Intelligent Tools for Policy Design

Deliverable 4.2 –FUPOL Simulator Software Interfaces Specification

Project Reference No.	287119
Deliverable No.	D 4.2
Relevant workpackage:	WP 4
Nature:	Report
Dissemination Level:	Public
Document version:	Final
Editor(s):	Egils Ginters
Contributors:	Susanne Sonntagbauer, Egils Ginters, Artis Aizstrauts, Girts Dreija, Miquel Angel Piera Eroles, Mujica-Mota Miguel-Antonio, Roman Buil, Nikolaus Rumm, Sergey Stepucev, Kawa Nazemi, Dirk Burkhardt
Document description:	The document comprises prerequisites and recommendations related to elaboration of the interfaces between FUPOL Simulator and main parts of the FUPOL software platform. In other words, there are shown the ways how FUPOL Simulator receives input data/policy use case models and how the simulation results are made accessible for other services. Interoperability of FUPOL Simulator with FUPOL Core platform and other services are described.

History

Version	Date	Reason	**Prepared / Revised by**
0.1	08-09-2012	Initial version	Egils Ginters
0.9	17-12-2012	Fupol Simulator concept described. Place of the Fupol Simulator software desing (WP4) in common WPs set is determined. Web services use is specified. Access to spatial data and database PostGIS, as well collaboration with simuation desktop are described beased on generic LUCC use case.	Artis Aizstrauts, Egils Ginters
0.96	20-12-2012	Generally finished draft. WP2, WP3, WP5 adds are expected.	Artis Aizstrauts, Egils Ginters
3.0	27-01-2013	Requirements to ESB API are defined.	Artis Aizstrauts, Egils Ginters, Nikolaus Rumm
4.0.	29-01-2013	Visualisation part is added.	Egils Ginters, Kawa Nazemi, Dirk Burkhardt
5.0	01-02-2013	Collaboration and logical interface with WP2. Easy Communication Environment description added. References to deliverables.	Miquel Angel Piera Eroles, Mujica-Mota Miguel-Antonio, Roman Buil, Egils Ginters, Artis Aizstrauts

Table of Content

1 LIST OF ABBREVIATIONS 5

2 INTRODUCTION 6

2.1 Purpose of the Document 6

2.2 Target group 9

2.3 Benefits 9

2.4 Structure of the document 9

3 FUPOL SIMULATOR ROLE IN COLLABORATION CONCEPT 9

4 FUPOL POLICY MODELLING AND SIMULATION PARTS COLLABORATION (WP2) 11

4.1 Policy domains use cases 11

4.1.1 Community Facilities 13

4.1.1.1 Area Design 13

4.1.1.2 Open Space 14

4.1.2 Urban Segregation 14

4.1.3 Sustainable Tourism 14

4.1.4 Land Use 15

4.1.4.1 Land Use Cover Change (LUCC) 15

4.1.4.2 Edgeland Industrialization 16

4.1.5 Urban Economics 16

4.2 Use case models specification and verification 17

4.3 The models transfer to FUPOL Simulator 18

5 CONCEPTUAL MODEL OF FUPOL SIMULATOR SOFTWARE ARCHITECTURE 22

6 THE PRELIMINARY DESCRIPTION OF FUPOL SIMULATOR INTERFACES29

6.1 Structural model of FUPOL Simulator architecture 29

6.2 Domain use case software architecture 33
6.2.1 Structural model.....33
6.2.2 Domain use case Web service 34
6.2.2.1 Function "startSimulation".....42
6.2.2.2 Function "getLandUseTypes".....45
6.2.2.3 Function "getLandUseDistribution".....48
6.2.2.4 Model utility library.....52
6.2.3 Domain use case simulation model 53
6.2.4 Collaboration with Geoserver 54
6.2.5 Access to PostgreSQL (PostGIS) database.....56
6.2.6 Domain use case Simulation Initialization and Control block (GUI).....57
6.2.6.1 Web page (GUI).....58
6.2.6.2 Java Servlet (SOAP Client) 61
6.3 ESB API – FUPOL Simulator interface of data import/export and interoperability.....62
6.3.1 User authentication 62
6.3.2 Geoserver functionality 63
6.3.3 Database functionality 64
6.4 FUPOL Simulator access to Visualisation services (WP5) 64
6.4.1 Interfacing to SemaVis (WP5) 65
6.4.2 Visualization of Statistical Data 66
7 CONCLUSIONS 74
8 REFERENCES 75

1 List of Abbreviations

ABM	Agent-based modelling
API	Application programming interface
ASCII	American Standard Code for Information Interchange
BMP	Bitmap image file
C/C++	Programming language
CAD	Computer-aided design
CPN	Coloured Petri Networks
CSV	File extension: Comma-separated values
DEVS	Discrete event systems
DGN	CAD file format supported by Bentley Systems Microstation and Intergraph's Interactive Graphics Design System
ECE	Easy Communication Environment
ESB	Enterprise service bus
FCM	Fuzzy cognitive maps
FP7	The Seventh Framework Programme for Research and Technological Development.
FUPOL	Future policy modeling. Research project for developing advanced information and communication technology tools to support policy design and implementation. FUPOL aims at a completely new approach to traditional politics.
GIF	Graphics Interchange Format
GIS	Geographic Information System
GUI	Graphical user interface
HTTP	Hypertext Transfer Protocol Secure
JAR	Java archive file format
JAVA	Programming language
JPG/JPEG	Joint Photographic Experts Group (image file format)
LUCC	Land use and cover change
MAS	Multi Agent System

MySQL	Open-source relational database management system
ODBC	Open database connectivity (standard C programming language interface for accessing database management systems
OGR	Vector graphics file format
OSM	OpenStreetMap (a collaborative project to create a free editable map of the world)
OSS	Open Source Software
PDF	Portable Document Format
PNG	Portable Network Graphics (a bitmap image file format)
PostgreSQL	Open-source object relational database management system
Python	Programming language
SD	Systems dynamic simulation
SDK	Software Development Kit
SDMX	Statistical data format
SVG	Scalable Vector Graphics
TIFF	Tagged Image File Format
WFS	Web Feature Service (vector format)
WMS	Web Map Service (raster format)
XML	Extensible Markup Language (markup language that defines a set of rules for encoding documents)

2 Introduction

2.1 Purpose of the Document

Policy modelling as the object of research is versatile and complex. Policy models depending on the domain could be described as discrete or continuous, and determined or stochastic systems. For simulation of the above-mentioned models different simulation tools could be used. Simulation of scenarios can ask for collaboration of some separate simulation models joining for implementation of the task or policy domain use case. Therefore distribution and multi-level simulation can

be realised. During generic and specific simulation software review and based on recommendations of D4.1 as main designing tool generic agent-based simulation package Repast Symphony for FUPOL Simulator software was selected. However if for long time changes forecasting other simulation approach will be necessary then on the macro-level Repast Symphony will be used for system dynamics simulation. Other architecture ones of FUPOL Simulator will be designed in Java, AJAX and php software environments. FUPOL Simulator must be implemented as OSS tool. The FUPOL Simulator (WP4) will be a part of the FUPOL software joining with other subsystems through FUPOL Core platform (WP3), therefore must be defined communication rules and interfaces with other parts via Enteprise Service Bus (ESB) API and direct way if necessary. In other words, there are must be shown the ways how FUPOL Simulator receives input data/policy use case models and how the simulation results are made accessible for other services. Interoperability of FUPOL Simulator with FUPOL Core platform and other services must be described. The objective of this document is to elaborate the requirements for the interfaces (logical and physical) to communicate with other parts of the FUPOL software.

Deliverable D4.1 and D4.2 has prescribed dependencies among deliverables of other work packages (see Figure 2.1). In Figure 2.1 the area of D4.2 interests are marked by crossed rectangles.

