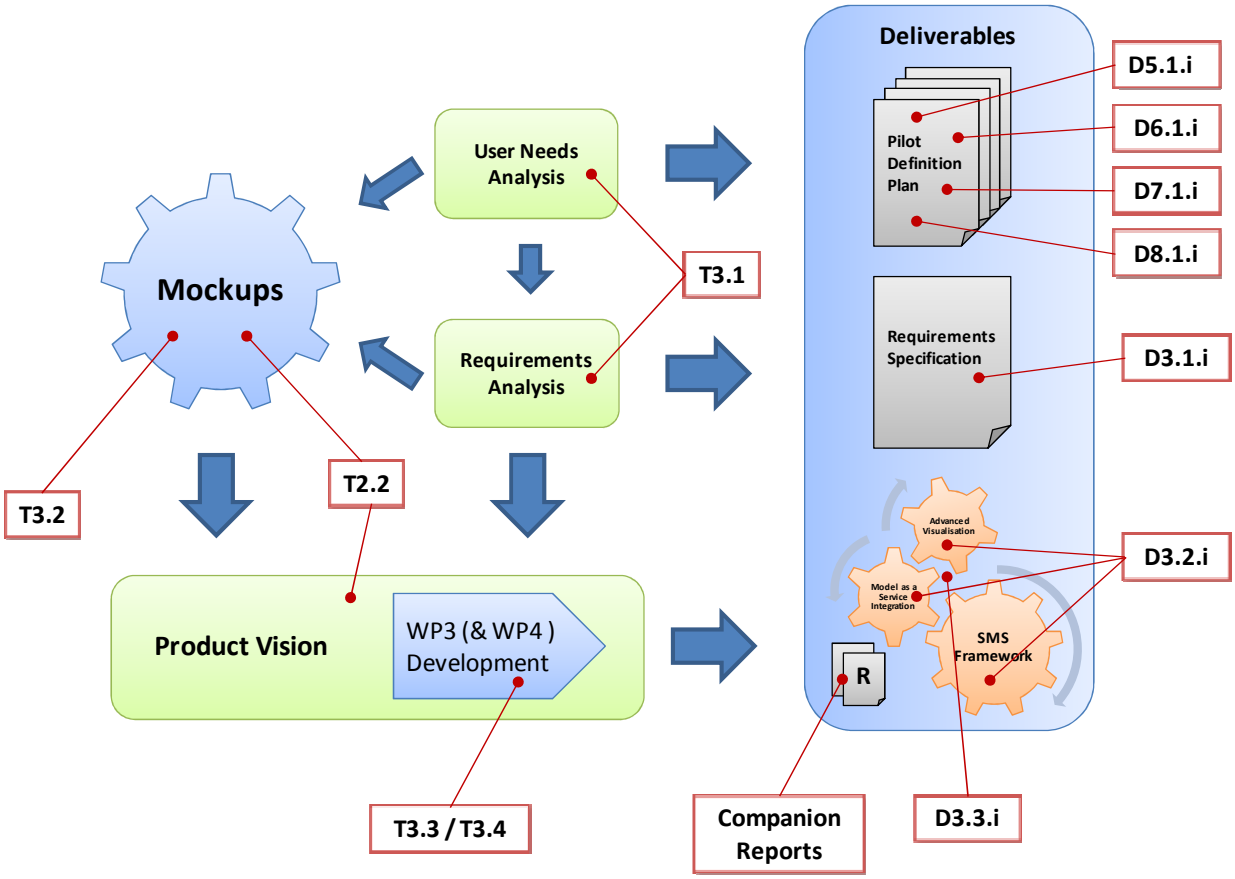

Figure 4: Dependencies in the Development Process

In the first year of SUDPLAN the developments were mainly driven by the *Product Conceptualisation* (T2.2), the *Product Prototyping* (T3.2), and the *Requirements Specification* (T3.1) Tasks. The requirements and user needs analysis, as performed in the course of T3.1, are further explained in the *D3.1.2 - Requirement Specification*. In short, they consisted of intense interview and discussion sessions with the pilot users and the study and analysis of the DoW, relevant projects, literature and other sources of requirements for the SMS. While T3.1 provided major input to the *Pilot Definition Plans (D5.1.2 – D8.1.2)* and T2.2 and T3.3, its immediate result is the deliverable D3.1.2. In addition, mockup activities were performed to be able to capture the user vision for the SUDPLAN product. Therefore, numerous mockups were produced on the basis of the requirements identified in T3.1 and were discussed with the pilot users.

In the second year, first validation results, an enhanced requirements document and more and enhanced mockups have been available. The refined and consolidated SMS requirements documented in D3.1.2, together with the mockups specified in T2.2 and T3.2, have been used by the developers of the SMS building blocks to identify and implement the concrete SMS functionalities. The immediate results of the WP3 development process (T3.3 and T3.4) are a set of distinct components, presented as SMS Building Blocks in this document, and the integrated

SMS itself, presented in the document *D3.3.3 Scenario Management System V3 – Companion Report*.

In the second year of SUDPLAN, the developments were not influenced solely by refined requirements (D3.1.2) and ongoing mockup activities but also by the outcome of the *Product Validation and Evaluation* Task (T2.3) as shown in *Figure 5: Second Year Development*. Thus, the findings of the Validation and Evaluation Report (D2.2.1) have been taken into account during further development and integration activities.

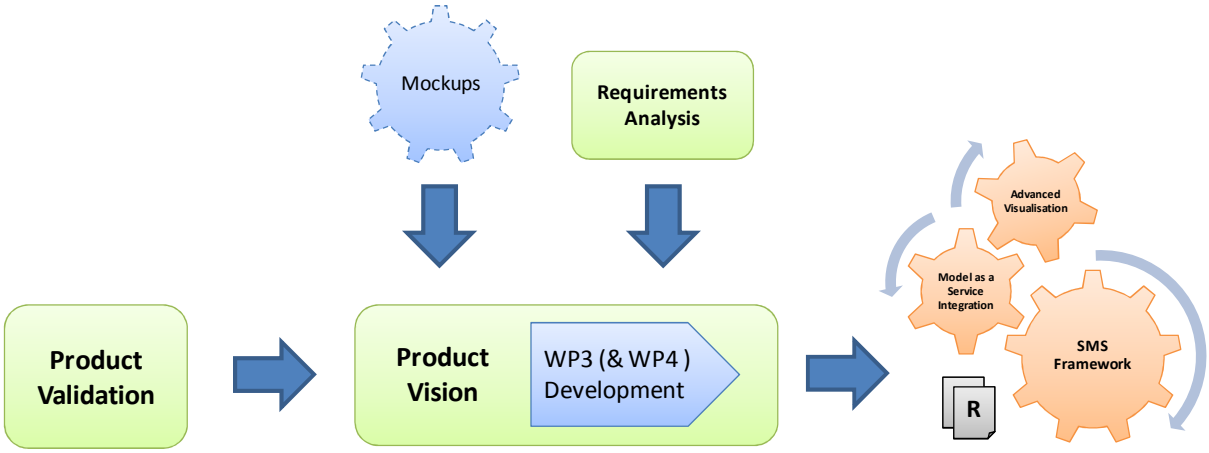

Figure 5: Second Year Development

In addition to the recommendations made in the first *Validation and Evaluation Report (D2.2.1)*, the early results of the second Validation and Evaluation Phase, which is based on the refined *Validation Plan (D2.1)* have been considered in the second year developments.

The developments of the third year had the focus to produce stable versions of the building blocks including the complete required functionality. The general nature of the building blocks is the same as for V2.

2.2. Choice of Core Technologies

The choice of technologies, products and software components for the implementation of the SUDPLAN product is driven by the requirement to provide a sophisticated yet simple to use planning tool by the end of the project. The goal of the project is to develop a mature software solution that can be adapted to the needs of any European city with minimal effort, rather than a proof-of-concept implementation that serves as the basis for further developments. In order to reach this goal it is necessary to select software components that already provide the infrastructure and a certain set of core functionality on which the SUDPLAN product can be built. Furthermore, the SUDPLAN product has to adhere to certain fundamental expectations expressed in the DoW, which imposes certain technical requirements on the type of software components that can be selected for the implementation. Moreover, the SUDPLAN partners involved in the implementation of the SMS Building Blocks must be able to extend, configure and adapt the selected software components in consideration of their personal expertise, the planned resources and the given time frame.

The core features of the Building Blocks including their relationship to the platform requirements specified in *D3.1.2 - Requirement Specification V2*, are described in the sections

4.1 Scenario Management System Framework. *4.2 Model as a Service Integration* and *4.3 Advanced Visualisation,* respectively.

2.2.1 Scenario Management

The aforementioned considerations have already been taken into account during the preparation of the project, and have led to the plan to use the cids product of cismet as the basis for the SMS Framework. Cids (http://www.cismet.de/en/cidsfacts.pdf) is an open source toolkit for the management and integration of complex, heterogeneous, multi-domain, multi-use data of both a temporal and spatial nature. It has been used in the ICT context, including FP4 and FP5 projects. cismet, as developer of cids, has the necessary knowledge to adapt it to SUDPLAN needs, thus developing SUPDPLAN specific extensions and providing integration and training support for the rest of the consortium.

2.2.2 Model as a Service Integration

The implementation of SOS and SPS related software is based on the Time Series Toolbox (TS-Toolbox) API from AIT. The TS-Toolbox API provides the means to conveniently deal with arbitrary time series.

We decided to use the TS-Toolbox because it represents a good starting point to implement dedicated services to wrap the various existing models needed in SUDPLAN and to establish the basis for the integration of new local models through standardised service interfaces. In the SANY project, we found that the use of existing SOS and SPS implementations, e.g. from 52° North Initiative [SOS52N, 2011] is not feasible as they are complete systems, and they are not built to act just as an interface implementation to wrap existing models or data bases. Many parts of the TS-ToolBox, especially on the client side, can be used or adapted to suit SUDPLAN needs.

2.2.3 Advanced Visualisation

It was decided to use the freely available World Wind Java SDK component as the basis for further developments of the Advanced Visualisation Component. After the integration of the World Wind SDK it was thus possible to focus only on the visualisation and animation of 3D results and predictions, in particular using the 3D landscape.

The World Wind SDK is a free and open source Java-API for a virtual globe released under the NASA Open Source Agreement (NOSA). The framework provides a powerful platform for giving the SMS the means to express, manipulate and analyse data of interest. World Wind provides many features for displaying as well as interacting with geographic data and representing a wide range of geometric objects. Moreover, extending the API is simple and easy to do. World Wind features include:

- Open-source, high-performance 3D Virtual globe API and SDK
- Open-standard interfaces to GIS services and databases
- Display high-resolution imagery, terrain, and geographic information from any open-standard public or private source

- Huge collection of high-resolution imagery and terrain from NASA servers
- Supports Coordinate System: Lat/Lon, UTM, MGRS
- Supports of GeoTIFF, JPG, PNG, and JPEG2000
- Supports standard GIS formats: Shape file, KML, GML, GeoJSON
- Different Navigation Modes are being supported
- Supports visualization for stereoscopic displays

Organizations across the world use and support the on-going development of World Wind to monitor weather patterns, and visualize cities and terrain. With World Wind taking care of the basic concepts of visualizing geographic data on a virtual globe, we are now able to focus on solving the domain specific problems and to focus mostly on the visualisation and animation of 3D results and predictions, in particular using the 3D landscape.

3. Scenario Management System Overview

The Scenario Management System is the platform on which any SUDPLAN Application (or SUDPLAN System) is built. It consists of the three distinct Building Blocks described in Chapter *4 - Scenario Management System Building Blocks* of this document and can be seen as a generic integration platform that will be able to facilitate climate change induced urban development planning in any city in Europe. The goal to provide a universal, flexible and adaptable planning tool is supported by the separation of the SUDPLAN System into several architectural layers as shown in *Figure 6: SUDPLAN Layered* Architecture.

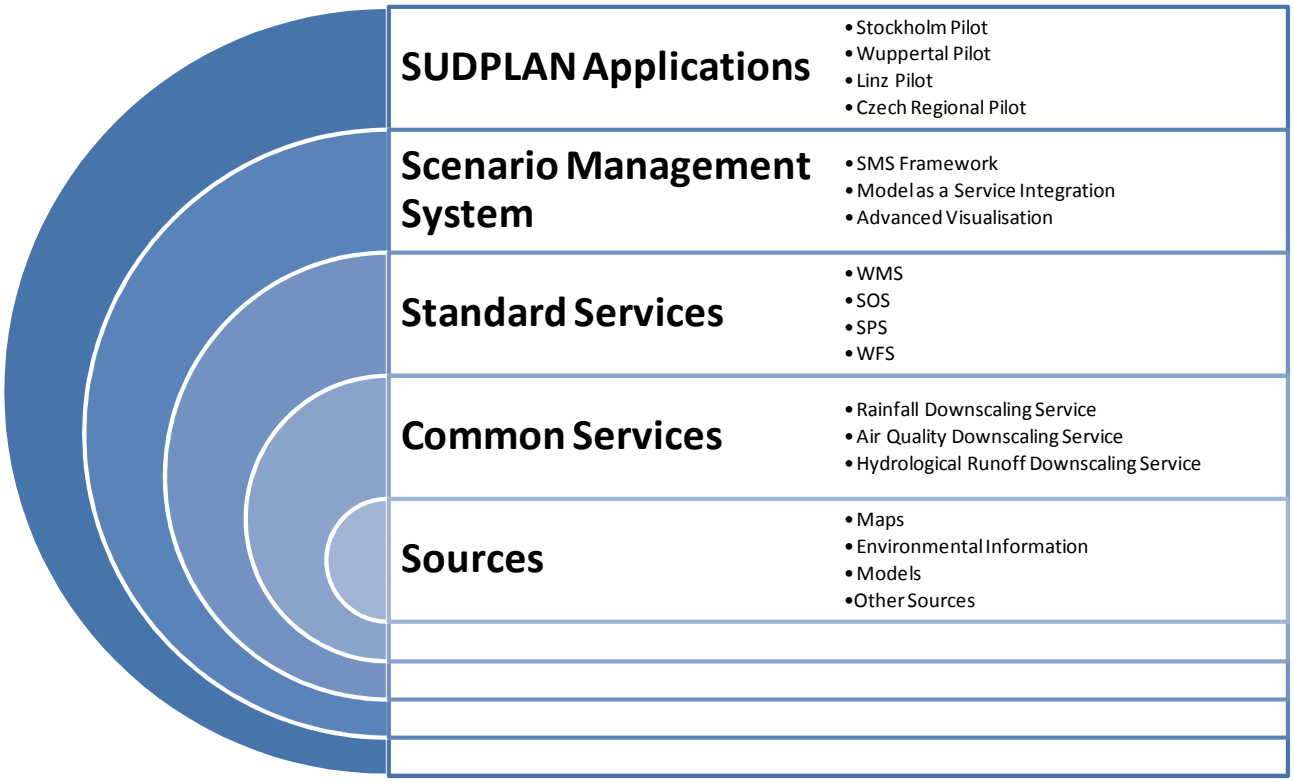

Figure 6: SUDPLAN Layered Architecture

The top-level layer, the SUDPLAN Application itself, is the result of an extension, customisation and configuration of the underlying SMS. The SMS comes with everything needed to provide common scenario management tasks including data integration, model management and execution, workflow management, basic and advanced visualisation, and comparison of various temporal and spatial data sets, etc. It therefore relies upon standard services for data access and model management and thus greatly facilitates the task of integrating new models and data sources. Consequently, the same mechanisms used for interfacing the SUDPLAN Common Services with the SMS can be used for local model and data source integration. As shown in *Figure 7: OGC Services* Integration, several services specified by the Open Geospatial Consortium (OGC) are supported by the SMS: Sensor Planning Service (SPS), Sensor Observation Service (SOS), Web Map Service (WMS) and Web Feature Service (WFS). Their usage is further explained in Chapters *4.1 - Scenario Management System Framework* and *4.2 - Model as a Service* Integration.

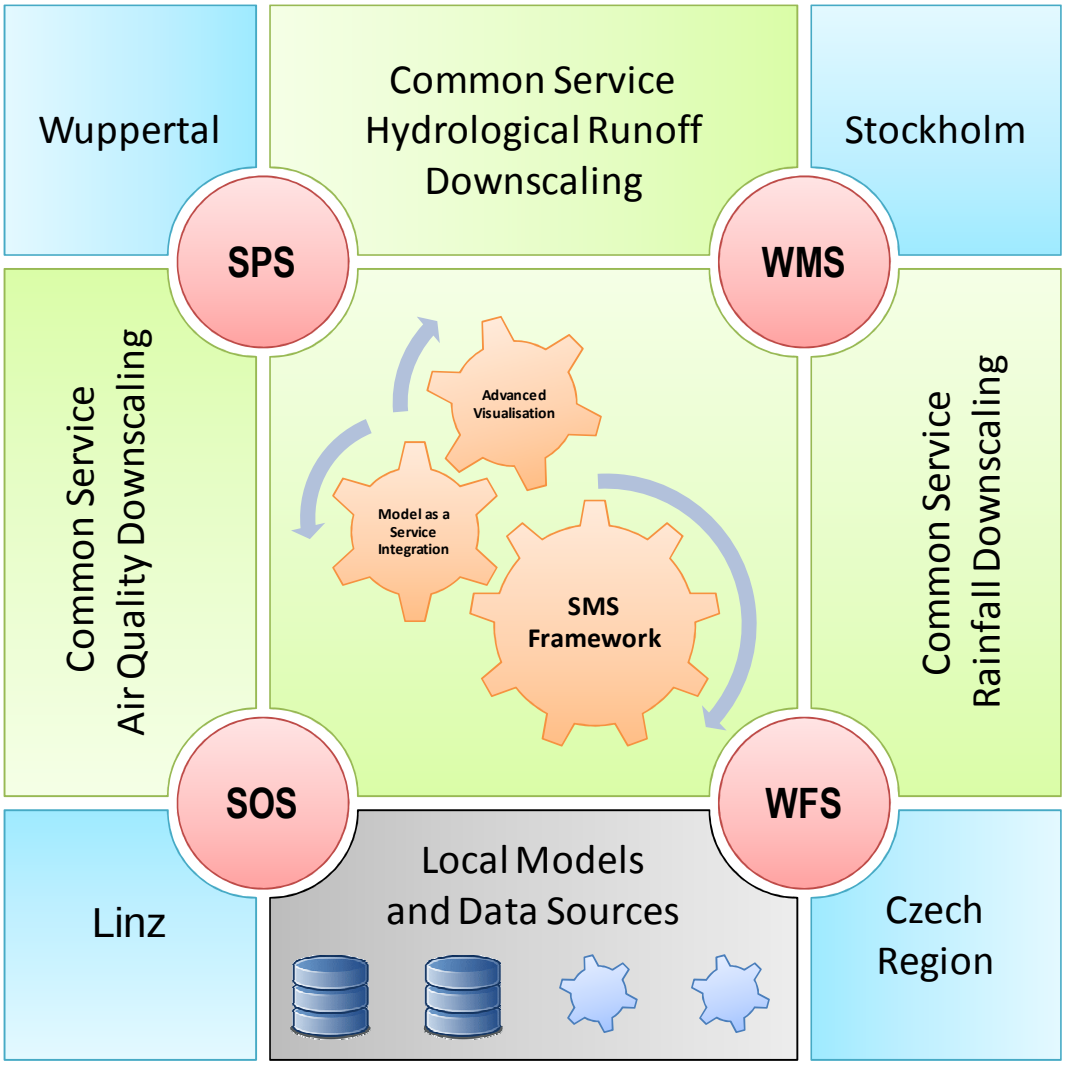

Figure 7: OGC Services Integration

It is also possible to develop a custom model integration solution with respect to particular user requirements. The SMS Framework allows both standard and custom integration without the need to change the SMS itself. For this purpose, the SMS Framework exposes an API that enables the developers of a SUDPLAN Application to extend the SMS with their specific functionalities. The four pilot applications of the SUDPLAN project therefore validate not only the general approach of the SMS but also its adaptability/transferability and thus its applicability to any city in Europe.

4. Scenario Management System Building Blocks

In this chapter, the three Building Blocks of the Scenario Management System and the incremental results of the second product implementation phase are presented in detail. These are described in the following subsections:

- Scenario Management System Framework
- Model as a Service Integration
- Advanced Visualisation Component

Each subsection starts with a short introduction of the role of the component in SUDPLAN and lists the expected properties and functionalities in relation to SUDPLAN requirements. It continues with the description of the chosen software components, presents their architecture, and briefly explains by which means it realises the anticipated functionalities. It further provides documentation of the developments carried out in the second and third year, specifically referring to new functionalities implemented. Finally, it gives a summary of the achievements of the third year and points to the software repository that contains the actual results.

4.1. Scenario Management System Framework

The Scenario Management System Framework (GUI shown in *Figure 8: SUDPLAN SMS GUI*) is the central component providing common SMS and integration functionality. Together with the Building Blocks for the integration of models through standardized services and for advanced visualisation capabilities it provides the basis for pilot specific implementations and the necessary workflows to support the use of models as a basis for decision making.

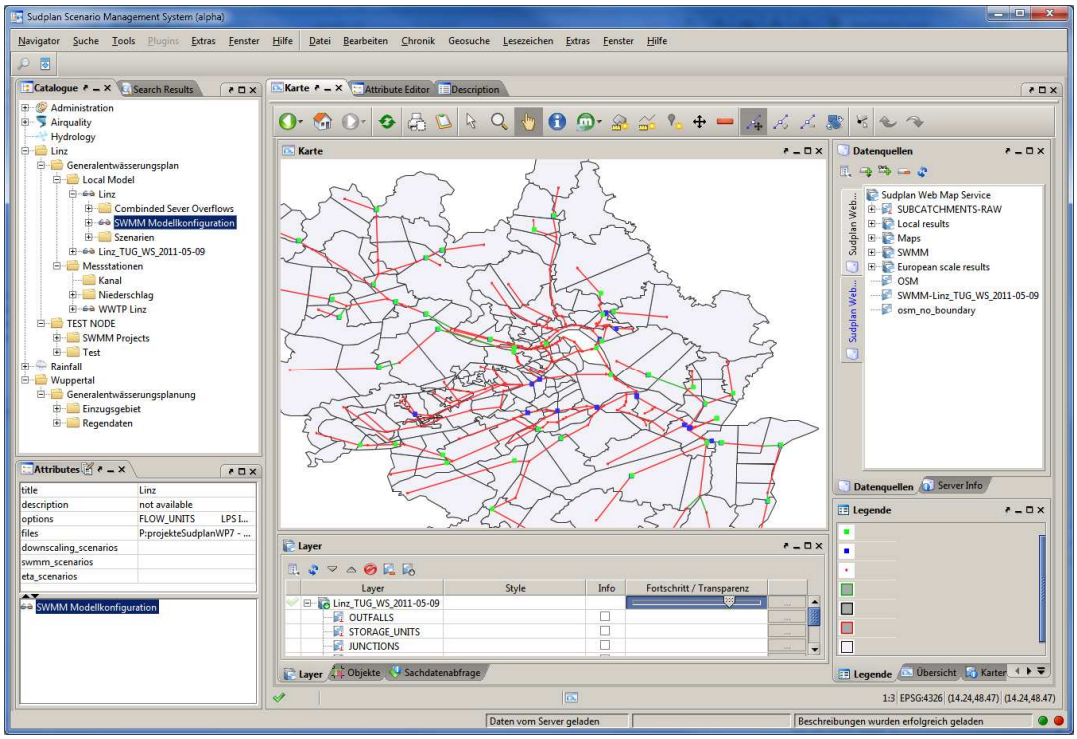

Figure 8: SUDPLAN SMS GUI

The core functionalities provided by the SMS Framework include, for example, support for the management of models, i.e. model execution, result storage, parameterisation and basic model result visualisation (such as 1D time series, and 2D maps).

The complete set of functionalities that has been provided by the SMS Framework as well as general properties of the anticipated scenario management system can be derived from the requirements specified in *D3.1.2 - Requirement Specification V2*. The SMS Framework must not be confused with the integrated Scenario Management System as described in Chapter *3 - Scenario Management System Overview*. Therefore, functionalities intended to be covered by the Common Services, pilot specific applications, the Advanced Visualisation Building Block, and the Model as a Service Building Block, are considered in the SMS Framework only when it directly contributes to their implementation.

The general properties of the SMS Framework as well as the overall Scenario Management System are related mostly to technical and high level requirements stemming from the DoW, ICT expectations, and best practices from various sources. In summary, the SMS Framework must

- be scalable, open and reliable
- be based on open standards, and make use of open standards
- be based on open source products, and be itself an open source product
- be based on technologies making use of the World Wide Web
- provide a generic infrastructure independent of a particular domain
- provide integration with SOA-based infrastructures and standards-based services
- provide security and access control mechanisms
- offer user-friendly graphical interfaces to improve usability

The core functionalities to be supported by the SMS Framework can be derived from the overall objectives of the project as well as specific requirements expressed by the different users of the SMS and requirements from developers that perform configuration, management, extension and integration tasks. In summary, the perceived functionalities of the SMS Framework are to

- provide functionalities for city management, planning and decision support in the context of climate change adaptation
- provide applications and tools for the management of scenarios
- provide integration, management, calibration, and configuration facilities for models
- provide automation of tasks like model runs, analysis and reporting
- allow the discovery of relevant sources
- provide spatio-temporal visualization of disparate information
- provide visualisation and comparison of model run results
- support sharing and publishing of information (e.g. model results, reports)
- support the management of various and disparate information sources

Each requirement can be further broken done into more detailed and concrete functionalities such that all relevant requirements defined in *D3.1.2 - Requirement Specification V2* can be taken into account by the developers.

4.1.1 Components and APIs used

As already indicated in *2.2 - Choice of Core* Technologies, SMS development efforts are based on the open source cids[1] geo-integration platform. cids is based on a 15 year development effort and was developed by the Environmental Informatics Group (EIG[2]). It was used in several projects including FP4 and FP5 projects. In 2001, it was turned over to cismet GmbH. All cids components are written in Java and thus are platform independent and web-enabled.

The cids product suite consists of a set of services, applications, software components, management tools, development tools, and application programming interfaces (APIs) for the management, integration, and development of heterogeneous information systems with a special focus on interactive geo-spatial systems. It provides a distributed integration platform, which is particularly useful for workflows that need a combination of information and processes from different source systems such as GIS systems, relational databases, and simulation models. In this way it already provides and supports a number of functionalities of the anticipated SUDPLAN SMS, including user management and access control, search and discovery of relevant information and advanced interactive 2D visualisation (OGC WMS and WFS clients). The developments carried out in the first and the second year to provide support for additional SMS-specific functionalities like scenario management and result visualisation and comparison are described in section *4.1.2 - First Year Development, 4.1.3* - Second Year Development *and $.1.4 Third Year Development*.

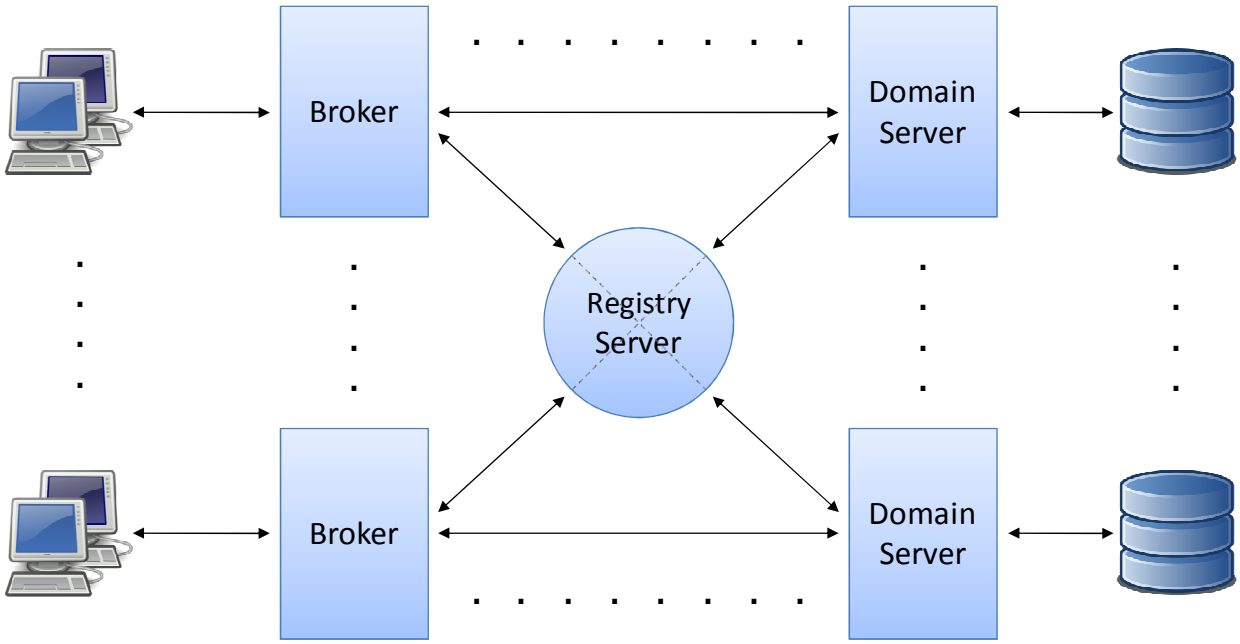

Figure 9: SMS Framework Client-Server Architecture

Figure 9: SMS Framework Client-Server Architecture shows that the SMS Framework is based on a client-server architecture in which an arbitrary number of client instances and server components co-exist in a service network, thus ensuring scalability and reliability. The components shown in Figure 9 are explained in detail in the following sections.

[1] http://www.cismet.de/en/products.html
[2] http://www.enviromatics.net

The main building blocks of the SMS Framework are the Navigator (client), the Kernel, and a set of system management tools. The building blocks and the components are shown in *Figure 10: SMS Framework Building Blocks*.

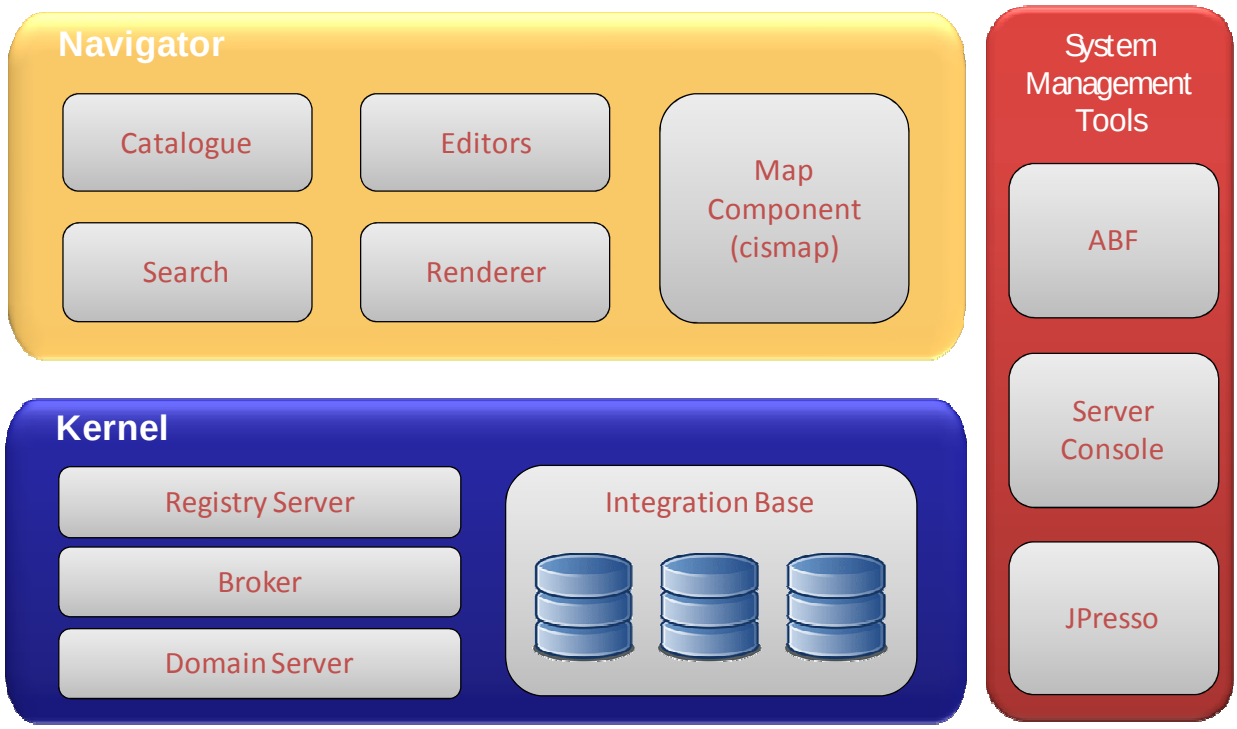

Figure 10: SMS Framework Building Blocks

The **Kernel** represents a network of distributed services and consists of the following four components:

- **Integration Base**
  The Integration Base is a distributed meta database which consists of a generic meta data model placed in a relational DBMS (Data Base Management System). *Figure 11: Generic Meta Data Model of the Integration* Base shows an excerpt of generic meta data model. It is the basis for a concrete information system and is able to describe arbitrary objects (real-world objects, services, models, geographical features, other information systems, etc.), their attributes (e.g. geographical location) and relationships by means of so-called meta classes and objects. The generic meta data model also supports a dynamic navigational catalogue structure, users, groups and permissions, and many other properties.

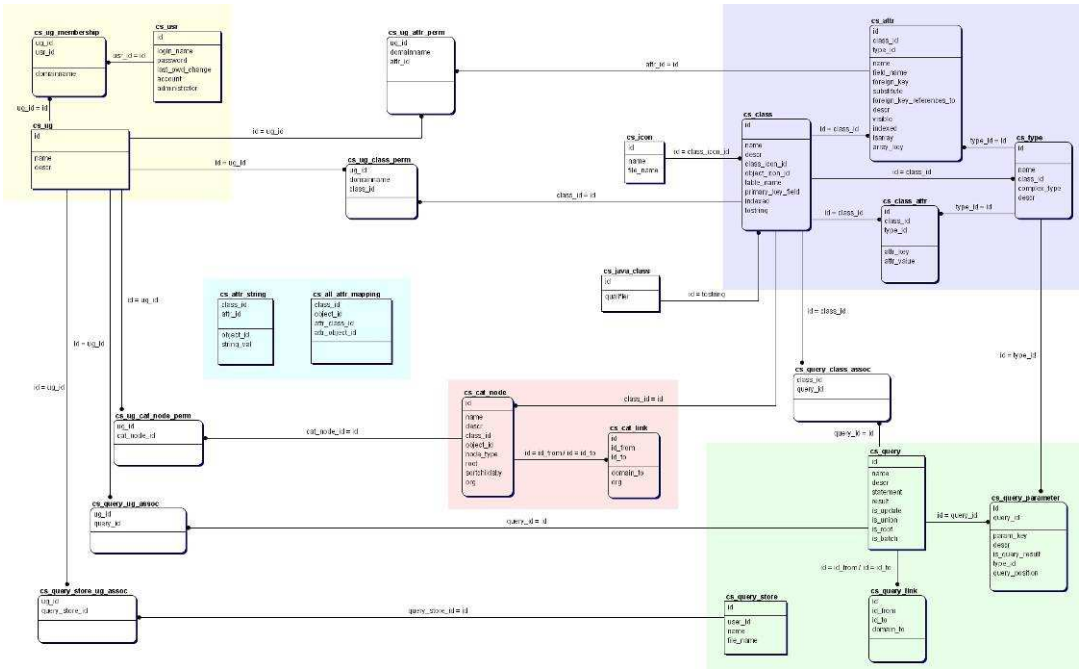

Figure 11: Generic Meta Data Model of the Integration Base

- **Domain Server**
  The Domain Server is the interface to an Integration Base and is responsible for the translation of the generic meta data structure into concrete meta objects and classes, thereby also supporting the creation and updating of meta data. It is also responsible for the construction of the dynamic catalogue structure at runtime.

- **Registry Server**
  The Registry Server is responsible for the resolution of distributed user privileges needed for the enforcement of access rights, the resolution of the distributed catalogue structure needed for the navigation, and the coordination of the distributed search. It also provides service infrastructure related functionalities like server name resolution, network monitoring, status information, etc.

- **Broker**
  The Broker is the interface to the clients and hides the distribution aspects of the system. It acts like a proxy and delegates client requests to the appropriate Domain Servers.

The **Navigator**, shown as a customized version in *Figure 8: SUDPLAN SMS GUI*, is the default client for user interactions with the system. It offers a uniform, user-specific view of the integrated information systems and is particularly useful for cross-system search and retrieval in space, time, and textual content. It can also be used to manage the information in the network. In SUDPLAN it will be used as the management client for OGC compliant data access and model execution services, which are described as customised information classes in the underlying meta data model, thus offering common scenario management functionalities for the various users of the system. The Navigator features a plug-in mechanism that allows the pilots to develop custom extensions in order to support workflows for their specific end user tasks. The Navigator provides the following core functionalities through dedicated GUI components:

- **Catalogue**
  The Catalogue presents a tree for navigation of the distributed catalogue and provides basic information about objects (*Figure 12: Catalogue Attribute* View). It supports caching of the dynamically constructed catalogue structure which can be distributed among several Domain Server and Integration Base instances. Furthermore, it provides functions to manage the catalogue structure as well as meta data objects (*Figure 13: Catalogue Node and Object* Management).

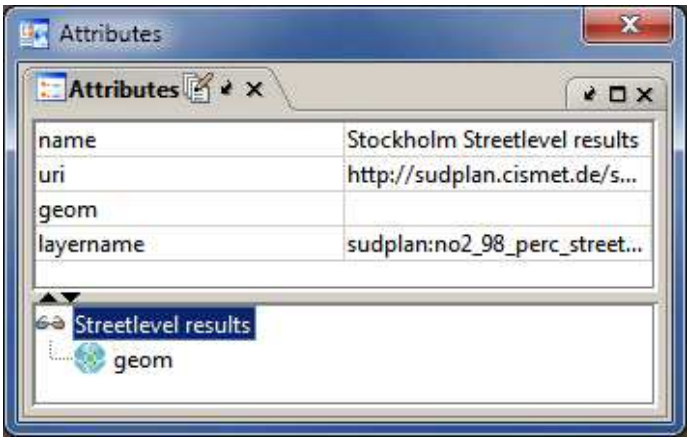

Figure 12: Catalogue Attribute View

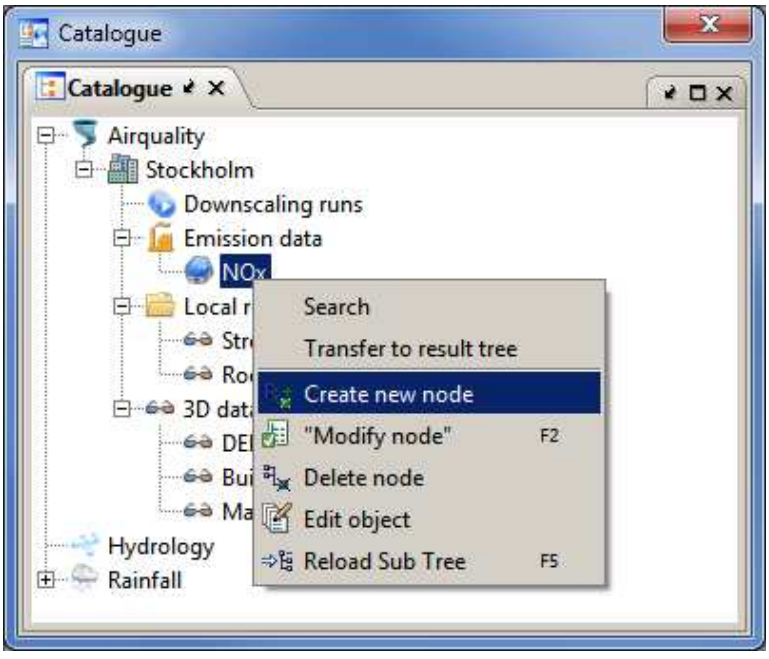

Figure 13: Catalogue Node and Object Management

- **Search**
  There is a highly customisable interface to the dynamic search capabilities of the Kernel, which takes a combination of different meta object attributes into account to provide thematic search to end users. It also features sophisticated geospatial search capabilities and direct interaction with the Map Component.

- **Renderer**
  Renderers are used to provide thematic, context and user dependent views on certain topics. The Navigator is equipped with a set of default Renderers for a variety of different object types and supports the integration of a customised Renderer on a per-object class basis. *Figure 14: Custom Rainfall Render as Client for* OGC SOS shows a custom Renderer that communicates with a Sensor Observation Service (see also *4.2 - Model as a Service* Integration).

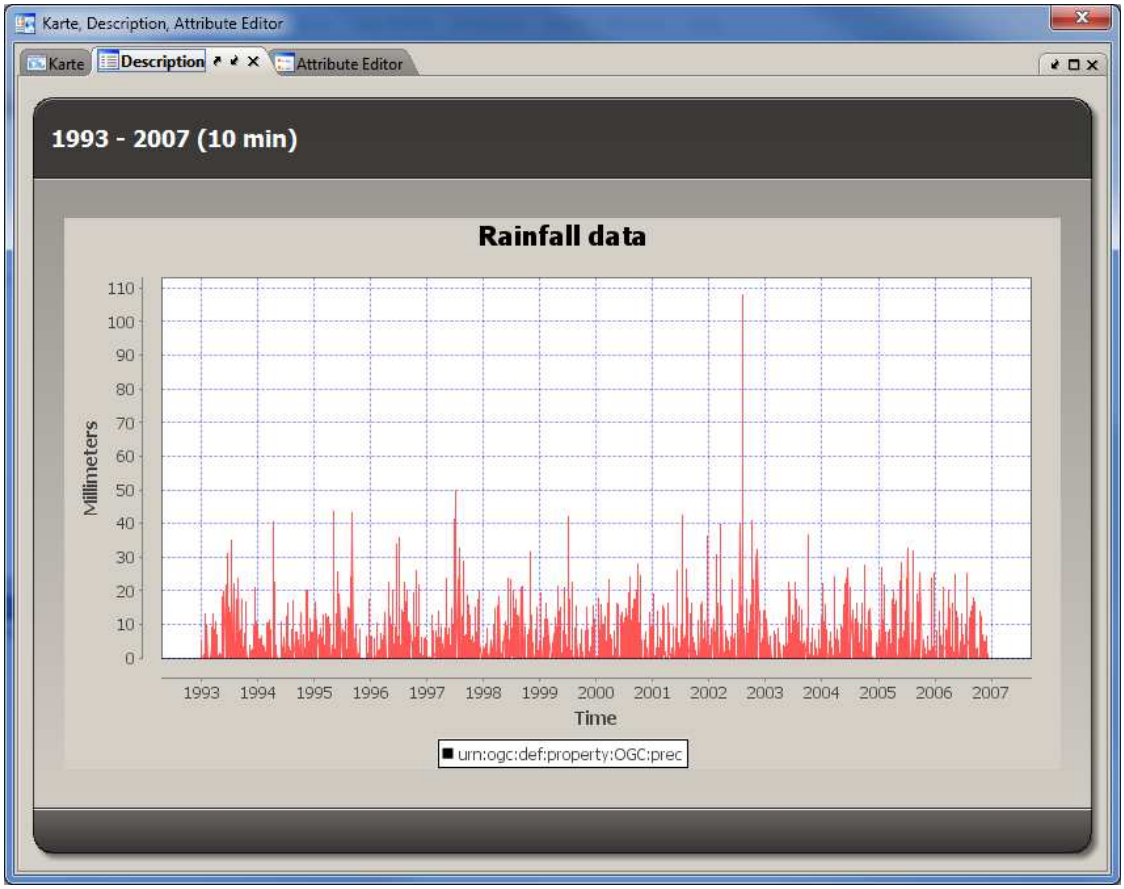

Figure 14: Custom Rainfall Render as Client for OGC SOS

- **Editors**
  Editors are used to manipulate meta objects in a uniform manner, such as scheduled model runs as shown in *Figure 15: Default Attribute Editor*. As already mentioned, meta objects can represent arbitrary physical or virtual objects from simple documents over geographical features to complex workflows. In addition to automatically generated editors, which provide attribute based editing functionality, custom editors are also supported. Custom Editors can be used in SUDPLAN, for example, to provide advanced functionalities for the common configuration of models or to support pilot specific tasks.