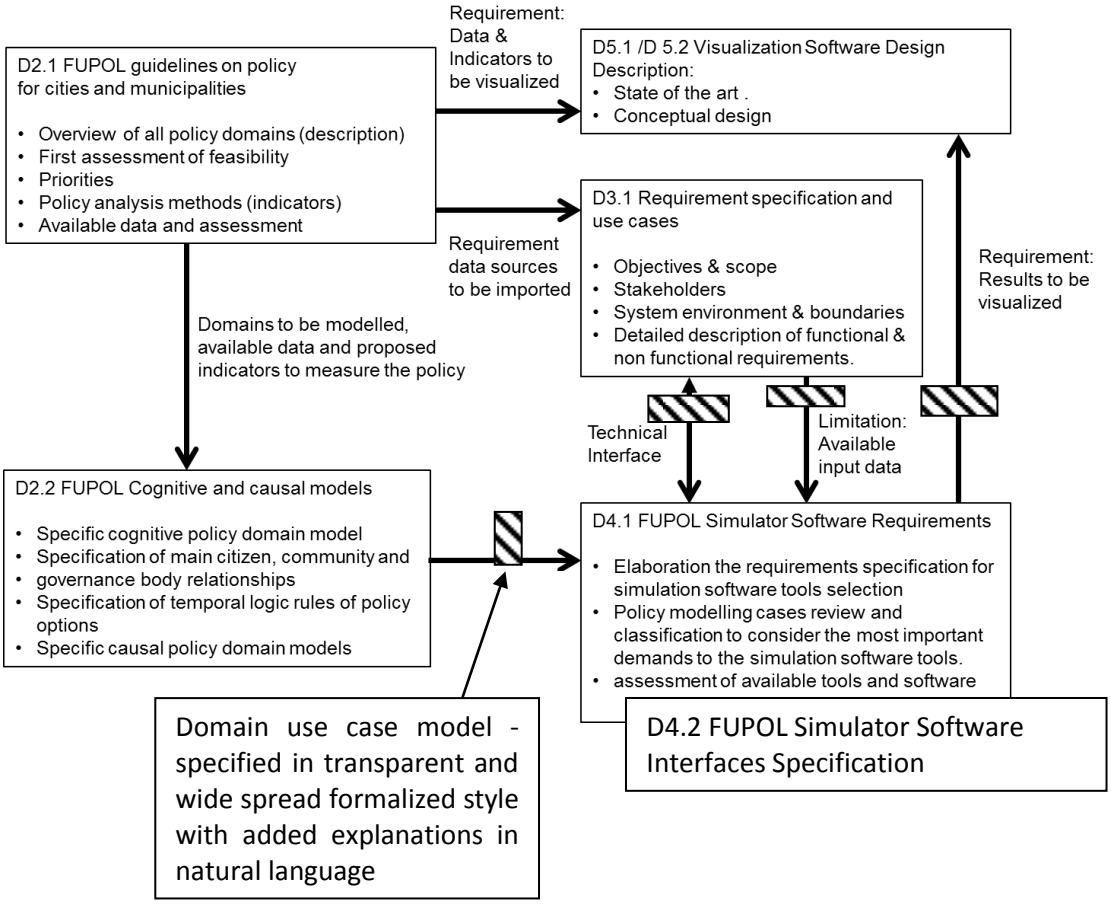


Figure 2.1: Dependencies among deliverables (see D2.1)

Deliverables D2.1 and D2.2 are one of main sources of policy domains uses cases set requirements.

Deliverables D3.1 and D3.2 are the information sources for understanding intended FUPOL Core platform architecture and FUPOL Simulator place in this architecture.

Deliverable D4.1 discuss the questions related with FUPOL Simulator software designing and it does not deal with integration of different tools or methods or modeling technologies (software) (Colored Petri nets, Fuzzy Cognitive Maps, etc.) used in WP2 for policy domain use cases models designing, specification and verification only.

Deliverable D4.2 defines what functionality is expected from ESB API by FUPOL Simulator to insure its operations. This deliverable is source a document for WP3 to create required ESB functionality.

In further parts of the document collaboration among FUPOL Simulator (WP4) and other FUPOL technical packages will be described more detailed.

2.2 Target group

Target group is FUPOL Policy modelling uses cases (WP2) authors, FUPOL Simulator software (WP4) and FUPOL Core platform (WP3) designers, and elaborators of FUPOL Visualisation tools (WP5).

2.3 Benefits

This deliverable contributes to a better understanding of features of FUPOL Simulator (WP4) software: functionality and architecture. The deliverable gives insight into FUPOL simulator software architecture and FUPOL Simulator collaboration with other FUPOL software parts. The report is the base for the FUPOL Simulator interfaces implementation and ESB API construction.

2.4 Structure of the document

The document consists of the following main parts. The document consists of the following main parts. The first part reveals FUPOL Simulator role in FUPOL WPs collaboration. The second part describes logical interface with WP2 and specify the form in which policy use case models are transferred to WP4 for software designing and simulation. The third part gives insight to FUPOL Simulator architecture concept, but forth part deals with collaboration with FUPOL Core platform and other parts of FUPOL software.

3 FUPOL Simulator Role in Collaboration Concept

FUPOL policy modelling platform consists of some different, however joined workpackages which collaboration is aimed to creating new knowledge and providing new possibilities to policy domain specialists related with forecasting of potential

decision making results through adding modern supporting tools as semantic search, simulation and visualisation.

Interoperability of WP4 and other FUPOL workpackages related with FUPOL Simulator designing and running is shown in Fig.2.2.

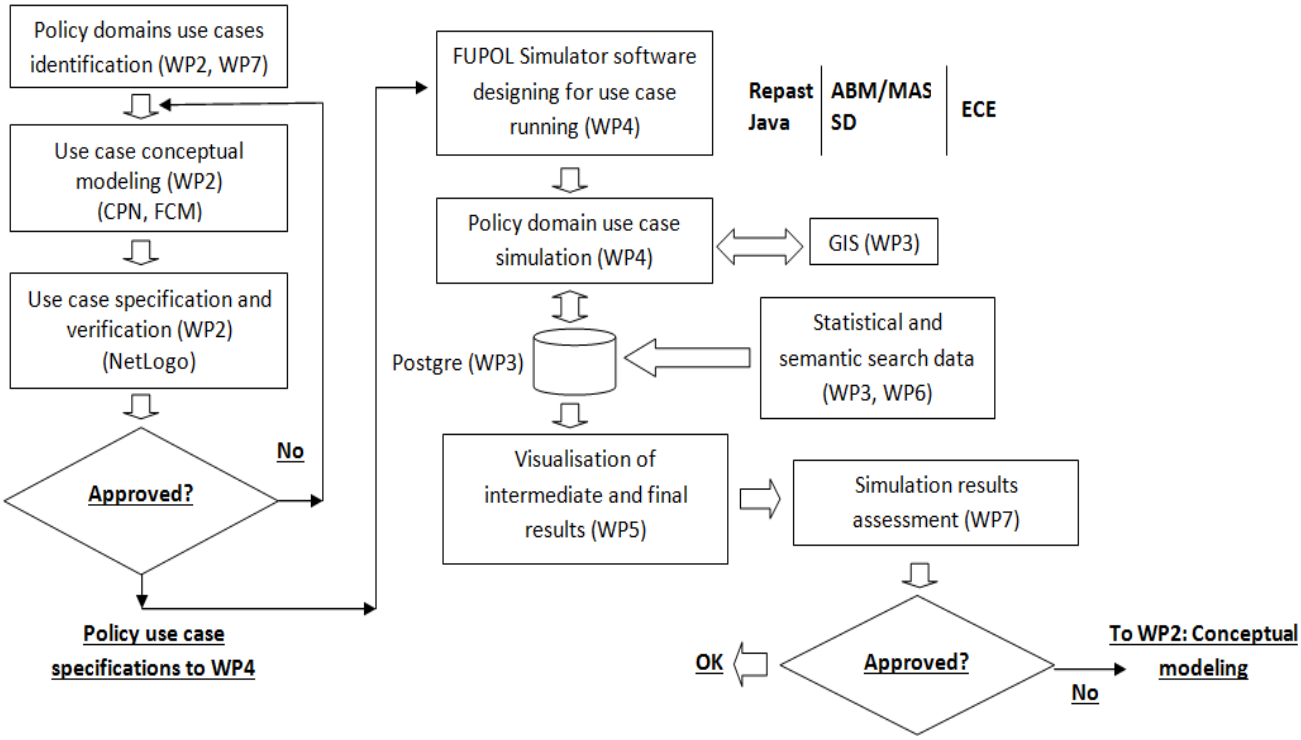


Figure 2.2: WP4 interoperability with other FUPOL workpackages related to FUPOL Simulator designing and running

Policy use case models identification is carried out together with beneficiaries. WP2 is responsible for policy use case model designing and verification, what is done using Coloured Petri Networks (CPN) and Fuzzy Cognitive Maps (FCM). For some critical contours verification ABM/MAS simulation tool NetLogo is used. Verified policy model use case specification in transparable, understandable and formalized form with explanations in natural language are transferred to WP4 for simulation software designing and running on FUPOL Simulator. FUPOL Simulator are implemented as two level simulation environment using generic simulation tool Repast Symphony

which is used for ABM/MAS and SD simulation as optional. If distributed simulation must be ensured then Easy Communication Environment (ECE) is used for simulation models communication. Data exchange is ensured by Postgre database. During simulation process access to GIS GeoServer and Postgre database are provided by FUPOL Core platform (WP3) through ESB API. Input data for simulation model have been provided by WP3 and WP6 through Postgre database. Mostly SDMX format is used for data store. Advanced visualization of the simulation results is ensured by WP5 using the access to Postgre database through ESB API. Use case policy models and results validation have been done by WP7 based on simulation results assessment. If the results are not satisfactory then conceptual model must be changed (WP2).

4 FUPOL Policy Modelling and Simulation Parts Collaboration (WP2)

4.1 Policy domains use cases

FUPOL aims at a new approach to traditional politics. Major innovations like multichannel social computing and crowd sourcing will change the way politicians communicate with citizens and enterprises and make decisions. The system will be able to automatically collect, analyze and interpret opinions expressed on a large scale from the Internet. This will enable governments to gain a better understanding of the needs of citizens. Likewise the software will have the capabilities to simulate the effects of policies and laws on different domains and to assist governments in the whole policy design process (http://www.fupol.eu).

Due to the European city and urban planning strategies the domains specified below could be simulated in the FUPOL project. The reviewed political areas are:

- Urban Planning;
- Land Use;

- Economy;
- Housing
- Shopping and Services;
- Community Facilities;
- Infrastructure;
- Waste Management;
- Transport and Movement;
- Consumer Protection;
- Urban Segregation;
- Consumer Production;
- Migration;
- Active Inclusion;
- Safety and Security;
- Health (care);
- Social Affairs;
- Education;
- Research and Innovation;
- City Treasury.

Due to the European city and urban planning strategies more than 20 domains were identified. D2.1 covered those having significant simulation/participation possibilities relevant to the FUPOL project. Finally, the main domains to be developed as demonstrators were selected, and they are:

- Community Facilities,
    - Area Design,
    - Open Space;
- Urban Segregation;
- Sustanaible Tourism;
- Land Use,
    - Land Use Cover Change (LUCC),
    - Edgeland industrialization;
- Urban Economics.

4.1.1 Community Facilities

Many factors influence the design of the public domain, including the underlying geography and historical pattern of development. In successful urban areas, the public domain provides access to the benefits of the city to all people. Urban policies should ensure the public domain is continuous and that infrastructure, universal access and circulation using public transport is equitably distributed. An urban policy model for e-participation should consider also the requirements to provide the setting for public events and the everyday life of cities and towns focussing on achieving high levels of public amenity and flexibility, among other aspects.

In FUPOL it is envisaged to specify the different drivers that could affect the succeess or the failure in deploying the investment of a certain reduced amount of typical facilities for public domain. Models will be formulated in such a way, that citizens and future end-users could adapt to other facilities of public interest.

4.1.1.1 Area Design

Area design domain is considered in order to specify the proper configuration of an area considering its functionalities according to citizens' context.

In the particular case of the Zagreb Municipality, they will use simulation to design a green park area near a new Autism Centre, for integration purposes. The simulation objective is to provide the best solution for the facilities that would be included in the 2000m^2 of green area situated near the Autism Centre. The design must satisfy most of the potential user demands and must encourage interactions between autist people and non-autist users, while avoiding possible conflicts between them. At the same time, possible conflicts between all kinds of users must be avoided.

4.1.1.2 Open Space

Open space domain is considered in order to determine which facility is most convenient in a certain spot of the city. The simulation model considers all the facilities around this spot, distances, and citizens' needs depending on their age, economic level, work, family status, etc. The result of the simulation is a certain percentage of acceptances of each facility considered to place in the studied spot.
The Municipality of Prato is interested in this domain and it is also the second interest for the Municipality of Zagreb in order to determine facilities to build around the Autism Centre to satisfy the citizens of the neighbourhood that are wooried about having an Autism Centre nearby.

4.1.2 Urban Segregation

Urban segregation is a concept used to indicate the separation between different social groups in an urban environment, which is a barrier for achieving social inclusion in cities. It occurs in various degrees in most large modern cities, including the developed and the developing world. Location is a key issue in many situations of urban segregation. For example, racial and ethnic ghettos are a persistent feature of most large cities. In some countries high-income families concentrate in areas that expand from the historical centre into a single geographical direction, whereas the poorest families mostly settle in the roughly equipped far peripheries.
The Municipality of Prato have a lot of immigration from diferents countries, and they start having some problems in some neighbourhoods. They plan to use simulation to find some solution to avoid conflicts.

4.1.3 Sustainable Tourism

Tourism itself has become a popular global leisure activity with around one billion tourists registered per year. In recent years sustainable tourism has been developed. It can be seen as having regard to ecological and socio-cultural carrying capacities and includes involving the community of the destination in tourism development

planning. It also involves integrating tourism to match current economic and growth policies so as to mitigate some of the negative economic and social impacts of mass tourism. Sustainable tourism is different to ecotourism.

New simulation models are required to explore how tourism in a region can grow in such a manner that the limit to the growth of tourism is not the adverse effects of the growth itself, but something of the region's choosing. Understanding the forces that shape the future of tourism in a holistic manner is essential for sustainably managing tourism development.

Municipality of Pegeia needs to apply this kind of simulation.

4.1.4 Land Use

Land use planning is required by governments to manage the development of land within their jurisdictions. In doing so, the governmental unit can plan for the needs of the community while safeguarding natural resources. To this end, it is the systematic assessment of land and water potential, alternatives for land use, and economic and social conditions in order to select and adopt the best land-use options. A land use plan provides a vision for the future possibilities of development in neighbourhoods, districts, cities, or any defined planning area.

The goal of land use planning is to further the welfare of people and their communities by creating convenient, equitable, healthful, efficient, and attractive environments for present and future generations.

4.1.4.1 Land Use Cover Change (LUCC)

Land use cover change is a complex matter, which is caused by numerous biophysical, socio-economical and economic factors. An obvious form of land use change in the suburbs of the metropolis is defined as urban spraw. The particular significance of this problem is the fact that humans are the main actors in the transformation of the environment, and impact upon the suburbs due to their settlement preferences and lifestyle choices.

This domain is selected as a generic domain to start working on the FUPOL Simulator software development by using well accepted models reported in the literature.

4.1.4.2 Edgeland Industrialization

Some 75 % of the population of the Central Europe region lives in urban areas, a figure which is likely to continue to grow. Towns and cities are perceived as engines of regional development, offering a broad range of services and economic and cultural opportunities.

Urban development is not only about planning buildings and activities, but also about creating industrial areas having a positive impact on their surroundings. The design of high quality urban spaces, involving inputs from community groups, is also an increasingly important aspect of the planning process. Such places help to define the public life of a village or town by strengthening the "local spirit".

Regarding industrialisation, the Municilapity of Barnsley plan to use simulation in order to know the effects of industrialization on land use, economics, etc.

4.1.5 Urban Economics

By definition economy consists of the economic system of a country or other area (city, region). This includes labour, capital and land resources, the manufacturing, trade, distribution and consumption of goods and services of that area.