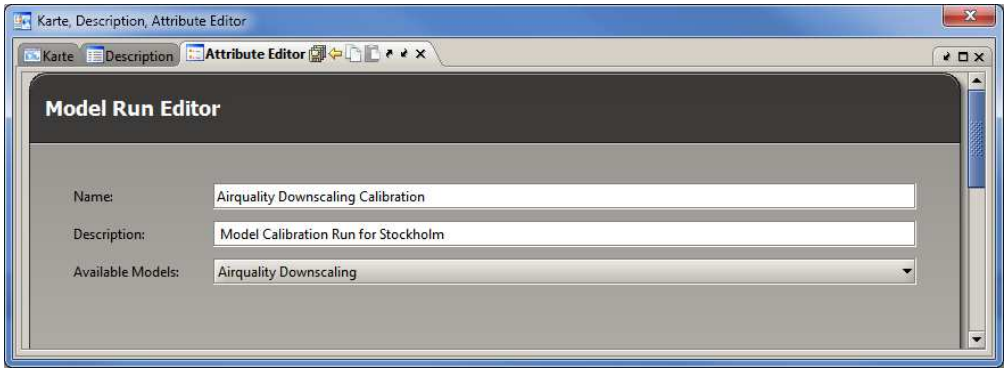

Figure 15: Default Attribute Editor

- **Map Component**
  The Map Component shown in *Figure 16: Map Component (cismap)* is based on cismap, a highly sophisticated 2D map viewer for geo data services such as Web Map Services (WMS) and Web Feature Services (WFS) which comply with the standards of the Open Geospatial Consortium (OGC). Since cismap supports both powerful visualisation and editing functionalities for geospatial information (including arbitrary geolocated meta objects), it can support SUDPLAN end users during configuration, management and decision and planning tasks. Some of the main features of cismap are

  - asynchronous WMS and WFS requests to load several WMS layers in parallel
  - dynamic addition of services, including drag and drop of the server URL
  - a powerful and easy to use geometry editor for the manipulation of geo-data
  - complete integration with cismap and support for the visualisation and manipulation of meta objects
  - a customisable user interface and the ability to save the layout on a per-user basis
  - the representation of geographical features on the map as complex widgets
  - customisable print and report generation facilities

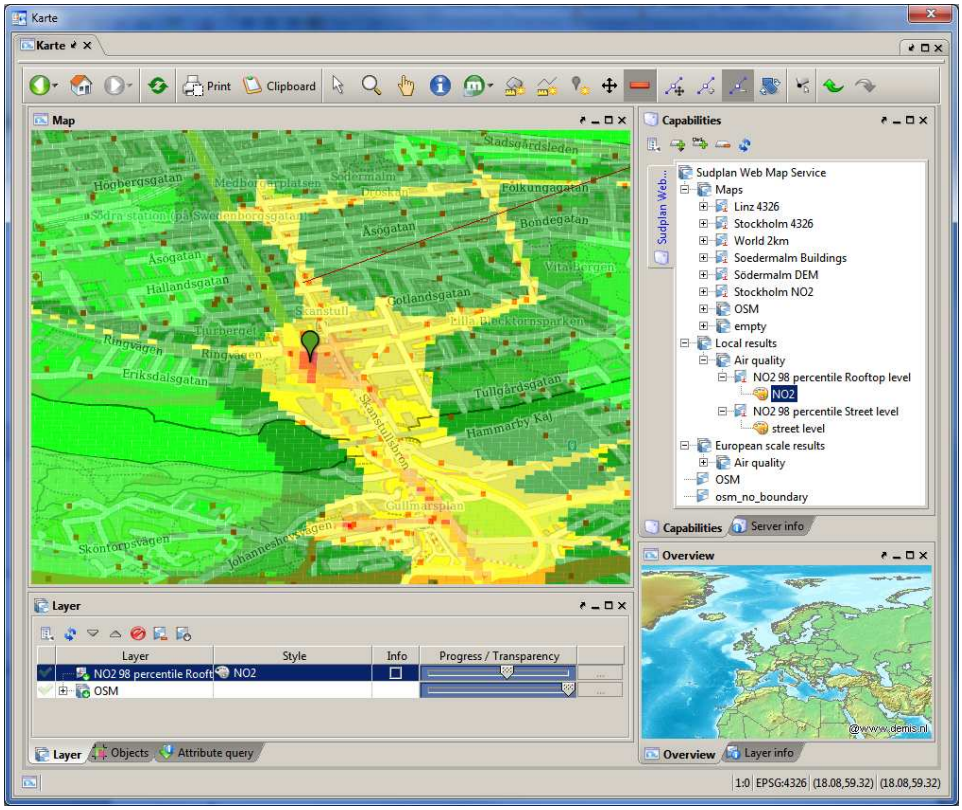

Figure 16: Map Component (cismap)

The cids product suite provides, in addition to the client and server components described in the previous sections, a set of System Management Tools. They support system managers during the installation and maintenance of the SUDPLAN applications as well as modellers during the integration and configuration of mathematical models to be used within SUDPLAN applications. They can also be used to carry out general system administration tasks like user management. The System Management Tools include the following.

- **ABF**
  ABF, which stands for "Administrators' Best Friend," is a powerful front-end for the configuration of the meta data base. It can be used to define new meta classes and their attributes, relationships, dynamic catalogue structures and so on.

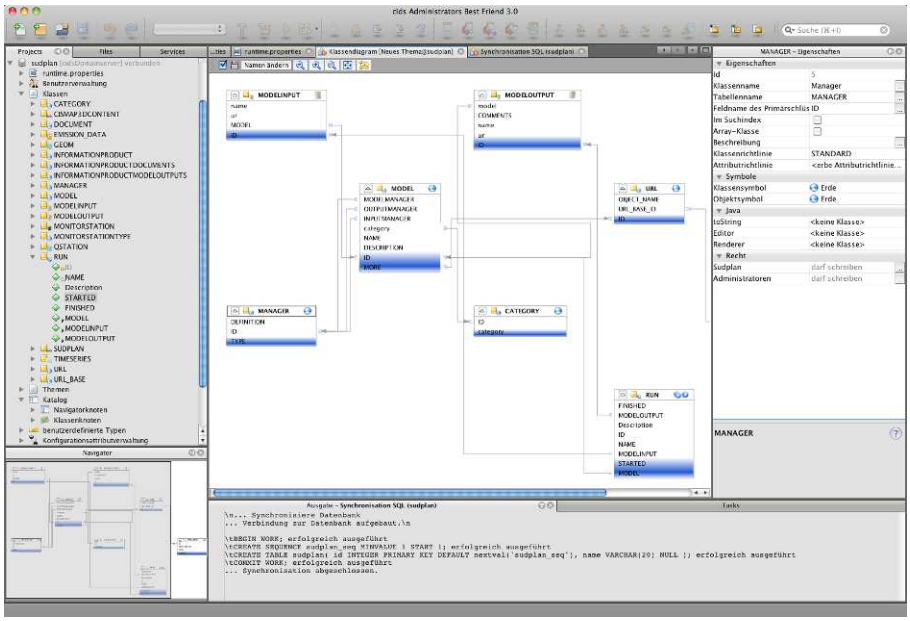

Figure 17: Administrators Best Friend (ABF)

- **Server Console**
  The Server Console is used to monitor and control a server component or a whole service network.

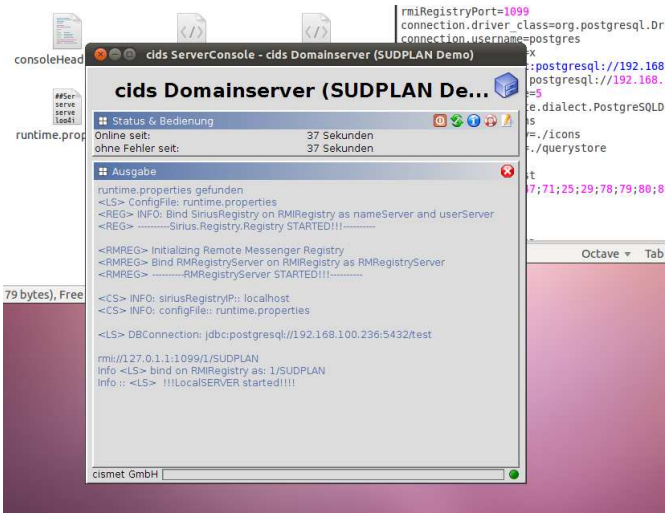

Figure 18: Server Console

- **JPresso**
  JPresso is a powerful ETL-Tool (Extraction, Transformation and Load Tool) for the integration of heterogeneous data-sources. It supports visual mappings that describe the connection between data-sources and the cids Integration Base.

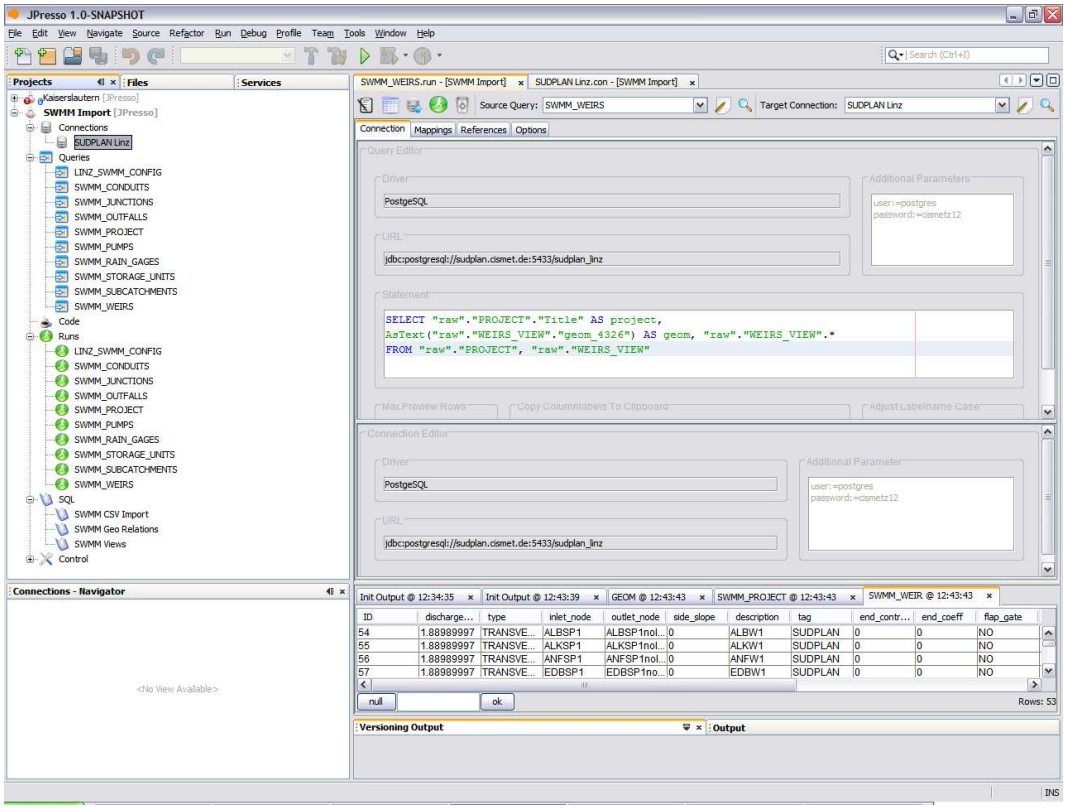

Figure 19: JPresso

The SMS Framework, based on cids components and tools described in this chapter, already fulfils most of the technical requirements that demand a generic infrastructure, open source components, web-based technologies, user-friendly graphical interfaces, security and access control mechanisms, and scalability. It represents a sound basis for the implementation of the SUDPLAN SMS core functionalities like scenario management, resource discovery, sharing and publishing of information, automation of tasks, and so on. The steps taken to use cids as a basis for the SUDPLAN SMS are described in the next sections.

4.1.2 First Year Development

The SUDPLAN SMS based on the cids platform described in the previous section has not only been deployed and configured to suit the needs of SUDPLAN, but the cids platform itself has also been extended to support the particular needs of data providers, data clients, and system integrators.

The first major task undertaken to support the SUDPLAN specific requirements was the extension of cids by a generic model management concept to support all aspects of model integration and management. The model management concept encapsulates model specific properties and provides user specific interfaces for the visualisation and manipulation of model input and output data. The information model of the model management concept shown in *Figure 20: Model Management Information* Model is flexible and adaptable such that it supports both SUDPLAN Common Services as well as arbitrary local models of the pilots or any other city applications.

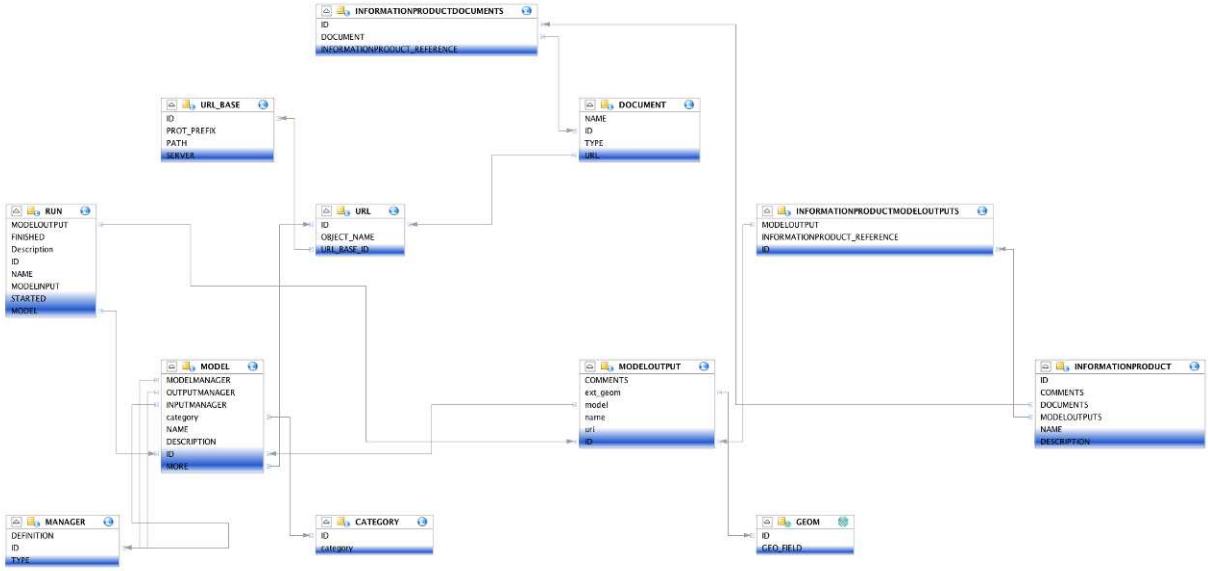

Figure 20: Model Management Information Model

The different entities of the Model Management information model shown in the diagram above are

- **Model Entity**, which defines the general properties of a model and is associated with Manager Entities.

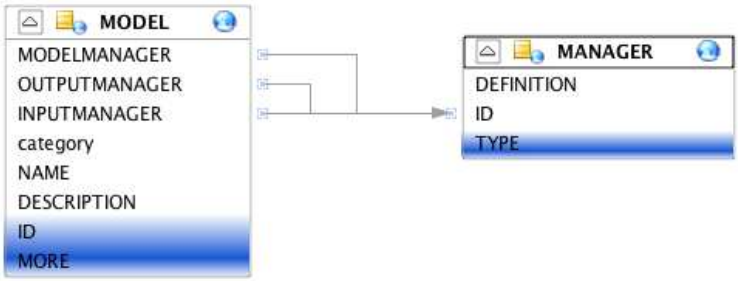

Figure 21: Model Entity

- **Manager Entity**, which defines the model-specific implementations and their type, for example Java, Groovy, BPEL, etc.

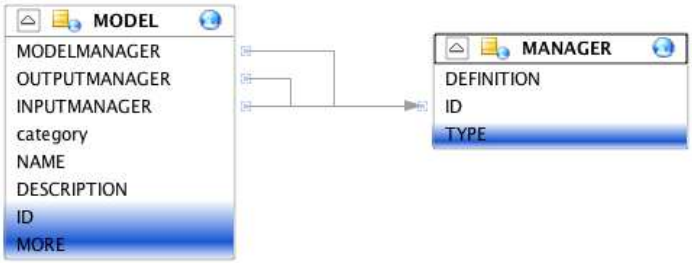

Figure 22: Manager Entity

- **ModelInput Entity**, which encapsulates the model input data and is associated with a Run Entity.

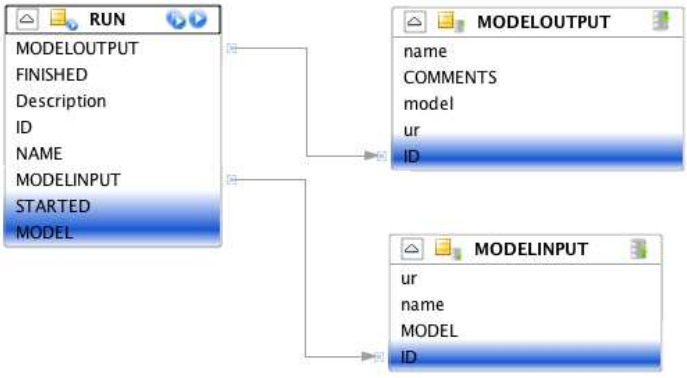

Figure 23: ModelInput Entity

- **ModelOutput Entity**, which encapsulates the model output data and is also associated with a Run Entity.

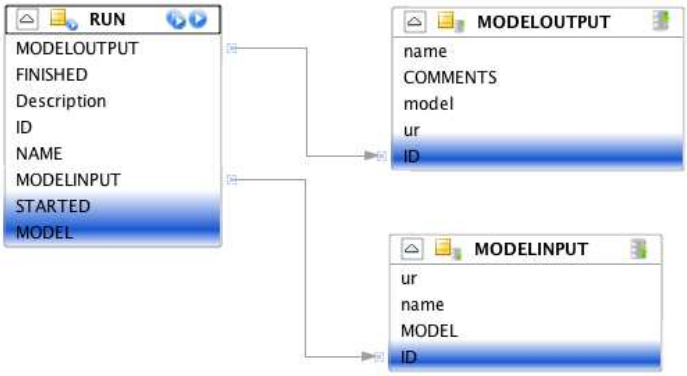

Figure 24: ModelOutput Entity

- **Run Entity**, which defines information needed for the execution of the model and is associated through the ModelInput and ModelOutput Entities with the input/output data of the model.

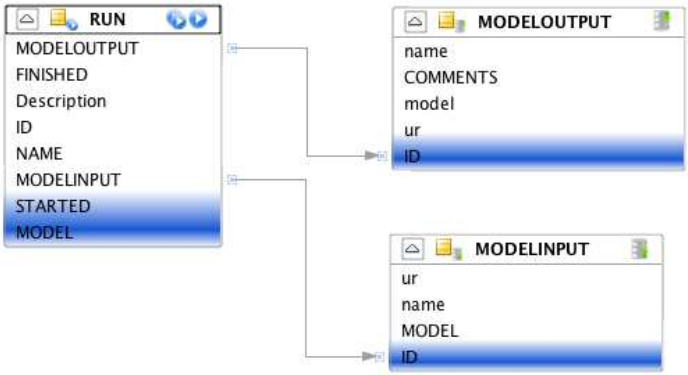

Figure 25: Run Entity

Due to the generic nature of the cids system, no changes to the Kernel or the server components are required to support the Model Management information model. Therefore, the

second major task completed in V1 was the extension of the client (Navigator) to provide convenient user interfaces for model integration and management. The class diagram in Figure 26 shows an excerpt (interfaces and base classes only) of the Navigator extensions realised in V1.

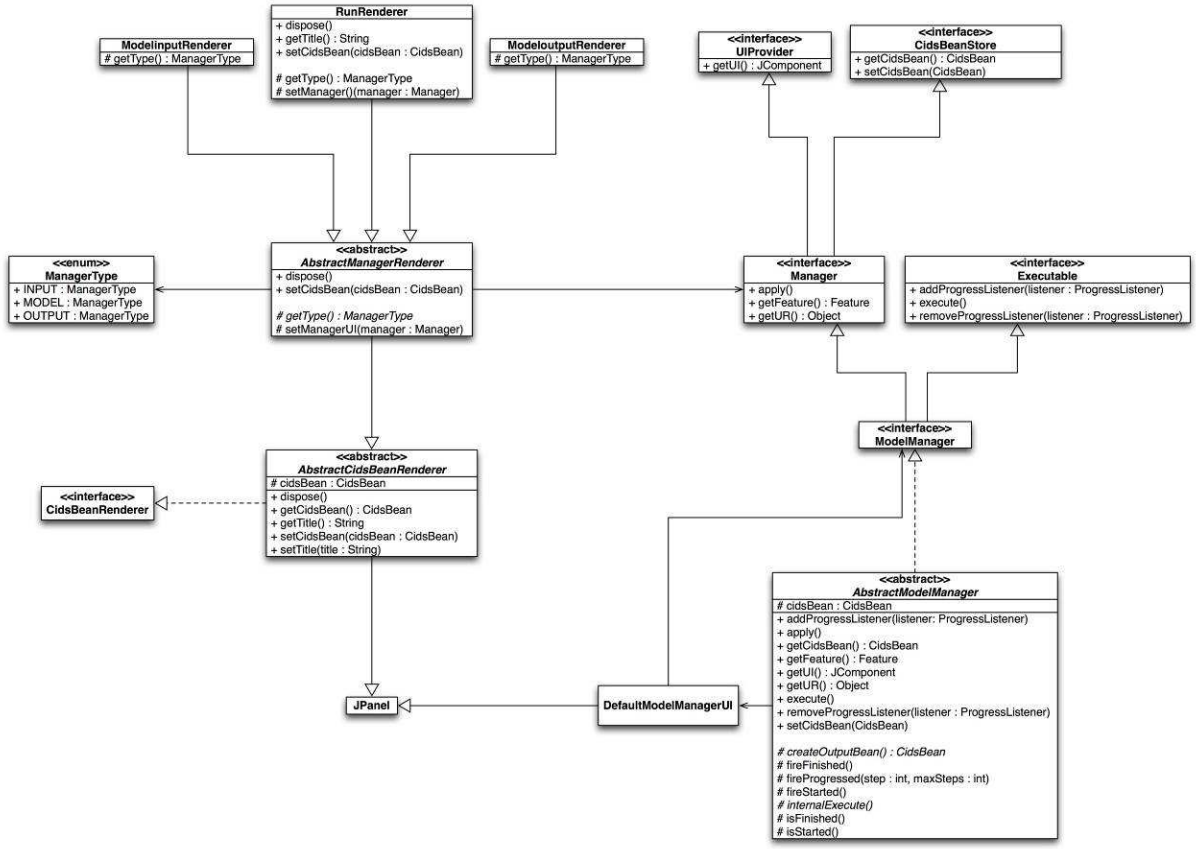

Figure 26: Model Management Navigator Extensions

The various Renderer classes refer to the custom Renderer concept presented in *4.1.1 - Components and APIs used.* They map directly to the different Model Management Entities (*Figure 27: Custom Model Management Renderer* Implementation) and provide support for visualisation of the model configuration, input/output data, etc.

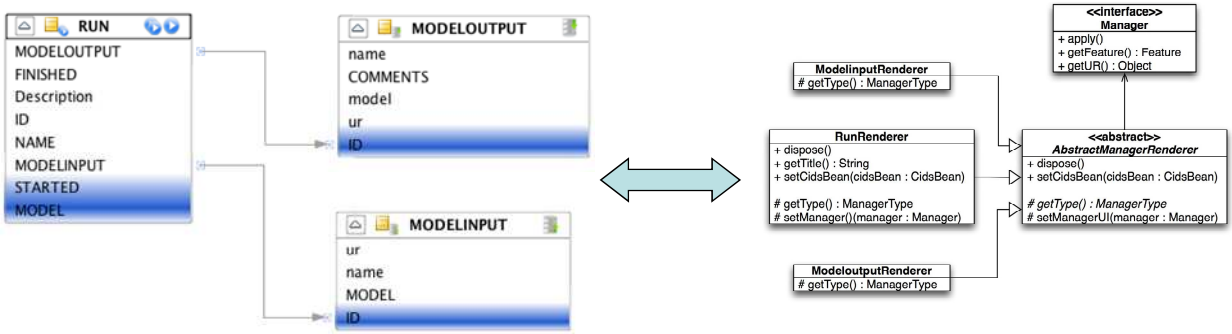

Figure 27: Custom Model Management Renderer Implementation

Figure 28: RunRenderer GUI shows the user interface of a custom RunRenderer implementation that displays the model execution status an of air quality downscaling run

performed by the corresponding Common Service. It includes references to the model results and parameterisation (input/output data) which can be visualised by the associated ModelInputRenderer, ModelOutputRenderer, or FeatureRenderer. For reasons of clarity, the FeatureRenderer is not shown in the class diagrams above. In short, a FeatureRenderer is used when data have to be visualised in the map.

In V1, the RunRenderer supports basic model management and asynchronous model execution, providing the possibility to initiate several model runs in parallel. Access to the Common Services is realised with the help of the Model as a Service Integration which provides a standardised OGC SPS (Sensor Planning Service) interface for model execution.

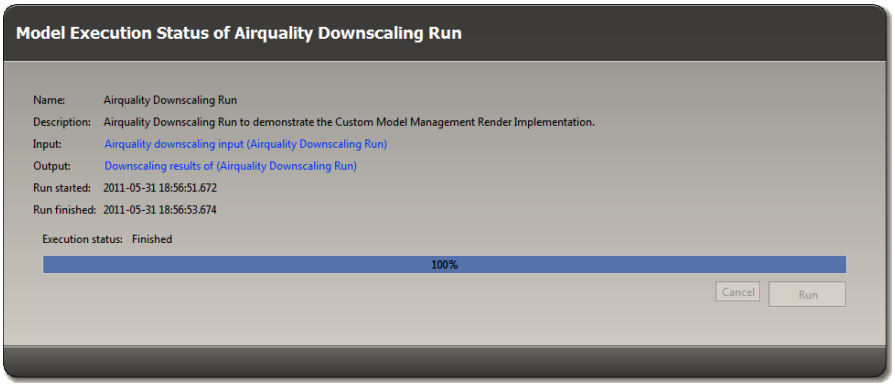

Figure 28: RunRenderer GUI Implementation

Access to model input/output data can be achieved through SOS (Sensor Observation Service), WMS (Web Map Service) and WFS (Web Feature Service) interfaces. SOS is commonly used when 2D data are to be displayed as a graph or table. WFS and WMS can be used when the data have to be shown in the map. An example of a custom ModelInputRender that retrieves model input time series from a SOS and visualises it as 2D graph was shown in *Figure 14: Custom Rainfall Render as Client for OGC* SOS.

Figure 29: Interactive FeatureRenderer shows a FeatureRenderer that visualises model input data (local Stockholm emission data). Retrieving data from a WFS and showing them in the map is a common feature of cismap. The feature implemented in the SUDPLAN V1 enhances interactivity through custom actions which can be performed on the individual features. In this case the feature representing the local emission data supports an action to initiate air quality downscaling.

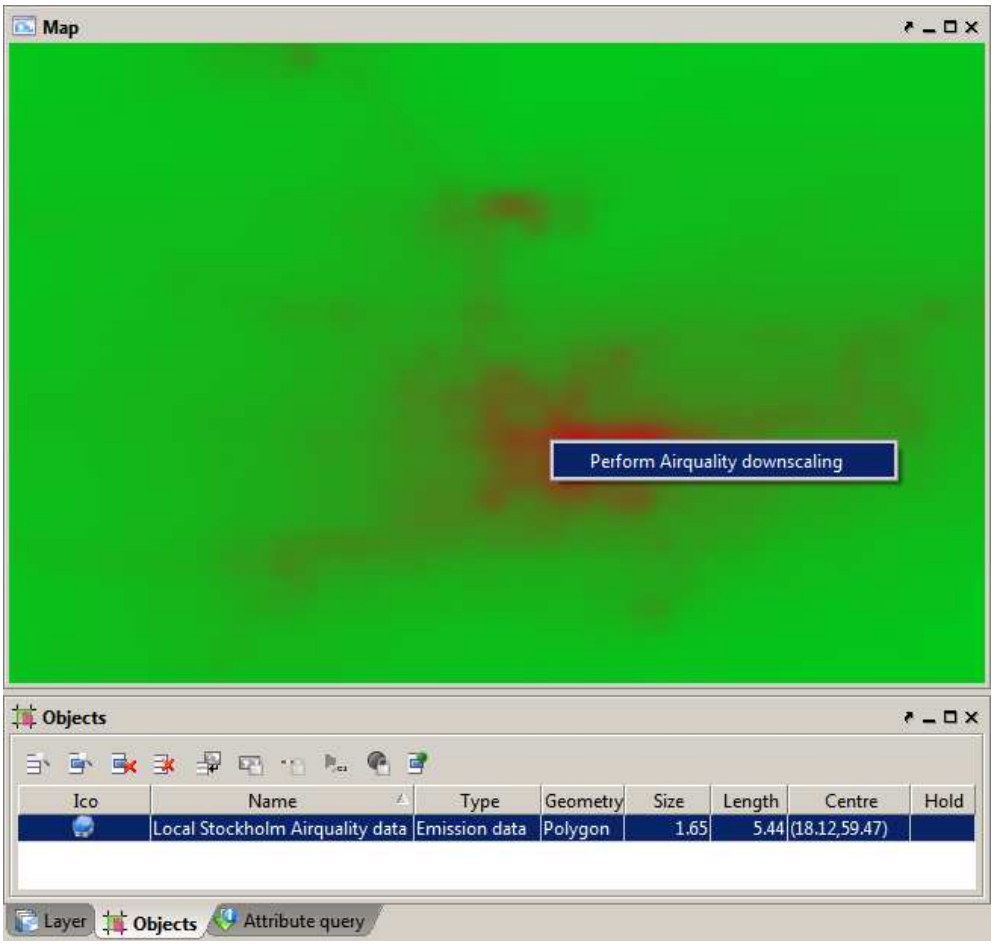

Figure 29: Interactive FeatureRenderer

The air quality downscaling action is associated with a model specific configuration wizard as shown in *Figure 30: Air Quality Downscaling* Wizard below.

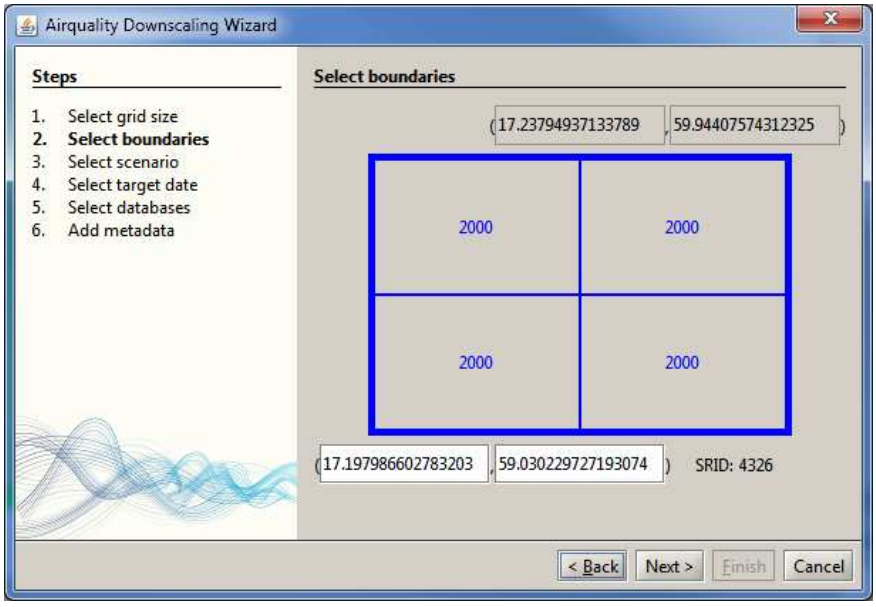

Figure 30: Air Quality Downscaling Wizard

Upon complete execution of the wizard a new model run is started, which can be monitored with the help of a RunRenderer already shown in *Figure 28: RunRenderer* GUI. Once the model run is finished, the output data can be retrieved and visualised. *Figure 31: Visualisation of Model* Output illustrates the possibility visualising model output data as a 1D time series, a feature which was also implemented in V1. The downscaled rainfall data produced by the corresponding Common Service in this example are managed by the SMS and thus can also be used as input data for custom local models.

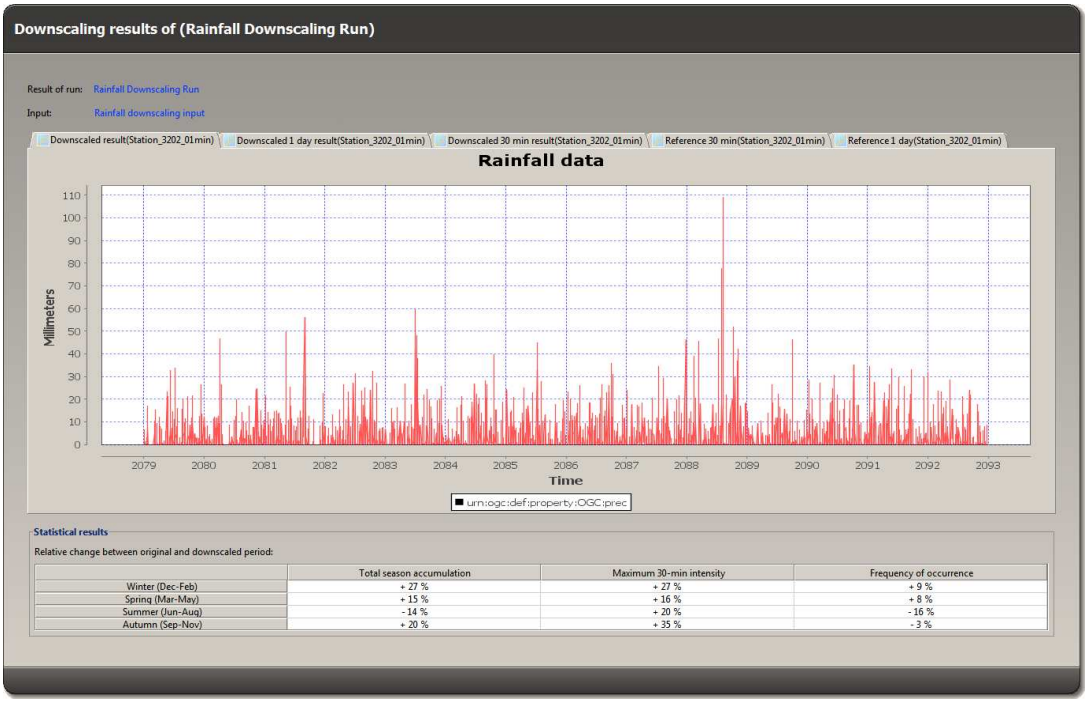

Figure 31: Visualisation of Model Output (Time Series)

There is also the possibility of visualising model results in the map while taking temporal aspects into account. SUDPLAN is required to visualise changes over time in the map, but geographic features retrieved from WFS or raster image layers retrieved from WMS are static objects and WFS and WMS clients do not in general support temporal representation. Therefore, in V1 a new type of FeatureRenderer was developed that is able to interactively change the feature shown depending on the selected point in time. In the example shown in *Figure 32: Interactive TimeseriesFeatureRenderer,* ozone (O_3) concentration over a period of 100 years can be seen in 10 year intervals.

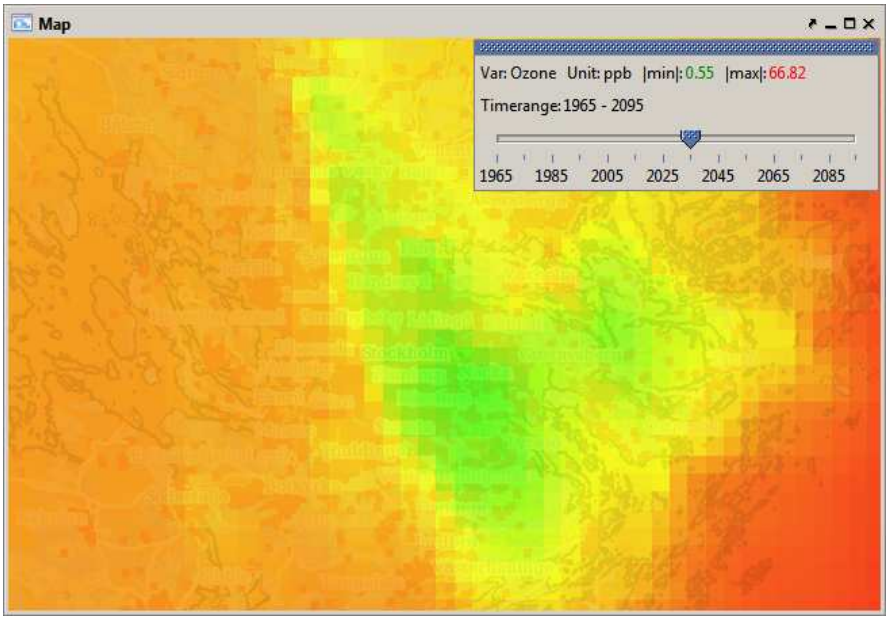

Figure 32: Interactive TimeseriesFeatureRenderer

4.1.3 Second Year Development

Besides the general improvements and enhancements of V1 features like the development of a Download manager, the enhancement of the SMS catalogue (soft refresh), IDF visualisation support, the integration of the new, World Wind based 3D component, the developments in the second year concentrated on the stable integration of Common Services (Pan-European and Rainfall in particular), the integration of local models (pilot specific), and local data (e.g. time series) and the support of the integration of common service downscaling results into local model runs, with the help of the Scenario Management System. The foreseen integration of the Hydrology CS was moved to V3, however interactive and collaborative preparatory work of WP3 and WP4 has prepared the ground for an easy integration in Y3 including the basic downscaled data, interface specifications and a first back-end implementation (Java-API) that needs to be wrapped by a set of OGC services to complete the work. Regarding the SMS, additional emphasis has been put on the enhancement of time series visualisation and comparison features and enhanced asynchronous model control. Both aspects are described in more detail in the following sections.

4.1.3.1 Time Series Visualisation and Comparison Framework

It is obvious that most of the data available in SUDPLAN consist of time series as they deal with environmental data that are measured or calculated for certain points in time. The objective of this work is to develop a component to visualise and compare these time series.

The component is designed as a highly flexible and extensible framework that abstracts and hides visualisation dependent tasks, and offers uniform interfaces for the integration and access of the visualisation in the SUDPLAN user interface. This permits an easy integration of different visualisations for time series depending on their individual visualisation requirements

(scalar, vector, gridded). The use of established software design patterns ensures easy access to and re-use of the framework.

The time-series visualisation component has to support the SUDPLAN users at their decision tasks. This requires the visualisation to be interactive and to support the user in the exploration of the processes that are represented by the time series. Besides multiple interaction techniques, particular emphasis is put on the comparison of multiple time series. In fact, SUDPLAN users need to compare results of various planning alternatives. The comparison abilities are extended by providing various operations that can be executed on time series.

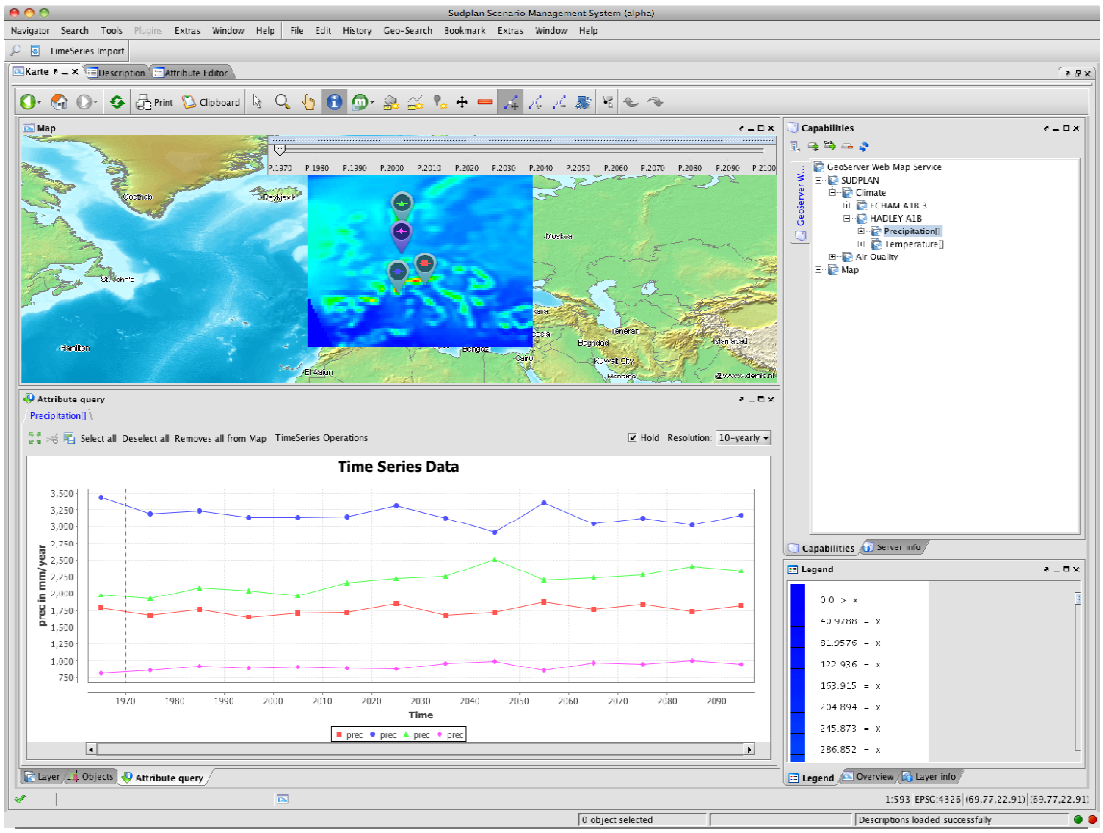

Figure 33: SMS Using the Time Series Visualisation and Comparison Framework

The basic idea of the new component is to support the user in the visual exploration of time series and the underlying processes. The component has to handle two different types of time series. Simple time series, such as temperature measured over a period of time and time series of grids. Time series of grids contain multiple data values (the grid) combined with a spatial reference for each time value. The visualisation of time series of 2D grids differs from regular 1D time series and thus demands dedicated visualisation methods. At this point it is not clear how multiple time series of grids should be visualised in SUDPLAN to gain an intuitive and comparative visualisation. Therefore we will concentrate on the visualisation of simple time series. However, due to the requirement of 2D grid time series visualisation, the component needs to be designed independent of the visualisation approach, which allows it to extend the component with a visualisation for time series of grids.

For simple time series the most frequent task is the display of the time series in different resolutions. This means that the user has a need to get very detailed views of time series as well as overviews which emphasize long term trends. The comparison of time series is also another important user task. As a consequence, the visualisation must display multiple time series simultaneously, in single or multiple graphs. Another feature requested by the end users is the ability to execute operations on time series, for instance computing the difference or average of two time series. In any case, to facilitate comparison of time series, interaction with the charts is mandatory. Making it possible to scroll the data over time, zooming in or out the data, showing detailed information of points of interest or changing the chart type increases the visual exploration experience significantly and is needed by the SUDPLAN users.

The possibility to zoom in and out the data leads to another more technical issue. SUDPLAN users generally handle time series with a huge amount of data because of high resolution over long term periods. Even the first prototype of the SMS contained time series up to 200MB of raw data. SUDPLAN is designed in a loosely coupled and service oriented architecture and therefore these data must be first transferred before they can be visualised. Moreover, it must be possible to dynamically load and cache data dependant on the area of interest and the chosen granularity.

4.1.3.1.1. Basic Chart Visualisation API

There are already charting solutions for Java. They all offer the possibility to create various types of charts such as point, bar and line charts and are therefore a good basis for creating a component for time series visualisation.

The following Java charting APIs have been examined with regard to their ability for time series visualisation and comparison and their ability for interaction.

ChartFX[1] is a commercial API for creating charts. It is the only API found so far that has built-in functionality for scrolling the dataset over the x-axis. It also has the largest number of chart types available. But due to the fact that SUDPLAN is developed as an open source project the charting API has to be available under open source license, which is not the case for ChartFX.

The Java Chart Construction Kit (JCCKit[2]) is a very small and flexible framework and is especially designed for scientific charts and plots. It is very effective for dynamic and real-time charts, in which it is necessary to reload the data permanently, but the interaction possibilities with the charts are very limited and the API is not well documented.

OpenChart2[3] is another open source Java charting API. The functionality is similar to the JCCKit. The documentation of the API is just made up of the Javadocs and a very limited set of developer tutorials. Furthermore the built-in interaction functionality is very limited e.g. OpenChart2 does not even directly support the export of charts to image files.

The most suitable API is JFreeChart[4]. It is the most popular Java Charting API and is developed under GNU LGPL[1]. It is especially designed for the use in Java Swing and has a large

[1] http://www.softwarefx.com/SFXJavaProducts/CFXforJava/
[2] http://jcckit.sourceforge.net/
[3] http://freecode.com/projects/openchart2
[4] http://www.jfree.org/jfreechart/

set of different chart types. Charts created with JFreeChart provide the ability to zoom in and out of the data, dynamically add data points, and add or remove complete time series. Additionally JFreeChart comes with an event handling system similar to Swing that makes it possible to react to users' input (such as mouse clicks), which is used to generate a popup menu that provides the user further functionality. The API is well documented and a very extensive Developer Guide with many code examples is available. Furthermore, a forum on the project website exists, which can be used to get help from other JFreeChart users and the developer team. Although JFreeChart does not support all necessary features out of the box, it is possible to extend the API according to our needs.

SUDPLAN users need to compare time series with different variables in one chart, e.g. temperature and ozone. This means that the chart has to provide different scales and labels on the axis. JFreeChart supports line charts with multiple axes, but as more axes are drawn on the chart the less comprehensible and clear is the chart for the user. Also not supported as a standard feature is the ability to scroll the data on the time axis. The JFreeChart API must be extended to add that feature. Also possible, but not yet implemented, is a combination of the zooming functionality and the possibility to reload data.

Because of the aforementioned issues simple line charts are used to visualise time series and JFreeChart is used only as a backend API handling the creation of the charts.

4.1.3.1.2. Framework Design

It is necessary to easily reuse and integrate the new component for time series visualisation in different parts of the cids platform or any other software. As a consequence a framework that provides the necessary functionality to create time series visualisations and hides the implementation details of the chart API, as they are described in the following chapter, has been designed. The framework is designed in a generic way, imposing no restrictions on the concrete implementation of the time series visualisation. One benefit is that it is easy to extend the framework with new kinds of visualisations or backend APIs, or to replace existing ones. Another reason for this generic approach is that SUDPLAN has the requirement of two different kinds of visualisation, namely for simple and gridded times series. While simple time series can be easily visualised with standard line charts, gridded time series need more complex and customised visualisations. Before the design and functionality of the new framework is discussed, the *TimeSeries API* which is used to represent time series objects within the constructed time series visualisation framework is introduced.

4.1.3.1.3. The TimeSeries API

The Time Series API (TS-API) is being developed by the Austrian Institute of Technology and is described in 4.2.1. It is a part of the time series toolbox (TS-Toolbox) which is "*a high level programming framework that allows efficient access to, processing, archiving and presentation of the semantically enriched time series.*"[2] It consists of generic interfaces to represent time series, which is the main reason it is used for data representation. It acts as a kind of adapter because it is easy for any user to develop their own implementation or an extension of

[1] http://www.opensource.org/licenses/lgpl-license
[2] Havlik, D., Bozic B. http://ts-toolbox.ait.ac.at/TS-Toolbox/pdf/TSTB_Documentation.pdf, Time Series Toolbox. Overview and Component Specification (2010)

an existing one. Furthermore, it abstracts the time series representation from the concrete visualisation API and the datasets used there.

The TS-API describes a time series as a set of slots and properties. The properties are valid for the whole time series and hold further information about it. Important properties that are used within the framework are the *unit keys* that describe the unit the time series values are measured in and *value keys* that act as an ID to access the data values. A slot always contains a `Timestamp` which acts as an identifier. It can contain zero or more values which can be identified by a value key. The prevailing value keys are stored in a time series property. Figure 34 illustrates the structure of a TS-API time series.

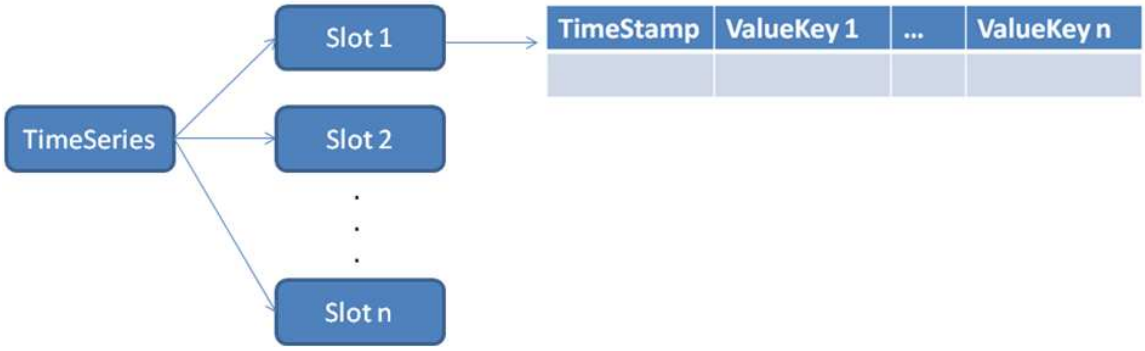

Figure 34 : the fundamental structure of a time series in SUDPLAN

The TS-API makes no assumption or limitations on the data that are stored, which allows the creation of gridded time series without special implementations or extension of the API, another benefit of using the Time Series API. But the TS-API has a further benefit. As already mentioned, the new component must be able to execute operations on time series data. The AIT TimeSeries Toolbox contains a module called *Formular 3* (F3) which allows definition and execution of operations on TS-API time series based on a customised input language. Operations on time series, their implementation in this framework and the functionality of F3 are introduced later.

4.1.3.1.4. Framework Conceptualisation

The basis of the framework is the interface `TimeSeriesVisualisation`. This Interface acts as the data model of the visualisation and basically offers methods to add and remove time series objects. `TimeSeriesVisualisation` defines a set of properties like the title of the visualisation. The use of properties enables each instance of a `TimeSeriesVisualisation` to be extended with individual properties. To keep the real visualisation of the time series data separate from the framework, the `TimeSeriesVisualisation` interface defines the method `getUI`(), which returns an instance of JComponent. As mentioned before this makes it possible to easily extend or replace the underlying visualisation. Each `TimeSeriesVisualisation` can define a *Toolbar* which offers the user specific interaction functionalities. To avoid integrating the toolbar at a static point within the visualisation user interface, the `TimeSeriesVisualisation` interface defines the `getToolbar()` method. This makes it

possible to determine the position of the toolbar individually or to leave it out completely if not needed.

To simplify the usage and creation of concrete `TimeSeriesVisualisation`, the framework is extended with a factory class `TimeSeriesVisualisationFactory`. This class defines the factory method `createTimeSeriesVisualisation(type)`. The type

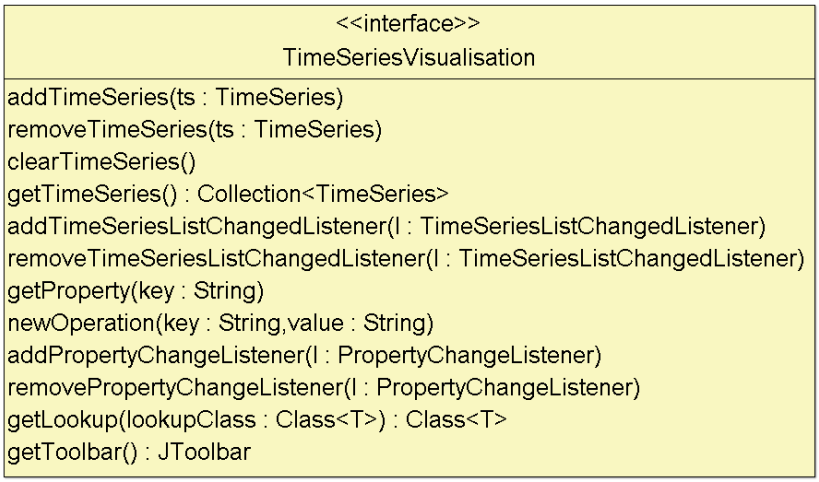

Figure 35: TimeSeriesVisualisation Interface Specification

argument makes it possible to distinguish between visualisation for simple and gridded time series, which have totally different requirements. The usage of a factory allows it to generate multiple and different implementations of the framework and offers easy access to them. Therefore the factory method creates an instance of `SimpleTSVisualisation`, which is the default implementation for simple time series in case the visualisation type is simple, which corresponds in design and functionality to the implementation described later.

4.1.3.1.5. Event Lookup Mechanism

It can be necessary that another component needs to know when time series were added or removed from the visualisation. The selection of time series can also be interesting for other components. Finally, it is likely that a concrete implementation needs to define visualisation specific properties or behaviours. Therefore the framework defines different event notification mechanisms according to the *observer pattern*. In general, the prevailing events can be divided in two classes. The first class is necessary for every visualisation that is created by the framework and notifies *listeners* about removed or added time series, for example. The second class is visualisation specific, such as the selection of time series, and cannot be provided in a general way.

Because of this difference, the realisation had to be executed in two different ways. The event notifications that are valid for all visualisations were integrated directly in the interface `TimeSeriesVisualisation`. These are only those methods to add and remove listeners that are interested in changes of the stored time series set. Those listeners are described by the interface `TimeSeriesListChangedListener.` The interface contains only one method that is called whenever the stored set of time series has changed. The state of the properties can also be observed.

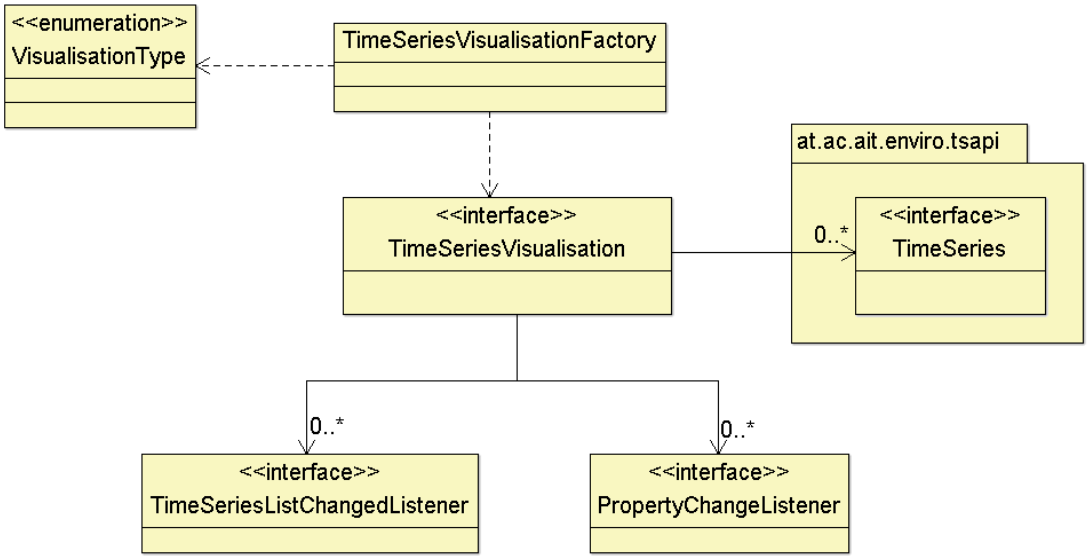

Figure 36: Framework Structure, Part 1

The event notification for all visualisation specific events as well as visualisation specific behaviour is defined in separate interfaces and must be implemented by the visualisation if it needs support for this event type or behaviour. Choosing this approach is much more flexible than defining all methods directly in the `TimeSeriesVisualisation` interface and allows fine grained control of the visualisation. Additionally it is much easier to define a new event notification mechanism and extend a visualisation to them if needed. But in doing so a new problem is created. The built in factory returns an instance of `TimeSeriesVisualisation`. How does the user of the factory know which event notifications mechanism and what visualisation specific behaviour is supported by the delivered `TimeSeriesVisualisation` instance.

To solve this problem the `TimeSeriesVisualisation` interface is extended with a lookup mechanism. A `TimeSeriesVisualisation` can be asked if it supports a specific interface. This is done by calling the method `getLookup`() which takes the class object of the interface as a parameter.

The lookup returns an instance of the interface if it is implemented by the `TimeSeriesVisualisation`class.

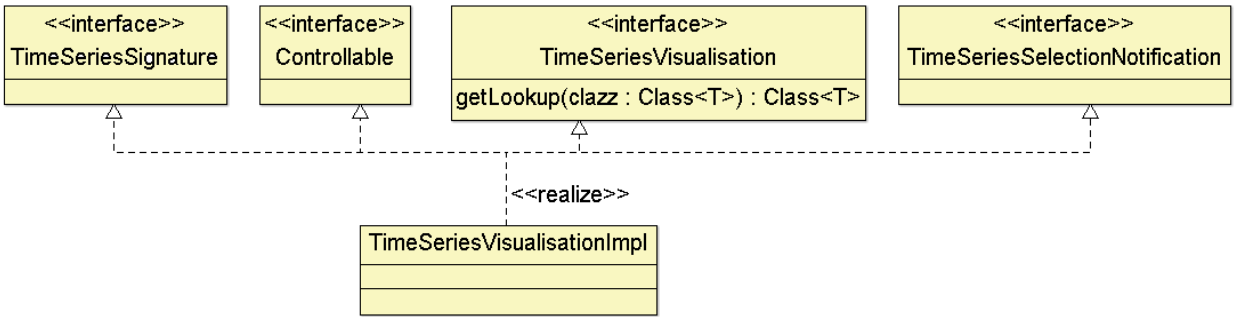

Figure 37: Framework Structure, Part 2

4.1.3.1.6. Comparison Operations

A requested feature for the time series visualisation framework is the ability to execute operations on time series. As mentioned before this task especially enhances the comparison of time series. The necessary extension of the former framework and the implementation of the operations will be topic of this section.

4.1.3.1.7. Framework Extension

An operation that can be executed on time series is represented by the interface `TimeSeriesOperation`. Because these operations must be executed from a `TimeSeriesVisualisation` and the result must be visualised there too, the `TimeSeriesVisualisation` interface is extended with methods to add and remove `TimeSeriesOperation` and a listener mechanism that makes it possible to be notified whenever an operation is added or removed from the visualisation. This flexible design makes it possible to add operations dynamically to the visualisation and to adjust the applied operations to the needs of the visualisation. It also is conceivable that the new operations are created and defined dynamically by the user. Of course this would cause further implementation effort, but the fundamental design is already flexible enough to tackle this challenge.

The operations are designed after the Command Pattern which is already realized in Java Swing in terms of *Java Actions*. Java Actions allow the definition of functionality at one place and access to it from multiple control elements. Furthermore they contain information like description texts and icons that are used to represent them in the user interface, the enabled state that allows it to enable or disable the execution of an action. The use of Actions is advantageous because they separate the execution code of the operation from the visual representation, and hence can be used for multiple visualisations at the same time. Simultaneously they are able to influence the visual representation of the operations in terms of defining the icons and text.

The loose coupling of operations and the visualisation as well as the use of Actions results in one problem. The method that is called if an `Action` is executed does not return any value. This means that the result of the operation must be communicated to the visualisation in a different way. Thus the `TimeSeriesOperation` interface is extended with a listener mechanism. After the execution of an operation all registered listeners are notified about the result.

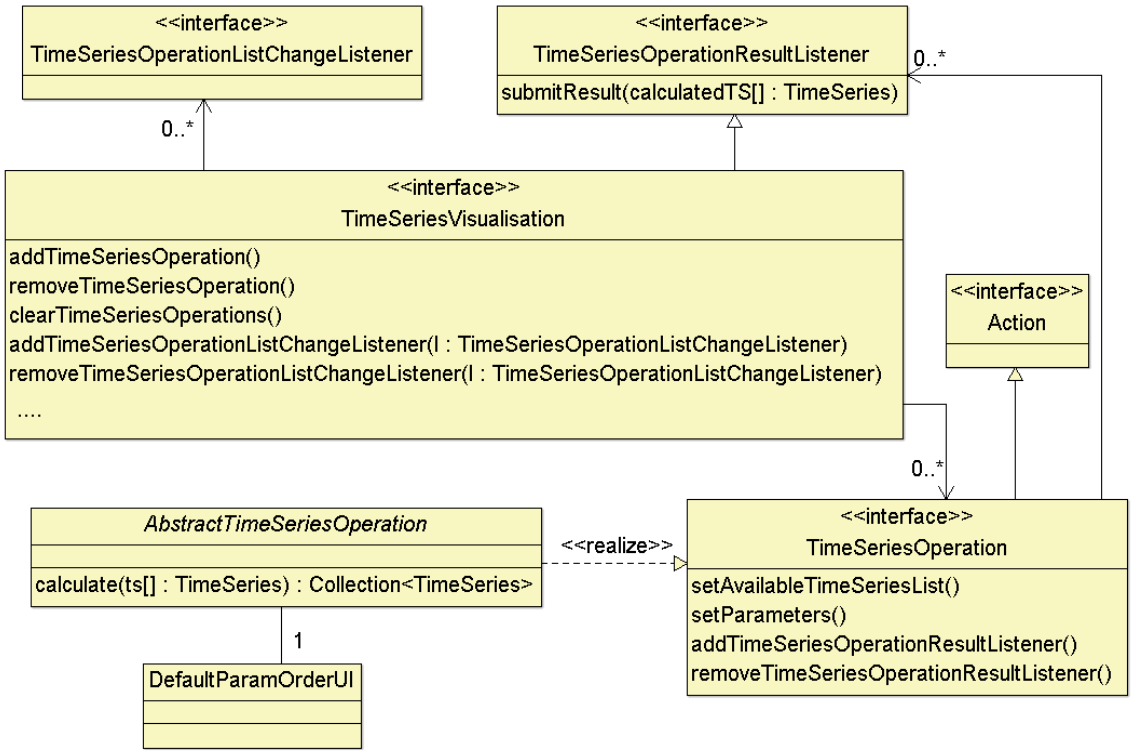

Figure 38: Time Series Operations

The operation has no information about what time series it should use as input parameters. The possible parameters are determined by the selection of time series. The operation must offer the possibility to select the parameters as a subset of these selected time series. The two sets can be set with the corresponding methods `setParameters()` and `setAvailableTimeSeriesList()` that are defined in the `TimeSeriesOperation` interface. The parameters are represented as an *array* which leads us to the next problem. The position of the parameters has an operation dependent semantic which must be taken into consideration when determining them. This is done with a special user interface that visualises this operation dependent semantic and allows the user to order the parameters as desired.

To keep the creation process of new operations as simple as possible, an abstract class `AbstracTimeSeriesOperation` is created that takes care of all the necessary steps to keep the enabled state up to date, to determine the parameters, notify registered listeners about the result and execute the real calculation in a separate thread. Also a `DefaultParamOrderUI` which takes care of the aforementioned problems regarding determining the parameters is implemented and used by this class as default. `AbstractTimeSeriesOperation` defines the abstract method `calculate()` which contains the operation dependent calculation code. To implement a new operation it is only necessary to instantiate subclass `AbstractTimeSeriesOperation` and to implement the `calculate()` method.

4.1.3.1.8. Integration in the SMS

As a first step, the visualisation for simple time series is embedded in a concrete use case called 'Information on the European Scale'. This use case defines how the user can extract environmental information from the visualisation of the downscaled European climate data. This visualisation of the European climate data is done with multiple layers in the cids *cismap* plugin that shows different information, such as maps. The cismap plugin is a full featured *WMS/WFS* client. WMS and WFS are standardized web services that allow the retrieval of cartographic data. In general cismap visualises a map layer that can be extended with a set of additional information layers. The map and the additional layers are retrieved from a WMS server.

In the above mentioned use case, one possible configuration of cismap shows a map of the European continent. The user extends this map with layers that visualise the desired European climate data. Such a layer represents one environmental factor, for instance temperature or precipitation, and basically consists of a pre-rendered image of a gridded time series. These information layers are visualised on top of the European map.

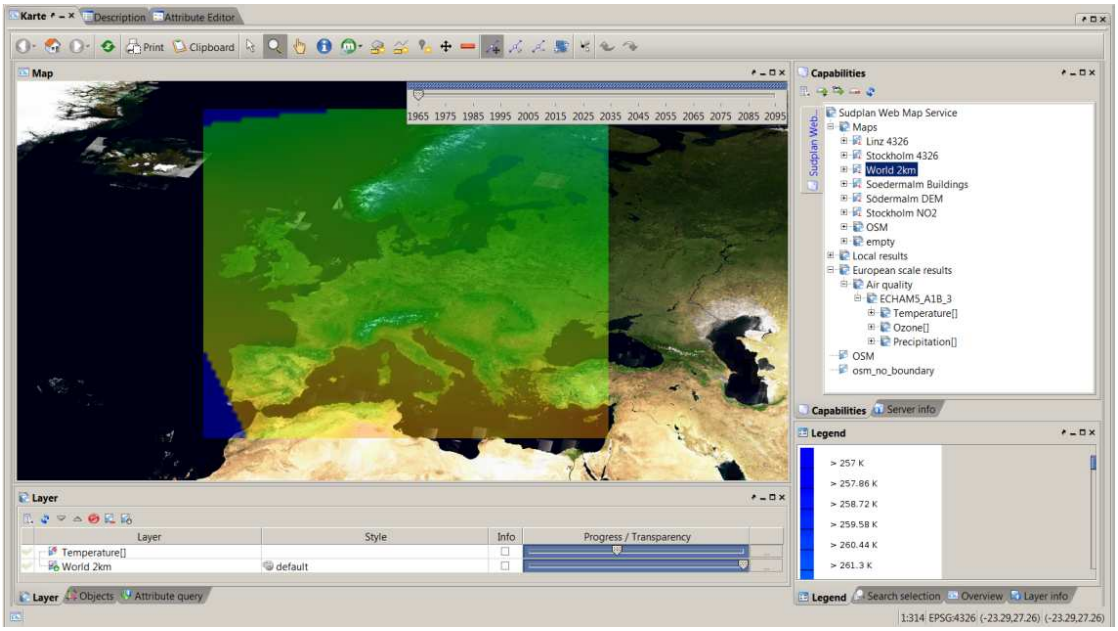

Figure 39: Pan-Europan Case - Screen Shot

If the user activates an information layer to allow feature information requests on it, the cismap plugin generates a FeatureInfoDisplay for this layer, and the WMS-Server is asked for the metadata necessary to request the corresponding Sensor Observation Service (SOS). The SOS is a standardized web service that allows the retrieval of sensor data. The SOS contains the time series for the European Climate data.

If the user now clicks on the map, a so called *feature info request* is actuated, and this is sent to the WMS–Server. In this case the server response contains the real world coordinates which are related to the position of the mouse click. In case the user has multiple information layers in the map, a feature information request for each activated layer is actuated. The coordinates and the previously retrieved SOS metadata are used to request a simple time series from the SOS which is related to the cursor position. This simple time series can be visualised with the new framework implementation for simple time series.

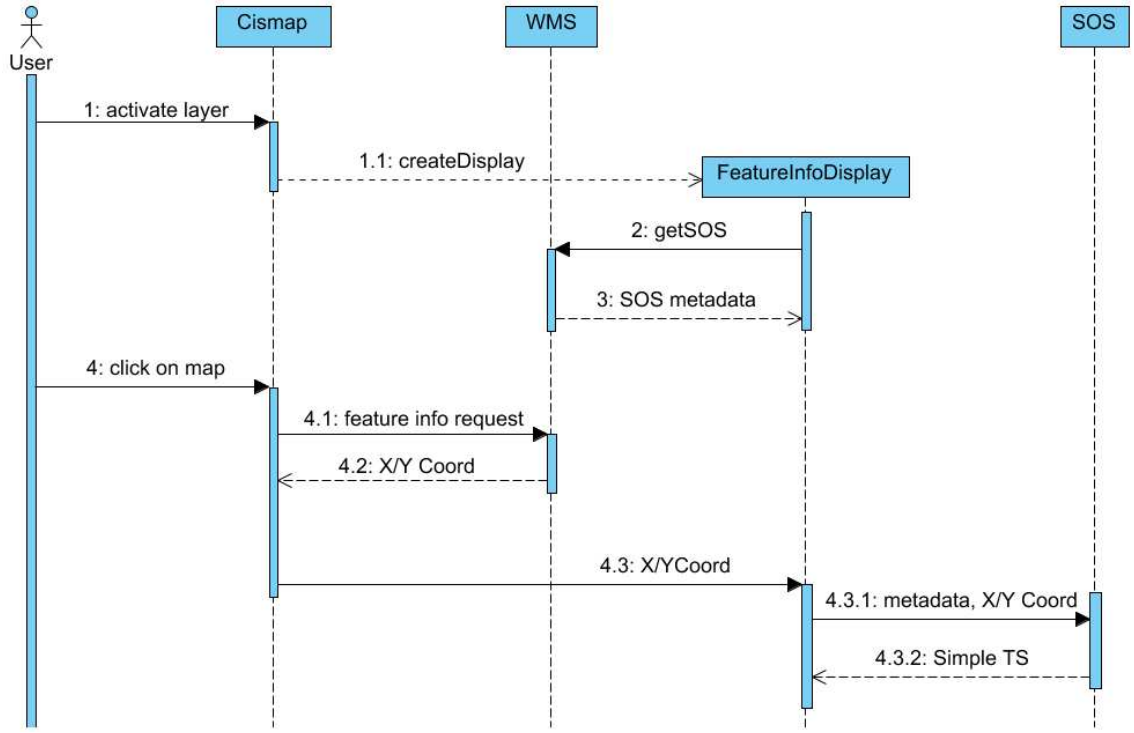

Figure 40: Time Series Retrieval v1

4.1.3.1.9. Retrieval of Multiple Time Series

The problem with visualising multiple time series is that each click on the map actuates a new feature information request for each activated layer, the server response only contains a single time series, and the corresponding display only visualises the new retrieved data. The `FeatureInfoDisplay` that visualises the information is extended with a hold flag. If the hold flag is set, the display takes care that all the information from all earlier requests is kept in the visualisation. To let the user control this behaviour a checkbox is added to the user interface of the corresponding `FeatureInfoDisplay`.

4.1.3.1.10. Multiple Axes

Another limitation of the cismap framework was that it generated a separate `FeatureInfoDisplay` for each layer. Because the individual environmental variables are separated in different layers to represent them individually on the map, it was not possible to combine the requested information of different layers in a single `FeatureInfoDisplay`.

JFreeChart supports charts with multiple axes by default and comes with all the necessary functionality to manage and display them. A JFreeChart object consists of a so called *plot area*. This is the area where the dataset of the chart, for instance multiple time series, and the corresponding axes are drawn. Axes can be arranged at all four sides of the data area. The whole plot also has an orientation to instantly switch the position of x and y axis. A plot is represented by an instance of `Plot`. JFreeChart also defines a set of subtypes that predefines the type of the used axis. As time series charts represent X/Y –values, a `XYPlot` is used which demands that instances of `ValueAxis` are used to represent the x and the y axis. The `Plot` offers methods to

add multiple x and y axes. They are used to generate separate y axes for each individual environmental factor like temperature or precipitation which usually have different units and therefore need individual scales.

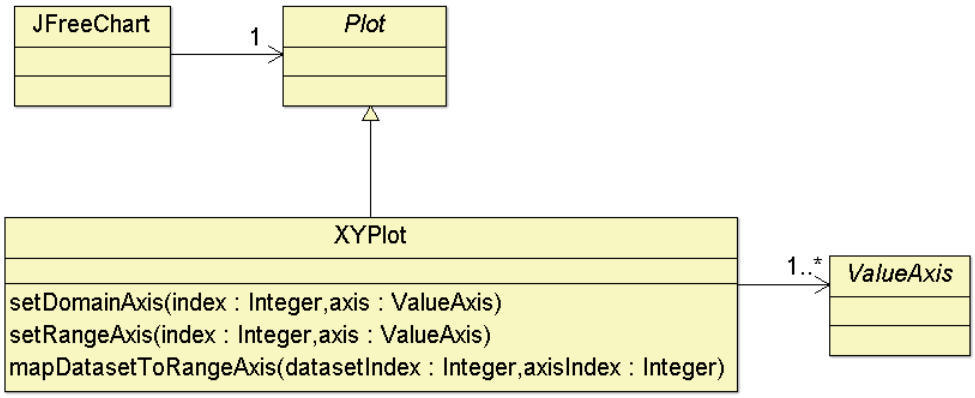

Figure 41: JFreeChart Class Structure –Multiple Axes

Basically all time series that are added to the chart are mapped to the primary axes of the chart. The primary axes are the ones which are generated during the instantiation of a JFreeChart object. This mapping ensures that the range of the axis always is in a consistent state with the datasets that are mapped to it. It is necessary to adapt this mapping to the new situation. To do this, the method `mapDatasetToRangeAxis(int datasetIndex, int axisIndex)` is used.

4.1.3.1.11. Time Series Selection

One issue that makes it necessary to extend the underlying JFreeChart API is the selection of time series in the chart. JFreeChart does not offer the ability to select single or multiple datasets within a chart. But this functionality is needed for multiple reasons. Selection of time series is essential to offer the user time series dependent interaction and information. The user should therefore have the ability to select time series to execute operations on them, to reflect the spatial geometry of the time series in cismap in order to show the user the spatial context of the data, and to remove single time series from the visualisation.

In JFreeChart, each dataset is drawn by an individual renderer. For time series charts a renderer is defined by the `XYItemRenderer` interface, and JFreeChart offers a set of previously implemented renderers, such as `XYLineAndShapeRenderer,` which is used as the default for time series charts and as the default for this implementation. An `XYLineAndShapeRenderer` draws, as the name suggests, the dataset as a combination of an individual shape for the data points within the dataset and a line between these data points. As a renderer is used to render a single dataset the selection status is stored directly in the renderer.

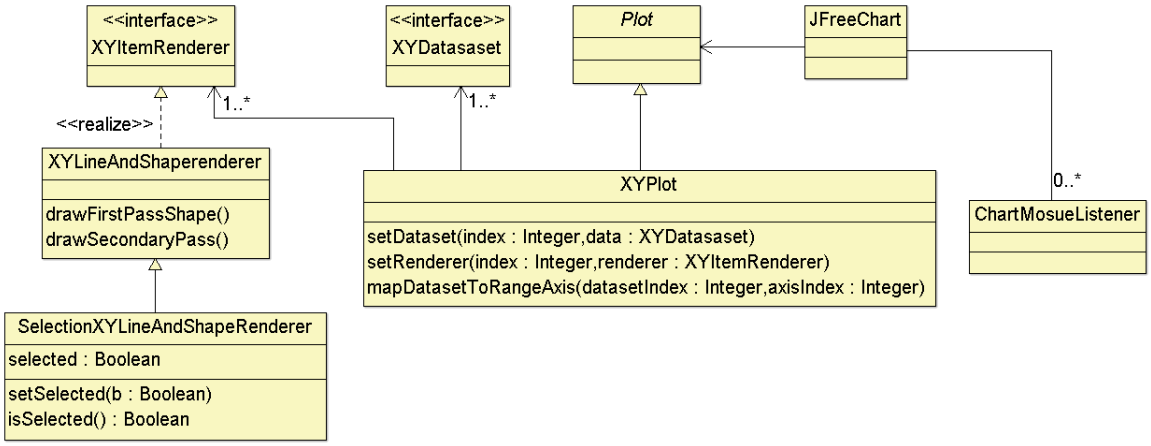

Figure 42: JFreeChart Classes Relevant for Selection

For this, a renderer class is derived from `XYLineAndShapeRenderer`. It contains the selection status and paints the corresponding dataset. The selection is visualised with a drop shadow, as it is known, from trees or lists. This helps the user to build his mental model of the functioning of the new component. The rendering of a time series dataset in JFreeChart is done in two steps. The first step draws the line between data points, the second one draws the shapes for the data points themselves. To draw a drop shadow for the selection the method `drawFirstPassShape()`, which takes care of drawing the line between the shapes, is overridden. If the selection flag is set, a second lightly transparent line is drawn on top of the real line. After that, the selection status of a time series still must be triggered by users' input. Although selection is not supported as a standard feature by JFreeChart it offers a customised event notification for mouse clicks. This event notification can be used by registering a ChartMouseListener to the chart object. This listener is notified whenever the mouse moves over the chart or a click is performed on it. The corresponding event of that notification contains an instance of ChartEntitity which represent the entity that is at the coordinates of the mouse click. Unfortunately JFreeChart does not distinguish these entities in a very detailed way. Hence it is not possible to identify the clicked entity as a time series dataset if the user clicks on the line between two data points and not at the data point itself.

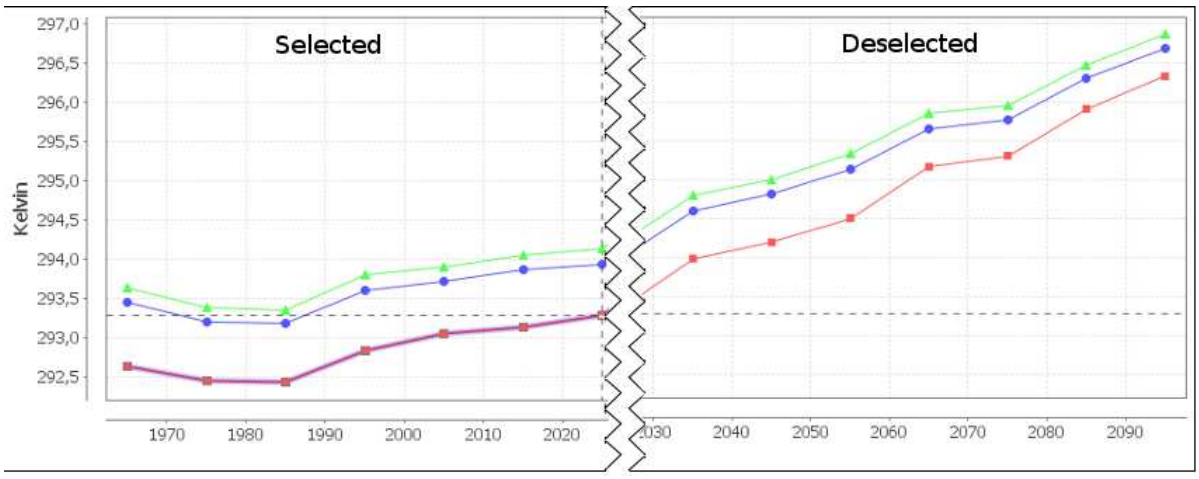

Figure 43: Selected / Deselected Time Series (red)

4.1.3.1.12. Time Series Scrolling

As mentioned in the introduction, one essential interaction method is to view the data in detail as well as in an overview and to navigate through the data. This is also one of the requirements of the new component. Charts created with JFreeChart can be zoomed in and out to view regions of interest in varying detail. During such a zoom action, the visual appearance of the chart is updated, which means that the axis labels are updated to the new viewing region and the datasets are redrawn. But in doing so, the user has no possibility to scroll the data over time. If they want to see the data in the same detail but for a different region they need to zoom out and then zoom in again. To avoid this, the chart is extended with a scrollbar that offers the user the ability to scroll over the whole time period of the dataset when zoomed in. To do this, the scrollbar must handle the time period that is represented by the chart's x-axis. The x-axis in a time series chart is always an instance or subtype of `DateAxis`. A `DateAxis` represents the values as a date. This is a serious problem, because dates in Java are represented as long values because a date is the time in milliseconds since midnight 1.1.1970. However, the scrollbar can only handle integer values. Therefore it is necessary to map the axis long values to integer values. In doing this the time period of the axis is divided by the number of possible integer values. This allows it to resize the scrollbar extent whenever the x-axis range has changed by a zooming event and to adopt the range of the `DateAxis` if the scrollbar is moved. In the case of the mapping between the long date values and integer scrollbar values, zooming in the data is only possible up to a limited granularity. This granularity depends on the time period that must be visualised in the overview. A possible solution of this disadvantage is to provide only two buttons that scroll the data to the left and to the right.

4.1.3.1.13. Spatial Context of Time Series

It is useful to show the spatial context of a time series directly during the retrieval and not only with the selection of it. Until now, the cismap plugin just displayed a standard icon, called *feature info icon*, for the last click on the map. Furthermore the feature info icon had no relation to the results of the feature info request that was actuated. Even if this use case is the first where it will be necessary to show multiple and customized symbols during a feature info request in cismap, such a requirement will possibly appear in the future again. Therefore the solution to that issue needs to be generic and easily applied to other situations.

A possible solution to this problem is to introduce a new interface in cismap, namely `MultipleFeatureInfoRequestDislpay`. This interface consists of two methods. The first one returns the status of a hold flag used to determine that the FeatureInfoDisplay visualises also old information requests. The second one returns a collection of `SignaturedFeatures`. In cismap a `Feature` is an object that can be displayed on the map and a `SignaturedFeature` additionally defines an icon that should visualise a relation to the information of the feature info request.

In this case this icon consists of the shape used to represent data items in the time series chart.

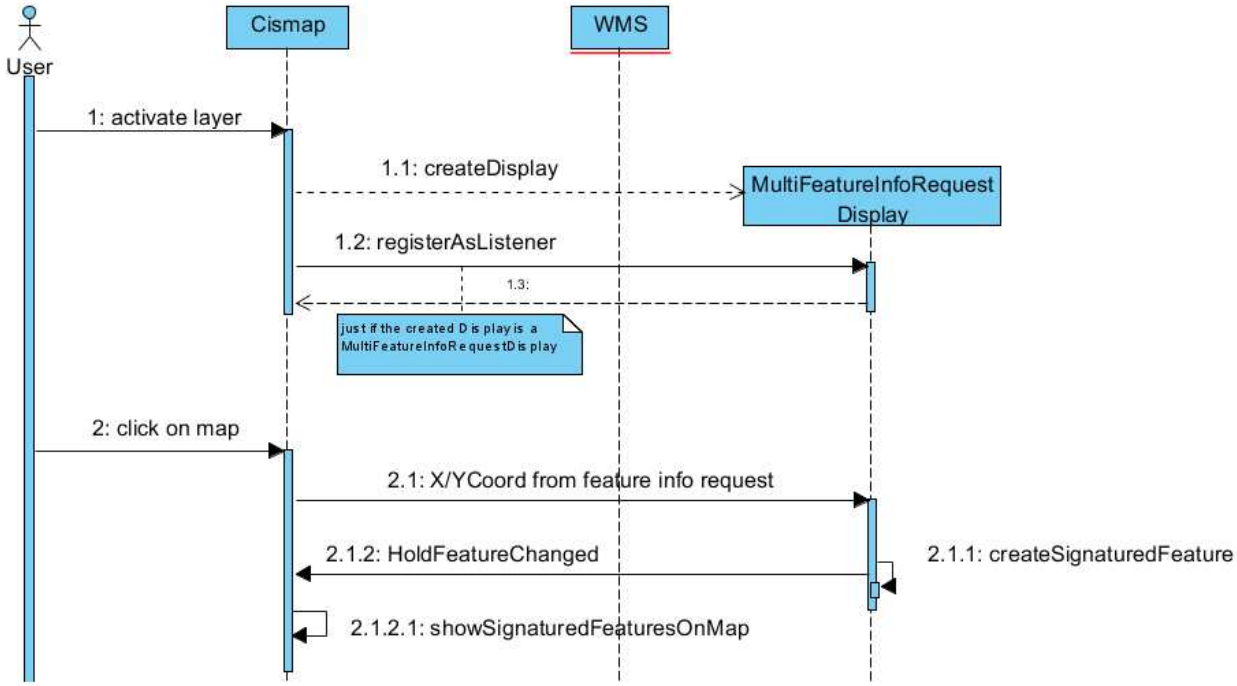

Figure 44: Time Series Retrieval v2

When the user activates a layer for feature info requests, cismap creates a `FeatureInfoDisplay` for that layer. As a consequence of the new interface, cismap is adapted to see if this `FeatureInfoDisplay` implements the new interface. If so, cismap registers itself as a listener to the display to be notified if the collection of `SignaturedFeature` has changed. If the user now clicks on the map and actuates a *feature info request*, the display creates a `SignaturedFeature` for this request and notifies cismap about that. Cismap then displays a feature info icon for each `SignaturedFeature` that the notification of the display contains. The standard feature info icon is overlaid with the icon of the `SignaturedFeature`. Because the feature information icon is customizable, and therefore the size and position where the overlay icon can be drawn can be changed, the feature information icon is extended with meta information that describes the position and the size of the overlay area and the background colour used.