In the FUPOL project not the whole economy should be simulated, which would exceed the scope of the project. It is advisable to select a sector, such as banking, manufacturing, or tourism, which can be linked to land use or participative budgeting etc. The model will focus on the main economic drivers for a certain domain (i.e. manufacturing in urban areas), in which the competitive advantages with respect the urban shortages will be highlighted by means of simulation. Citizens and end users will be able to deal with a trade-off evaluation between the penalties (noise, street

bottlenecks, etc.) with respect to the benefits (i.e. expenses of the workers in the neighbourhood commerce, security, etc.).
Yantai municipality needs to face the problem of high energy consumption in the city. They need to confront the problem from an economic prespective because their main factor to reduce the energy consumption is the renovation of the industrial sector.

The task of the FUPOL Simulator (WP4) is to imitate the policy use case models (WP2) of the domains defined.

4.2 Use case models specification and verification

The formal specification of FCM and Behavioural rules of the different policy user cases into Coloured Petri Nets will allow the use of state space analysis tools to compute the different reachable states that could be obtained by a proper combination of firing the different transitions described in the CPN model. Thus, the reachability tree provides a first verification proof that at local level, an agent with a behaviour specified by the CPN model would act in a deterministic way.
The 'Sense – Think – Act' method of operation can show some limitations when applied to modelling cognitive behaviour, and an alternative approach embracing embodied, situated cognition can also be implemented if needed. However, a question remains concerning how to implement such an approach since it effectively entails sensing, thinking and acting all occurring at the same time i.e. concurrently, rather than sequentially. This problem will be analyzed and solved in WP2 through the CPN state space analysis tools, since CPN formalism provides formal qualitative and quantitative analysis of concurrent processes. In case the policy user case would require a concurrent MAS approach, and the concurrency state space analysis would reveal sensitivity in the results, WP2 will implement (in Repast Symphony or NetLogo) the right control procedures in a flow control Agent for the aims of verification most critical contrours of policy use case models.

Despite the formal results that could be obtained by the reachability analysis, it is important to evaluate if the interaction between the different agents designed to represent the main actors in each policy user case, generates the effects predicted during the modelling abstraction process. For this verification, the causal models will be implemented as Agents in Repast and the different agent's interactions will be verified at a small scale.
However simulation model can be heterogeneous and be distributed among different platforms, i.e. MAS and SD. Then for SD specification algebraic equetions must be used.

The task of the WP2 is to determine in which form that is transparable enough for beneficiaries and software designing specialists as well the earlier verified policy use case models will be transferred to WP4 for simulation software designing and imitation of FUPOL Simulator. The aim of this document is to define the concept for the models transfer.

4.3 The models transfer to FUPOL Simulator

In this subsection it is specified how the models developed in WP2 will be described to support different simulation exercises within WP4, in such a way that policy use case model elaborated from informal and fuzzy statements, once verified in WP2 will be described under a formal specification that is understood and agreed by WP4 for simulation software designing and running purposes.
The proposed model specification must preserve certain requirements such as completeness (i.e. all relevant knowledge is defined), correctness (i.e. the knowledge defined is true for the domain), unambiguous expression and traceability in order to preserve the model verification task implemented in WP2 while avoiding the full work (i.e. reinventing the wheel) of a new model language specification that would be required using an informal approach based on a natural language. However partial explanations of model specifications will be added. Note that informal knowledge is

often incomplete, ambiguous and inconsistent; hence there is a need to preserve expressive freedoms in requirements, i.e. varying degrees of formality.

Despite the modelling task in WP2 is based on the abstraction process of defining the main dynamics that takes place in the different policy domains at a generic level, that will be particularized in 5 pilot use cases by means of a casual modelling approach, in WP2 it will be necessary to codify the causal models in a Multi Agent System platform for verification purposes in order to ensure that the model implementation and its associated data accurately represent the policy user case conceptual description and specifications: what the model or simulation will represent, the assumptions limiting those representations, and other capabilities needed to satisfy the policy use case simulation in FUPOL. Due to some shortages of NetLogo, in WP2 to satisfy the verification needs in WP2, models will be codified in Repast Symphony 2.0. Algorithms specifying system dynamics in two-llevel use case model (optional) will be described by algebraic equations with comments in natural language.

Agent Specification

The behaviour of each agent will be specified explicitly by the Repast commands with a set of comments illustrating at conceptual level the agent behaviour. Agent behaviour will be codified from a structural point of view by means of procedures each one with a series of commands.

Agent Interaction Specification

Agent specification is insufficient to describe the behaviour of the agents. Its behaviour is represented by the actions the agent performs which results in some change to its own state, to the state of other agents or to the state of the environment. The type of change that occurs represents the outcome of the behaviour.

In D2.2 it is well described the affinity and proximity interactions required in urban policy modelling, which will be codified also in WP2 as procedures in Repast

Symphony and properly commented in informal natural language. This approach ensures the traceability of the cause and effect for transparency purposes when analyzing simulation results.

Time Specification

The specification of a role-management protocol will be particular to each urban policy domain.

Time specification of laws that allow agents to achieve their goals at run-time will be formalized also in Repast Simphony and properly commented in informal natural language. This approach ensures the traceability of the cause and effect for transparency purposes when analyzing simulation results.

Flow Model Specification

A general and well accepted approach to manage the flow of interaction between Agents is to empower each agent to decide the right interactions to deal with at any time. This approach is well supported by a procedural flowchart in which at every time advancement of the simulation, or \tick, each Agent will execute his own step method.

Howerver, some policy user case may need an Observer Agent to implement a more complex policy-related logic, monitoring, and control, contained in the knowledge base policy objects that must be continuously performed for individual simulation targets.

To define the overall flow of each policy user case model, WP2 will provide the Observer Agent implemented in Repast Simphony together with a flowchart specification of the information flow and Agent interaction. The Flowchart provided by WP2 will plays an engine role by:

i) Integrating different information flows (such as citizens characteristics, biophysical environment and policy information) from the different Agents into the decision-making
ii) Scheduling the agent's decision-making processes and subsequent actions
iii) Governing the behaviour of the different types of agents.

Thus Observer Agent developed in WP2 will have mainly the following responsibilities:

- Monitor specific parameters regarding the evolution of the indicators described in each particular policy user case.
- Communicate information to other agents regarding their status.

Finally, the verified policy use case model will be specified in following form and transferred from WP2 to WP4 for simulation software designing and running on FUPOL Simulator:

- Agents, Agents Interaction and Time specifications will be specified in Repast Symphony with comments in natural language;
- Flow Model will be specificified by means of an Observer agent codified in Repast Symphony together with flowcharts comments;
- System dynamics models (optional) will be specified using algebraic differential equations with comments in natural language;
- Interoperability and data exchange in distributed policy use case model (two-level) will be described in natural language;
- Drivers, boundary conditions (exogenous variables) and values, and other quantitative parameters will be specified in tables form with comments in natural language;
- Input data sources and links to them must be defined;
- Other important notes related use case model in natural language.

FUPOL Simulator interoperability will be explained further based on LUCC use case defined as one of potential policy domains in D2.1 and D4.1.

5 Conceptual Model of FUPOL Simulator Software Architecture

The architecture of the FUPOL Simulator is responsible for implementing the simulation of policy use case models elaborated in WP2 and must ensure collaboration with other FUPOL software especially FUPOL Core platform. D4.1 and D4.2 are not related to other software use or integration designed and used under the framework of other workpackages aimed to semantic search, visualisation and designing, and verification of policy domain use case models (CPN, FCM etc.).

The simulation under the FUPOL framework will be implemented at two levels: micro and macro simulation levels. In the micro level the agent-based simulation (ABM/MAS) operations are related to versatile and small basic components interaction and forecasting of the interaction results will be carried out. The FUPOL Simulator will be designed as multilevel and even more as system for implementing distributed simulation models. At the macro level (if it is necessary) the system dynamics (SD) simulation will be realised as optional using the same generic simulation software Repast Symphony. The micro level ABM/MAS simulation will be the data source for the macro SD simulation model (if necessary), for example, in economic development models.

The FUPOL Simulator software will consist of following functional parts (see Figure 5.1):

- Simulation Initialization and Control block (GUI);
- Multilevel and Distributed Simulation Models Management block.