Because multiple layers can be activated for feature information requests, it can occur that both `MultipleFeautureInfoRequestDisplay` and normal displays are activated at the same time. Therefore it is necessary to emphasise the icon for the last feature info request to show the spatial context for normal displays too. This is done by using a different colour for the feature info icon.

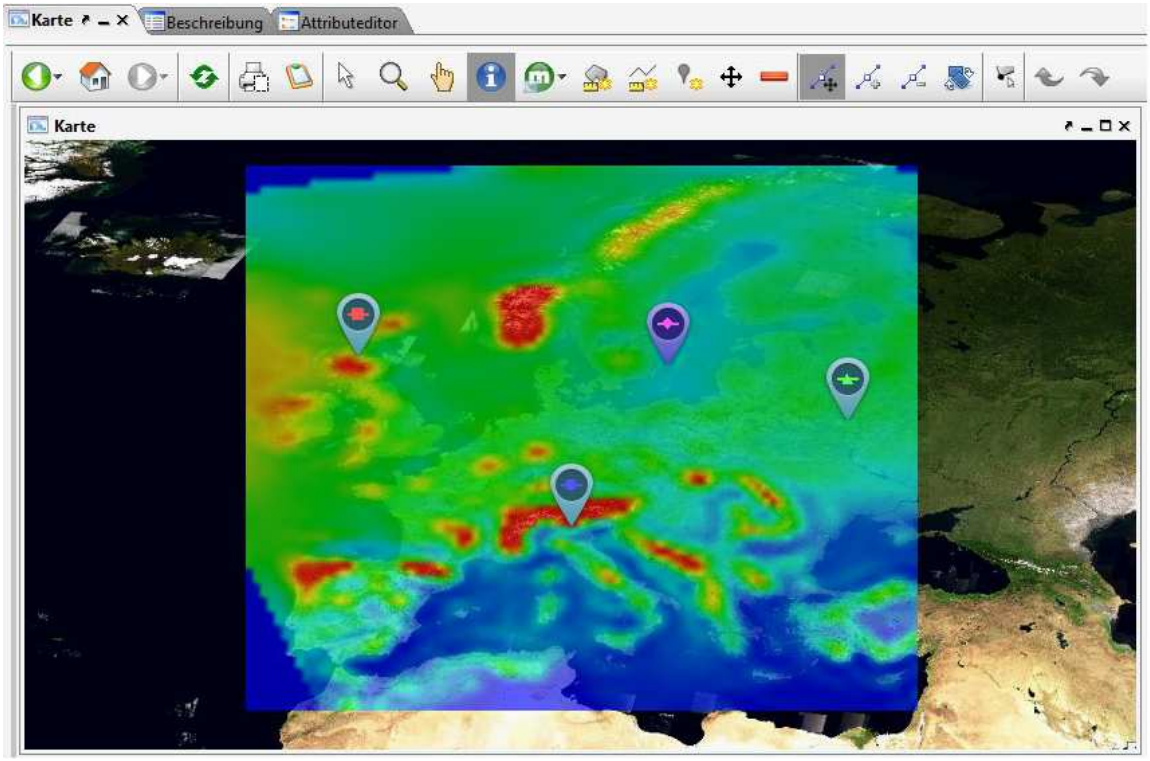

Figure 45: Time Series Spatial Context

4.1.3.2 Asynchronous Model Execution Framework

SUDPLAN Applications integrate a large variety of models –Common Services and Local Models - that can be executed via the SMS. The execution time of these models differs from seconds to several weeks. Thus the SMS provides an Asynchronous Model Execution Framework so that implementers can integrate models quickly and easily without the burden of status polling and user notification. Additionally the framework offers automatic unfinished execution recovery so that model status monitoring can be continued if the SMS is reopened. This is especially useful for long running tasks.

4.1.3.2.1. Framework Classes

Figure 46 depicts a UML class diagram that contains the core classes of the Asynchronous Model Execution Framework.

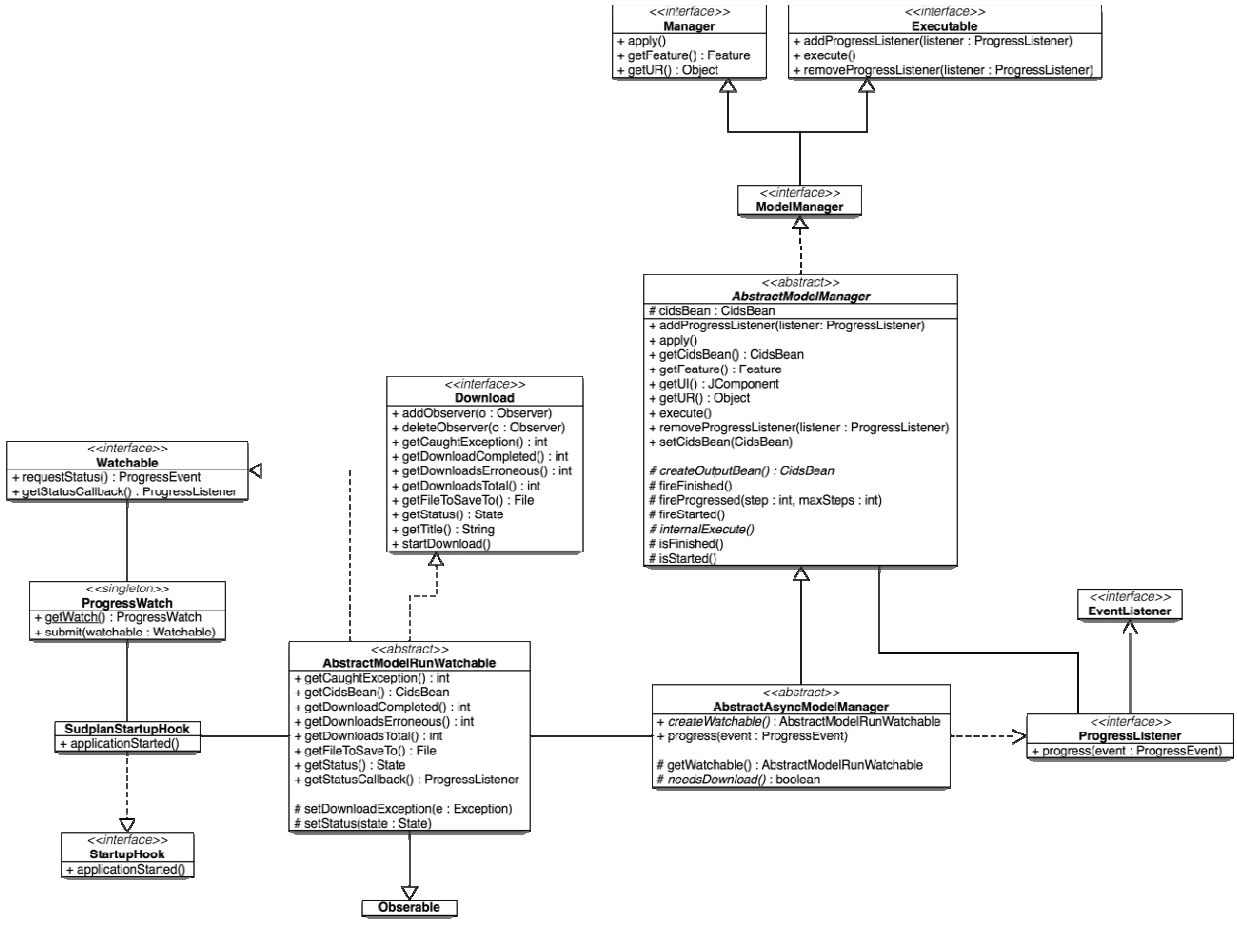

Figure 46: Asynchronous Model Execution Framework

The most relevant classes and their responsibilities are listed in the following.

This description focuses the asynchronous model execution framework and may not necessarily be sufficient to gain a comprehensive understanding of the SUDPLAN Model Management Framework, let alone the cids Application Framework.

- *Manager:* The *Manager* interface is the most basic interface of the SUDPLAN Model Execution Framework. Any model integration must provide three classes: One responsible for the model input, model output and model execution, respectively. Each of the classes has to adopt the *Manager* interface.
- *Executable:* Each class that is responsible for the actual model execution must adopt the *Executable* interface, additionally.
- *ModelManager:* The *ModelManager* interface simply merges the *Manager* and the *Executable* interfaces for reasons of convenience.
- *AbstractModelManager:* The *AbstractModelManager* class adopts the *ModelManager* interface and provides implementations for the required operations. Moreover it offers facilities to handle model execution progress. However, it forces an actual model manager implementation to be able to actually perform the model execution and to be able to create a so-called *CidsBean* (*CidsBean*s are part of the cids Application Framework and are not further described here) that represents the output of the model. Model integrators are encouraged to subclass this class as it handles most of the task involved in the model execution by default. However, it does not provide support for asynchronous execution by default.
- AbstractAsyncModelManager: This is one of the core classes involved in the Asynchronous Model Execution Framework. Model integrators must subclass this class to take advantage of it. The *AbstractAsyncModelManager* class forces the adopter to provide some more information that is necessary to integrate smoothly with the AMEF, but takes care of model execution progress and potentially download dispatch on execution finish.
- AbstactModelRunWatchable: This is another core class of the AMEF. It adopts the *Watchable* as well as the *Download* interface and subclasses the *Observer* class. This *Watchable* implementation provides default settings to instruct the *DownloadManager* to potentially download the model results and provides the preferred status callback implementation. The only two things the adopter of this class must take care of is an appropriate way to provide status information of the specific model execution and an appropriate way to download the model results if this is necessary.
- Watchable: The *Watchable* interface describes the necessary operations to monitor a specific resource. *Watchable* instances can be submitted to the *ProgressWatch* class.
- ProgressWatch: The *ProgressWatch* class is a singleton instance that is responsible for handling of *Watchable* instances. Thus it polls the model execution status using the *Watchable* interface and propagates the status via the provided status callback.
- Download: The *Download* interface describes the necessary operations if a resource shall be downloadable via the *DownloadManager* (the *DownloadManager* is part of the cids Application Framework and is not further described here).
- SudplanStartupHool: The *SudplanStartupHook* is an implementation of the *StartupHook* interface and thus is responsible for the recovery of unfinished runs on SMS startup, so that they can be monitored again.

- StartupHook: This interface is part of the cids Application Framework and provides a means for interested parties to perform operations on application startup.

The Asynchronous Model Execution Framework is highly parallelized and thread-safe. Thus adopters won't have to take care of threading and can directly implement the required operations of the framework classes in a synchronous (not synchronized) fashion.

4.1.3.2.2. Framework Sequence of Actions

This section provides an overview of the sequence of actions for a single model execution via the Asynchonous Model Execution Framework and thus illustrates how it works. However, in order to keep things simple threading is mostly omitted. Moreover some workflows that are not directly related to the model execution or that are part of other frameworks are indicated by a "virtual" operation call. Ultimately the depicted sequences may not be exhaustive, in order to keep things simple.

4.1.3.2.2.1. Initiation of the Model Execution

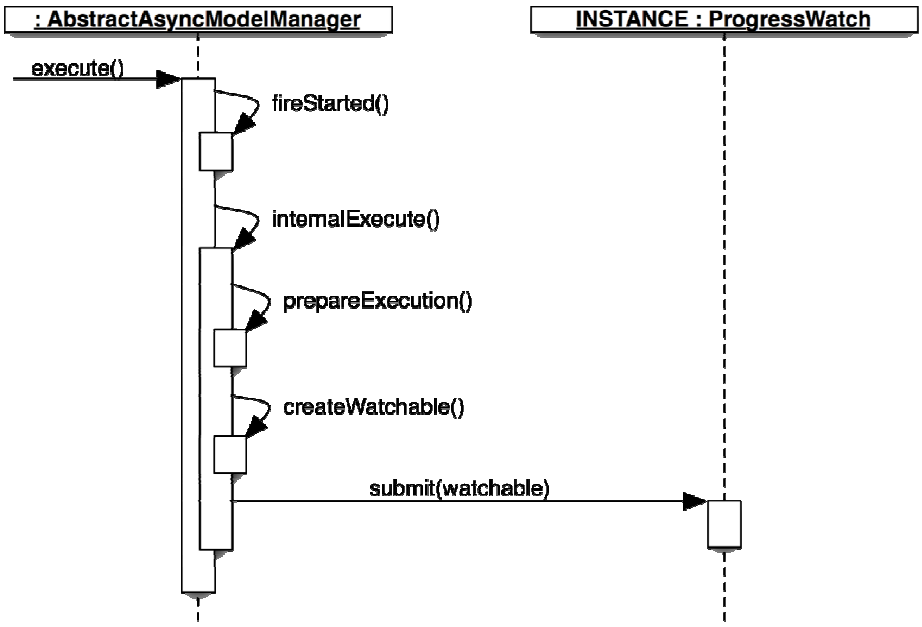

Figure 47: Initiation of the Model Execution

- The model execution is initialised through a call of the `execute()` operation. In most cases this by the SMS-internal *ExecutableThreadPool*.
- The *AbstractAsyncModelManager* (*AAMM*) instance indicates the start of the model by calling the `fireStarted()` operation.
- After that the `interalExecution()` operation calls `prepareExecution()`.
- The `perpareExectution()` operation implementation is actually responsible the start the specific model and has to make sure that a call to `createWatchable()` will provide an adequate *AbstractModelRunWatchable* (*AMRW*) implementation for the specific model execution.

- The *AAMM* fetches the *AMRW* via `createWatchable()` and submits it to the *ProgressWatch.*

4.1.3.2.2.2. Execution Status Monitoring

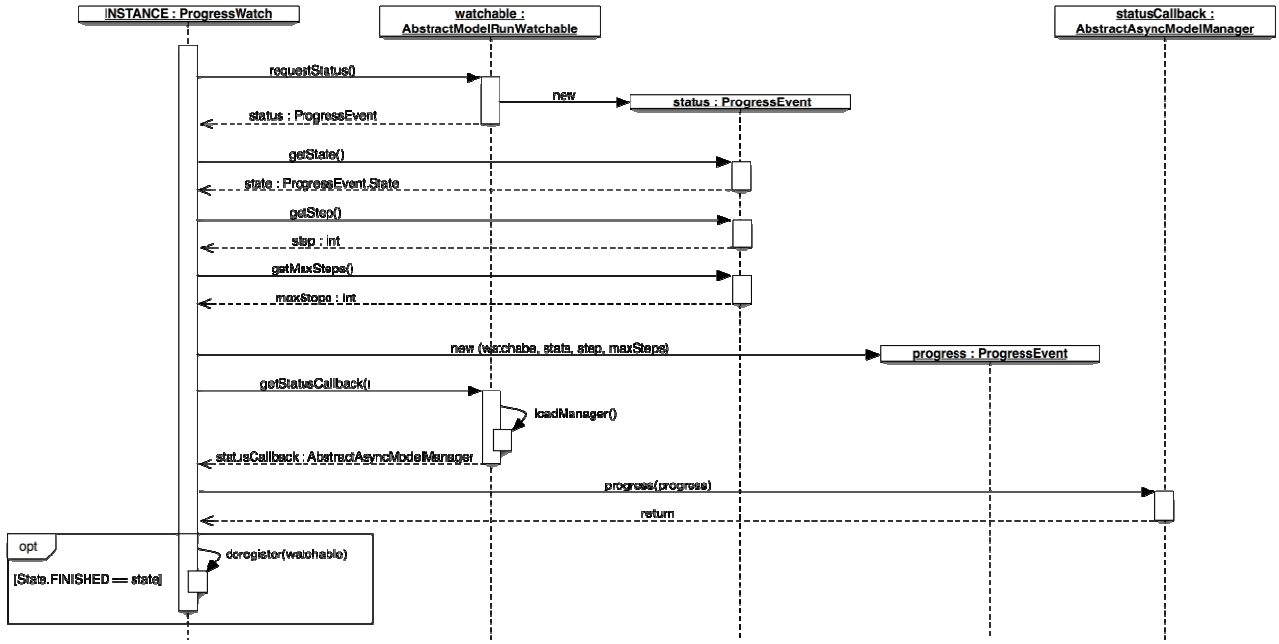

Figure 48: Execution Status Monitoring

- The *ProgressWatch* is responsible for monitoring the status of all submitted *Watchable* instances. Thus it originates polling the `requestStatus()` operation of the *Watchable.*
- The *Watchable* instance creates a new *ProgressEvent* reflecting the current status of the specific model execution and returns it.
- The *ProgressWatch* in turn creates a new *ProgressEvent* on basis of the *ProgressEvent* returned by the *Watchable* instance but sets the actual *AMRW* as event source.
- After that the *ProgressWatch* requests a *ProgressListener* using `getStatusCallback()`. By default the *AMRW* loads its corresponding *AAMM* instance and returns it
- The previously created *ProgressEvent* is then dispatched to the *AAMM.*
- If the status of the model execution is finished the *ProgressWatch* deregisters the corresponding *Watchable*.

4.1.3.2.2.3. Model Execution Status Event Processing

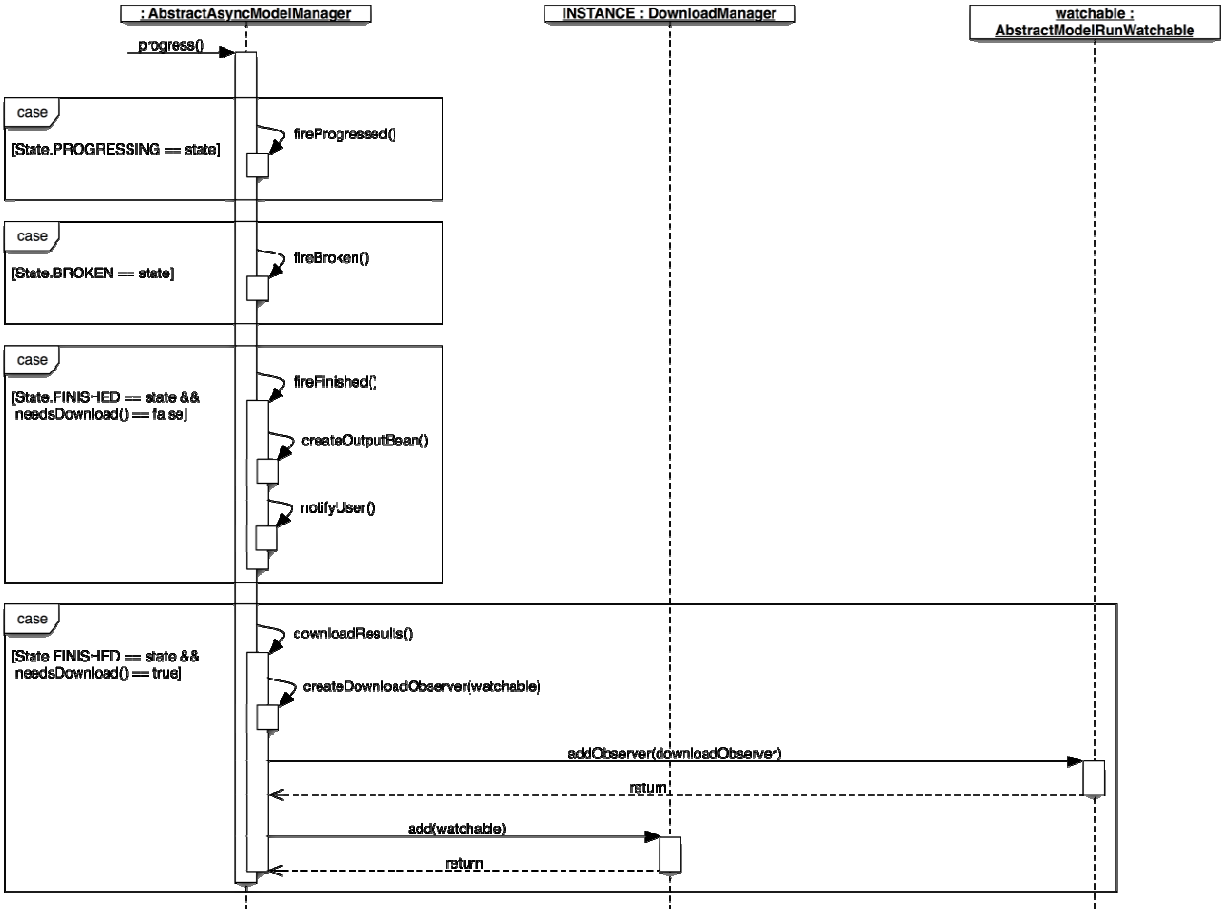

Figure 49: Model Execution Status Event Processing

- The status of a model execution is received by the specific *AAMM* instance through a call to the `progress()` operation.
- In case of a PROGRESSING state the `fireProgressed()` operation is used to propagate the model execution state.
- In case of a BROKEN state the `fireBroken()` operation is used to indicate that the model execution has stopped and will not be recovered.
- In case of a FINISHED state:
    - If the `needsDownload()` operation indicates that a download is not necessary the *AAMM* uses the `fireFinished()` operation to propagate that the model execution is successfully stopped. The `fireFinished()` operation automatically uses `createOutputBean()` to fetch the model results and notifies the user about current state.
    - If the `needsDownload()` operation indicates that the results must be fetched before the execution is of value for the initiator the *AAMM* initiates the download via the `downloadResults()` operation
        - The `createDownloadObserver()` operation is used to provided a means to watch over the download progress. The download observer is

initialised with the *Watchable* instance associated to the current model run and is additionally added to it as an actual *Observer* instance.

- Finally the *Watchable* instance is submitted to the *DownloadManager* via `add()`.

4.1.3.2.2.4. Model Result Download Processing

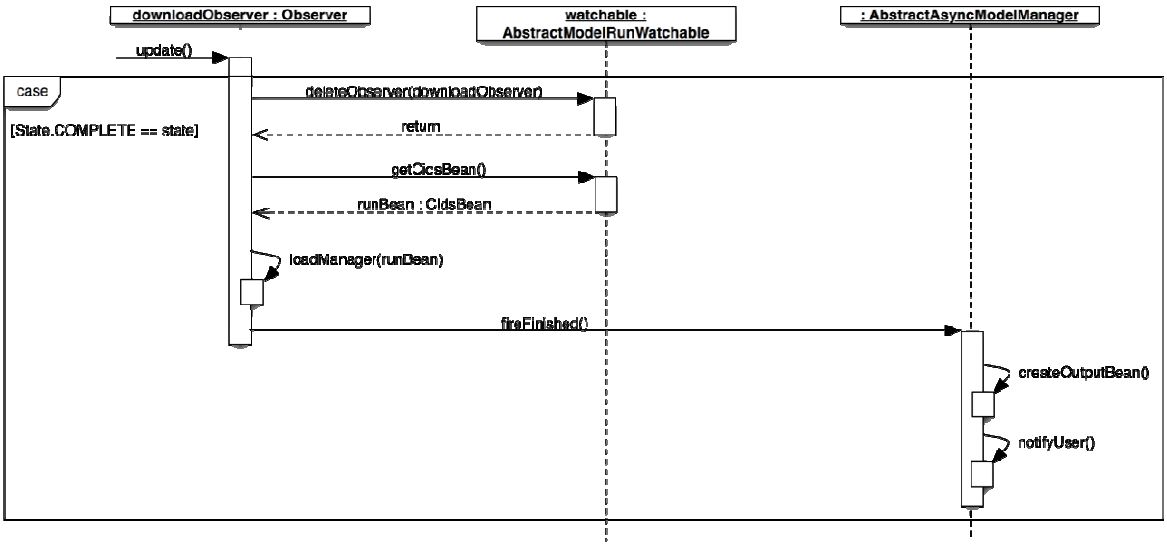

Figure 50: Model Result Download Processing

- If the `update()` operation of the download observer receives the state COMPLETE it detaches itself from its *AMRW* instance.
- After that it calls the `getCidsBean()` operation of the *AMRW* and tries to load the *AAMM* instance that is responsible for the particular model run and thus its download
- Finally a successful model execution is indicated by the `fireFinished()` operation.
- `fireFinished()` uses `createOutputBean()` to fetch the model results and ultimately the user is notified of the status of the particular model execution.

4.1.3.2.2.5. Model Monitoring Recovery on Startup

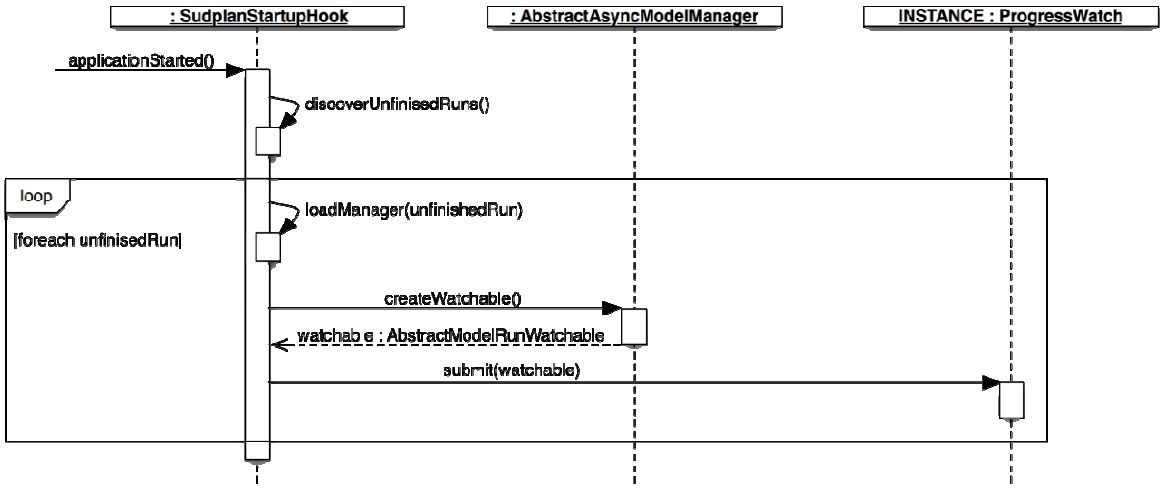

Figure 51: Model Monitoring Recovery on Startup

- On application startup `applicationStarted()` operation of the *SudplanStartupHook* is invoked.
- The *SudplanStartupHook* has facilities to discover unfinished model runs and receives a list of them.
- For each of the unfinished runs:
  - The *SudplanStartupHook* loads its corresponding *AAMM*
  - It uses the *AAMM*'s `createWatchable()` operation to get a *AMRW* instance for the specific run
  - The fetched *AMRW* is then submitted to the *ProgressWatch*.

The Framework is part of the current release of the SMS and thus is available for all Pilot Applications.

4.1.4 Third Year Development

Based on the second year developments the focus of year three's WP3 work was on:

- Integration of the Hydrology Common Service
- Enhancement of the Air Quality Common Service integration
- Visualisation and comparison features for gridded time series
- General GUI enhancements towards a stable product

The integration activities (Hydrology and Air Quality) are reported in D3.3.3 whereas the basis software development during the third year of the SUDPLAN project mainly focused on enhancements and usability improvements of already existing features, such as time series and IDF curve import and export for local and downscaled data, 3D component integration, Air Quality and Rainfall Common Services, local models (pilot specific) as well as general GUI enhancements towards a stable product. Moreover, the Hydrology Common Service integration was pushed forward on basis of enhanced map interaction features, Euler 2 Rain Event generation from IDF curves was introduced, a new 3D Component Integration Framework was created, Gridded Time Series Visualisation was developed and Gridded Time Series Comparison features were added. More detailed information about the new features is available in the following sections.

On basis of the work in V1 and V2 and the additional improvements in V3 the Scenario Management System can now be considered well prepared not only for the fast integration of the Hydrology Common Service but also the improvement of the various Pilot applications and does now constitute a stable application for all SUDPLAN use cases.

4.1.4.1 GUI Enhancement: Map interaction

In order to provide a smooth user interface the SMS must offer means to interact with the mapping component in an intuitive way. Hence the SMS was extended by the possibility to create context actions on the map with geo-spatial reference only. Moreover, as geo-spatially referenced objects play a major role in the SUDPLAN project the possibility to access WFS provided features is crucial. Thus a simple to use action is made available that takes care of background feature retrieval and the correct display on the map. That way, pilot applications and Common Services are easily able to implement their workflows, such as the "Show Catchment Area for this Location" workflow of the Hydrology Common Service (see. D3.3.3).

The UML in **Fel! Hittar inte referenskälla.** shows the related classes:

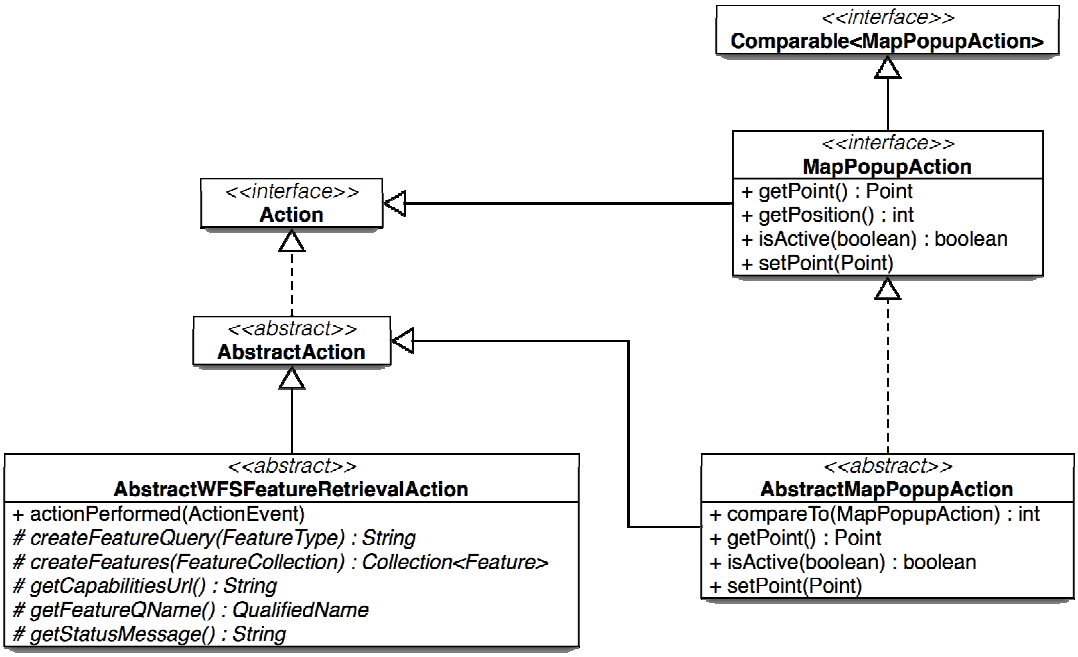

Figure 52: Map interaction enhancement: UML class diagram of additional actions

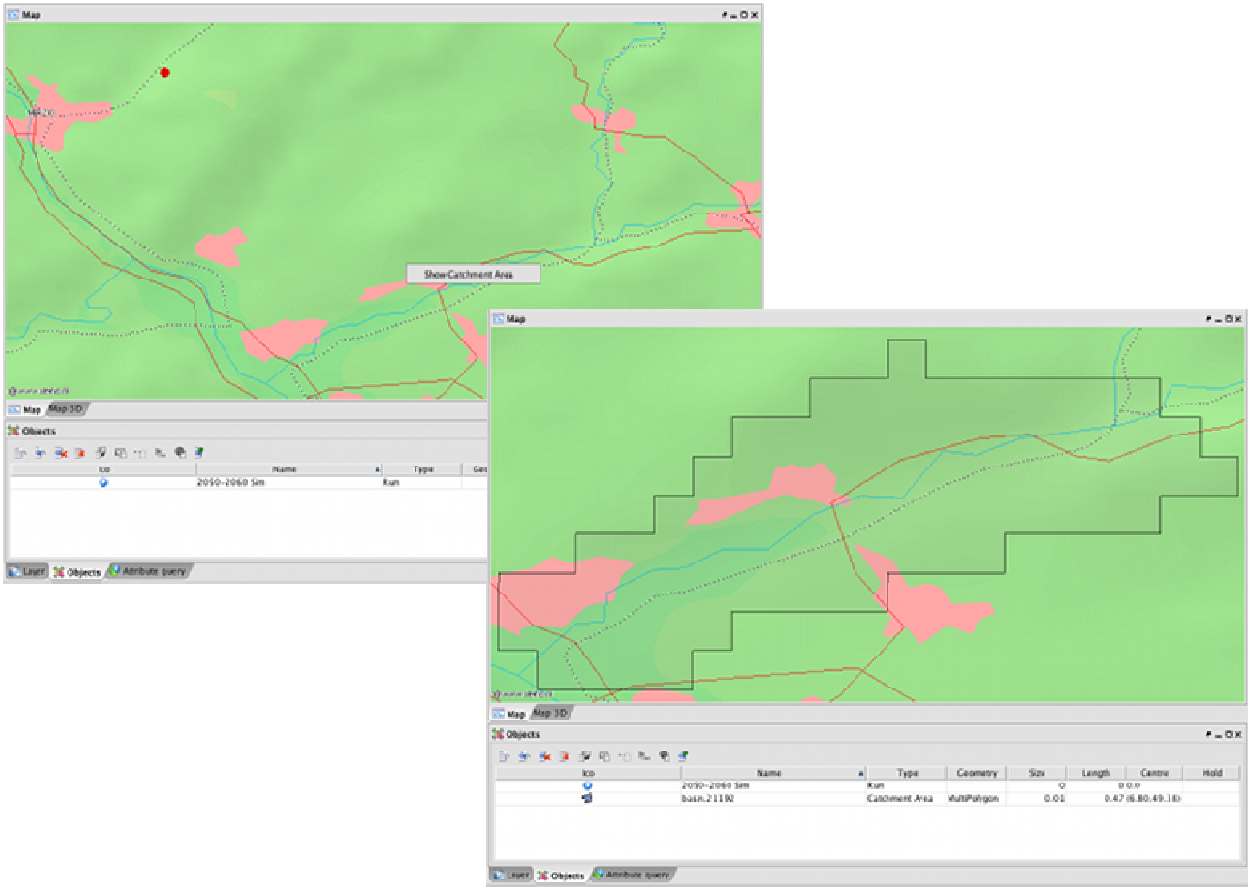

Figure 53: Map interaction example "Show Catchment Area"

4.1.4.2 Euler 2 Rain Event generation

To date SUDPLAN supports management and especially downscaling of rainfall time series as well as IDF curves. However, for certain use cases it is required to contemplate short term rain events. Moreover, they may be crucial input for other, so called, local models, e.g. the Runoff model used in the Wuppertal pilot (see. D6.2.x). For this reason SUDPLAN supports Euler 2 Rain Event generation from historical or downscaled IDF curves. This way, the Scenario Management System not only closes the gap between Rainfall Common Services and the WP6 Pilot but also provides added value for every SUDPLAN user.

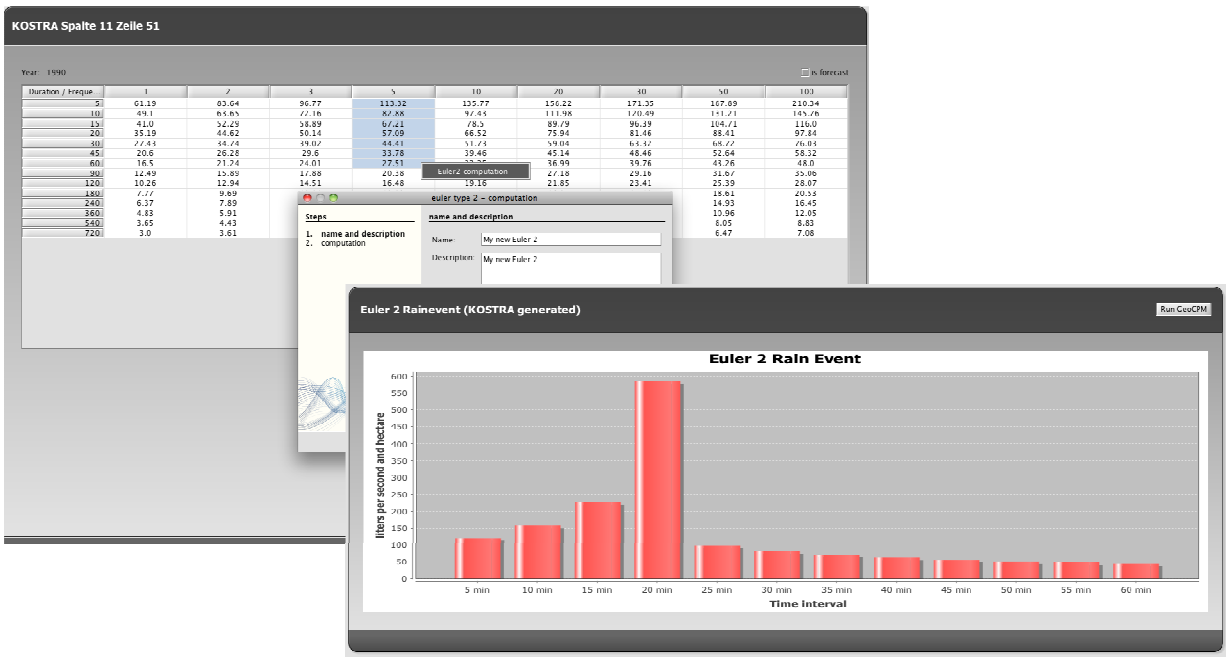

Figure 54: Euler 2 Rain Event from IDF curve

4.1.4.3 3D Component Integration Framework

The SUDPLAN SMS makes use of advanced visualisation techniques such as 3D visualisation. To ensure fast and easy integration in the SMS a small yet powerful framework has as been created to further abstract from the actual 3D component (see chapter **Fel! Hittar inte referenskälla. Fel! Hittar inte referenskälla.**) and consequently to support and speedup the integration cycle of the SUDPLAN 3D component. Figure 55: 3D Component Integration Framework UML class diagram

depicts the UML of the core classes of the framework.

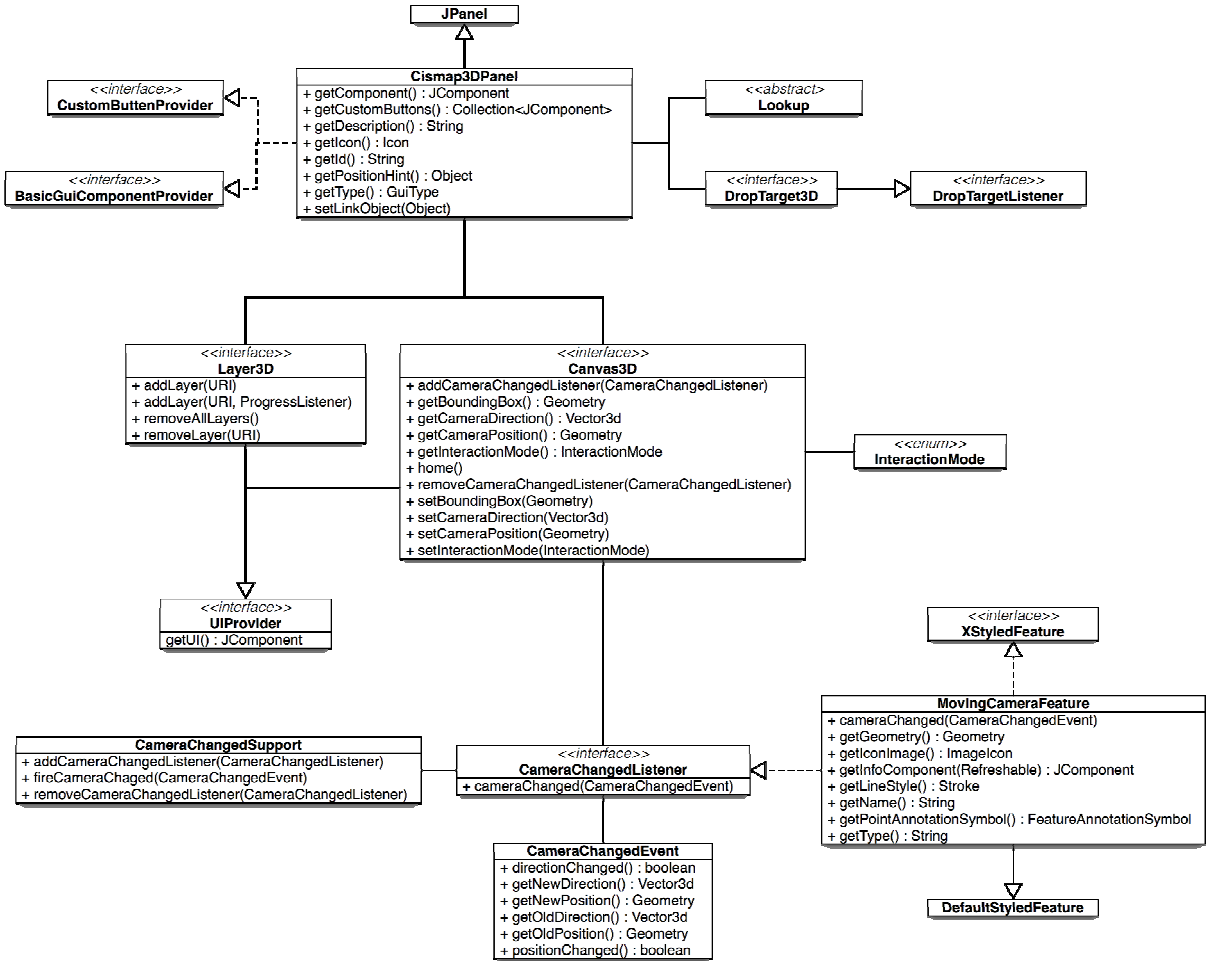

Figure 55: 3D Component Integration Framework UML class diagram

Further information about the responsibilities of the different classes is provided in the following listing. However, the focus is on the functionality regarding integration of 3D components in the SMS and may not necessarily explain the integration of the framework in the SMS itself.

- *Canvas3D*: The *Canvas3D* interface is the most important component for integration. It is necessary that an implementation is provided because it makes the main 3D user interface available to the SMS. Moreover, it provides means to control the viewport and also allows other components (such as the 2D mapping component of the SMS) to register delegates that inform about viewport changes. If the underlying 3D component support interaction mode changes the *Canvas3D* interface provides means to control them as well. Implementers shall put themselves in the global *Lookup*.
- *Layer3D*: The *Layer3D* interface serves as a basic layer control unit and thus allows for addition and removal of 3D layers identified via *URI*s. If the underlying 3D component offers a more complex control interface the SMS can reach it from here. Implementers shall put themselves in the global *Lookup*.
- *DropTarged3D*: A 3D component may indicate its capability to deal with Drop events during a Drag and Drop operation by implementing this interface. Implementers shall put themselves in the global *Lookup*.
- *UIProvider*: This is a simple interface that indicates that implementers are able to provide a user interface for their purposes. Both, the *Layer3D* and the *Canvas3D* can do so.

- *InteractionMode*: A simple enumeration of different interaction modes that can be used to control the 3D component if it supports it.
- *CameraChangedListener*: This interface must be used if a component is interested in camera changes of the 3D component.
- *CameraChangedEvent*: If the camera of the 3D component is changed a corresponding event is generated and propagated to all registered *CameraChangedListeners*.
- *CameraChangedSupport*: A simple yet helpful implementation for listener registration and event propagation. *Canvas3D* implementations may use this and thus do not have to care about proper event propagation themselves.
- *MovingCameraFeature*: The *MovingCameraFeature* is a special component that provides instantaneous reflection of 3D camera movement within the 2D context (that is the SMS internal 2D mapping component) if activated. To fulfil this task it implements some of the SMS internal feature operations.
- *Cismap3DPanel*: This is the core integration panel. It uses the global *Lookup* to find *Canvas3D*, *Layer3D* and *DropTarget3D* implementations. If provided it uses its own viewport to display the *Canvas3D* UI and if appropriate the *Layer3D* UI. Moreover it registers itself as a *DropTarget* in case of a *DropTarget3D* implementation is in the *Lookup* and delegates any Drop events to it. The *Cismap3DPanel* also registers itself in the SMS docking framework and can thus be freely positioned. Additionally it provides a toolbar with default actions such as 2D-3D syncing, a "Home" button and the *MovingCameraFeature* activation.

The following screenshot shows how a 3D component can be integrated in the SMS using the 3D component integration framework using the example of the SUDPLAN 3D component:

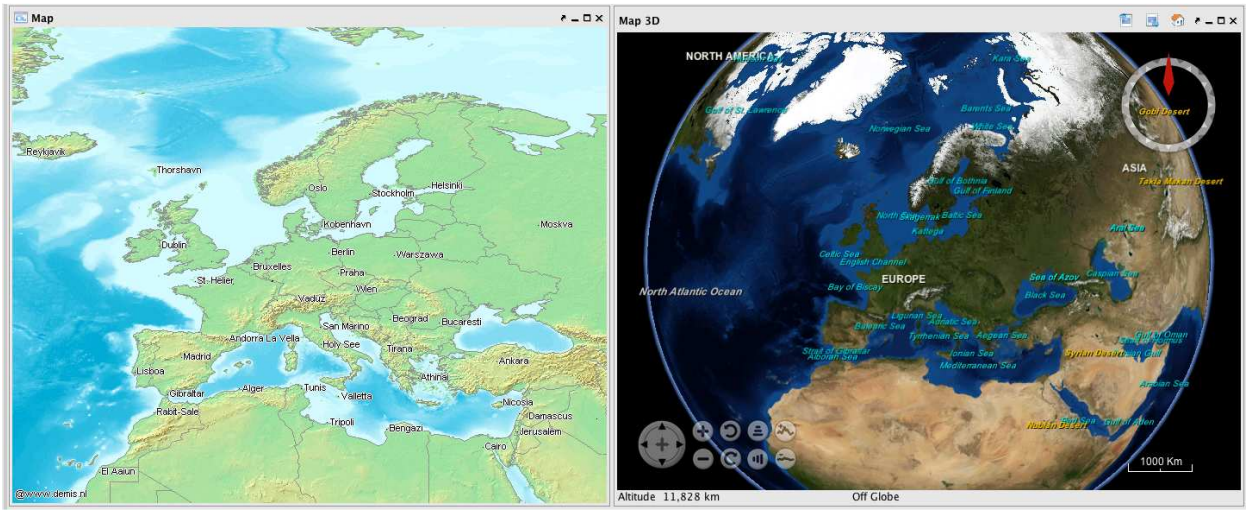

Figure 56: 3D Component Integration Framework in use: 2D mapping component alongside the 3D mapping component

4.1.4.4 Visualization of gridded time series

With the integration of the Air Quality Common Service the SUDPLAN user is able to generate a forecast on air pollutant concentrations until 2100. As a result of the air quality forecast, the user is given a spatially gridded time series containing concentrations of a specific air pollutant. For further processing these time series can be exported e.g. to a CSV file. In order to increase

the usability of SUDPLAN SMS and to improve the support of the decision making process, spatially gridded time series are visualized. As a spatially gridded time series provides a spatial and temporal location for its values, it is possible to visualize them in Map View components e.g. cismap that is uses as integral part of the SMS. In order to keep the visualization standard-compliant and thus allowing access for external users, it is realized by generating WMS layers.

4.1.4.4.1. Workflow

The visualization of gridded time series requires the user to already have appropriate data available e.g. by running a Common Service resulting in gridded time series. A gridded time series provided by a Common Service actually is accessible via an SOS offering. For example, the Air Quality Common Service calculates several time series, each of it representing a specific combination of an air pollutant and a temporal resolution of the time series. So the SUDPLAN SMS can use the TS-Toolbox to retrieve the actual Common Service results.

By using the custom renderer of a Common Service the user is able to select a gridded time series for visualization. After the user has chosen a gridded time series for visualization, its offering is retrieved from the SOS and written in a database, which is accessed by a WMS server to render an image using these values. So the WMS server creates a layer to visualize the values of a specific point in time of a time series. Since each layer visualizing a time series covers the same spatial area, the user has to be supported by the SUDPLAN SMS to bring those layers in the correct temporal order and to draw a conclusion from its development over time. With the help of the interactive `TimeseriesFeatureRenderer` introduced in Y1, this support already is available. Following figure depicts the described steps and the involved components in a sequence diagram:

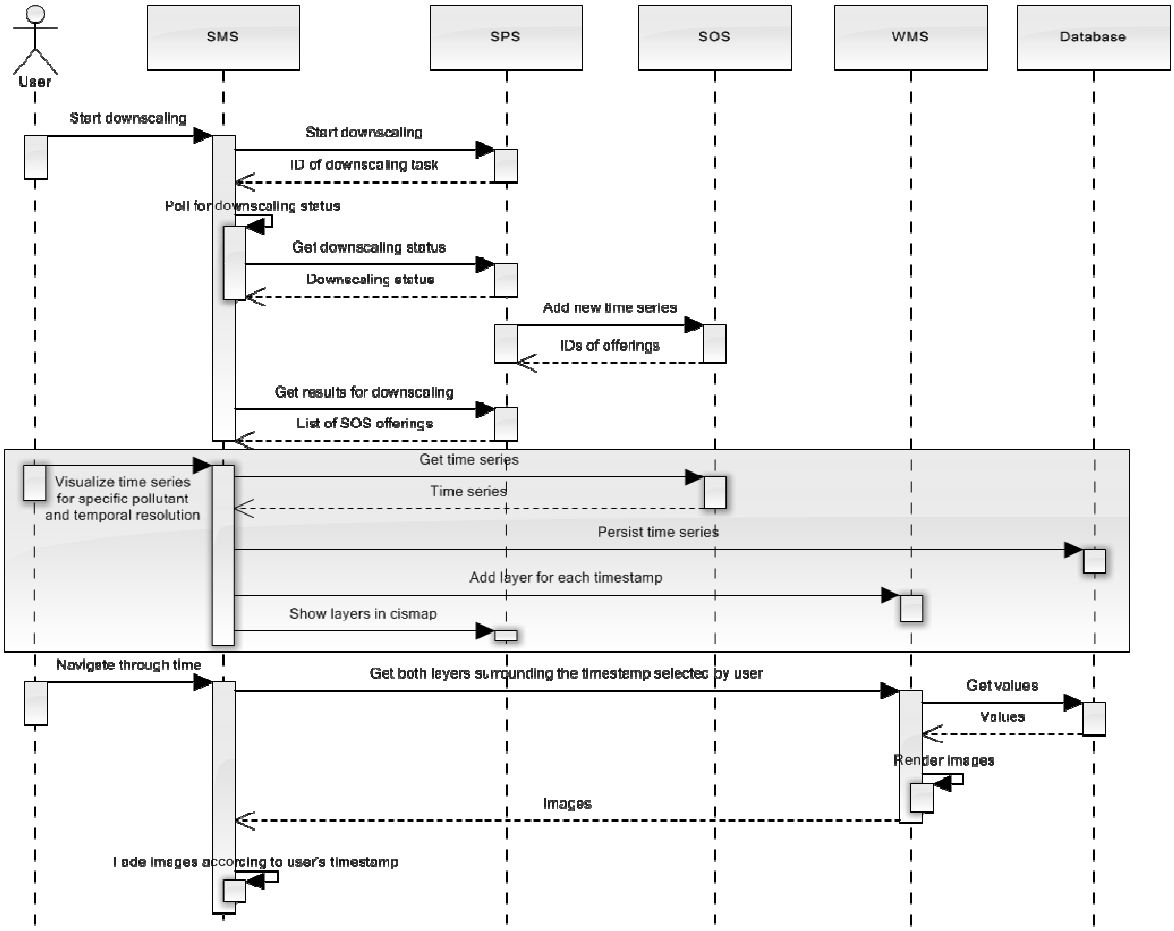

Figure 57: Visualizing Air Quality downscaling results

The first step in the above sequence diagram outlines the Air Quality Common Service Integration which was already realized in Y1/Y2. This integration in the SUDPLAN SMS is realized with the help of a customized `ModelInputRenderer`, a customized `ModelOutputRenderer` and corresponding user interfaces. The highlighted step shows the workflow to realize the result visualization. These will be discussed in the following. The last step shows what is necessary to allow the user to navigate through time in context of an Air Quality downscaling result. This step already is realized as well by the TimeseriesFeatureRenderer.

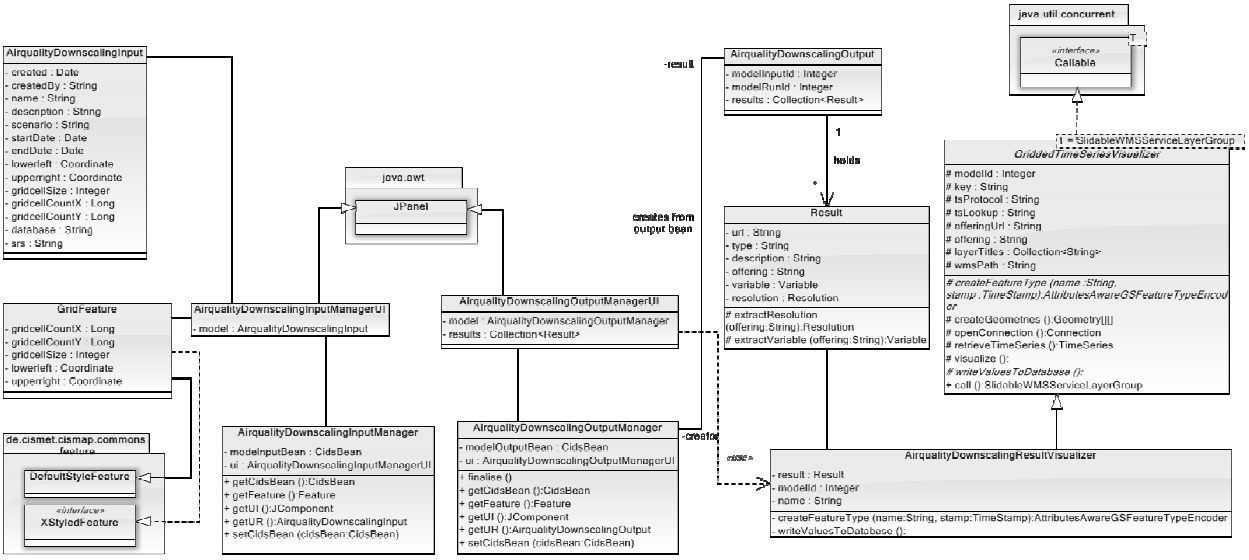

Figure 58: Classes responsible for Air Quality Common Service Integration

The classes `AirqualityDownscalingInput`, `AirqualityDownscaling-InputManager` and `AirqualityDownscalingInputManagerUI` manage and visualize user's input parameters once an Air Quality downscaling was defined. The class `GridFeature` realizes the visualization of the spatial grid. Handling and visualizing the result of an Air Quality downscaling is realized by the classes `AirqualityDownscalingOutput`, `AirqualityDownscalingOutputManager` and `AirqualityDownscaling-OutputManagerUI`. The result of an Air Quality downscaling is represented by an instance of `AirqualityDownscalingOutput`. The offerings which are provided by a result are encapsulated in objects of class `Result`. As soon as the user selected one specific result to visualize, `AirqualityDownscalingOutputManagerUI` hands over the corresponding `Result` object to an instance of `AirqualityDownscalingResultVisualizer`. This class is responsible for the visualization of the provided `Result` object, i. e. it retrieves the time series, persists it in the database and creates a corresponding layer in the WMS server. The retrieval of the time series is accomplished by the TS-Toolbox. `AirqualityDownscalingResultVisualizer` itself is derived from the abstract base class `GriddedTimeSeriesVisualizer`. It provides most of the functionalities to visualize gridded time series, i. e. it implements the time series retrieval, opens a connection to the database where the time series is to be persisted and calculates the spatial grid from the information provided by the time series. The figure also shows that `GriddedTimeSeriesVisualizer` implements the `Callable` interface. This is necessary since the time series visualization may take a considerable amount of time and therefore be processed in an own thread or task. After instantiation of a `GriddedTimeSeries-Visualizer`, the caller only needs to invoke `call()` in order to start the time series visualization. The return value of the `call()` method fits the requirements of the TimeseriesFeatureRenderer.

4.1.4.4.2. Persisting the result

Once the desired time series is retrieved, it has to be written into a database. The database has to be accessible by the WMS server which later renders these time series. Since a database and a WMS server are requirements for a SUDPLAN application, both can be used for the purpose of

gridded time series visualization and there is no need to additionally set up further components. The following figure depicts the structure of a Air Quality downscaling result set.

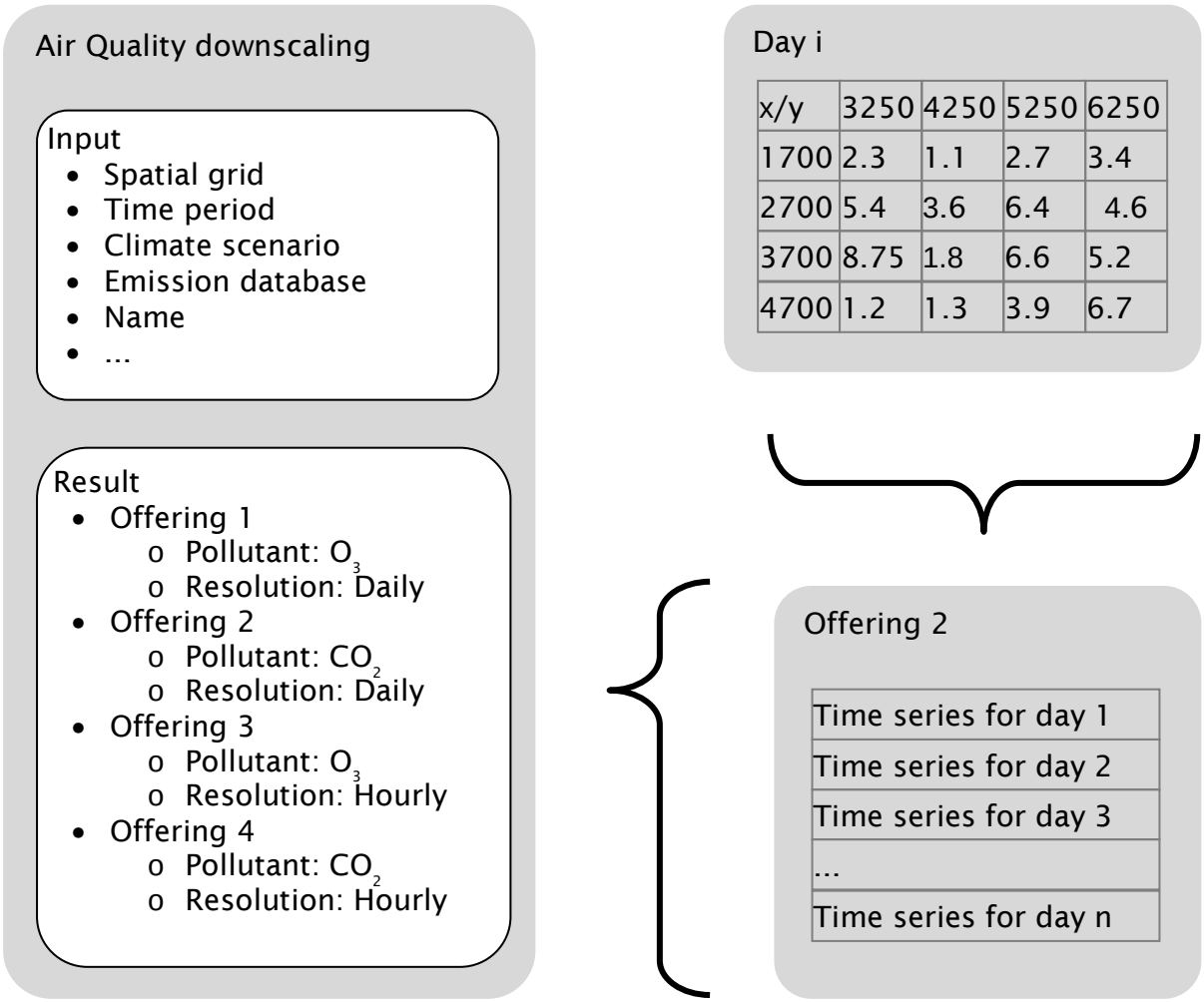

Figure 59: The structure of a Air Quality downscaling result offering

As mentioned earlier, the SPS generates an SOS offering for every combination of an air pollutant and a temporal resolution. Each SOS offering provides a spatially gridded time series for each timestamp. Every value of this grid is persisted - enhanced by its spatial location and some metadata like the air pollutant - as a single row by `AirqualityDownscalingResultVisualizer` via JDBC in the configured database. Additionally, a database view is created which selects all values of this specific grid.

Regarding the structure of a gridded time series offering, the database contains `n` new views, each view providing the values for one timestamp. In the next step, these views play an essential role to enable the WMS server to render the offering's grids.

4.1.4.4.3. Creating WMS layers

Most WMS implementations are able to manage layers which retrieve the necessary spatial information from a PostgreSQL/PostGIS database. This can be used to generate layers rendering images from the information in the corresponding view. Using the TimeseriesFeatureRenderer,

the SUDPLAN SMS is then able to let the user navigate through time for a gridded time series visualization.

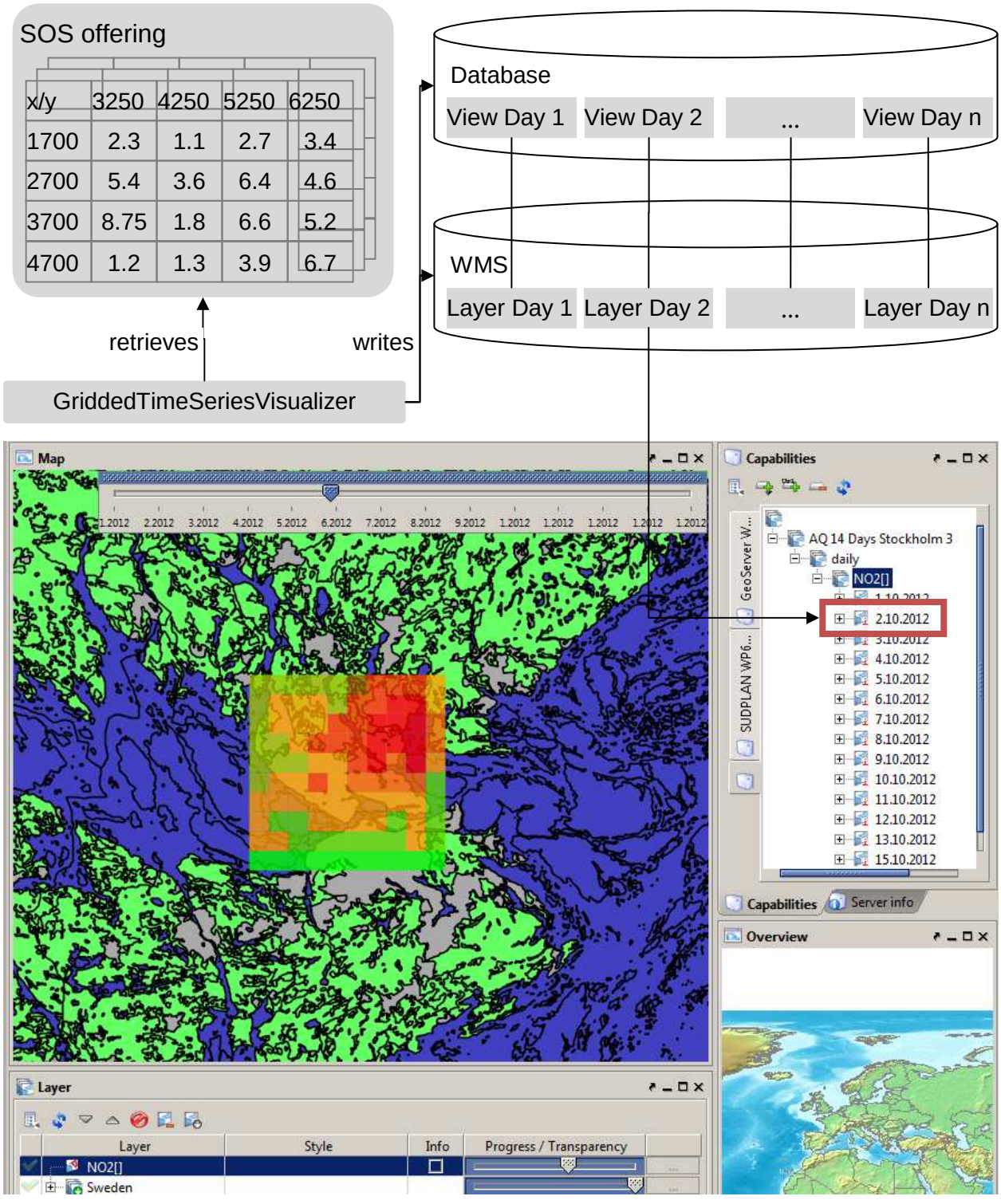

Figure 60: Components involved in gridded time series visualization

The above figure illustrates the steps necessary to visualize gridded time series to the user. An important prerequisite for the visualization is to ensure that the WMS server has read access to the database which is used for persisting the result. This has to be done manually. Since the database already provides views for each timestamp, `GriddedTimeSeriesVisualizer` now creates one layer for each timestamp. Each layer is configured to retrieve its information

from the corresponding view. All layers need to reside in the same layer group in order to allow the TimeseriesFeatureRenderer fade through the layers.

4.1.4.5 Comparison of gridded time series

To comprehensively support the decision making process of the SUDPLAN user, it is necessary to offer the user a visual comparison of gridded time series. With the help of this visualization it is e.g. possible to draw conclusions from simulating the same user scenario using different climate models. In the following, two considered approaches for comparing gridded time series are described, which one was realized and how it was realized.

4.1.4.5.1. Approaches

In contrast to a regular time series a gridded time series is enriched by an additional dimension, a spatial location. As the spatial location consist at least of two dimensions, a gridded time series is made up of three dimensions: every value is valid for a two-dimensional coordinate and a timestamp. Mainly two approaches were considered on the question how to visualize three-dimensional time series on a two-dimensional screen.

With the Advanced Visualisation Component the SUDPLAN SMS provides a component capable of visualizing 3D/4D data. Gridded time series, obtained by an SOS, can be visualized by 3D diagrams or 3D data vases.

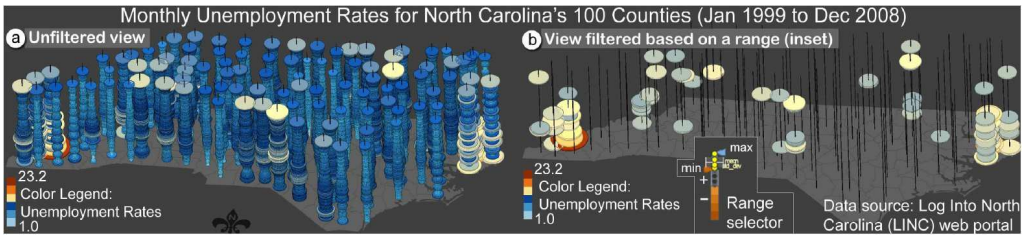

Figure 61: Example of 3D data vases

(Thakur, S., Rhyne, T.-M. (2009): Data Vases: 2D and 3D Plots for Visualizing Multiple Time Series. In Advances in Visual Computing (pp. 929-938). Springer Berlin / Heidelberg)

Although this approach relates the values of a gridded time series with their spatial aspect, it has two major drawbacks. First, it is not compliant to standards used in SUDPLAN and second this approach has to be realized from scratch.

The other approach is to use image manipulation algorithms. Using the result visualization of gridded time series the SUDPLAN user is able to create WMS layers which make the gridded time series available as a series of images. This means every value is converted into a colour, which is just another interpretation for this value from a computational point of view. The comparison of time series is accomplished by applying a formula, e. g. in a simple case by subtracting both values to see the difference. Since image manipulation is the same for colours, it can be used to compare already visualized gridded time series. In order to navigate through the third dimension, time, a similar approach as it is used in the gridded time series visualisation should be used for a consistent user experience.

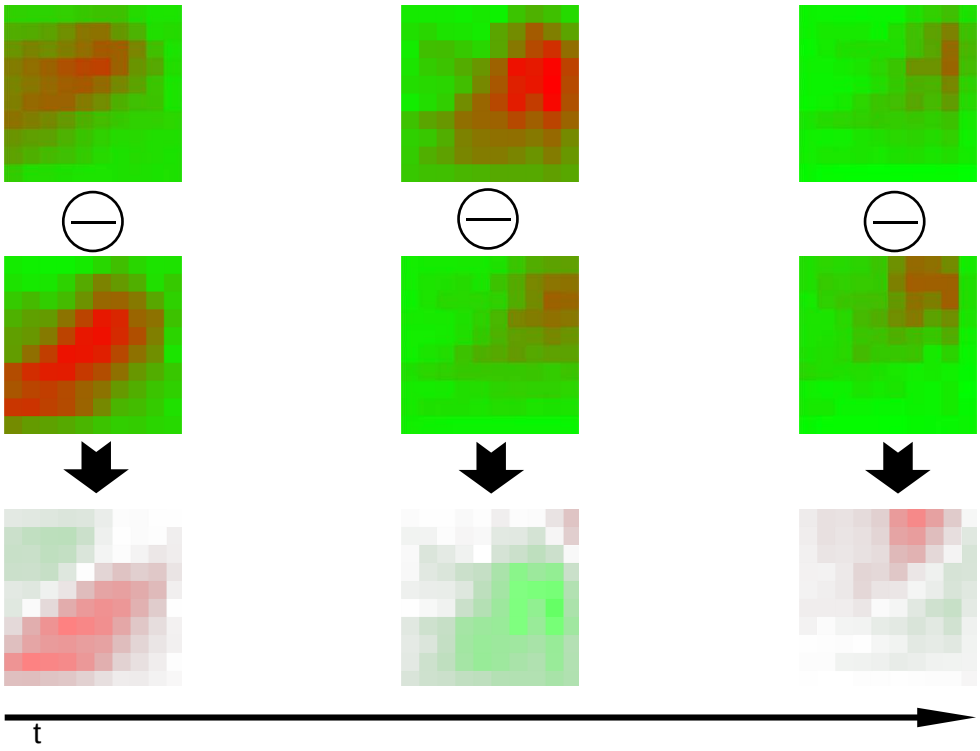

Figure 62: Calculating the difference (last row) of two gridded time series (first and second row)

This approach doesn't require as much development effort as the first approach. Additionally it is compliant to the OGC WMS standard as it could be enhanced to generate a WMS layer out of the comparison result. A major drawback of this approach is that humans can't tell the difference between two similar 32bit colours. Therefore the colour space has to be reduced which results in an aggregation of the values of the time series.

In order to realize the comparison of gridded time series in time, the second approach was chosen. Its only drawback seems to be the reduction of the colour space. But as the grid comparison is used to support a decision making process it isn't necessary to exactly reproduce the values of the gridded time series, which wouldn't be possible with 3D data vases as well.

4.1.4.5.2. Workflow

Since image manipulation is used to compare gridded time series, it's obvious that the SUDPLAN user has to visualize a gridded time series before it can be used for comparison. Thus the user has to fulfil the steps outlined in 4.1.4.4, Visualization of gridded time series, first. As a result the WMS server provides layers for the gridded time series which can be compared. The user now can add those WMS layers to cismap and then select the WMS layers to compare along with a comparison method. The result will be visualized in cismap over the WMS layers along with a control to enable the user to navigate through result's timestamps.

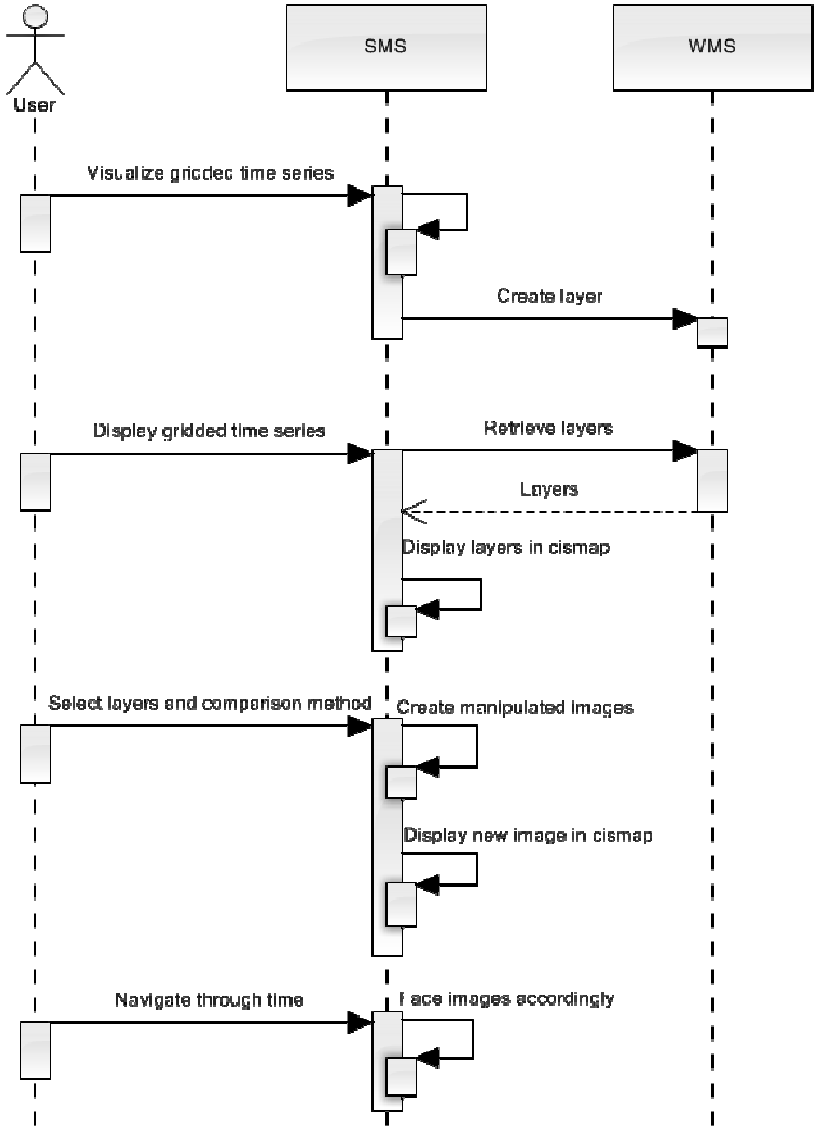

Figure 63: Comparing gridded time series

The first step in Figure 63 illustrates that the user has to visualize gridded time series before a comparison is possible. Since this is already explained in detail in 4.1.4.4, it is shown in a short form here.

In the next step the user adds those WMS layers, which he later wants to compare, to the map view in cismap. This feature already is supported by the SUDPLAN SMS. But forcing the user to do so seems to be needless as the SMS already lists all available WMS layers. So the user could select the WMS layers which are to be compared from this list. But this would break the usual handling of cismap and its components. Additionally, by adding the WMS layers to cismap first, the images of the gridded time series already are loaded from the WMS server. This allows a smooth visualisation, as the images haven't to be loaded in a later step.

By using a new UI component, the user is able to select two WMS layers for comparison and a comparison method in the third step. This component carries out the image manipulation in memory and displays the result in cismap.

Finally, similar to the visualization of gridded time series, the user can now navigate through time in the comparison result. Since this result resides in the memory and isn't available

as a WMS layer, its time navigating UI component differs from the time navigating component of WMS layers. It is included in the UI component introduced in the third step.

4.1.4.5.3. Selecting gridded time series for comparison

As mentioned before, the user has to add WMS layers of gridded time series to cismap in order to be able to compare them. In 4.1.4.2.5 it is described how to define cismap actions. For a consistent user interface such an action is implemented to let the user select the gridded time series for comparison.

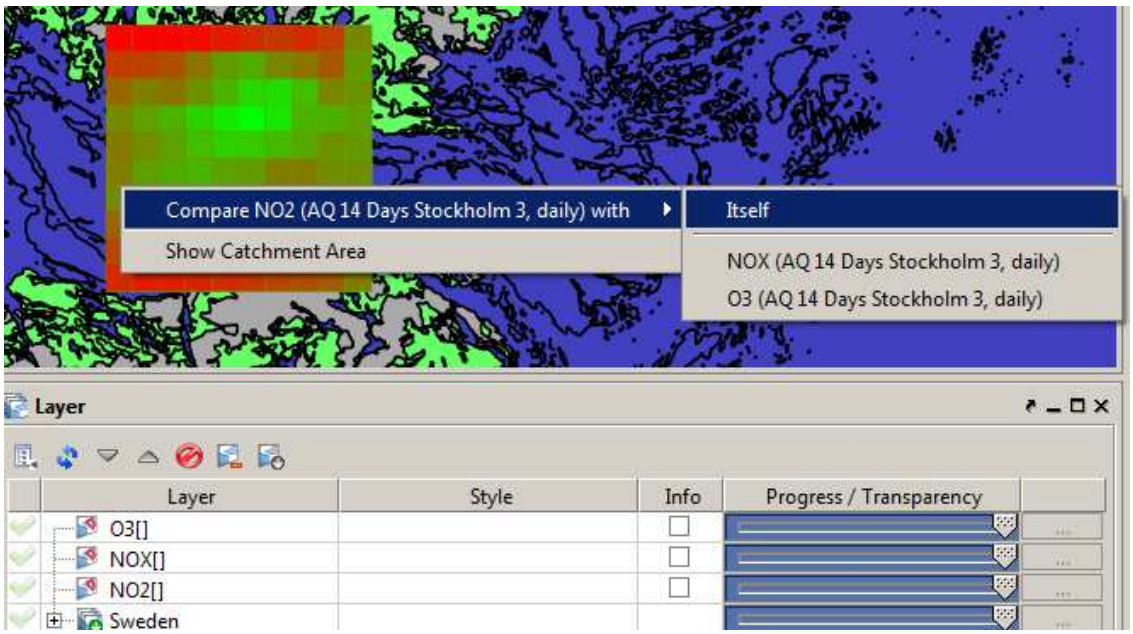

Figure 64: Action allowing selection of gridded time series for comparison

Another way to select the gridded time series for comparison is provided by the UI component which allows the comparison of gridded time series. This component will be explained in detail in the next chapter.

4.1.4.5.4. Comparing gridded time series

In order to compare gridded time series, the user has to determine which gridded time series are to compare and which comparison method should be applied. For this purpose, the following UI component can be used.

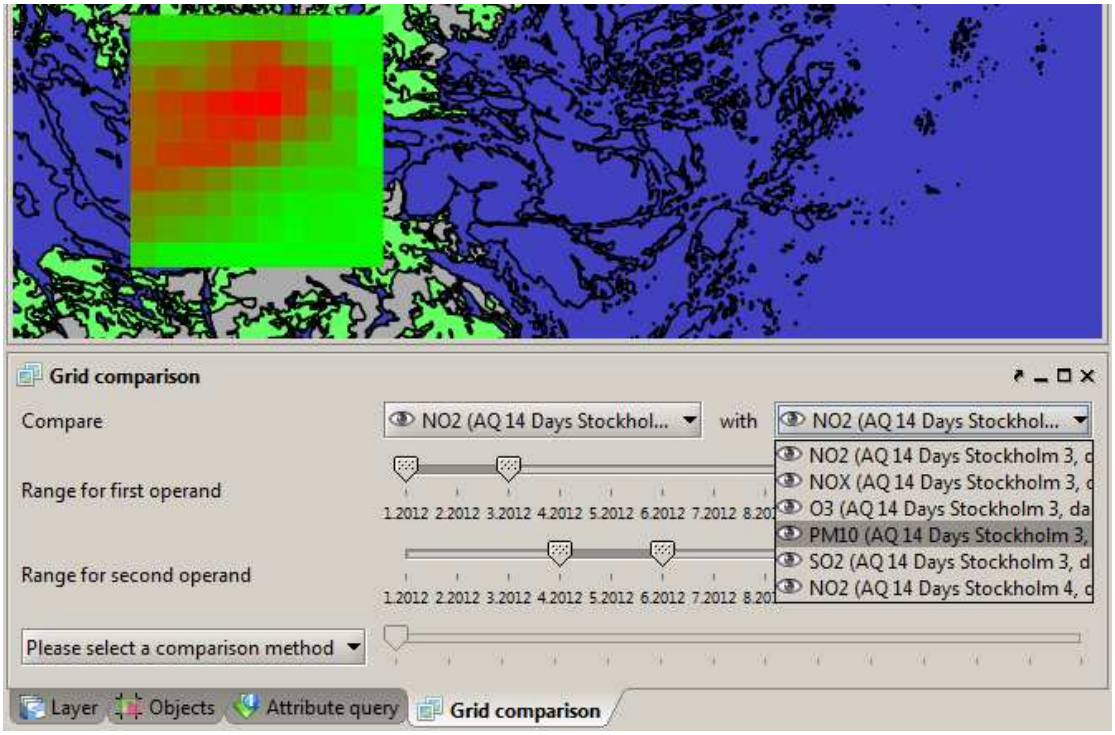

Figure 65: Grid comparison component

The grid comparison component is split into four rows. The first row provides two drop down lists allowing the selection of the gridded time series to compare. These lists contain all gridded time series added to cismap. After selecting a gridded time series the following both rows are adapted to the selection. These rows show the time period of the chosen time series similar to the gridded time series visualization in Figure 65. Using the range sliders the user can restrict the time period which is to be compared in both gridded time series. This allows the comparison of two parts of the same gridded time series, e. g. comparing the first and the second half of a year in the same time series. The fourth row provides a drop down list of comparison methods. After selecting a comparison method, the resulting images will be drawn over the already displayed WMS layers in cismap. With the help of the slider besides this drop down list, the user is able to navigate through time in the result.

4.1.4.5.5. Realization

The following figure depicts an UML class diagram which shows the classes responsible for the comparison of gridded time series.

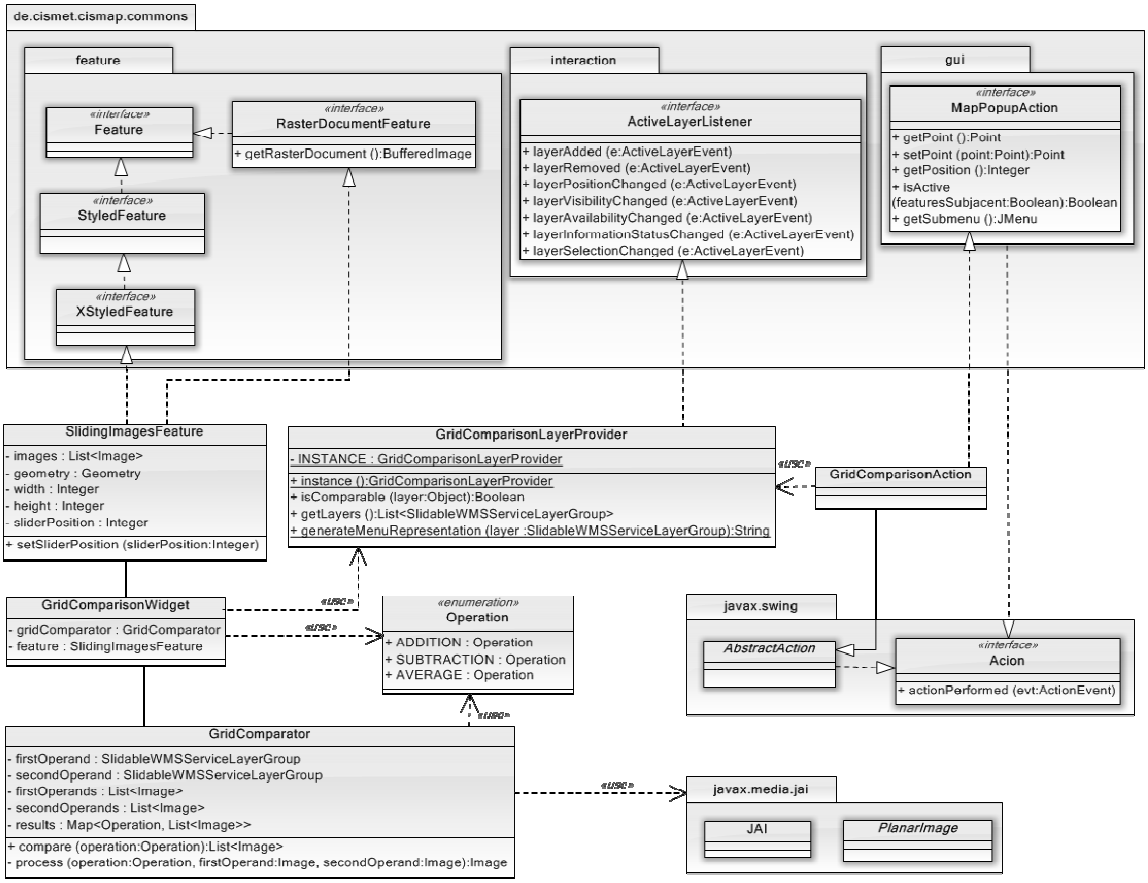

Figure 66: Classes responsible for comparison of gridded time series

As mentioned before, each gridded time series is represented by a WMS layer acting as a parent for one WMS layer for each timestamp in the time series. The cids internal class `SlidableWMSServiceGroupLayer` represents such a WMS layer. The user starts his workflow by adding gridded time series to cismap. As being a registered `ActiveLayerListener`, the singleton `GridComparisonLayerProvider` is informed about every addition and removal of layers to cismap. With the help of the method `isComparable(Object)`, this singleton filters all the layers added or removed to cismap for layers of type `SlidableWMSServiceLayerGroup` with a special marker indicating that this layer can be compared with another layer. This list of comparable layers is then used by `GridComparisonAction` and `GridComparisonWidget`. The `GridComparisonAction` uses the internal lookup mechanism of cids to register itself as `MapPopupAction`. Thus it is displayed in the context menu of cismap and provides a submenu listing all available, comparable layers. `GridComparisonWidget` is a subclass of `JPanel` and realizes the grid comparison component described in 4.1.4.5.4. As soon as the user decided which gridded time series to compare and selected a comparison method (represented by enumeration `Operation`), the `GridComparisonWidget` creates a corresponding `GridComparator`. The `GridComparator` uses JAI (Java Advanced Imaging) to perform the image manipulation and caches the resulting images in a `Map`. The results of `GridComparator` is then used by `GridComparisonWidget` to create a `SlidingImagesFeature`. This feature extends `RasterDocumentFeature` and thus is used by cismap to display a `BufferedImage` at a spatial location determined by `SlidingImagesFeature`'s `geometry`. Additionally, `SlidingImagesFeature` offers

its user, i. e. `GridComparisonWidget`, the method `setSliderPosition(Integer)` to enable the navigation through time. The given `Integer` is a indicator of the slider's position. By accordingly fading the images which correspond to this position, the user is able to see the progression of the comparison result.