In the FUPOL project at least two kinds of visualisation of data must be distinguished:

- Visualisation of the modeling results (WP5);
- Visualisation of the simulation desktop, intermediate and final results (Simulation Initialization and Control block (GUI)) (WP4).

Simulation Initialization and Control block (GUI) is responsible for:

- Restrictions and limitations (red lines, red polygons) – interaction with GIS and FUPOL GIS DB;

- Initial Simulation Window Marking, Zooming and Selection (rectangle) – interaction with GIS and FUPOL GIS DB;
- Data Input (CORINE, INSPIRE, EUROSTAT, User Data) – interaction with FUPOL GIS DB;
- Data Output (CORINE, INSPIRE, User Data) – interaction with FUPOL GIS;
- Data Visualisation (map, tables, graphs, set of pictures, movie) – interaction with FUPOL GIS DB and GIS;
- Launching (Start, Stop, Back) and Timing (time periods);
- Interaction with FUPOL Core platform (authentication etc.).

Multilevel and Distributed Simulation Models Management is responsible for:

- Simulation Domain Selection:
  - Simulation Platform (environment where policy domain use case model is simulated) selection – interaction with ABM/MAS or SD platforms;
  - Simulation Algorithm Calibration (Probability, Distance or Surrounding, Switching on-off) – interaction with simulation platform;
  - Drivers and Categories (selection, creation, weighting) – interaction with simulation platform.
- Simulation Models Repository Management – interaction with simulation models base or repository;
- Simulation Models Connection, Routing and Multi-level management – interaction with simulation models base or repository, simulation platforms and ECE;

- Easy Communication Environment (ECE) Configuration - interaction with simulation models base or repository.

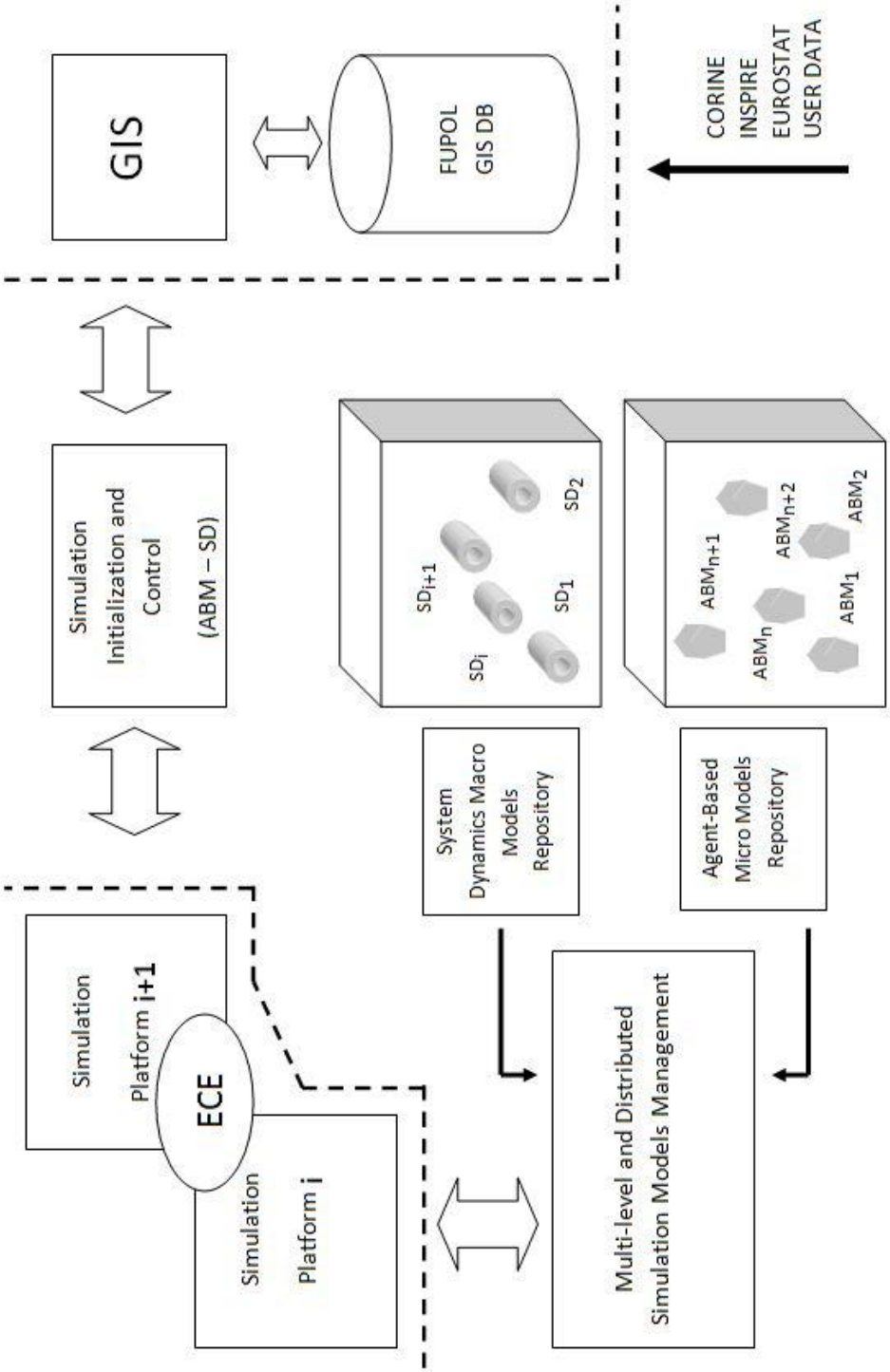


Figure 5.1: FUPOL Simulator preliminary architecture

Agent-Based Micro Models Repository comprises ABM/MAS models set oriented to FUPOL policy domain use cases models simulation. It is possible, that one scenario requests some separate models interaction and micro simulation results must be sent to the macro level for system dynamics simulation (for example, economics). The functions mentioned above are realized by the Multi-level and Distributed Simulation Models Management block. The block is responsible for attaching the predefined simulation platform, models deployment, calibration and initialization.

In order **to join and integrate distributed simulation models**, the Easy Communication Environment (ECE) environment is used (Aizstrauts et al 2012). ECE is communication mechanism provided for data exchange among simulation tools and models. The base is a reduced HLA (Perumalla 2009) exchange mechanism – principle of broadcasting. One of the main statements of ECE is simplicity of use to promote designing of distributed simulation models for domain specialists who have no specific knowledge in IT and programming (see Fig.5.2).

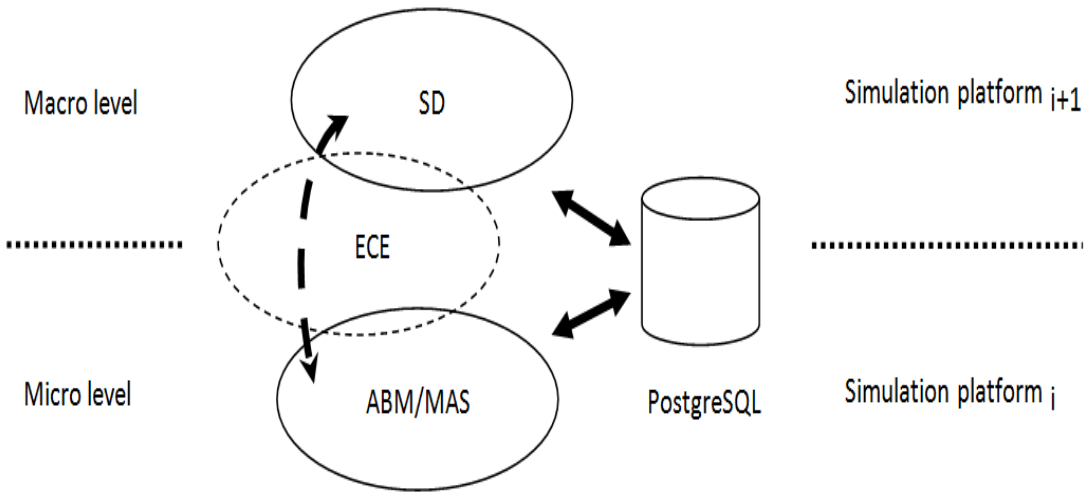


Figure 5.2: Simulation models integration in FUPOL Simulator preliminary architecture

ECE is used for communication and data exchange in simulation architecture between both levels – micro and macro. The micro level then is data source for

macro simulation level. On macro level system dynamics simulation will be done as optional.