4.1.5 Current Release Feature Summary

The current release of the SMS Framework supports a number of vital features that are used to access, visualise and compare Common Service as well as local models results. In summary, the features included in the third year release of the software are:

- Enhanced 3D component integration
- Enhanced local and downscaled data (time series, IDF curves as well as custom pilot data) import and export facilities
- Support for Hydrology Common Service
- Euler 2 Rain Event generation
- Enhanced model result visualisation (2D time series)
- Model Result (Scenario) Comparison (2D time series) including visual comparison support, time series operations and 2D map interaction
- Model management (information related to a model, i.e. meta information)
- Enhanced Asynchronous model execution (incl. model state persistence)
- Management of model runs, results and parameterization
- Basic model result visualization (1D time series, 2D maps)
- Model Result (Scenario) Comparison (1D time series) including visual comparison support, time series operations and 2D map interaction.
- Integration of temporal aspects of 2D map visualisations
- Search and discovery of objects of concern (e.g. results, scenarios)
- OGC SOS (TS-API based), SPS (TS-API based), WMS and WFS integration capabilities.
- Support for Rainfall (time series & IDF)
- Air Quality Common Services
- Pan-European Common Services (climate change parameters)
- Local model integration support
- Integration facilities for local data (time series)

The current version of SMS related software (*cids SUDPLAN extensions*) can be found at http://sudplanwp3.cismet.de/sms/.

4.2. Model as a Service Integration

The main objective of this part of the SMS is to provide the means to control model implementations and access model results, including both SUDPLAN Common Services and local pilot specific models, via standardised web services. The selected standards are members of the OGC SWE [SWE, 2007] family, specifically SOS (Sensor Observation Service) and SPS (Sensor Planning Service), which are used for model result access and model control. This part of the SMS can be used to access the corresponding common service (*see D4.1.1 – Concerted Approach Report V1*) as well as to encapsulate local models as in some of the four pilot applications of SUDPLAN.

Within SUDPLAN we concentrated on the use of the OGC service interfaces. The main reasons were:

- The OGC service interfaces are an accepted standard in the GIS community, and cover a large number of use-cases relevant to environmental applications.
- All OGC specifications are freely available to everyone.
- All OGC service specifications share common data encodings and descriptions such as O&M – Observation and Measurement [OM1, 2007], GML - Geography Markup Language [GML, 2004], and SensorML - Sensor Model Language [SensorML, 2007].
- Implementations of libraries and some ready to use services are already available with open source.
- The DoW requires the use of an open approach in consideration of the technical requirements of SISE [SISE, 2009] (See D3.1.2, REQ-DoW-2 ff).

Additionally, the use of these established standards enables the integration of SUDPLAN services and the SMS with existing as well as emerging data and model services.

A precondition for the integration of data services is that all data need to be self-describing [ORCHESTRA, 2008] which means that there is information about the values, such as unit of measurement, description of phenomena, precision and uncertainty, and methods of measurement in the form of sensor descriptions. The same is true for models as we expect to access more models over time and they should be usable without prior knowledge of any details about them. There are many standards for data transport and remote service invocation. SANY SensorSA [SANY, 2009] and Fusion Architecture [SanyFMA, 2010] documents already anticipate most of this and were therefore used as important sources of requirements regarding the Model as a Service Integration Building Block of the SMS.

4.2.1 Components and APIs Used

The implementation of SOS and SPS related software is based on the Time Series Toolbox (TS-Toolbox) API from AIT. The TS-Toolbox API provides the means to conveniently deal with arbitrary time series.

We decided to use the TS-Toolbox because it represents a good starting point from which to implement dedicated services able to wrap the various existing models needed in SUDPLAN and to establish the basis for the integration of new local models through standardised service interfaces. In the SANY project, we found that the use of existing SOS and SPS implementations, e.g. from 52° North Initiative [SOS52N, 2011] is not feasible as they are complete systems, and they are not built to act just as an interface implementation to wrap existing models or data bases. Many other parts of the TS-ToolBox, especially on the client side, can also be used or adapted to suit SUDPLAN needs.

In the following a short introduction on the TS-Toolbox architecture and the related APIs is given. More information on the TS-Toolbox can be found at http://ts-toolbox.ait.ac.at.

TS-Toolbox is a high level programming framework that allows efficient access to and processing, archiving and presentation of semantically enriched time series. It consists of three layers (*Figure 67: Elements of the TimeSeries ToolBox*): the ”applications“ layer provides examples of complete TS-Toolbox applications with CLI, GUI and web-service interfaces; the ”components“ layer provides the functional building blocks for commonly used functions; and the ”TS-API“ layer provides the basic interfaces and methods for presentation and manipulation of semantically enriched time series.

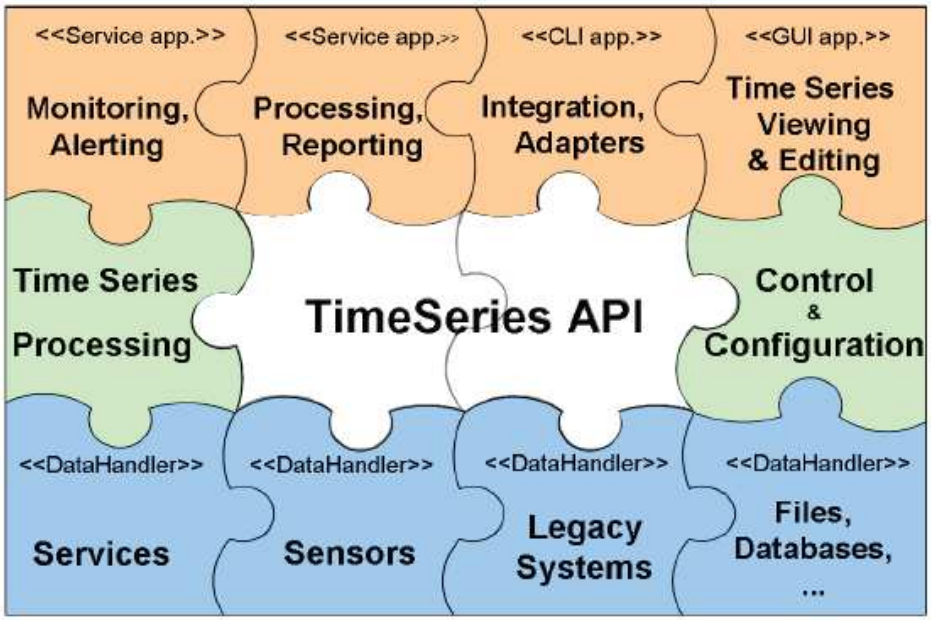

Figure 67: Elements of the TimeSeries ToolBox

The functionalities implemented by the TS Toolbox components provide application developers with higher level building blocks than typical general purpose libraries, and allow rapid development of full-fledged service, GUI, and CLI applications. Moreover, the TS-Toolbox based applications are highly modular, network-aware, and easily re-configurable, and the use of TS-API interfaces simplifies the task of extending and adapting the TS-Toolbox applications to new environments. The main advantages of the TS-API are:

- Data from different legacy systems, including local models and related environmental, geographical or background data, can be easily integrated, processed, visualized, and made available to a larger audience by means of standardized service interfaces.

- Its pipe-oriented architectural design greatly simplifies the task of component and data flow configuration within an application, e.g. using the output of a certain model (SUDPLAN Common Service) as the input to another model (pilot-specific local model).

- The components can be easily chained, e.g. in order to provide more sophisticated processing capabilities, and used in parallel (e.g. asynchronous model execution), in order to provide alternative means for accessing, storing and presenting the data.

- Components can be easily extended or replaced with alternatives, e.g. in order to access new types of data, integrate new models, or improve the processing or storage performance.

The TS-Toolbox "components" provide the functional building blocks for commonly used functions, such as:

- Sensor configuration and access to sensor data. For example, the AnySenDataHandler provides a means to access various sensors accessible over serial and network interfaces.

- Read- and/or write- access to time series stored in services, files and databases. For example, the SOSDataHandler provides access to standardised Sensor Observation Service, the CSVSimpleDataHandler to ASCII files and the GenericDBDataHandler to relational databases.

- Service interfaces. For example, the Remote DataHandler allows exchange of time series between two TSToolbox applications over the network.

- Processing and annotation of the time series. In particular, the F3 Processor component provides a generic functional language for manipulation of the time series.

- Control components provide a convenient way to dynamically control the data flow and the behaviour of the DataHandler and Processing components.

The TS Toolbox is designed with extensibility in mind, and more components, such as model integration and management components developed in SUDPLAN, can be added in the future.

4.2.2 First Year Development

In the interest of model and service integration into the SUDPLAN SMS three tasks has been performed:

- Definition of a data and meta-information encoding conformant to OGC standards
- Integration of a client part into the SMS
- Development of a generic service providing OGC SOS and SPS interfaces to encapsulate models

The generalised course of action of a model invocation is shown in *Figure 68: Invocation of a Model Encapsulated as a* Service.

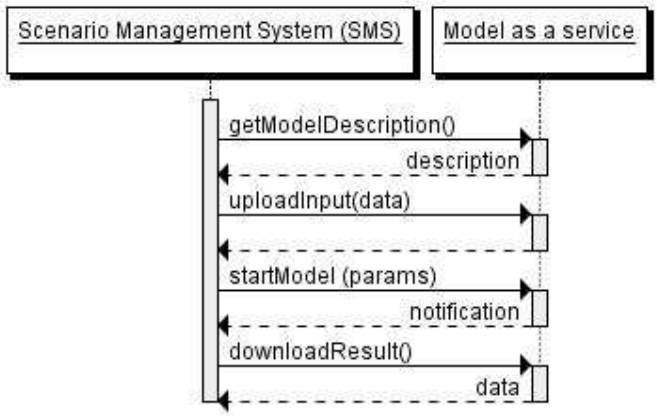

Figure 68: Invocation of a Model Encapsulated as a Service

4.2.2.1 O&M Encoding of Time Series of Fields

Based on the size of the data, a distinction is made between parameters for a model run and input data. While the parameters are typically of simple type and small in size (except for calibration data, for example), the input data for a model are complex in nature and quite commonly of several GB in size. Because the model needs fast access to the input data these data have to be uploaded to the model site. The user of the model does this (sometimes over institutional boundaries) through an implementation of the SOS interface. This interface is also used to enumerate and retrieve the model results. Model input and result data transferred through the SOS interface are encoded using the O&M (OGC Observation and Measurement) [OM1, 2007] information model.

The O&M encoding is straight forward for time series of scalar values and used by many SOS implementations such as 52° North [SOS52N, 2011]. The encoding of time series of coverages is not that well defined. Observations and Measurements - Sampling Features (O&M-SF) [OM2, 2007] specification provides a means to encode discrete coverages and provides dedicated elements to do so, but it lacks dedicated elements necessary to describe continuous coverages.

The SANY 'Fusion and Modelling Architectural Design' [SanyFMA, 2010] document describes two methods for encoding coverages in O&M. Both use the O&M-SF sa:SamplingSurface element. While there is no dedicated element in O&M-SF v1.0 to specify a grid on which the sampling takes place, both methods describe the sampling points and grid by embedding its description in sub-elements of the sa:SamplingSurface, which was not designed for this purpose.

In SUDPLAN an additional method has been considered for describing continuous coverages. The optional SOS method DescribeFeatureOfInterest allows for retrieval of the feature type description (xml schema) for a given feature of interest. This allows us to introduce a new namespace containing additional Sampling Feature types.

The new SamplingGrid type inherits from the sa:SpatiallyExtensiveSamplingFeatureType defined in O&M-SF and contains a gml:RectifiedGrid element that describes the rectified grid on

which the sampling takes place. The advantage of the mentioned inheritance is that the sampledFeature relation is retained. SUDPLAN SamplingGrid schema and one Sampling-Grid instance example are shown below.

SamplingGrid schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xlink="http://www.w3.org/1999/xlink"
        xmlns:gml="http://www.opengis.net/gml"
        xmlns:sa="http://www.opengis.net/sampling/1.0"
        xmlns:aitsa="http://www.ait.ac.at/sampling"
        targetNamespace="http://www.ait.ac.at/sampling"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified">
  <import namespace="http://www.opengis.net/gml" schemaLocation="./gml4sos.xsd"/>
  <import namespace="http://www.opengis.net/sampling/1.0"
          schemaLocation="http://schemas.opengis.net/sampling/1.0.0/ samplingManifold.xsd"/>
  <element name="SamplingGrid" type="aitsa:SamplingGridType" substitutionGroup="gml:_Feature"/>
  <complexType name="SamplingGridType">
    <complexContent>
      <extension base="sa:SpatiallyExtensiveSamplingFeatureType">
        <sequence>
          <element ref="gml:RectifiedGrid"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</schema>
```

SamplingGrid instance example:

```
<?xml version="1.0" encoding="UTF-8"?>
<SamplingGrid xmlns="http://www.ait.ac.at/sampling"
        xmlns:gml="http://www.opengis.net/gml"
        xmlns:xlink="http://www.w3.org/1999/xlink"
        xmlns:sa="http://www.opengis.net/sampling/1.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.opengis.net/gml http://schemas.opengis.net/gml/3.1.1/base/gml.xsd
                            http://www.ait.ac.at/sampling ./SamplingGridSchema.xsd ">
  <gml:description>Sweden grid</gml:description>
  <gml:name>Sweden Grid</gml:name>
  <gml:location />
  <sa:sampledFeature xlink:href="urn:MyOrg:feature:sthlm_1"/>
  <gml:RectifiedGrid dimension="2">
    <gml:limits>
      <gml:GridEnvelope>
        <gml:low>0 0</gml:low>
        <gml:high>50 60</gml:high>
      </gml:GridEnvelope>
    </gml:limits>
    <gml:axisName>x</gml:axisName>
    <gml:axisName>y</gml:axisName>
    <gml:origin>
      <gml:Point srsName="urn:x-ogc:def:crs:EPSG:3021">
        <gml:pos>6546000.0 1580000.0</gml:pos>
      </gml:Point>
    </gml:origin>
    <gml:offsetVector srsName="urn:x-ogc:def:crs:EPSG:3021">0 2e-005</gml:offsetVector>
```

```
      <gml:offsetVector srsName="urn:x-ogc:def:crs:EPSG:3021">2e-005 0</gml:offsetVector>
   </gml:RectifiedGrid>
</SamplingGrid>
```

4.2.2.2 Using UncertML to Describe Statistical Data

The descriptive model language Uncertainty Markup Language – UncertML [UncertML, 2007] developed by the INTAMAP project can be used to encode the accuracy of the observation collection. The SUDPLAN rainfall downscaling service generates a 2D table of the predicted precipitation values for the total seasonal accumulation (TOT), maximum 30-min intensity (MAX), and frequency of occurrence (FRQ).

Because of the statistical characteristics of these data, UncertML could be used to encode them, similar to the way uncertainties in sensor data and model outputs were encoded in the SANY project. This would mean treating these data as descriptions of the time series of rainfall, not as an independent result. At the same time these data have the characteristics of a time series, meaning that they provide the above mentioned statistical information for every season. At this point no decision has been taken about whether these data should be encoded in SUDPLAN as observations (using O&M), or as observation uncertainties (using UncertML).

4.2.2.3 Using SensorML to Describe Models and Required Parameters

The SOS and SPS interfaces provide process descriptions encoded in SensorML through the describeSensor operation. In SUDPLAN, the process is a model and can be described as a non-physical (pure) process. The information provided in the form of a SensorML document can be quite extensive, encompassing model inputs, parameters, outputs, the model algorithm itself, and details of the implementation module. Currently, our applications require descriptions of constant model parameters necessary for the human interpretation of the model results. This includes model identification, responsible party, input and output model. Our processing services act as clients to several Sensor Observation Services. The following simplified SensorML document shows basic model identification as well as the inputs and outputs of the rain downscaling model used in SUDPLAN. The model takes as input a SOS offering name containing the historical rain measurements as well as a future timestamp around which the downscaling results are generated. The model generates a timeseries of coverages with the resulting data described in the element “Downscaled rain”.

```
<?xml version="1.0" encoding="UTF-8"?>
<sml:SensorML xmlns:sml="http://www.opengis.net/sensorML/1.0.1"
xmlns:swe="http://www.opengis.net/swe/1.0.1"
xmlns:gml="http://www.opengis.net/gml"
xmlns:xlink="http://www.w3.org/1999/xlink" version="1.0.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sensorML/1.0.1
                    http://schemas.opengis.net/sensorML/1.0.1/sensorML.xsd ">
   <sml:member>
      <sml:System gml:id="SUDPLAN_A1B3">
         <gml:description>Simple rain downscaling model</gml:description>
         <sml:identification>
            <sml:IdentifierList>
               <sml:identifier name="UID">
                  <sml:Term definition="urn:x-ogc:def:identifier:OGC:uuid">
                     <sml:value>urn:x-ogc:object:model:SUDPLAN:prec:A1B3</sml:value>
                  </sml:Term>
```

```
            </sml:identifier>
            <sml:identifier>
               <sml:Term definition="urn:x-ogc:def:identifier:OGC:shortName">
                  <sml:value>SUDPLAN A1B3</sml:value>
               </sml:Term>
            </sml:identifier>
         </sml:IdentifierList>
      </sml:identification>
      <sml:inputs>
         <sml:InputList>
            <sml:input name="ObservationOfferingName">
               <swe:Text />
            </sml:input>
            <sml:input name="centerTime">
               <swe:Time />
            </sml:input>
         </sml:InputList>
      </sml:inputs>
      <sml:outputs>
         <sml:OutputList>
            <sml:output name="Downscaled_rain">
               <swe:DataArray>
                  <swe:elementCount>
                     <swe:Count />
                  </swe:elementCount>
                  <swe:elementType name="CoverageType">
                     <swe:DataRecord>
                        <swe:field name="Timestamp">
                           <swe:Time definition="urn:ogc:data:time:iso8601"/>
                        </swe:field>
                        <swe:field name="Grid">
                           <swe:DataArray>
                              <swe:elementCount>
                                 <swe:Count />
                              </swe:elementCount>
                              <swe:elementType name="value">
                                 <swe:Quantity definition="urn:ogc:def:property:OGC:1.0:precipitation">
                                    <swe:uom code="mm"/>
                                 </swe:Quantity>
                              </swe:elementType>
                           </swe:DataArray>
                        </swe:field>
                     </swe:DataRecord>
                  </swe:elementType>
               </swe:DataArray>
            </sml:output>
         </sml:OutputList>
      </sml:outputs>
   </sml:System>
</sml:member>
</sml:SensorML>
```

Based on these encodings a client part (called a DataHandler in the TimeSeries ToolBox context) was implemented and integrated into the SUDPLAN SMS.

After implementing a generic server providing SOS as well as SPS functionality the integration of the Common Service started with

- Climate scenario data

- Rainfall downscaling for rain time series
- Air Quality downscaling

In general all Common Services models require the upload of input data (handled by an SOS interface), the invocation of the actual model run (handled by an SPS interface) and the download of results (again handled by an SOS interface). This is shown in more detail in *Figure 69: Model Invocation* Details.

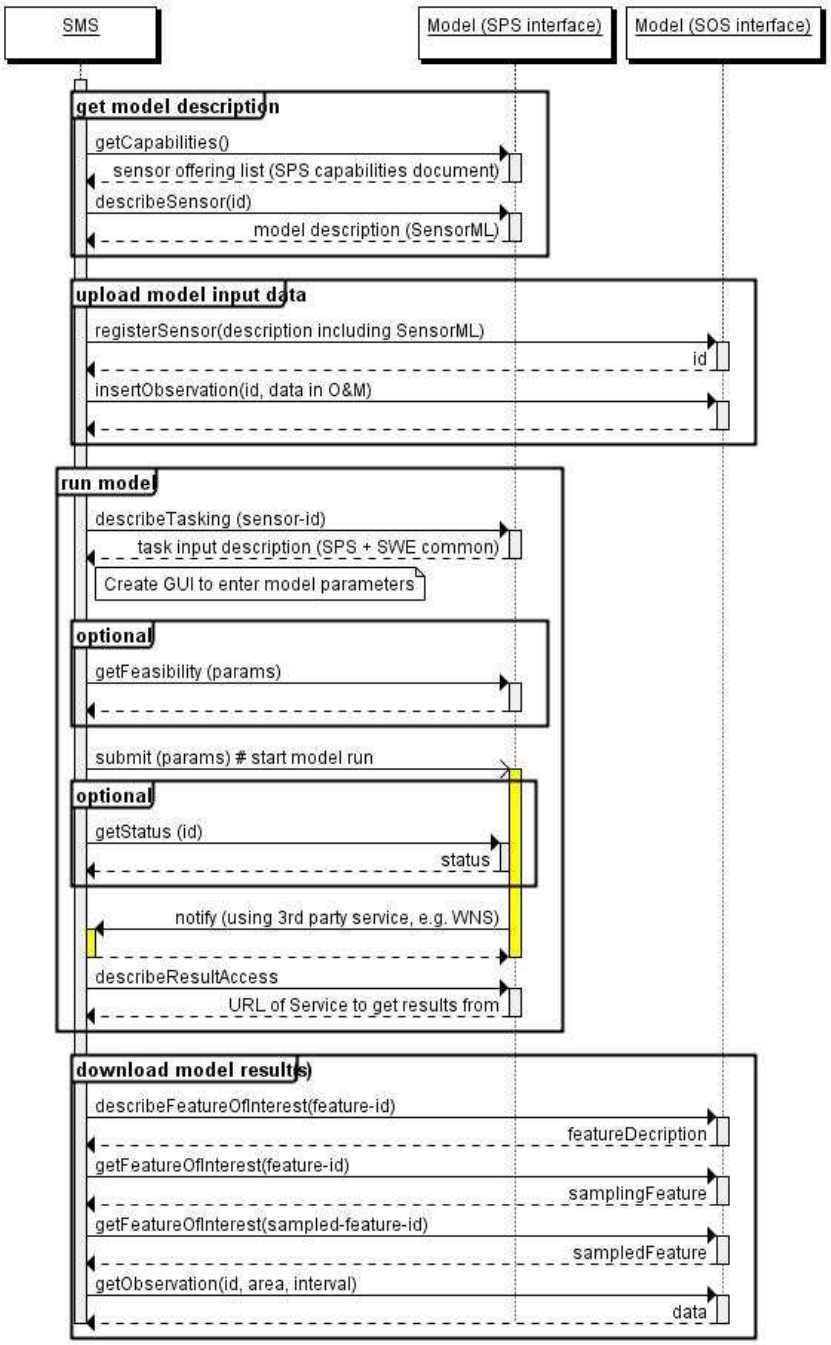

Figure 69: Model Invocation Details

4.2.3 Second Year Development

For the model and service integration into the SUDPLAN SMS the following tasks have been performed:

- Further development in the generic OGC SOS and SPS service and client counter part.
- Providing a service offering climate scenario data
- Providing a service for rainfall time series downscaling
- Providing services for the Linz pilot: Sensor data and local model
- Providing a service for rainfall IDF curves downscaling

Contrary to the first year's prioritisation the integration of hydrology has been moved to the 3rd year. Instead the IDF downscaling was implemented in the 2nd year. The reasons for this are concrete and reflect higher priority user needs of the pilot applications.

4.2.3.1 Rainfall Downscaling Services

There is an important difference between the two rainfall downscaling services. The long rain time series (more than 10 years, at least one value every 10 minutes) allows access to long-term efficiency requirements, as it is done in the Linz pilot. But it is not easy to use this long time series to support access to extreme events. On the other hand IDF curves don't tell anything about the overall amount of rain in the future. Instead they reflect the probability of a specific rain intensity and duration. This is needed in the Wuppertal pilot to assess flooding risks.

Technically the rainfall time series downscaling takes a rain time series as input and projects this time series to a future period under the assumption of a specific climate scenario. What was not explicitly stated in this sentence is the meta-information contained in the time series, in particular the time interval of the input rain time series and matching coordinates. In contrast to time series downscaling IDF downscaling takes an IDF curve as input and projects this to a future period under the assumption of a specific climate scenario. But in this case the time interval for which the input IDF curve is valid, as well as the coordinates, need to be given explicitly.

4.2.3.2 Linz Pilot Application Services

After integration of the local model a complete model run cycle involving access to local data, the upload of these data as input to a common service, running the common service, downloading the result from the common service, and uploading this result as input to a local model can be demonstrated. These steps are followed by a local model run and the download of the results from the local model back to the SMS.

An example of how this is used in the Linz pilot is shown in *Figure 70: Complete Model Run (Linz).*

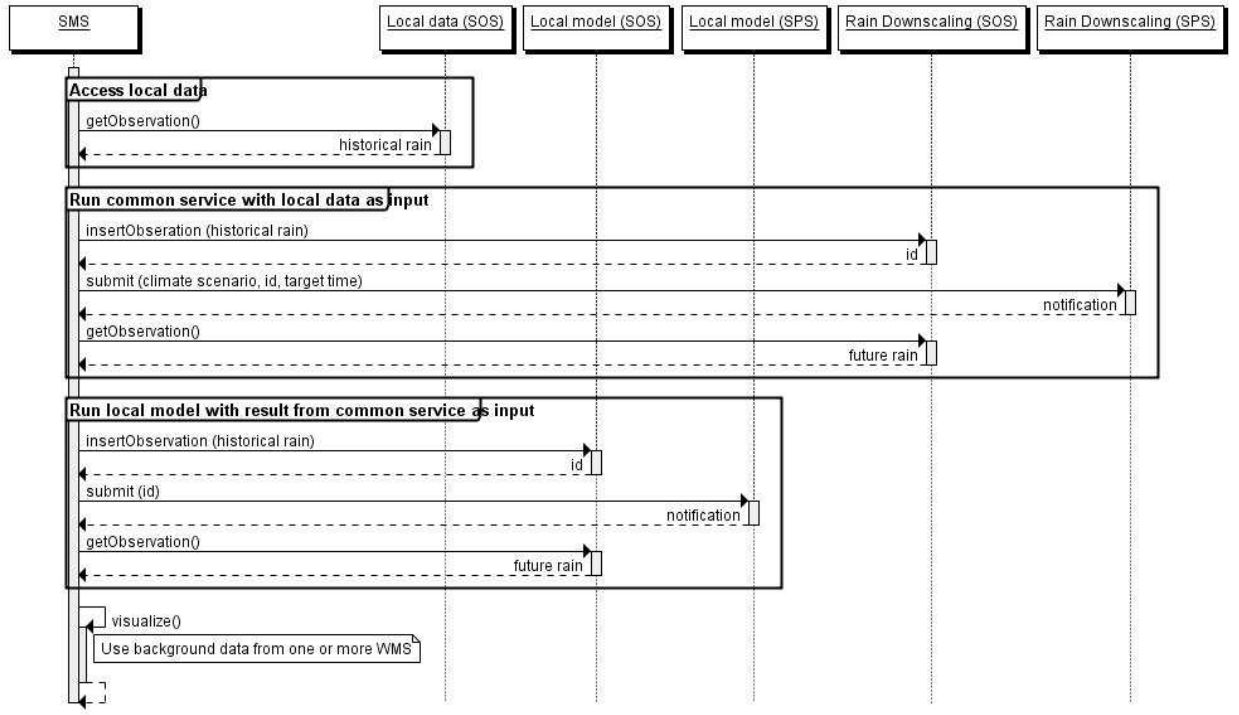

Figure 70: Complete Model Run (Linz)

4.2.4 Third Year Development

For the model and service integration into the SUDPLAN SMS the following main tasks had been performed in the 3rd year:

- Further development in the generic OGC SOS and SPS service and client counterpart
- Finalisation of the air quality downscaling integration
- Integration of the hydrology model
- Deployment of stable OGC services (at SMHI)

The generic OGC SOS and SPS work contained some extensions for result filtering, meta-information and stability enhancements.

The air quality downscaling integration now allows the execution of models on a backend supercomputer.

The integration of the hydrology models enables access to river runoff data.

4.2.5 Current Release Feature Summary

The focus of the third implementation cycle was to complete the development to support the access to all of the downscaling functionality through a set of common services. In particular, dedicated SOS and SPS services have been developed that are accessed from the SMS framework to execute a model (SPS) or retrieve model results and upload input data and parameters (SOS). To facilitate easy integration into the platform the software comes with a set of client components (stubs/proxies) to easily access model related information without the need to deal with the service itself.

The current release supports a number of essential features that are used to access Common Services implementing OGC service interfaces and which can be used to control local models.

- Access to local data through SOS
- Upload model input data through SOS
- Model execution through SPS
- Access to model results through SOS

The usability of these services within SUDPLAN has been demonstrated in the pilots. Furthermore there was also a demonstration of the usability of the services from "outside" SUDPLAN in the scope of a publication at EGU 2012.

The results are available as open source and accessible at http://ts-toolbox.ait.ac.at/SUDPLAN/staging/.

Further development of these services is already on-going in some other projects, so the SUDPLAN results will be available beyond the project lifetime.

4.3. Advanced Visualisation Component

According to the DoW SUDPLAN SMS functionality will include "an advanced 3D/4D visualisation component for the visualisation and animation of 3D results and predictions, in particular using the 3D landscape." This component will help the planners, modellers and decision makers to gain deeper insights into the correlations of phenomena related to their particular workflow. Unlike the previously described parts of the SUDPLAN SMS, the Advanced Visualisation Component has to be built from scratch. This implies several basic tasks which must be performed, besides the actual implementation efforts, such as component design and state of the art research. The efforts carried out in the first and second development phase will be described briefly in the following paragraphs.

In order to meet all of the requirements regarding the visualisation of the SUDPLAN SMS stated by the various formal documents (DoW, D3.1.2 - Requirement Specification V1 etc.) substantial research and evaluation had to be done. This was necessary in order to guarantee a suitable and flexible visualisation solution. The SUDPLAN SMS, more precisely the Scenario Management System Framework described in paragraph 4.1 - Scenario Management System Framework, is based on Oracle's Java technology (in the following only referred as Oracle). Since the Advanced Visualisation Component will become an integral part of the final SUDPLAN SMS, it also has to be based on the same platform-neutral and web-based technologies. Platform neutrality and the ability to invoke the SMS over the web impose certain restrictions on the selection of 3D visualisation APIs. To avoid a proprietary solution that is limited in its interaction capabilities with the SMS Framework, Oracle's own high level API Java 3D [Oracle, 2011] and low level OpenGL Java bindings, such as JOGL, were chosen for the implementation of the Advanced Visualisation Component. The underlying framework was changed to the World Wind Java SDK in the course of the second year.

The goal of the first development cycle was first to create a running system which visualises pilot data related to the Stockholm (Södermalm) pilot. The idea behind this is to incrementally enhance the functionality. In the first version the data were opened and processed directly, but this approach will be gradually replaced by a service-oriented concept. In the course of the project the visualisation component will evolve over each development cycle. To achieve the goals planned for the second development cycle and to accommodate for new and more detailed requirements (general and pilot specific) the underlying framework needed to be changed. Using the World Wind Java SDK it was possible to incorporate the pilot specific data more easily and progress towards the envisaged service-oriented concept.

The overall goal is to produce a flexible and modular component which can be used in any application which has a need for geospatial visualisation in 3D. Special emphasis is placed on the reuse of available APIs and software. Through the course of the project we will use suitable and stable open source software as much as possible. Additionally, we will use publicly available proven geospatial standards such as WMS, WFS etc. in order to facilitate the easy integration of new and existing content for visualisation.

4.3.1 Components and APIs used

In contrast to the initial development (Year 1) it became evident in the second year that it will not be possible to accommodate all (previous and new) user requirements with a Java3D based, built from scratch, Advanced Visualisation Component within the timeframe of the project.

Therefore, a switch in the components used and APIs was necessary. It was decided, in agreement with the work packages involved and the consortium, to use the freely available World Wind Java SDK component as basis for the further development of the Advanced Visualisation Component. After the integration of the World Wind SDK it was thus possible to focus only on the visualisation and animation of 3D results and predictions, in particular using the 3D landscape. More detail on the decision process is given in chapter 4.3.3 Second Year Development.

The following paragraph gives a short overview of the functionalities already provided by the World Wind SDK.

The World Wind SDK is a free and open source Java-API for a virtual globe released under the NASA Open Source Agreement (NOSA). The framework provides a powerful platform for giving the SMS the means to express, manipulate and analyse data of interest. World Wind provides many features for displaying as well as interacting with geographic data and representing a wide range of geometric objects. Moreover, extending the API is simple and easy to do. Following an extract of important World Wind features:

- Open-source, high-performance 3D Virtual globe API and SDK
- Open-standard interfaces to GIS services and databases
- Capable to display high-resolution imagery, terrain, and geographic information from any open-standard public or private source
- Huge collection of high-resolution imagery and terrain from NASA servers
- Supports Coordinate System: Lat/Lon, UTM, MGRS
- Supports of GeoTIFF, JPG, PNG, and JPEG2000
- Supports standard GIS formats: Shape file, KML, GML, GeoJSON
- Different Navigation Modes are being supported
- Supports visualization for stereoscopic displays

Finally, organizations across the world use and support the on-going development of World Wind to monitor weather patterns, visualize cities and terrain. With World Wind taking care of the basic concepts of visualizing geographic data on a virtual globe, we are now able to focus on solving the domain specific problems and to focus mostly on the visualisation and animation of 3D results and predictions, in particular using the 3D landscape.

4.3.2 First Year Development

Note: Due to the change from Java3D to the World Wind Java SDK most parts of this section are obsolete but are left in the document for completeness.

In this paragraph a short introduction to the Java 3D API is given and references are provided for further study. The Java 3D tutorial describes Java 3D as follows:

"*The Java 3D API is an interface for writing programs to display and interact with three-dimensional graphics. Java 3D is a standard extension to the Java 2 JDK. The API provides a collection of high-level constructs for creating and manipulating 3D geometry and structures for rendering that geometry. Java 3D provides the functions for creation of imagery, visualizations, animations, and interactive 3D graphics application programs. The Java 3D API is a hierarchy of Java classes which serve as the interface to a sophisticated three dimensional graphics rendering and sound rendering system. The programmer works with high-level constructs for creating and manipulating 3D geometric objects. These geometric objects reside in a virtual universe, which is then rendered. The API is designed with the flexibility to create precise virtual universes of a wide variety of sizes, from astronomical to subatomic. Despite all this functionality, the API is still straightforward to use. The details of rendering are handled automatically. By taking advantage of Java threads, the Java 3D renderer is capable of rendering in parallel. The renderer can also automatically optimize for improved rendering performance. A Java 3D program creates instances of Java 3D objects and places them into a scene graph data structure. The scene graph is an arrangement of 3D objects in a tree structure that completely specifies the content of a virtual universe, and how it is to be rendered. Java 3D programs can be written to run as standalone applications, as applets in browsers which have been extended to support Java 3D, or both.*" [Sun, 2000].

As described in the definition of the Java 3D API the primary advantage is the high level aspect of the library. The programmer does not have to deal with the low level concepts such as the drawing itself, output device handling, parallelisation etc., and is therefore free to concentrate on the content of the 3D representation itself. Instead of rendering the objects directly like in low level OpenGL the objects are represented in a tree like data structure which allows various optimisations before rendering and easy interaction/modification with individual objects shows a simple example of a scene graph in Java 3D. The figure describes the minimal scene graph necessary to display a shape, using the generic three-dimensional representation of objects in Java 3D.

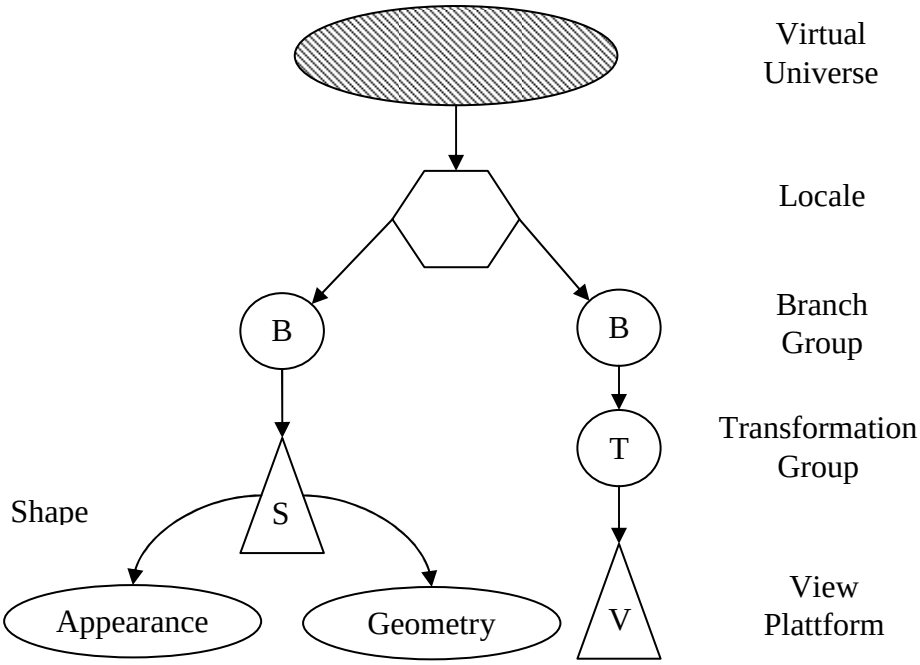

Figure 71: Example of a Java 3D Scene Graph

In this paragraph the design and current implementation of the Advanced Visualisation Component will be explained briefly. In order to develop a visualisation component for an integrated system, a default development process has to be customised. DFKI decided together with the technical partners of the SUDPLAN consortium that it would save time and thus more cost efficient to develop the visualisation component in a sand box. A sand box, in this context, is a small executable application delivering a lightweight environment for testing the visualisation component independently. This procedure has the advantage that the complete SMS does not have to be started in order to test the features of the visualisation component. Consequently, in order to guarantee a nearly seamless integration the commonly known facade pattern [Gamma, 2005] is used to enable design by contract [Meyer, 1992] and thus to deliver an independent component. The overhead this approach produces will be compensated for by the benefit of a modular architecture, which is clearly defined through its interfaces and facilitates reuse of the developed components.

Figure 72: Advanced Visualisation Component Building Blocks shows the core building blocks of the visualisation component, which will be explained in detail on a technical level in the following sections. Core classes of the single components will be presented in standard Unified Modelling Language (UML) Class Diagrams [Booch, 2005] combined with simple tables describing the single classes in order to advance comprehension. The classes are reduced to the important methods and attributes in order to lay special emphasis on the important parts. The description of the renderer is left out intentionally because the Java 3D rendering engine is used as described in *4.3.1 - Components and APIs used*.

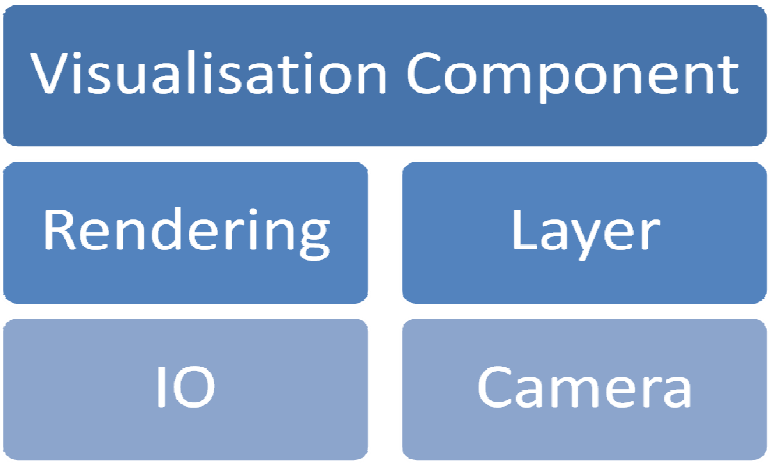

Figure 72: Advanced Visualisation Component Building Blocks

4.3.2.1 Layer

The concept of a Layer is ubiquitous in the visualisation component. All content is added to the visualisation component in the form of layers. A layer encapsulates information and meta information about the content which should be displayed. For example, a layer 'knows' which content it provides, e.g. elevation data, buildings, textures etc. There are more layers available than represented in *Table 1: Core Layer* Classes in order to simplify the overview. *Figure 73: Layer Class Hierarchy* presents the layer management classes.

Figure 73: Layer Class Hierarchy

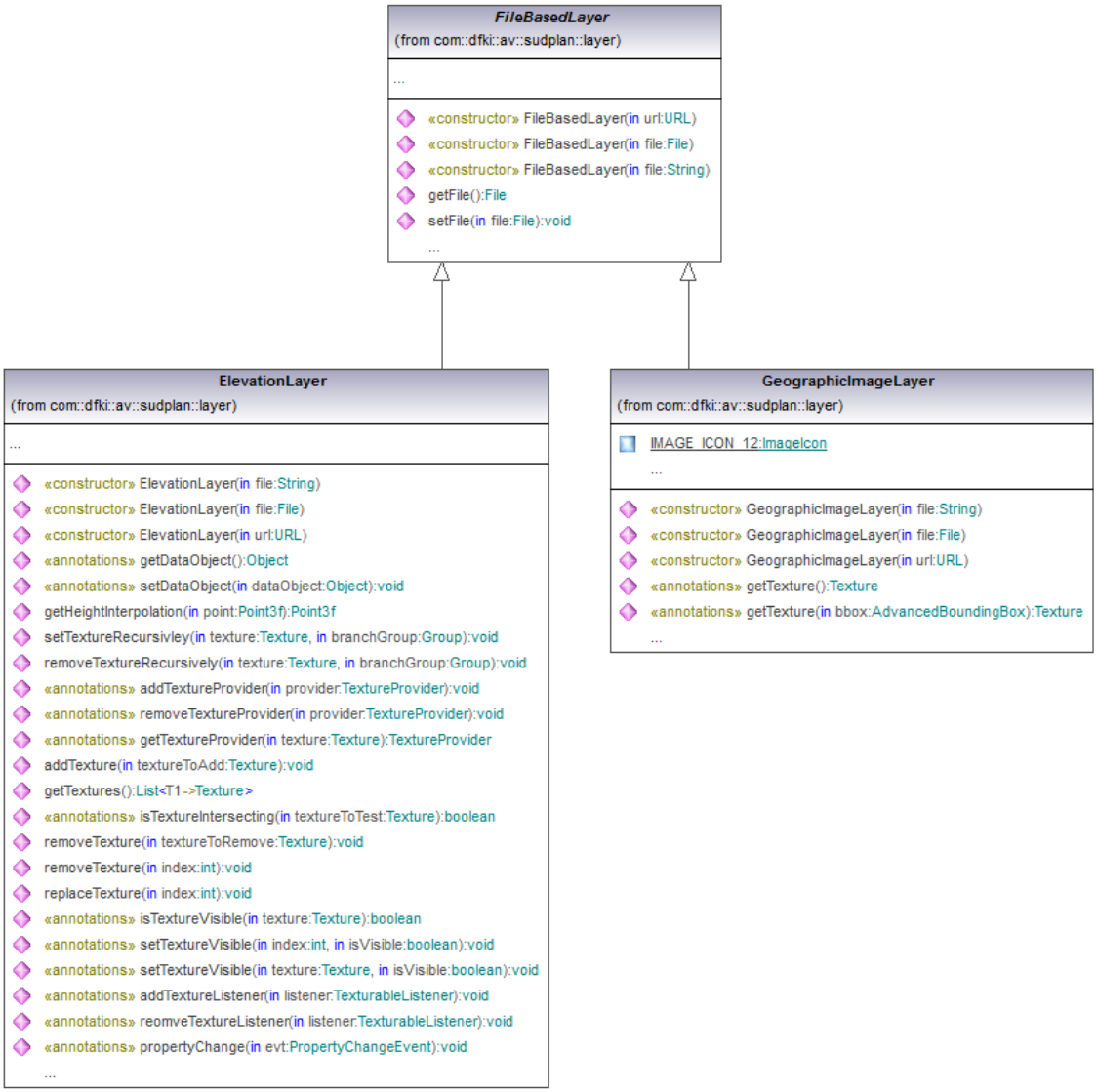

Figure 74: Layer Instances

Class	Description
Layer	The ***Layer*** class is the base class for all layers and defines the basic operations for controlling attributes such as name, icons, visibility, geospatial extend etc.
AbstractLayer	The ***AbstractLayer*** implements most of this functionality in order to provide a base class for all more specific layer classes.
FileBasedLayer	Specialisation of ***AbstractLayer*** which can handle files conveniently. Similar as the ***AbstractLayer*** class this class serves as a base class for other concrete implementation.
ElevationLayer	***Layer*** for handling digital elevation models and to control the texturing of the same.
GeographicImageLayer	***Layer*** for handling geospatial referenced images (e.g. Geo Tiffs) and to display them on top of elevation data.

Table 1: Core Layer Classes

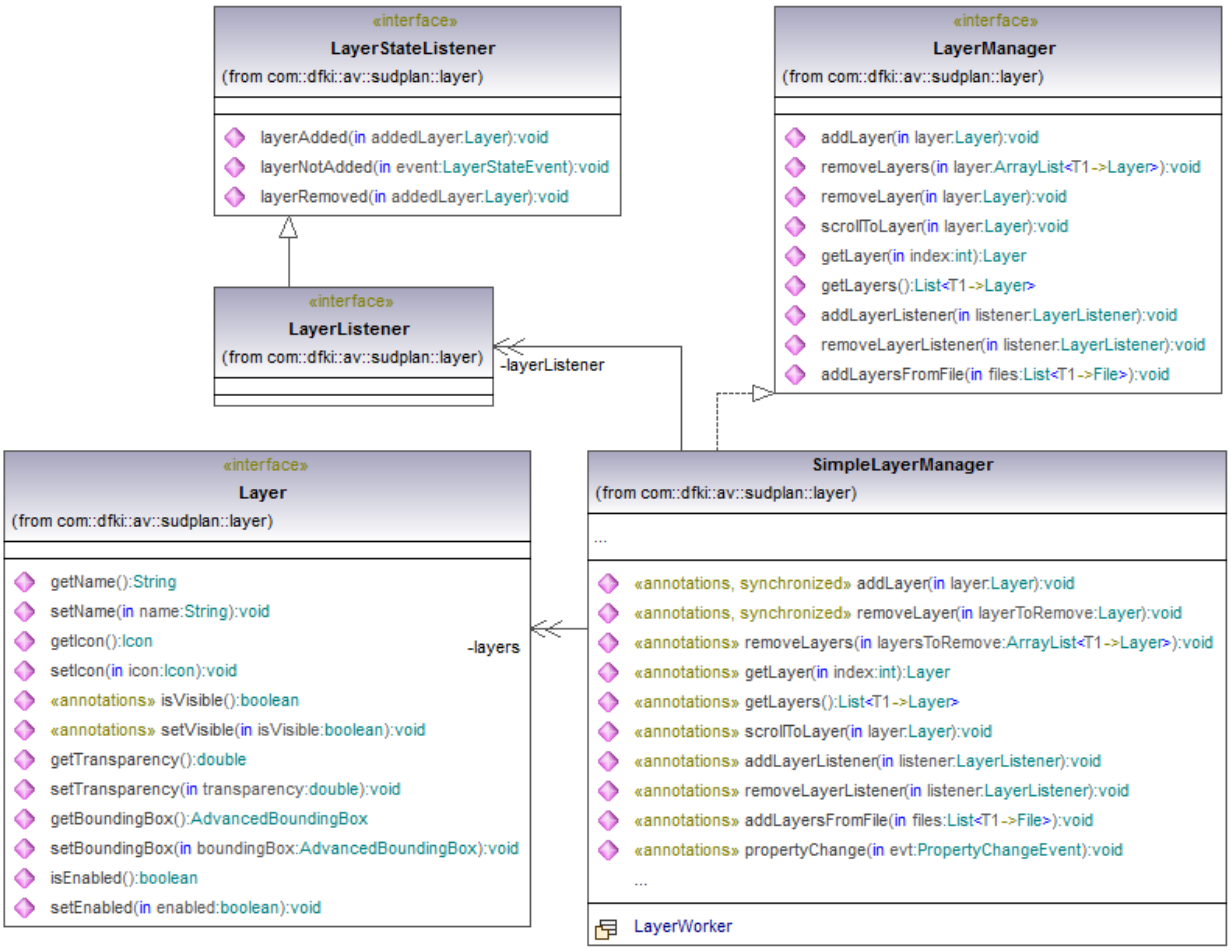

Figure 75: Layer Management Classes

Class	Description
LayerManager	The ***LayerManager*** interface defines methods providing the means to handle layers. Layers will be added and removed through the ***LayerManager***. The ***LayerManager*** also notifies registered listeners about layer state events.
SimpleLayerManager	A ready to use implementation of the ***LayerManager***
LayerListener	Combines ***LayerState*** notifications e.g. layer added/removed with ***PropertyChangeEvents***.

Table 2: Layer Management Classes

4.3.2.2 Camera

In order to create a picture of the interactive 3D scene a camera model is used to control the positioning of the user view in the virtual world. Camera management gives developers the ability to position and query the view of the visualisation scene. A geographical adapter allows the use of the camera directly with geographical units instead of the internal coordinate system. The camera listener mechanism, combined with the calculated 2D view bounding boxes, can be used to synchronise the Advanced Visualisation Component with other components, such as a cadastral map or a Satellite Map.

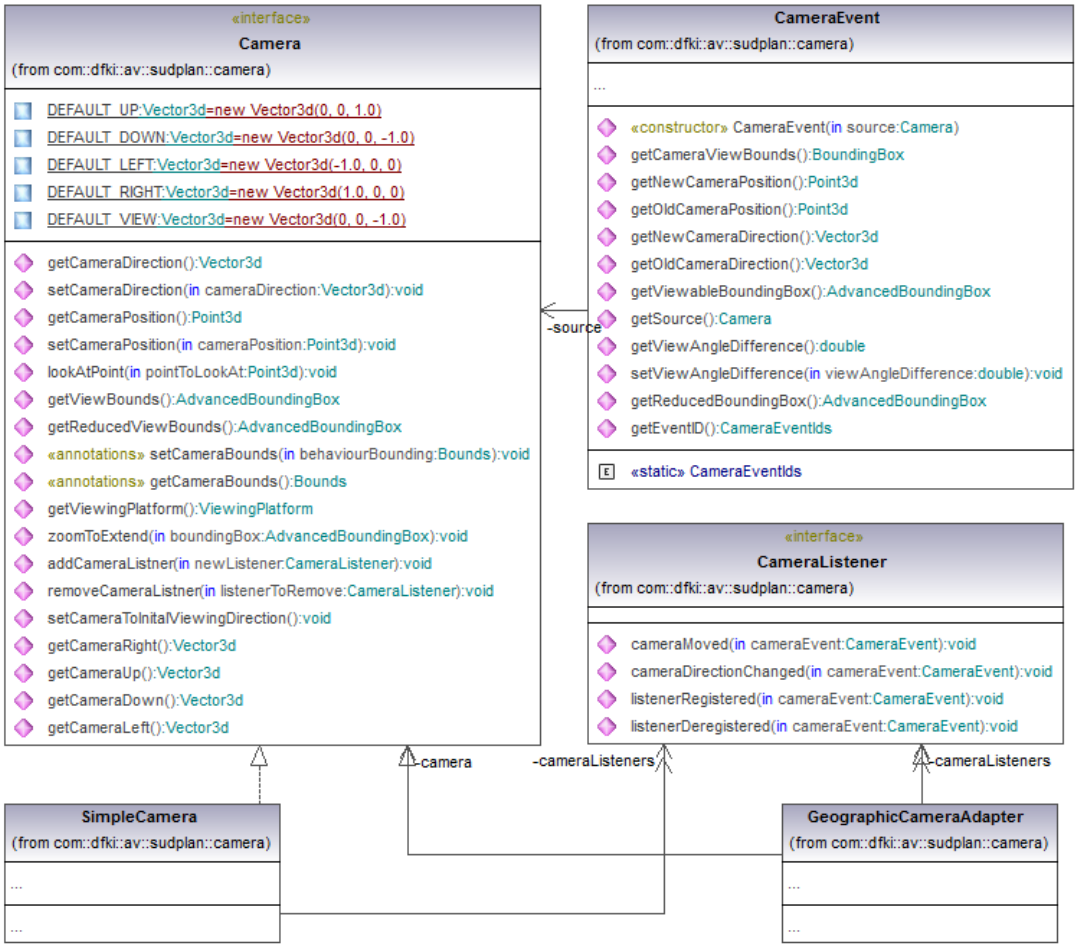

Figure 76: Camera Classes

Class	Description
Camera	The ***Camera*** is the base interface for all camera operation. It provides methods for setting and retrieving the camera position and direction. It also provides convenient methods for setting the camera to look at a specific point in the scene. It also provides access to orientations vectors (up, down, left, right). It defines methods for calculating the view bounding box, which is the visible part of the surface.
SimpleCamera	An implementation of the above specified camera.
GeographicCameraAdapter	Encapsulates a ***Camera*** object and performs the transformation from geographic coordinates e.g. WGS 84 to the internal coordinates used by the visualisation component and vice versa. This allows direct interaction between the visualisation Component and GIS.
CameraListener	Delivers ***CameraEvents*** to registered listeners about camera updates. See ***CameraEvent***
CameraEvent	The event containing ***Camera*** changes such as position and view direction changes.

Table 3: Core Camera Classes

4.3.2.3 Input and Output (IO)

The IO component is mainly used as a support library to transform input files into data types more appropriate for visualisation purposes. The IO component creates **Layer** objects out of well-known spatial data types such as Shape files, ESRI ASCII grids, etc. There are more loaders available than represented in *Table 4: Core IO* Classes in order to simplify the overview. A more complete list may be found in the source code and developer documentation.

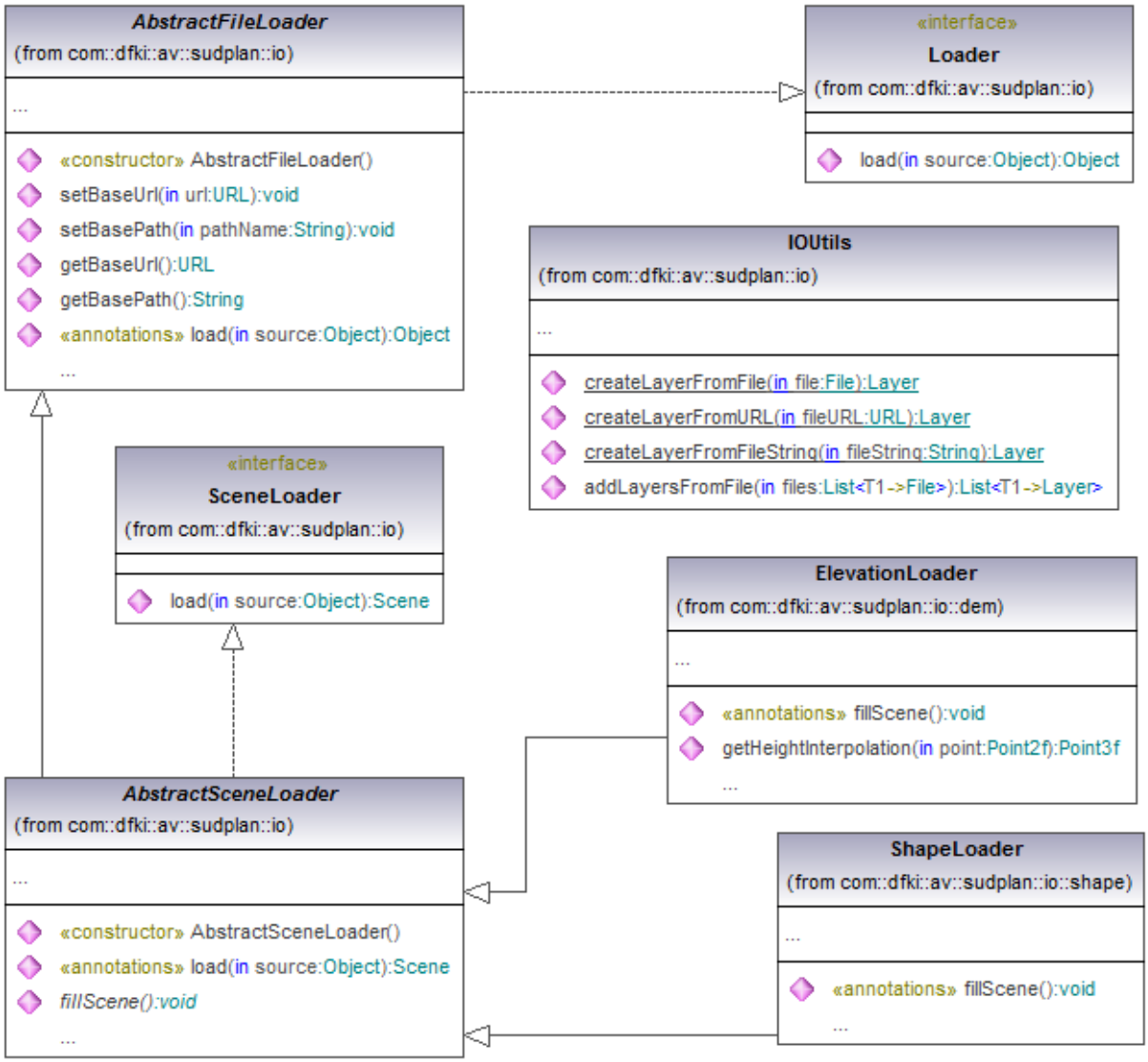

Figure 77: Loader Classes

Class	Description
Loader	Simple interface for a generic loader, which takes an object as configuration and returns the loaded instance.
AbstractFileLoader	Implements basic functionality to handle files either in the host system or via unified resource locator (URI)
SceneLoader	A Java 3D based loader which creates a Java 3D scene object, a data structure representing a scene graph for representing 3D content.

AbstractSceneLoader	Partially implementation of a ***SceneLoader***.
ElevationLoader	The ***ElevationLoader*** parses a digital elevation model from the specified files. Currently ESRI ASCII grids are supported.
Shapeloader	The ***ShapeLoader*** processes ESRI Shapefiles. Currently only lines and polygons are supported.
IOUtils	Convenience class for creating ***Layer*** objects directly without having to use or know about the loader hierarchy directly.

Table 4: Core IO Classes

4.3.2.4 Rendering

The rendering component is responsible for producing pictures from the available layers. These pictures will be the final visualisation the user will perceive. The complete rendering is done in the first version by the Java 3D API as described in *4.3.1 - Components and APIs used*.

4.3.3 Second Year Development

As already stated in section 4.3.2 it was necessary to make a switch in the components used. In the beginning of the second year it became evident that it would not be possible to provide all the needed features – like generating terrain from elevation models, selecting and displaying images from imagery servers, automatic caching and tiling, etc. – within the project scope if it must be built from scratch. The initial decision to use Java3D as a basis of development was reviewed and possible solutions were discussed with the technical partners within SUDPLAN.

The plan for the second year of the development, as described in the previous version of this document, was to

- Implement a Virtual Globe metaphor for visualisation
- Support more Digital Elevation Models
- Support the use of standard geospatial services such as WFS/WMS
- Enhance integration/visualisation of pilot specific data and adequate visualisation metaphors
- Improve navigation and user experience of the first version
- Further improve the infrastructure of the Advanced Visualisation Component

Reviewing the initial possibilities of the first year under the constraints of seamless integration into the SMS it became clear that the World Wind Java SDK would be the best match for further development of the Advanced Visualisation Component, as it already provides a Virtual Globe metaphor, improved navigation capabilities compared to the year one prototype and support for basic geospatial services.

It was decided to use the World Wind Java SDK component as a basis for all further developments of the Advanced Visualisation Component, because it is available as open source, delivers a lot of features needed within the project and has the smallest overhead when integrating it into the SMS.

Furthermore, the virtual globe provided by the World Wind Java SDK already provides some of the aforementioned features planned for the second year, thus we could focus the development on the remaining features and the integration of pilot specific data/results.

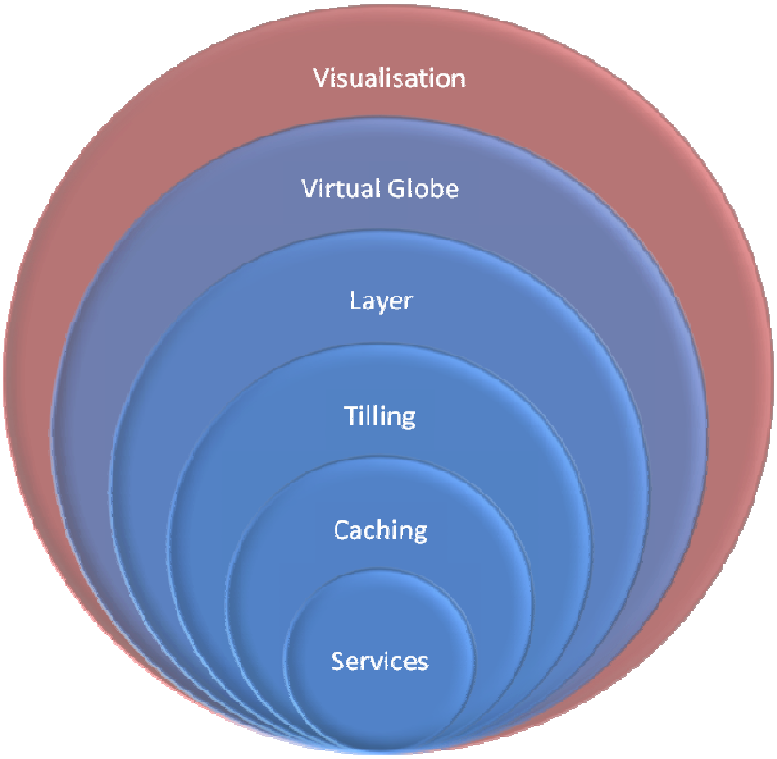

Figure 78: Features provided by the World Wind Java SDK (blue)

Because of the above mentioned decision – to switch to the World Wind Java SDK – the first task was to implement and integrate the new version using World Wind into the SMS that provides the same set of basic features as V1. Since the implementation of the basic feature set could be completed quite quickly, we were able to focus on special visualization tasks. In addition, the integration of input data from the pilots was performed as data became available.

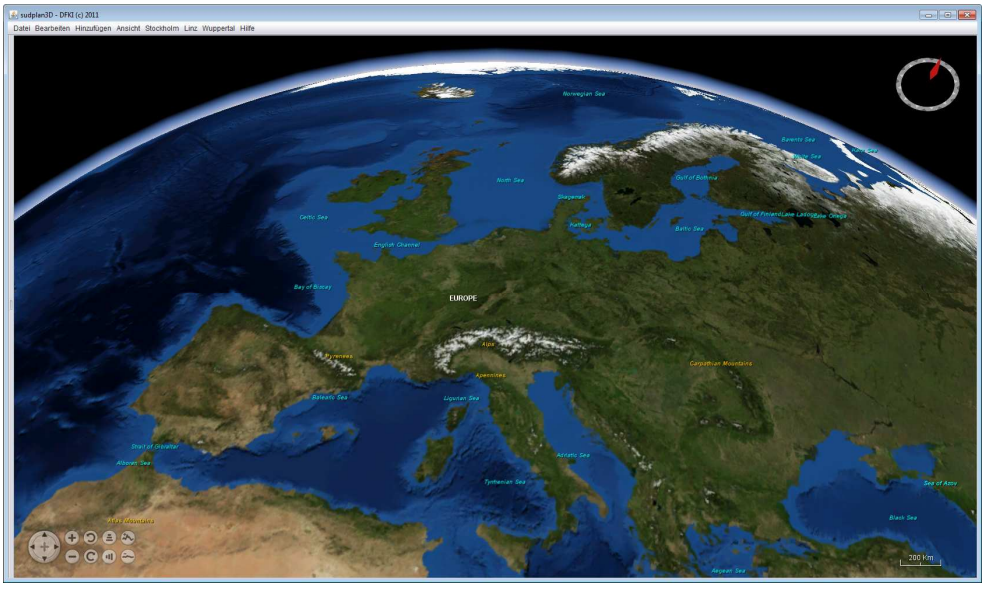

Figure 79: 3D Visualization Component Using World Wind Java SDK

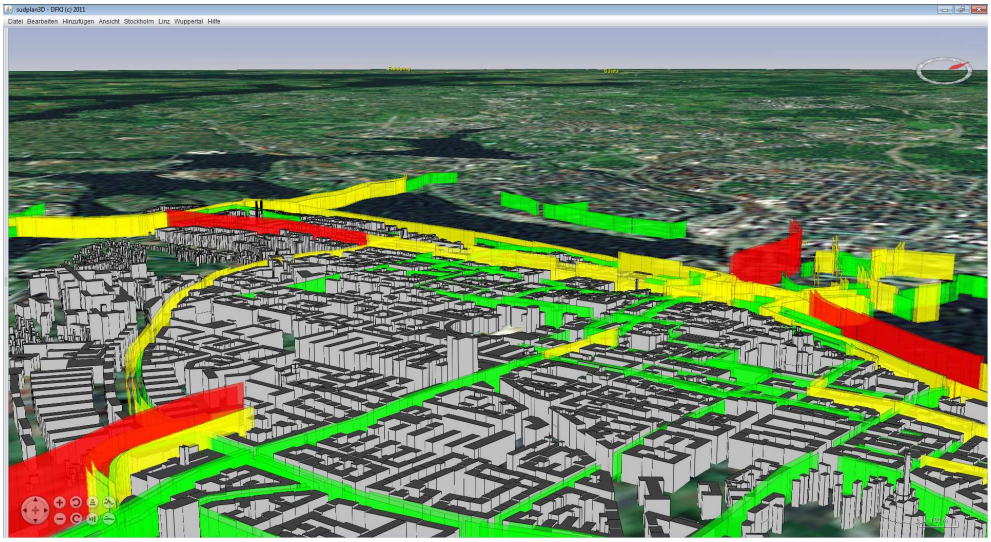

Figure 80: 3D Visualisation of Air Quality and Traffic Density in Stockholm

Figure 80 illustrates the relation of air quality and traffic in the city of Stockholm. Since the World Wind SDK already provides many standard GIS features (e.g. elevation model, compass, the possibility to increase the digital elevation model etc.) the 3D visualization is able to focus on suitable visualization techniques.

4.3.3.1 Integration of the World Wind SDK

The integration of the World Wind SDK into the Advanced Visualisation Component and thus into the Scenario Management System was the first task that had to be completed. In order to achieve this, different interfaces of SUDPLAN's 3D visualization component had to be enhanced.

First, the interface VisualizationComponent with its implementation VisualizationPanel has been changed (see Figure 81). The class LayerAction is an event type to provide the mean for the Scenario Management System to turn on / off a selected Layer in the 3D visualization component. Moreover, we added methods to add and remove a layer. The input for this method is of type Object to be able to support an unlimited number of possible data sources. Which kinds of data sources are being supported depends on the implementation. Currently, the only implementation of the interface VisualizationComponent is the class VisualizationPanel**.** This class supports data sources of type File, URI, and URL to be able to support local as well as any remote data source (e.g. web services, etc.).

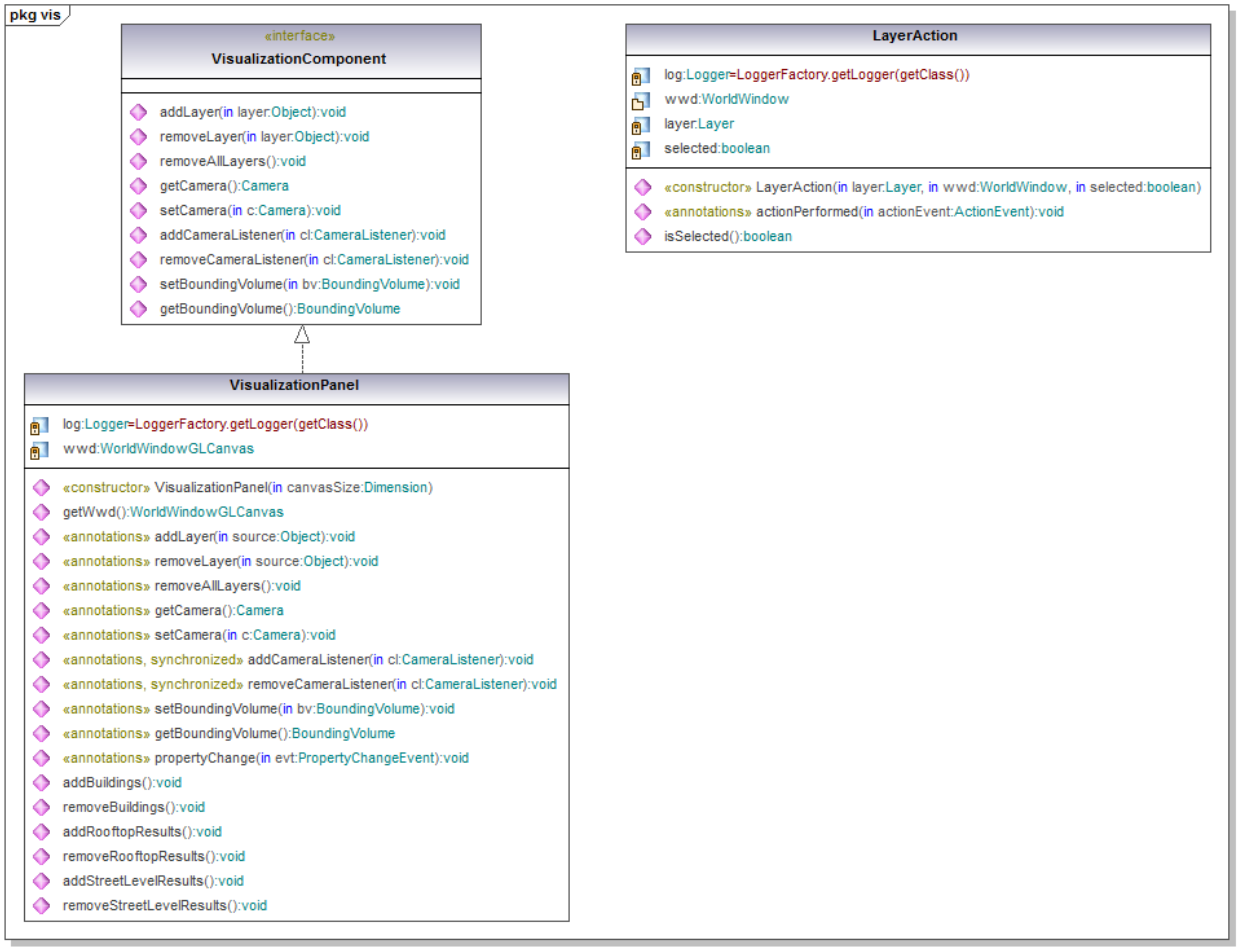

Figure 81: Class LayerAction

Considering the integration and usage of the World Wind SDK the interface definition for the VisualizationComponent and its implementation VisualizationPanel had to be changed slightly. The class LayerAction provides the means to disable/enable different layers shown by the 3D visualization component. As in the former implementation the synchronization of the 2D map view with the 3D virtual globe view and vice versa was a challenging task. In order to address this problem the methods add-/removeCameraListener(), get-/setCamera(), and get-/setBoundingVolume() have been added to the VisualizationComponent.

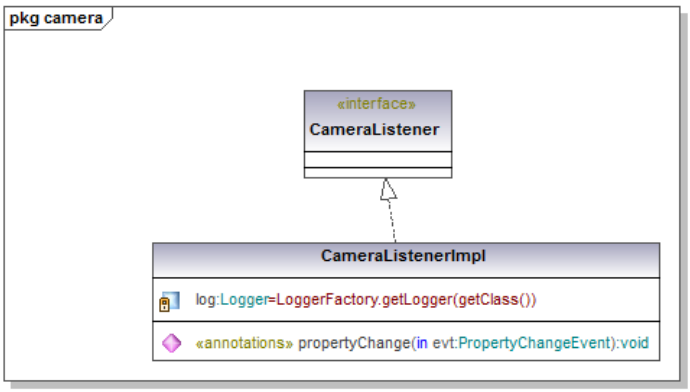

Figure 82: CameraListener

The CameraListener is derived from the java.beans.PropertyChangeListener. Every change of the camera position is sent to the classes implementing this interface. While add-/removeCameraListener() ensures that any change of the camera position or orientation within the 3D visualization is also displayed in the 2D view the methods get-/setCamera() and get-/setBoundingVolume() ensure that any change in the 2D view is also visible in the 3D component. The former one uses a kind of an observer pattern. In order to get any changes from the 3D component the "surrounding" Scenario Management System only needs to implement the interface CameraListener (see Figure 82).

The interface Camera (see Figure 83) is the interface for the camera operations in the 3D visualisation component and serves as a data structure. It provides methods for retrieving the camera position (latitude and longitude), the viewing direction, and the BoundingVolume (see Figure 83). Using the class SimpleCamera the user is able to define the position and orientation of the camera and thus the view onto the virtual globe. Here, the view immediately switches from one position to another without any animation. In contrast to the SimpleCamera class the AnimatedCamera class provides a mean to animate the camera view in a flight mode to a new camera position.

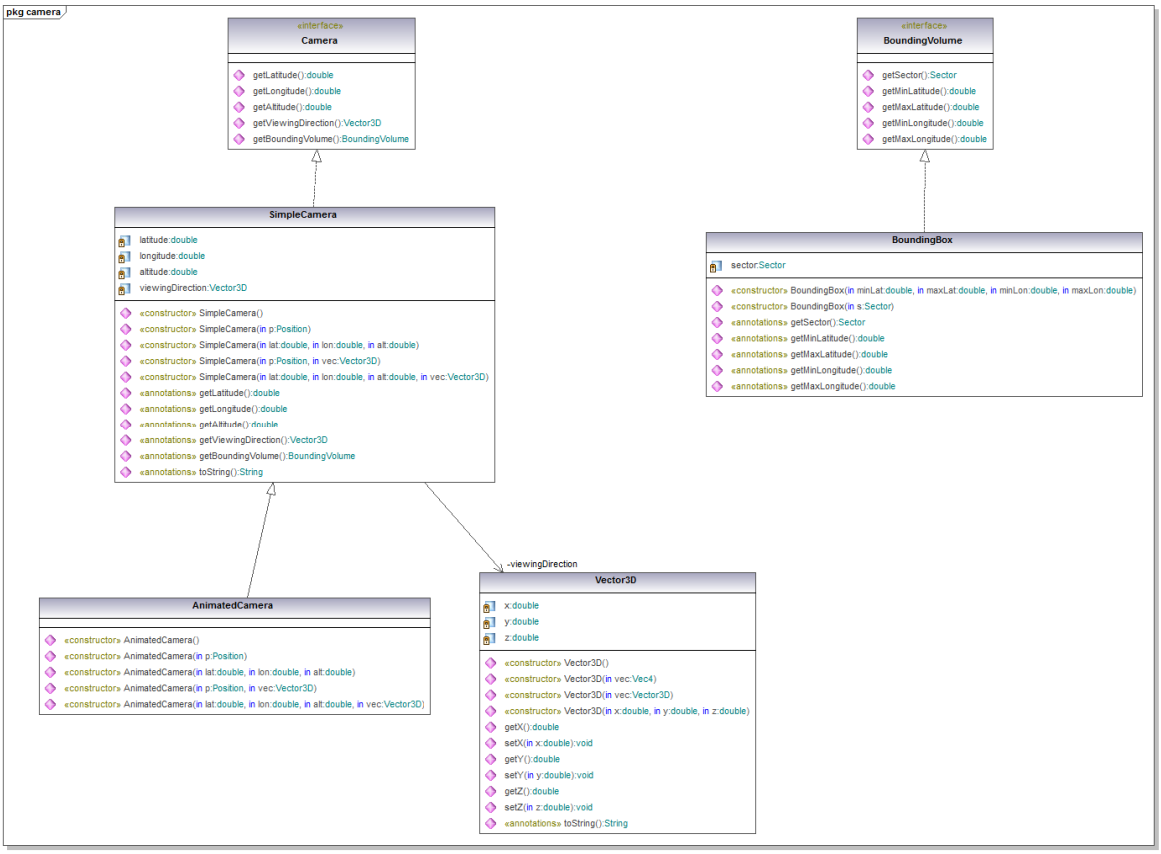

Figure 83: Animated Camera and BoundingBox

The Camera provides an interface for a data structure used to control the camera position and orientation. Using the class AnimatedCamera one is able to animate the camera on an interpolated path to a desired position in a flight mode. The class BoundingBox covers a geographic area.