In Easy Communication Environment (ECE) each distributed simulation model has to publish data to be used by other models. Distributed simulation model can receive data from other models only if it is subscribed for such data. Similar to HLA the data is only send to those modes that have subscribed for such data (see Fig.5.3).

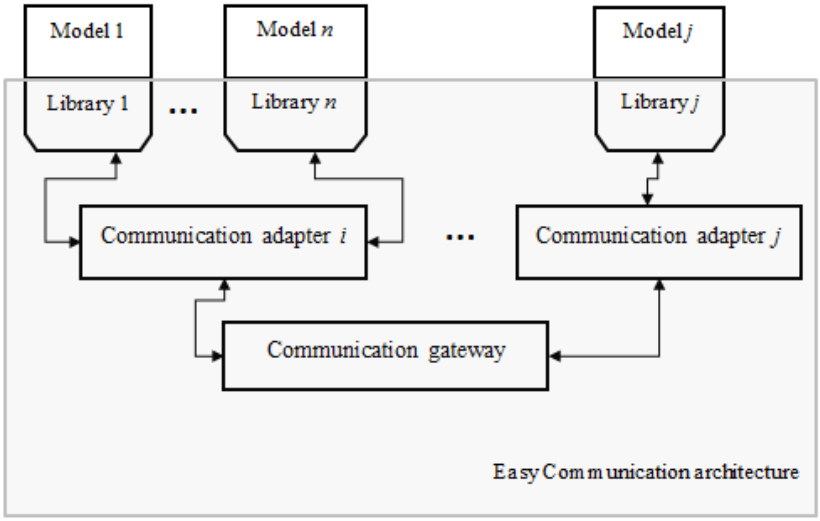


Figure 5.3: Simulation models interoperability through ECE

Figure 5.3 shows Easy Communication Environment architecture with all the main components of it. The Model component represents simulation model that is executed with particular simulation tool. Each model can be made and executed with different simulation tool as long as they support ECE connectivity. To connect model/simulation tool to ECE there must be Library (i.e. "Library 1" from Figure 5.3) component which is made for each simulation tool apart. As it was mentioned before ECE allows communication among various simulation platforms which means there cannot be easy, straight forward communication because there is no universal communication protocol/technique among simulation tools. Therefore ECE must have Library component that works as proxy between Model and Communication adapter (i.e. "Communication adapter i" from Figure 5.3) component. It "translates" Communication adapter messages to particular simulation tool understandable

format. Communication adapter is "local communication manager", with it model can subscribe/publish simulation data. Various simulation tools run with different interpretation of time, therefore Communication adapter stores all incoming messages in right order, so Model can retrieve them when they are needed. Communication gateway component works as a router in Easy Communication Environment, it handles all message transportation to legitimate destinations. Communication gateway manages data subscription database, each Communication adapter registers subscription for data to Communication gateway. Based on subscription database Communication gateway sends out data to particular Communication adapter (technically it is Communication adapter, but afterwards model receives data from it). Communication among gateway and adapters can be over LAN or WAN.

Macro and micro level simulation design will be implemented based on particular domain use case specification which is provided by WP2. Models that require multi-level simulation will implement the main process with System Dynamic, but sub-processes with ABM/MAS (see Fig.5.4).

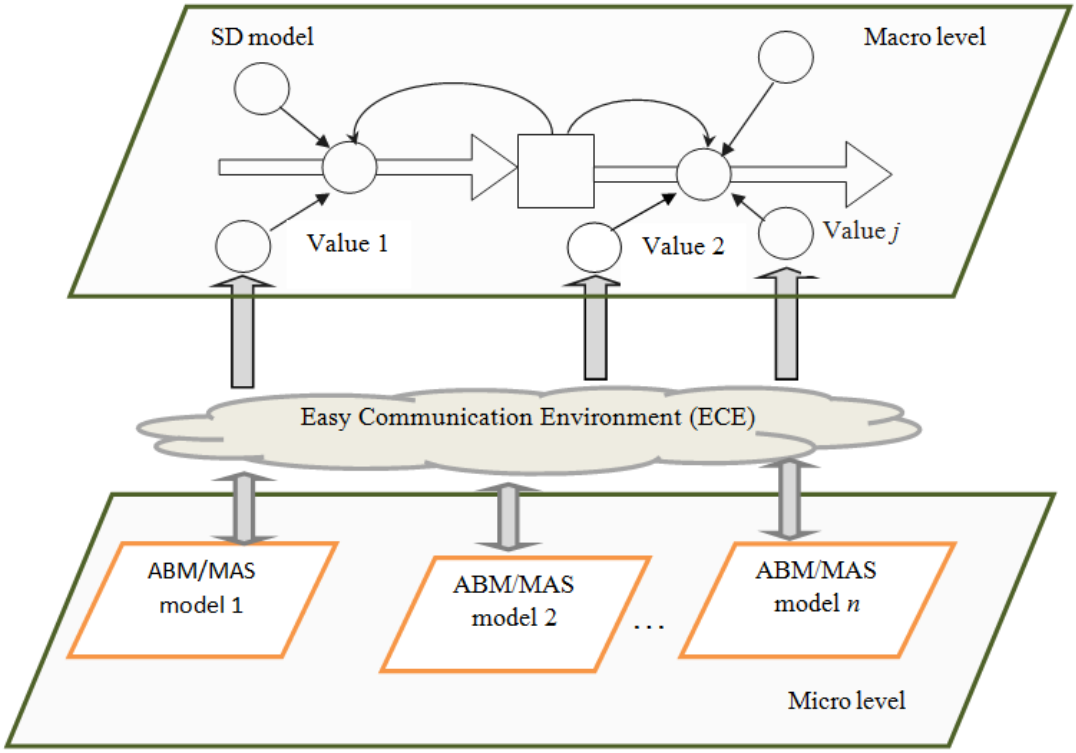


Figure 5.4: ABM and SD simulation models integration in FUPOL Simulator preliminary architecture

Figure 5.4 shows interaction between Macro and micro level simulation. Macro level simulation uses System Dynamic (SD) simulation technique, but Micro level simulation uses ABM/MAS simulation technique. SD model simulates higher scale processes, but ABM/MAS is supposed to simulate small scale process, compared to SD simulated process. In this case, Macro and micro simulation design is implemented in the way where ABM/MAS model simulates some of SD model sub-processes.

System dynamic itself is model of interactions among sub-processes and in this case (Macro and micro simulation design) some of the SD model sub–processes are displaced with ABM/MAS model. In Figure 5.4 Value 1 (for example) stands for a SD model sub-process and is displaced with ABM/MAS model 1. System dynamic sub-processes have incoming data as interaction from other sub-processes implemented by ABM/MAS model.

ECE also ensures integration the set of ABM/MAS models if it is determined by policy domain use case modeling scenario. Policy use cases simulation models data integration in FUPOL Simulator will be realised through PostgreSQL/PostGIS database. FUPOL Simulator will store the simulation data in Postgre mostly in SDMX and broadcast data dictionary to other workpackages for data search and use.

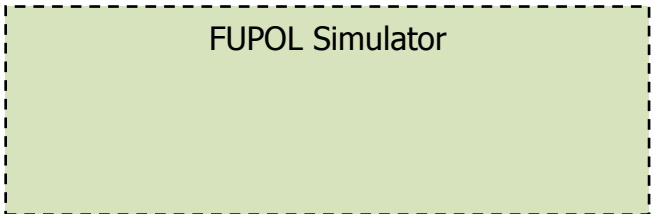
The Simulation Initialization and Control block is responsible not only for simulation process control, but also for interaction with FUPOL Core platform ESB and FUPOL PostgreSQL/PostGIS spatial database. To control the simulation process the visualisation of simulator desktop, intermediate and final results also must be ensured. To store user data, intermediate and final results the access to FUPOL Spatial Database must be provided which is based on Postgre solutions.