Finally, the methods setBoundingVolume() and getBoundingVolume() provide a means to set or get the camera position – and move it to the correct position – according to the area covered by the BoundingVolume. Currently, the only implementation is the class BoundingBox with its private attribute sector. This object member is defined by four corners (see Figure 83). Later, any other BoundingVolume could be added to support different kinds of primitives or geometric volumes (e.g. bounding sphere, etc.).

4.3.3.2 The Visualization Wizard (VisWiz)

Because of many features already provided by World Wind's virtual globe we were able to come up with a more general concept in order to cope with the pilots' visualisation needs. We developed a new visualisation wizard called VisWiz (see Figure 69). The idea of VisWiz is to provide a means to support the user in selecting a suitable and state-of-the-art visualization technique both in the sense of scientific visualization and information visualization.

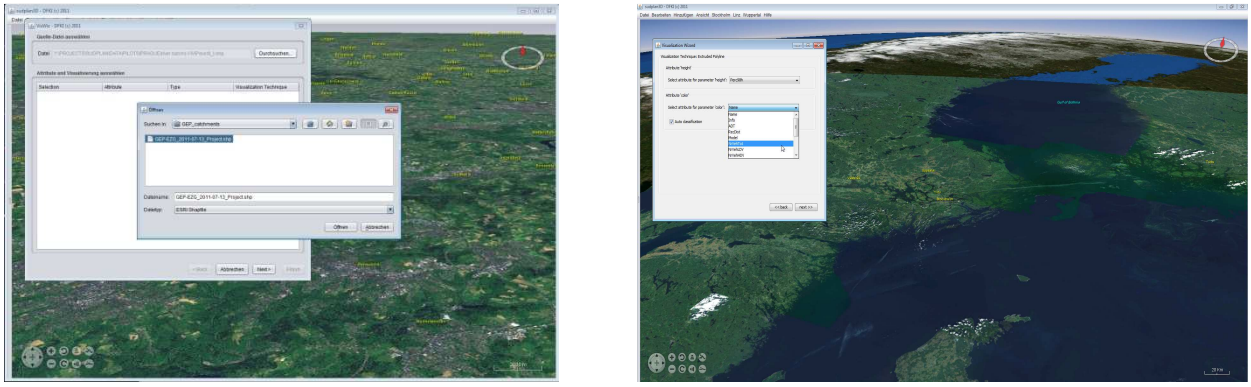

Figure 84: VisWiz Screenshot

Figure 84 depicts an example of the visualisation Wizard (VisWiz). With VisWiz the user is able to select between different visualization techniques for one or multiple attributes provided as input source.

The VisWiz was designed to support all user types for selecting suitable visualisation techniques for the selected data even if they are not familiar with visualisation in general or the content of the data source. In order to be usable for a wide variety of data sources and by a broad user group, great emphasis was put on the following points for the design of the VisWiz:

- **Independence from data source:** The user should be able to use data even if they are not in the SUDPLAN SMS (e.g. results produced by external sources)
- **Extendable visualisation collection:** It should be possible to easily extend the collection of visualisations with new techniques.
- **Intelligent proposal:** The system should be able to propose suitable visualisation techniques based on the selected data so the user does not need to have any prior knowledge.
- **Simple user interaction:** An inexperienced user should be able to use existing data sources to produce suitable visualisations without further knowledge of the data or visualisation techniques

Figure 85 illustrates an abstract overview of the design and workflow of the VisWiz component. VisWiz is divided into several modules to allow for an easy extension of supported data formats/sources, suggestion functions as well as state-of-the-art visualisation techniques. Techniques in the area of scientific visualization and information visualization are of special interest.

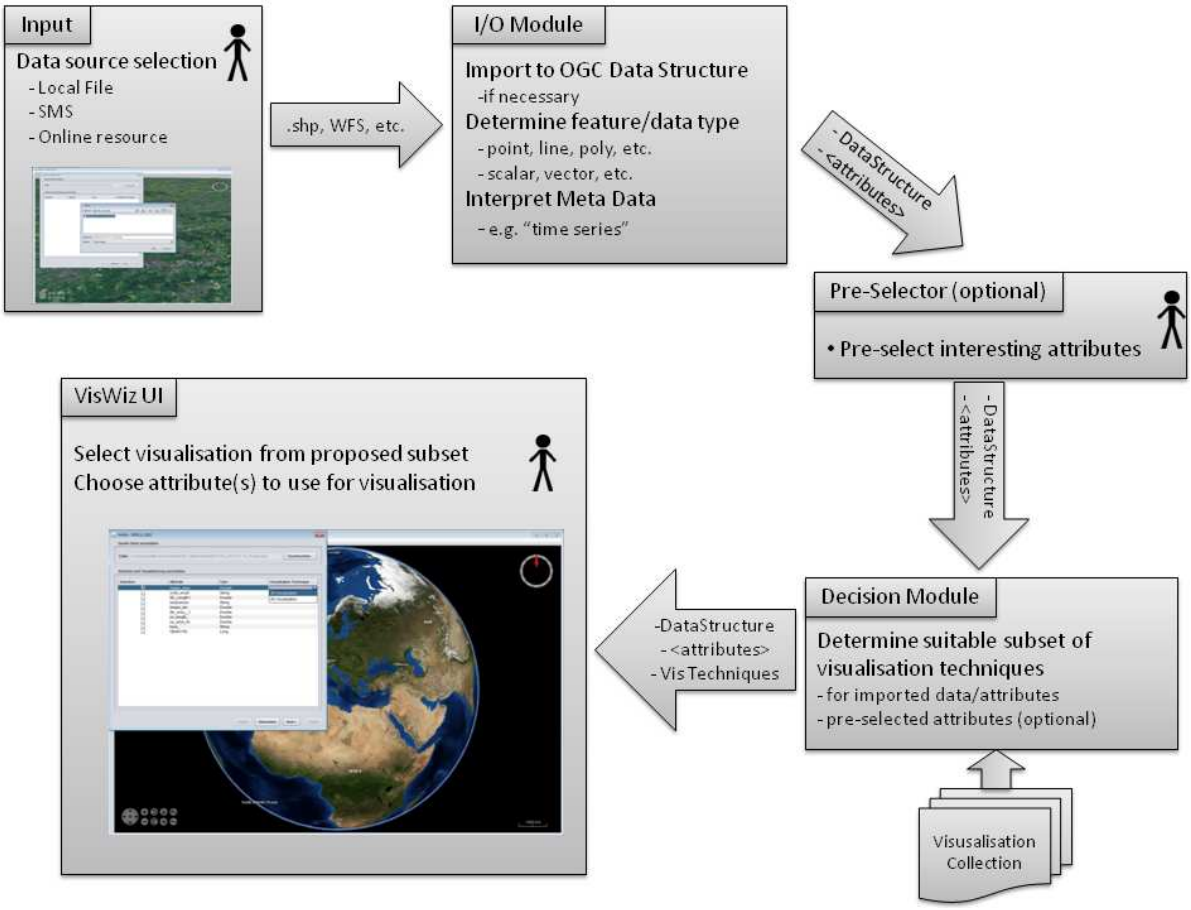

Figure 85: Abstract VisWiz Modules and Workflow

Input module: The input module provides an easy interface to allow the user to choose a data source which should be used for the visualisation. This could be a result file from an external simulation tool, a result stored within the SUDPLAN SMS or an online resource (i.e. remote WFS server).

I/O module: The I/O module imports the selected data source into the internal data structure for the visualisation component which is based on the OGC Data), determines what feature and data types are included within the chosen data and interprets additional meta information which could be available. The I/O module computes an *attribute vector* describing the imported data (feature types, attribute types, meta data) in a way to automatically determine suitable visualisation techniques in the *Decision module*.

Pre-selector module (optional): If the user is familiar with the selected data or already knows which attributes they want to visualise they can select attributes of interest and thus constrain the *attribute vector* which is used to determine the possible visualisation techniques.

Visualisation Collection: The visualisation collection consists of all available visualisation techniques. Additional meta information for each technique, which is used by the *decision module* to determine if it is suitable for the selected data, is also include in the collection.

Decision module: Based on the *attribute vector* computed by the *I/O module* and the meta information from the *Visualisation Collection* the decision module determines a subset of visualisation techniques suitable for the selected data.

VisWiz UI: The VisWiz UI presents previews along with further information on the usage of the suitable visualisation techniques, e.g. minimum number of needed attributes, to the user. The user can select a visualisation technique and is then guided through the attribute selection for the different visualisation parameters.

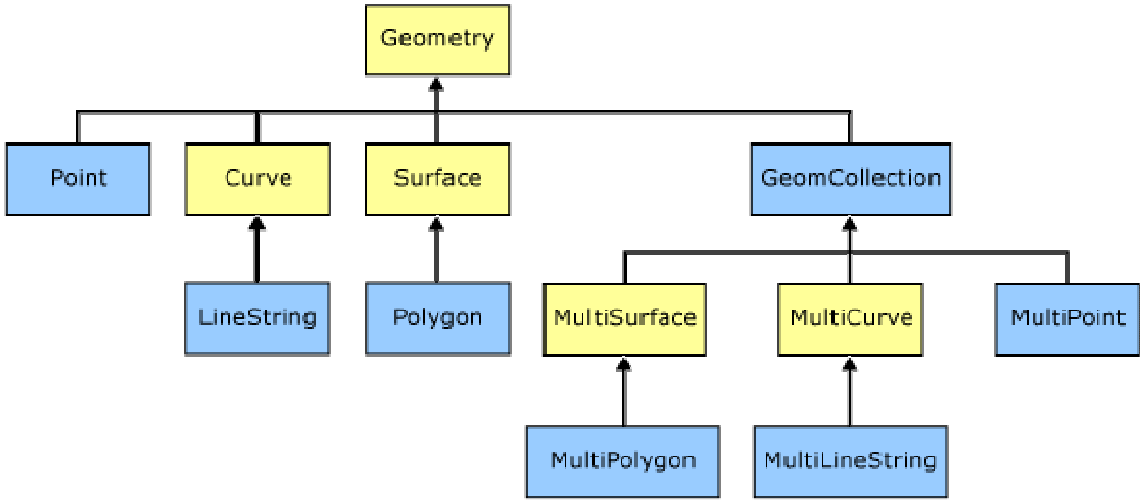

Figure 86: OGC Based SUDPLAN 3D Data Structure

An example workflow is depicted in Figure 87. The user opens a data set and the VisWiz presents the suitable visualisation techniques. In this example, street level results for air quality were chosen by the user. As the features in the data set are LineStrings, the system proposes the visualisation techniques “Extruded PolyLine” and “PolyLine”. The user chose the “Extruded PolyLine” technique which needs two parameters, one for the height and one for the color of the generated polygon. The user is then presented a dialog where they can easily map attributes from the selected data source to the two parameters of the visualisation. In this example the user wants to use the attributes “Perc98h” (98 percentile for NO concentrations) and “NrVehTot” (total number of vehicles). In the example these two attributes are mapped to the ‘height’ and ‘color’ parameter of the visualisation (left) and vice versa (right). The result of the different combinations is shown at the bottom.

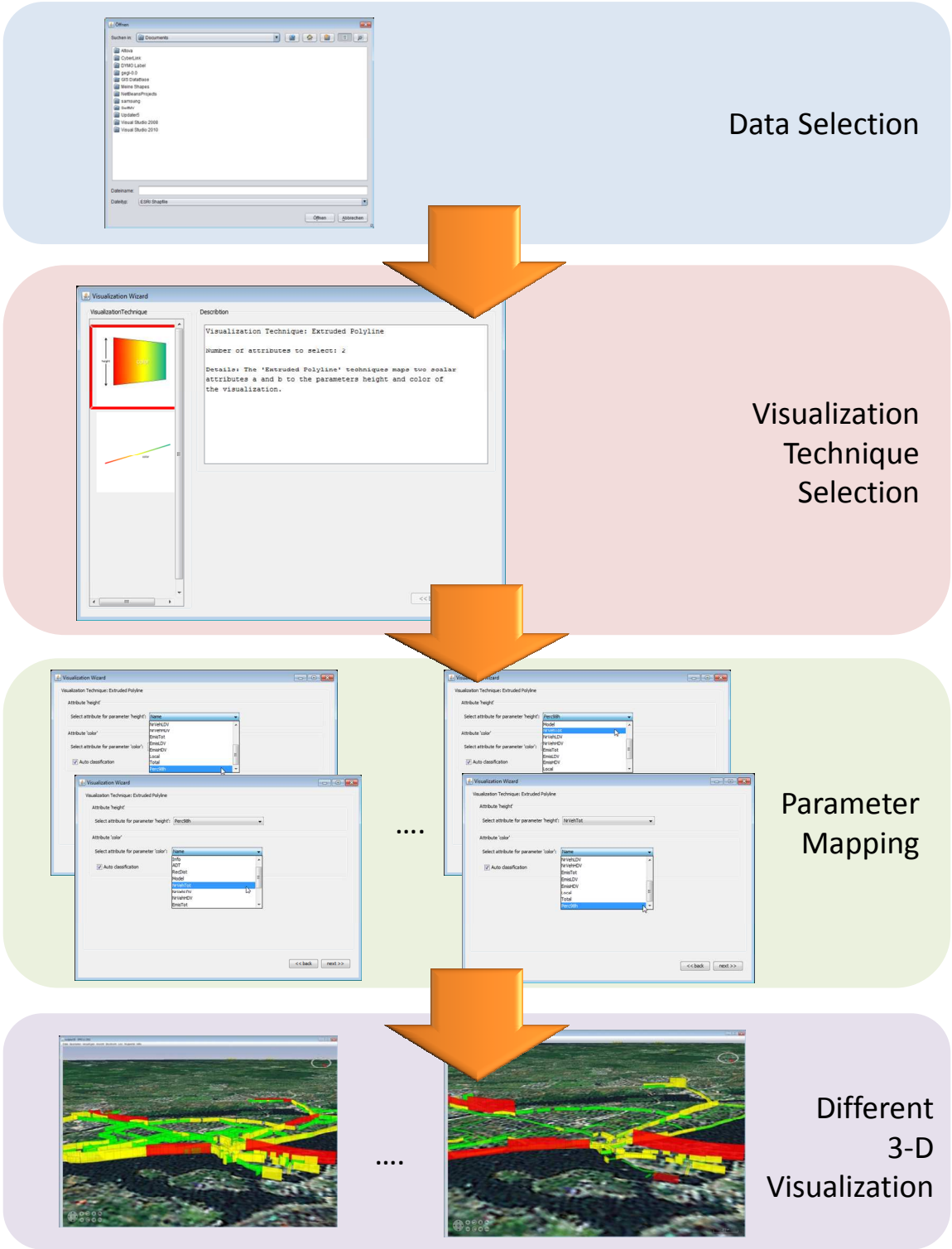

Figure 87: VisWiz workflow

Figure 87 shows an example for a VisWiz workflow. After the selection of the data source the user selects a visualization technique. Depending on their selection and the input parameters the user is able to map the attributes of the data source. According to the mapping of attributes to visualisation parameters different presentations will be computed.

4.3.4 Third Year Development

Based on the developments of the second development year, goals for the third year were:

- to implement further visualization techniques from the current state-of-the-art in information and scientific visualization, e.g. volume visualization techniques like marching cubes for iso-surface generation. Special focus was on information visualization in combination with suitable interaction means.

- to integrate the visualisation techniques into the VisWiz to enable the user to change and test different types of visualisation in an easy way.

Moreover, the 3-D visualization took advantage of new display devices like 3-D stereoscopic desktop screens as well as 3-D stereoscopic projections. Finally, we have investigated how modern input devices like multi-touch devices support users in the context of a GIS environment.

The following section will describe the main features of the current release. We will detail a new plug-in framework. Moreover, we will illustrate the current state-of-the art visualization results for the city of Stockholm and the city of Wuppertal. Finally, we show how we have enhanced the 3-D stereoscopic mode for the WorldWind API to be used by the SUDPLAN component.

4.3.4.1 The Visualisation Plug-in Frameworks

The integration of visualisation techniques into has been an important task. The visualisation techniques should be easy to integrate and without the need to modify the original code base in case of changes related to the visualization itself. Moreover, one should have the possibility to enhance the functionality with new plug-ins or modules. Thus, not only developers but also third parties can add new functionality (e.g. a new state-of-the-art visualization for a particular use-case) to the application.

To address the above mentioned problems we decided to use Java's Service Provider Interface[1] (SPI). A service provider framework is a system in which multiple service providers implement a service, and the system makes the implementations available to its clients, decoupling them from the implementations.

There are three essential components of a service provider framework: a service interface, which providers implement; a provider registration API, which the system uses to register implementations, giving clients access to them; and a service access API, which clients use to obtain an instance of the service. The service access API typically allows but does not require the client to specify some criteria for choosing a provider. In the absence of such a specification, the API returns an instance of a default implementation. The service access API is the "flexible static factory" that forms the basis of the service provider framework.

"A service is a set of programming interfaces and classes that provide access to some specific application functionality or feature. The service may simply define the interfaces for the

[1] For detailed information see: http://docs.oracle.com/javase/tutorial/sound/SPI-intro.html

functionality and a way to retrieve an implementation."[1] In our case, a service is e.g. a visualization or classification algorithm, but it does not implement the underlying features. Instead, it relies on a service provider to implement that functionality, i.e. a particular implementation like a bar visualization. "A service provider interface (SPI) is the set of public interfaces and abstract classes that a service defines. The SPI defines the classes and methods available to the application. A service provider implements the SPI. An application with extensible services will allow vendors, and perhaps even third parties to add service providers without modifying the original application."[2]

The following section describes briefly the different services we implemented and how they can be used to enhance the current state of the Advanced Visualization Component.

4.3.4.2 Extensible Visualization Algorithms

As mentioned above both developers and third parties should have the possibility to extend the 3-D visualization without the need of base code changes. Figure 88 illustrates the usage of the SPI framework for the visualization algorithms.

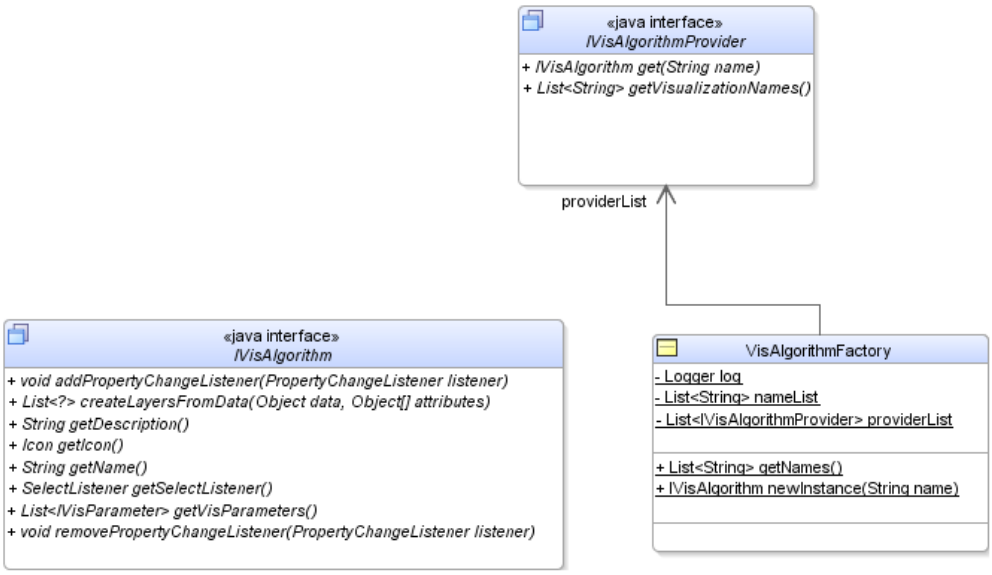

Figure 88: The service provider interface definition for the visualisation plug-in framework

The class `VisAlgorithmFactory` is the responsible for collecting all classes that implement the interface `IVisAlgorithmProvider`. An implementation of the `IVisAlgorithmProvider` interface contains a reference to all the implementations of the interface `IVisAlgorithm` that it provides. Finally, the interface `IVisAlgorithm` provides the necessary methods for a visualization technique. In detail the method `createLayersFromData(…)` is the main method responsible for setting up the 3-D visualization for the virtual globe. The method `getVisParameters()` returns all visualization parameters that can be changed by the user. Figure 89 presents the `IVisParamter` interface and its currently available implementations. Using those parameters the developer of the visualization technique can decide which settings the user can change.

[1] See http://java.sun.com/developer/technicalArticles/javase/extensible/
[2] See http://java.sun.com/developer/technicalArticles/javase/extensible/

Providing a `ColorParameter` for the roof in case of the `VisBuildings`, for example, gives the user the possibility to change the colour of the roof. The way how he can change the colour will be explained in the section 4.3.5.3 and 4.3.5.4.

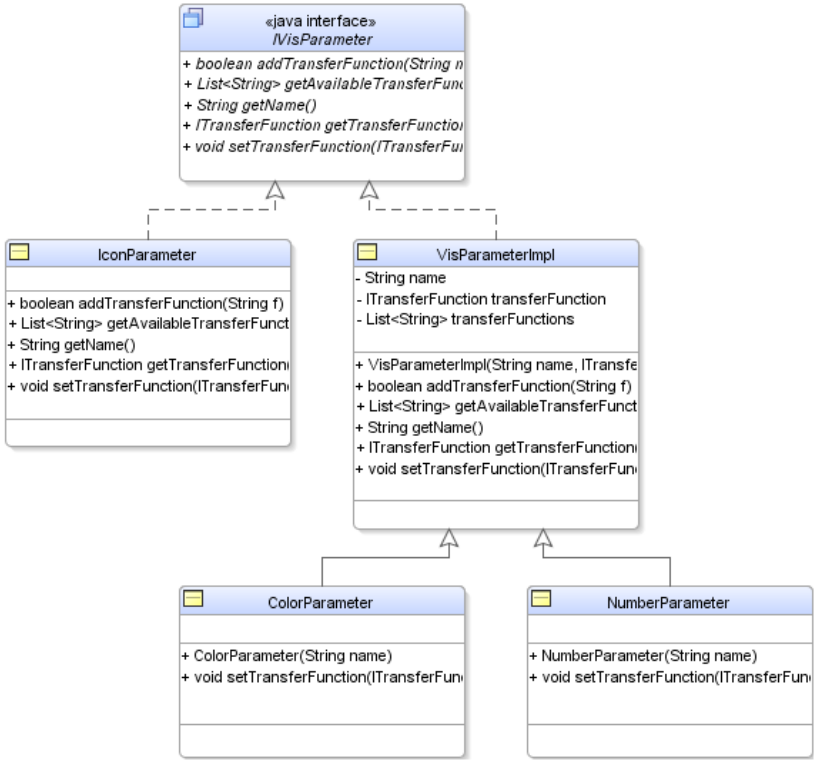

Figure 89: The `IVisParamter` interface and its implementations

Additionally, the developer of a new `IVisAlorithm` implementation does not have to deal with the user interface. The information how to set up the user interface for the VizWiz is taken from structure and Meta information of the implementation. For example, returning a ColorParameter and NumberParameter results in an user interface shown in Figure 72).

Currently, the 3-D visualization component supports multiple visualization techniques. Among others:

- **VisBuildings** – This visualization aims at displaying 3-D buildings on the virtual globe. Different parameters can be configured. Besides the color of the cap the height of the building can be defined.

- **VisTimeseries** – This technique can be used to visualize and animate layers over different timesteps.

- **VisPointCloud** – The VisPointCloud visualization displays the points of a given 3D-data set. It serves as default visualization in case no other visualization is available or fits the underlying dataset.

- **VisDelaunayTriangulation** – This visualization can be used for a point cloud dataset. It shows the convex hull triangulation of the point set using the Delaunay triangulation.

- **VisMarchingCubes** – Provides an iso-surface for a given gridded data set. The user can choose one or more isovalues. Moreover, he can also use serveral time steps. A detailed description will be given in section 4.3.5.5.

4.3.4.3 Extensible Transfer Functions

In order to give the user the means to adjust the configuration of the visualization (e.g. the color of the cap, the height of building, etc.) we provide the `IVisParameter` as mentioned above. The type of the parameter (i.e. color, number, or icon) declares which kind of setting can be changed but not how it can be changed. This is specified by the transfer function.

Figure 90 shows the SPI definition for the `ITransferFunction` interface. As for the `IVisalgorithm` the `ITransferFunctionProvider` is the interface for all provider implementation (e.g. `BasicTransferFunctionProvider`). The `TransferFunctionFactory` collects all available implementations of the `ITransferFunctionProvider` interface and thus all implementations of the `ITransferFunction`.

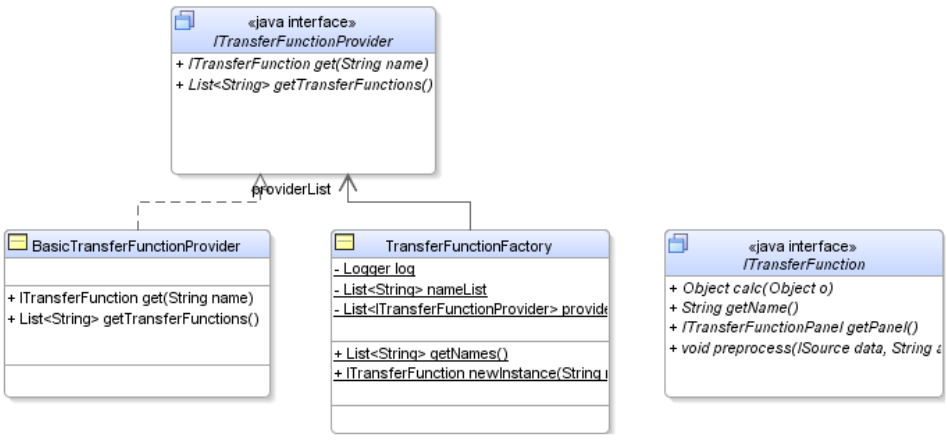

Figure 90: The SPI definition for the transfer function plug-in framework

According to the classification of the `IVisParameter` in `ColorParameter`, `NumberParameter`, and `IconParameter` the `IVisTransferFunction` architecture is devided into `NumberTransferFunction` and `ColorTransferFunction` (see Figure 91). The abstract class NumberTransferFunction is used to transfer any object into an object of type Number (e.g. Integer, Double, Float, etc.). For example: the `ConstantNumber` transfer function returns always a constant number value regardless of the input argument. For both types we already implemented the basic operations/transfer functions. However, using the SPI functionality the user / developer can easily develop and aggregate his own transfer functions. He can combine existing ones and thus increase the complexity of the transfer functions.

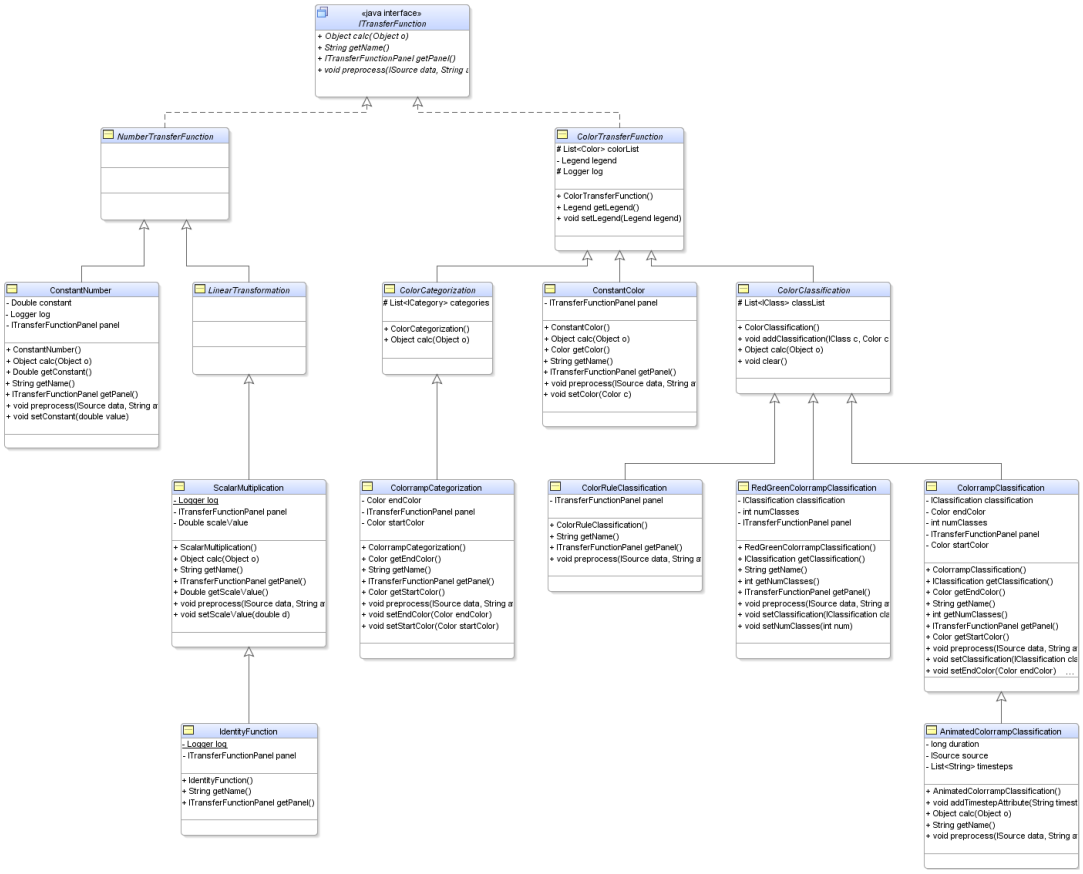

Figure 91: The `ITransferFunction` and current implementations

Finally, the user may need to classify/group the input data, e.g. values from zero to ten should be coloured red. The following section explains the usage of basic classification techniques.

4.3.4.4 Extensible Classification Techniques

As for the `ITransferFunction` we set up the SPI framework for the `IClassfication` interface. Figure 92 illustrates the UML diagram. Again, the class `ClassificationFactory` collects all implementations of the `IClassificationProvider` interface. Currently, we include the `BasicClassificationProvider`. This class contains all basic implementations of state-of-the-art classification algorithms (see Figure 93).

However, those classification algorithms only work on data that can be ordered based on certain criteria. Thus, they cannot be applied were no order relation is given (e.g. colour, street name, etc.). Of course one can use the alphabetic order but it is not always meaningful. In order to tackle this problem we introduced the mean of categorization. Here, the user defines an attribute that determines the category were a data item belongs to.

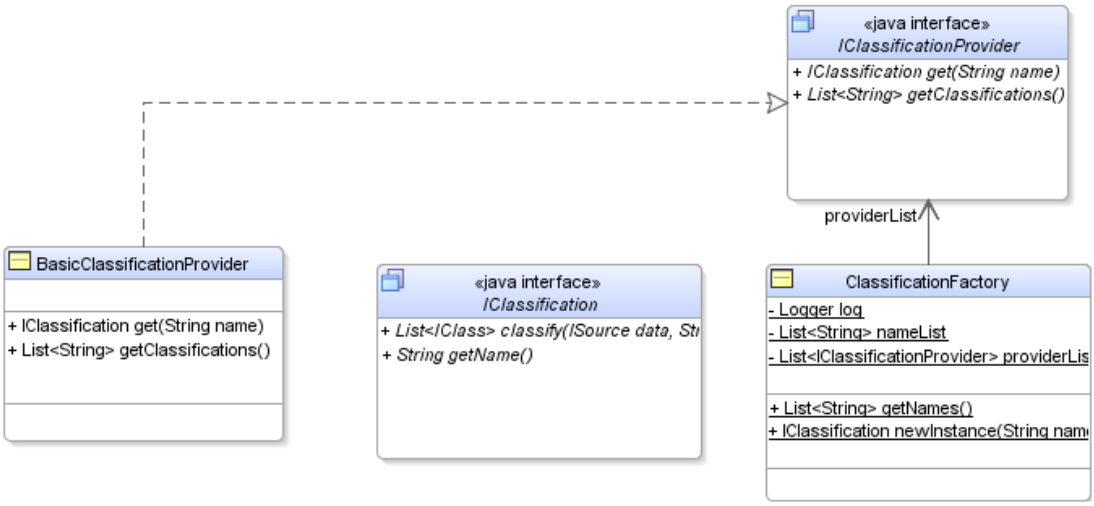

Figure 92: The SPI definition for the classification plug-in framework

Figure 93 presents the basic classification techniques that have been implemented:

- **Natural breaks** – classification that seeks to partition data into classes based on natural groups in the data
- **Quantile** – Points taken at regular intervals from the cumulative distribution function of a data attribute.
- **Standard deviation** – Showing the variation from the average value of an data attribute.
- **Equal intervals** – This method sets the value ranges in each class equal in size. The entire range of data values (*max* - *min*) is divided equally.
- **Pretty breaks** – Compute a sequence of equally spaced "nice" values which cover the range of the data values.

Besides the basic classification algorithms we included the user defined classification transfer function `ColorRuleClassification` (see Figure 91). Here, the user can define the classes and map the output (e.g. colour).

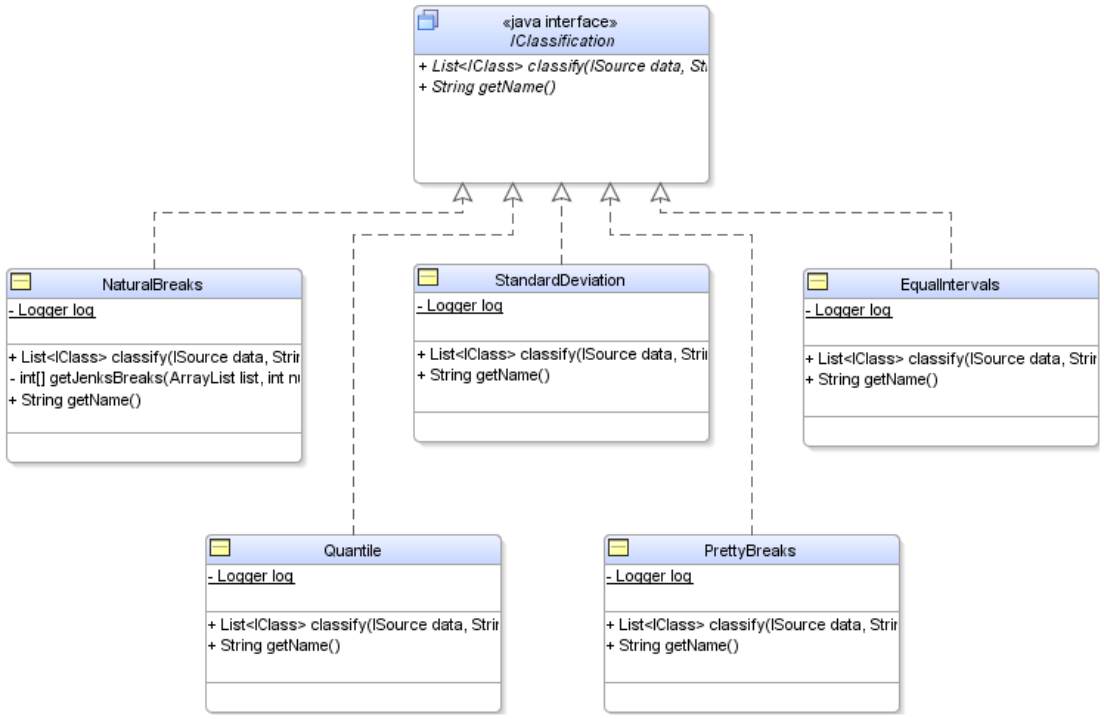

Figure 93: The `IClassification` interface and its implementations

4.3.4.5 Advanced Visualization: Isosurface Generation for 3-D Data

The Marching Cubes algorithm is a state-of-the-art visualisation technique to present isosurfaces. As described above the package provides a `MCVisualizationProvider` class which implements `IVisAlgorithmProvider` in the SUDPLAN project. One use-case is to visualize air quality results for the city of Stockholm both for different time steps and different isovalues (see Figure 94). The algorithm uses 2-D air quality measurements at different time steps and at different heights. Those "layers" are then combined as a 3-D / 4-D grid and serve as input for the "Marching Cubes" algorithm.

In order to produce an iso-surface the algorithm divides the 3D data set into smaller cubes. For each cube, each vertex is tested whether it is smaller than the predefined iso-value or not. Based on 256 possible configurations the triangulation can be set up to produce the iso-surface for the given iso-value.

Additionally, the user can animate the iso-surface. This is important in case of (a) having different iso-values allowing the user to examine how the iso-surface / volume evolves and (b) exploring one iso-value at different time steps to observe the changes of the iso-value over time. To control the animation additional control elements have been integrated to start, pause, stop the animation.

Using the VisWiz, as mentioned above, the user is able to choose transfer function and enter the iso-value/s as well as different time steps. Thus, the user can get a quick and easy understanding of a certain air quality measurement (e.g. NO_x) at a certain place by viewing surfaces and how they grow and shrink, rather than reading some values and numbers.

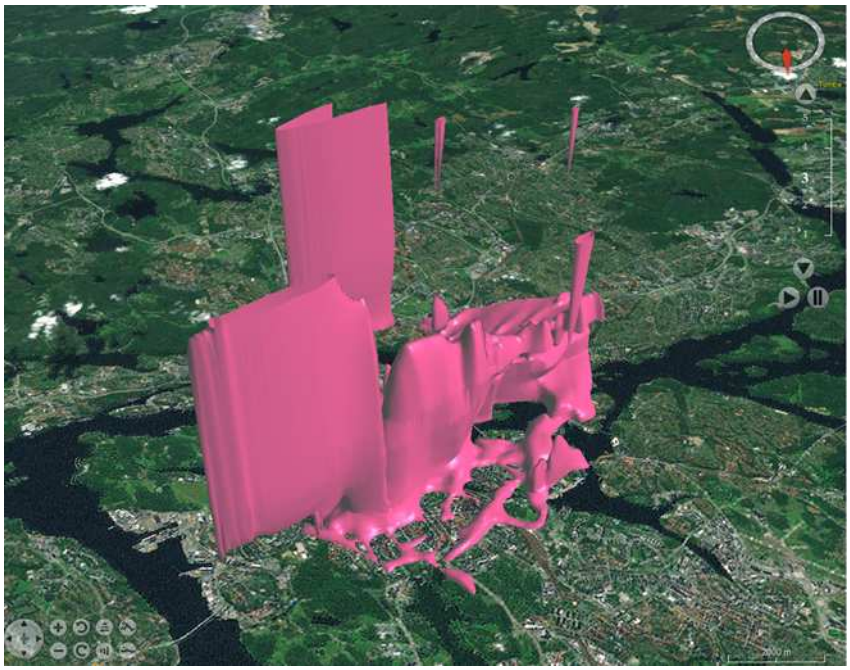

Figure 94: An iso-surface visualization produced by the `VisMarchingCubes` algorithm for the city of Stockholm.

4.3.4.6 Wuppertal Pilot Visualization

For the Wuppertal pilot we had to provide a customized solution. Here, we used again the SPI framework to easily integrate the new tailored visualization technique. The input data is a terrain triangulation of the Lüntenbeck neighborhood in the city of Wuppertal. For each triangle a series of water level results have been simulated. The simulation assumed an event of one hour heavy rain fall. The goal was to show an animation of a coloured water layer simulating the changes in water level as time passes.

In the beginning we had to tackle the problem of the size of the dataset. Furthermore, in order to interact and to investigate the data set we provide the mean to play an animation to see the flow of the water. Since the investigated area is not that flat we provided two additional modes. The user is able to (1) offset the visualization from the terrain and (2) to lift the visualization layer to a common reference plane to be able to compare the water level results. Figure 95 illustrates the visualization results.

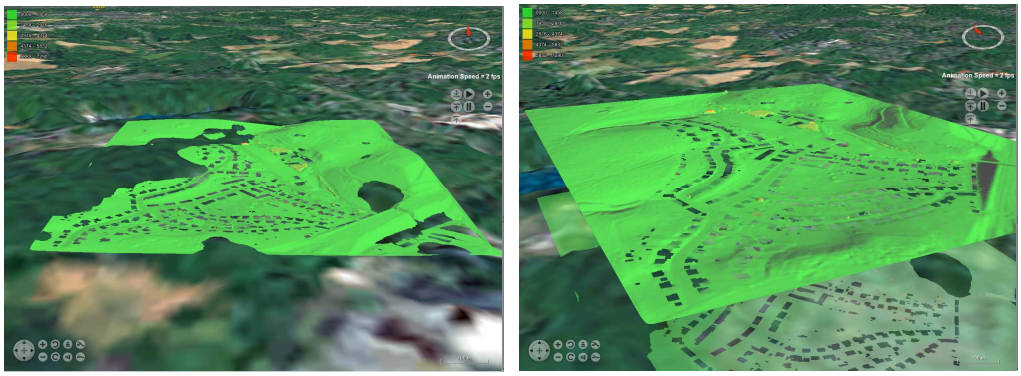

Figure 95: GeoCPM visualization of the water level simulation results for the city of Wuppertal. On the terrain (left) and lifted (right).

4.3.4.7 Stereoscopic Visualization Mode

The stereoscopic 3-D visualization was also part of the 3rd year development. The WorldWind API already provides some basic stereoscopic rendering options like anaglyph stereo. DFKI maintains a PowerWall™; a stereoscopic 3D projection system using side-by-side mode. Considering stereoscopic side-by-side projections nothing was implemented in the WorldWind API until now. To address this problem we implemented a stereoscopic side-by-side plug-in as a dual window view. By clicking a button the side-by-side stereoscopic mode is turned on, and the virtual globe can be seen in 3-D wearing polarized glasses.

4.3.5 Current Release Feature Summary

The current release of the SUDPLAN 3D advanced visualization component supports a number of features that are used, visualize and compare GIS related simulation data. In summary, the features included in the third year release of the software are:

- Implementation of Java's Plug-in architecture
- Enhanced user interface for the visualization wizard VisWiz
- Advanced parameterization of visualizations using VisWiz component
- Management of visualization techniques, results
- Time series visualization
- Support for multiple state-of-the-art classification algorithms to support visualization techniques
- Enhanced IO-Module for GeoCPM data
- Visualization of iso-surfaces using Marching Cubes algorithm
- Visualization of the simulation results from Wuppertal Pilot
- Integration of Animation for selected visualization techniques
- Integration of WMS layer at defined heights above sea level
- Ability to render the 3-D virtual globe on a back-projected stereoscopic large display

A current version of SUDPLAN 3-D advanced visualization component as stand-alone Java Webstart application can be found at http://sudplan.kl.dfki.de.

Conclusions

This document accompanies the software deliverable *D3.2.3 Product Implementation V3*. In this report we presented the results of the three development cycles of WP3 *Scenario Management System* of the SUDPLAN project. In addition, we explain the relationship of the deliverable to other tasks and deliverables, and provide a justification for the choice of technologies and basic software products used in the context of the Scenario Management System development.

In contrast to *D3.3.3 Scenario Management System V3* which will present integrated software, this document focuses on the individual building blocks of the scenario management system. The software presented here is used to implement the four pilot applications, which is an on-going activity.

The work in WP3 can be categorized according to three building blocks as:

a. **scenario management platform functionality** including model management, basic visualisation etc.

b. **model as a service integration** based on related OGC SWE standards and

c. **advanced visualisation** functionality including an interactive 3D Map component

For each building block information on the technical baseline is given. Moreover, the text includes a list of core features, the documentation of the main developments carried out as well as a brief outlook on the future development plans.

Regarding the individual building block development activities, the main achievements of the third year are:

- General enhancements of GUIs and service implementations to provide a stable and user friendly system
- The implementation of visualisation and comparison features for gridded time series
- The implementation of visualization techniques and interaction means and the general enhancement of the VisWiz.

⇨ The release of fully functional and stable versions of the Scenario Management System building blocks.

A large part of WP3 work, the concrete Common Service integration activities as well as the documentation of the Integrated Scenario Management System functionality will be subject to D3.3.3 Scenario Management System V3 (Companion Report).

References

GML, 2004 Opengis geography markup language (gml) implementation specification (February 2004), http://portal.opengeospatial.org/files/?artifact_id=4700

OM1, 2007 Opengis observations and measurements part 1 - observation schema (December 2007), http://portal.opengeospatial.org/files/?artifact_id=22466

OM2, 2007 Opengis observations and measurements part 2 - sampling features (December 2007), http://portal.opengeospatial.org/files/?artifact_id=22467

SensorML, 2007 Opengis sensor model language (sensorml) implementation specification (July 2007), http://portal.opengeospatial.org/files/?artifact_id=21273

SanyFMA, 2010 SANY fusion and modelling architecture. Discussion Paper OGC 10-001, Deliverable D3.3.2.3 of the European Integrated Project SANY FP6-IST-033564 OGC 10-001, Open Geospatial Consortium (2010), http://portal.opengeospatial.org/files/?artifact_id=37139

SOS52N, 2011 Sensor observation service (March 2011), http://52north.org/communities/sensorweb/sos/index.html

OOSTETHYS, 2011 Sensor observation service (March 2011), http://www.oostethys.org

SWE, 2007 Botts, M., Percivall, G., Reed, C., Davidson, J.: Ogc sensor web enablement: Overview and high level architecture (July 2007), http://portal.opengeospatial.org/files/?artifact_id=25562

SISE, 2009 Hrebicek, J., Pillmann, W.: Shared environmental information system and single information space in europe for the environment: Antipodes or associates? In: Hrebicek, J.,Hradec, J., Pelikn, E., Mrovsk, O., Pillmann, W., I.Holoubek, Bandholtz, T. (eds.) Proceedings of the European conference of the Czech Presidency of the Council of the European Union, TOWARDS eENVIRONMENT. pp. 447–458. Masaryk University, Prague, (03 2009), http://www.epractice.eu/en/library/289287

ORCHESTRA, 2008 Klopfer, M., Kanellopoulos, I. (eds.): orchestra - an open source architecture for risk management. In: The ORCHESTRA Consortium (2008), http://www.eu-orchestra.org/, © 2008 by the ORCHESTRA Consortium

SANY, 2009 Klopfer, M., Simonis, I. (eds.): SANY - an open service architecture for sensor networks. SANY Consortium (2009), http://www.sany-ip.eu/publications/3317

UncertML, 2007 Williams, M., Cornford, D., Bastin, L., Pebesma, E.: Uncertainty markup language, UncertML (2007), http://www.intamap.org/uncertml/uncertml.php

Booch, 2005 Grady Booch et al.: The Unified Modeling Language User Guide, 2005 , Addision Wesley Longman

Gamma, 2005 Erich Gamma et al. : Design Patterns: Elements of Reusable Object-Oriented Software 2005 Addision Wesley

Meyer, 1992 Meyer, Bertrand: Applying "Design by Contract", in Computer (IEEE), 1992, pp. 40–51

Sun, 2000 Sun Microsystems Java 3D Engineering Team: Java 3D API Tutorial, 2000

Oracle, 2011 Oracle, 2011 Oracle: Java 3D http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138252.htm, 2011

Annex 1: Glossary

Climate scenario	*Climate scenarios* means the resulting climate evolution over time, as simulated by global (GCMs) and regional (RCMs) climate models. Climate scenarios are products of certain emission scenarios that reflect different economic growth and emission mitigation agreements.
Common Services	*Common Services* are the climate downscaling services for rainfall, river flooding and air quality, developed in the SUDPLAN project and accessed through the SUDPLAN platform (Scenario Management System)
Emission scenario	European cities will also handle different local *emission scenarios* (to the atmosphere) that to a large extent influence future air quality, but with little influence on global climate.
Information product	Raw data, such as the results of mathematical modelling, and the analysis thereof, will often need to be packaged in such a way as to be accessible to the various stakeholders of an analysis. The medium can be one of a wide variety, such as print, photo, video, slides, or web pages. The term *information product* refers to such an entity.
Model	A *model* is a simplified representation of a system, usually intended to facilitate analysis of the system through manipulation of the model. In the SUDPLAN context the term can be used to refer to mathematical models of processes or spatial models of geographical entities.
Profile	Within SUDPLAN a *profile* is a set of configuration parameters which are associated with an individual or group, and which are remembered in order to facilitate repeated use of the system.
Report	A *report* is a particular type of information product which is usually static and might integrate still images, static data representations, mathematical expressions, and narrative to communicate an analytical result to others.
Scenario	A *scenario* is a set of parameters, variables and other conditions which represent a hypothetical situation, and which can be analysed through the use of models in order to produce hypothetical outcomes.

Scenario Management System	*Scenario Management System* (SMS) is synonymous with SUDPLAN platform
Scenario Management System Framework	The *Scenario Management System Framework* is the main Building Block of the Scenario Management System. It provides the Scenario Management System core functionalities and integration support for the other Building Blocks.
Scenario Management System Building Block	Scenario Management System Framework is composed of three distinct *Building Blocks*: The Scenario Management System Framework, the Model as a Service Building Block and the Advanced Visualisation Building Block.
SUDPLAN application	A *SUDPLAN application* is a decision support system crafted by using the SUDPLAN platform and integrating models, data, sensors, and other services to meet the requirements of the particular application.
SUDPLAN platform	The *SUDPLAN platform* is an ensemble of software components which support the development of SUDPLAN applications.
SUDPLAN system	*SUDPLAN system* is synonymous with SUDPLAN application
User	The term *user* refers to people who have a more or less direct involvement with a system. Primary users are directly and frequently involved, while secondary users may interact with the system only occasionally or through an intermediary. Tertiary users may not interact with the system but have a direct interest in the performance of the system.
Web-based	Computer applications are said to be *web-based* if they rely on or take advantage of data and/or services which are accessible via the World Wide Web using the Internet.