6 The Preliminary Description of FUPOL Simulator Interfaces

The FUPOL Simulator is a part of FUPOL software system and therefore must comply (with some constraints) with the interoperability requirements accepted in FUPOL software environment. Therefore requirements for the FUPOL Simulator software interfaces must be formulated. In other words, must be shown the ways how FUPOL Simulator receives input data or policy use case models and how the simulation results are made accessible for other services. Logical interface between WP2 and WP4 is already described in Part 4.

6.1 Structural model of FUPOL Simulator architecture

FUPOL Simulator is one of FUPOL software services. It is used for simulation of several and different policy domain use cases models. Each policy domain use case is available to use for all other FUPOL software services (see Figure 6.1).



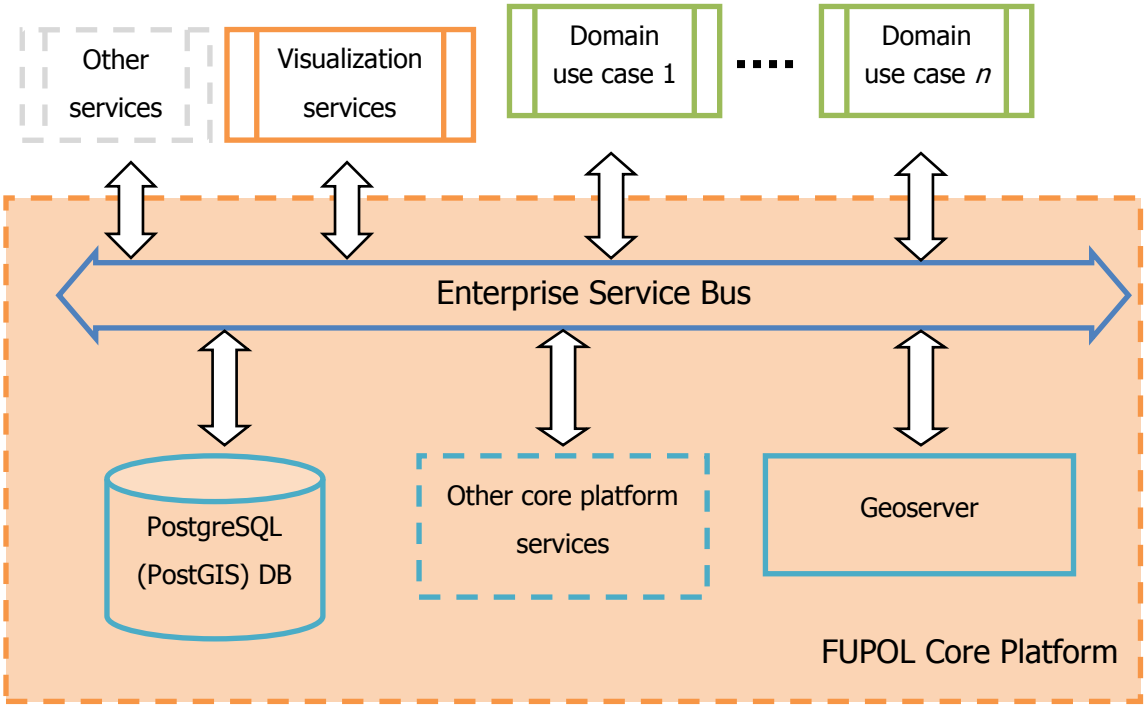


Figure 6.1: FUPOL Simulator in joint FUPOL software architecture

Figure 6.1 shows part of FUPOL software architecture – indicating the connections of FUPOL Simulator services and other FUPOL software services. This Figure 6.1 shows that all Fupol software services use Enterprise Service Bus (ESB) for communication among other services. ESB is the main communication platform for FUPOL software services (see WP3 D3.1).

Block "Other services" represents all FUPOL software services which are not mentioned in Figure 6.1 Statistical data or user management service, for example.

Visualization services, that will be the main recipient of FUPOL Simulator output, will receive data from PostgreSQL (PostGIS) DB.

FUPOL Simulator uses PostgreSQL (PostGIS) DB, Geoserver and other FUPOL Core Platform services (Figure 6.2 describes in more detail other FUPOL Core platform services collaboration with FUPOL Simulator). These services are available through ESB.

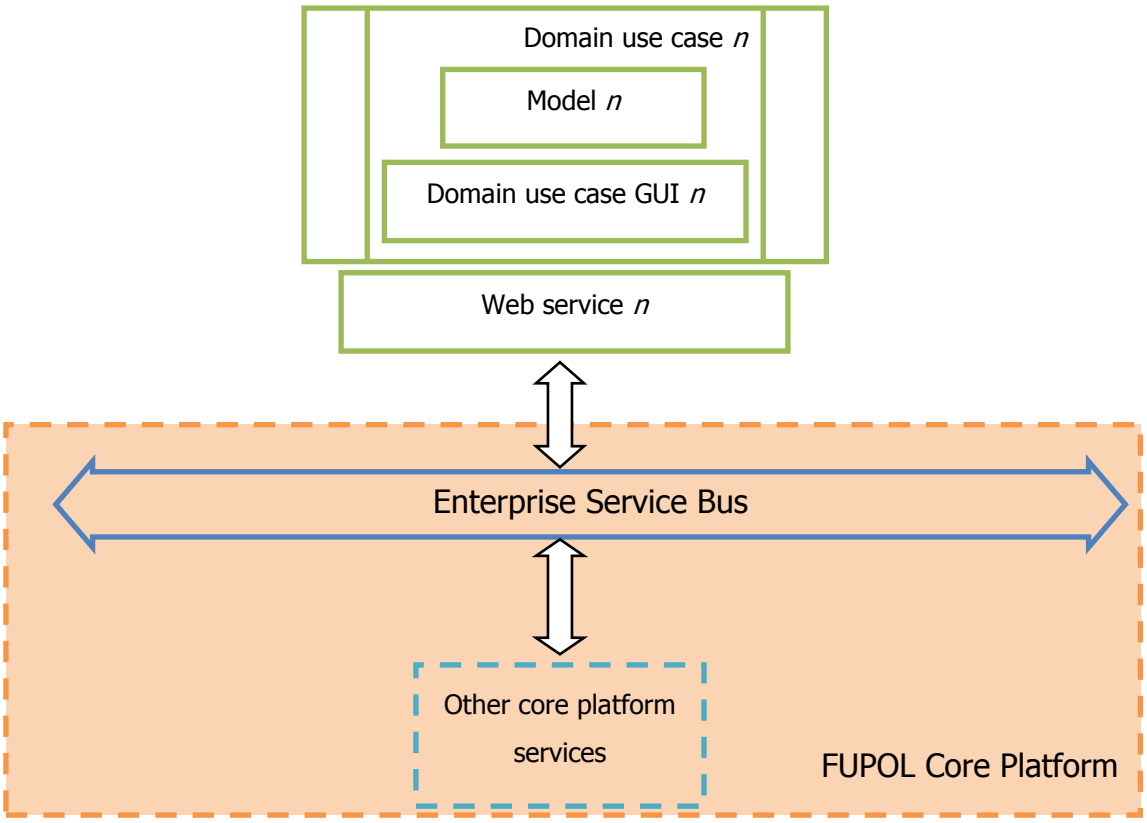


Figure 6.2: FUPOL Simulator collaboration with Other core platform services

Figure 6.2 shows that every policy domain use case model has two main elements – Model (simulation model) and Web service. Model does all the modeling, but Web service allows user to exploit particular policy domain use case model using Domain use case GUI. Other FUPOL Core platform services may have more services than these, but they are irrelevant to FUPOL Simulator in this case.

FUPOL Simulator (see Figure 6.1) is composed of several and different policy domain use cases. Each policy domain use case represent specific policy domain use case model. Each policy domain use case model contains simulation Model itself and Web service with its "published" functions – functions to be used Domain use case GUI, for example, to get land use information about specific point on map necessary in LUCC domain use case. Web service accepts SOAP requests over HTTP. Each policy

domain use case Web service describes its functions using Web Services Description Language (WSDL).

PostgreSQL (PostGIS) DB is used to store different data, which can be useful to other modeling tools or FUPOL software services (for example, Visualization services). Access to PostgreSQL DB can be established only via ESB. All FUPOL software services (including Simulator) must use ESB API to fetch/store data from PostgreSQL database.

Domain use case GUI blocks represent user interface for each domain use case model. FUPOL Simulator user uses particular domain use case GUI to manipulate with specified domain use case model parameters and view modeling results. Domain use case GUI is built for each domain use case model separately, meaning – each GUI is build based on particular domain use case nature and FUPOL Simulator users' expectations on this particular domain use case model usage. Each GUI is also a SOAP client and it makes SOAP requests to particular domain use case Web service. Domain use case GUI is made with HTML and JavaScript, but SOAP client is made with Java Servlet technologies which are implemented into FUPOL Core platform. Simulation Initialization and Control (GUI) and its communication with domain use case Web service will be described in more detail in the following chapters.

Geoserver is one of the FUPOL software services, it can be accessed via ESB. It can be used for data reading and data writing as well. FUPOL software services can store maps into Geoserver, read them later and even read maps which are stored by other services.

Geoserver is used as output/input data source in many policy domain use case models. Models will use Geoserver both for data reading as well as writing. Maps that are changed during the modeling process are stored on Geoserver. For example LUCC domain use case model uses Geoserver CORINE maps as the main initial data, however new maps generated during modeling session are stored on Geoserver.

Domain use case GUI would be implemented into FUPOL Core platform. FUPOL Core platform provides authorization/authentication for policy domain use case GUI. FUPOL software service users can access FUPOL Simulator only by passing FUPOL

Core platform authorization. Domain use case GUI receives user information from FUPOL Core platform using ESB API provided by WP3. Afterwards user information is forwarded to domain use case model and model utility library. Domain use case model may need user information in modeling process, for example to include users' previous modeling data in current modeling. In LUCC domain use case, for example, user data is needed to store maps on Geoserver, it saves modeling result maps by user identification.

6.2 Domain use case software architecture

6.2.1 Structural model

In this section domain use case is described in more detail including information on each element and more technical information (see Figure 6.3).

Figure 6.3 shows FUPOL Simulator domain use case structure and collaboration of its main components with other FUPOL software services. The main components of domain use case software are – domain simulation Model, Model utility library, Web service and Simulation initialization and control unit (Domain use case GUI). The Model utility library is not mandatory. All blocks will be more detailed described in the following sections.

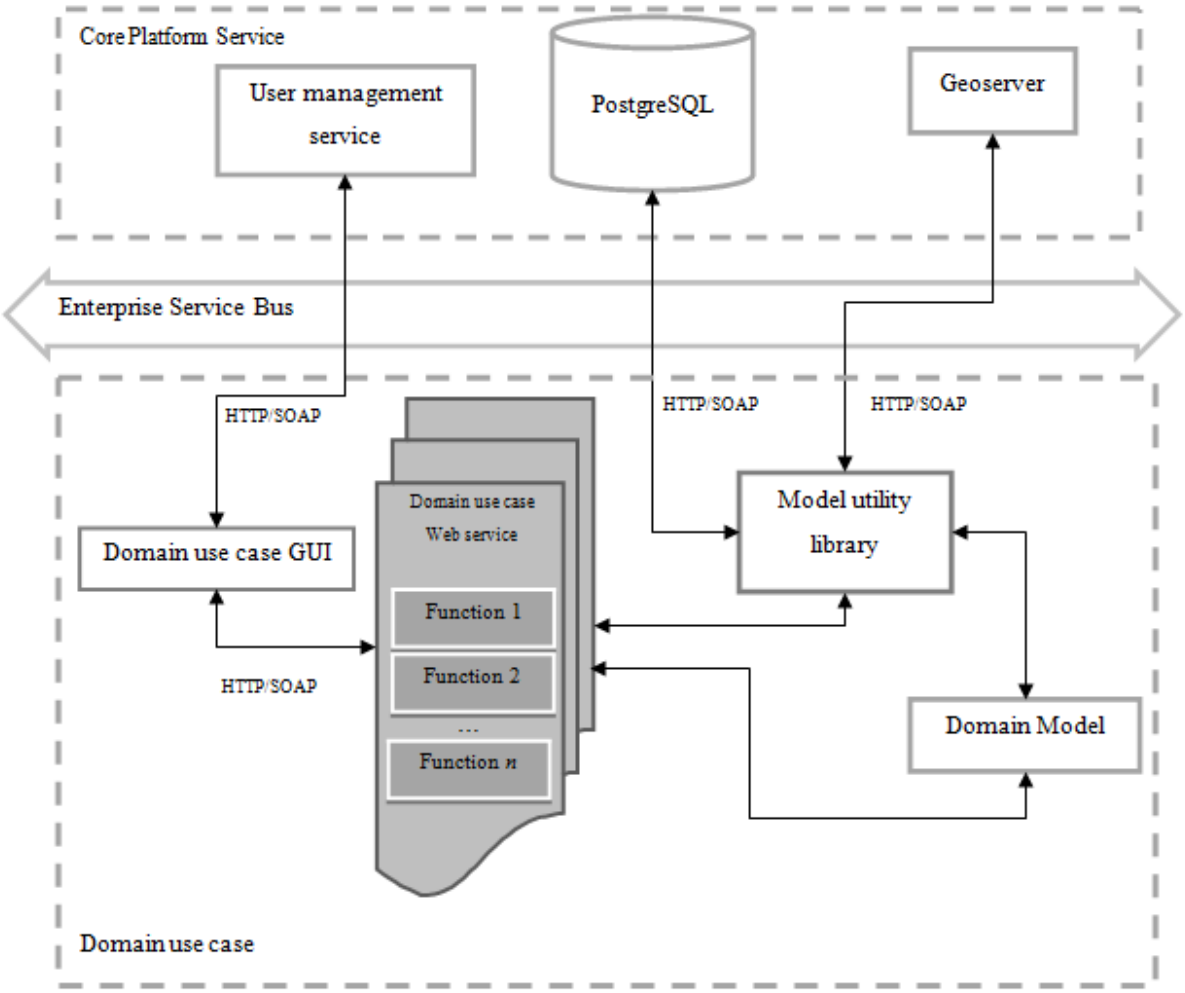


Figure 6.3: FUPOL Simulator policy domain use case software architecture

6.2.2 Domain use case Web service

To ensure that interface of any domain use case is uniform, the Web services that operate on uniform principles are used, apart from domain use case simulation models and the way they are developed. Domain use case simulation model development does not need to follow any unified interface, however it must be convenient and will be implemented in Repast Symphony and Java. Therefore Web service is necessary to make FUPOL domain use case model usage uniform.

A domain use case Web service is web applications that implements Web service specifications. Domain use case web server as web application is hosted on Apache Tomcat 7 web server. It is an open source software implementation of the Java

Servlet and JavaServer Pages technologies. Apache Tomcat is developed in an open and participatory environment and released under the Apache License version 2 [http://tomcat.apache.org/]. Apache Tomcat 7 requires Java 6 or later JRE [http://tomcat.apache.org/tomcat-7.0-doc/setup.html]. For Web service deployment Apache Axis2 is used. It is a Web services / SOAP / WSDL engine . Apache Axis2 web application itself is deployed on Apache Tomcat 7 web server. It requires JDK 5 or above to run [http://axis.apache.org/axis2/java/core/docs/installationguide.html]. Apache Axis2 ensures WSDL generation and SOAP request/response handling over deployed Web services.

Figure 6.4 shows how domain use case Web service is deployed. Domain use case Web service as web application is deployed on Apache Axis2 and Apache Axis2 itself is deployed as web application on Apache Tomcat 7.

In Figure 6.4 it can be seen how the domain use case Web services function request is handled. Request comes from SOAP client which is part of domain use case GUI. Request has two layers – the first is HTTP layer which ensures that request with second layer information reaches the precise web server and the second layer is SAOP layer which has the SOAP document that describes requested Web service, function and all the necessary function parameters.

There are three request processing steps (see arrows 1..3 in Figure 6.4). The first step is to handle HTTP connection (see arrow 1 – HTTPS Layer in Figure 6.4) which is done by Apache Tomcat. It receives HTTP connection based on its configuration.

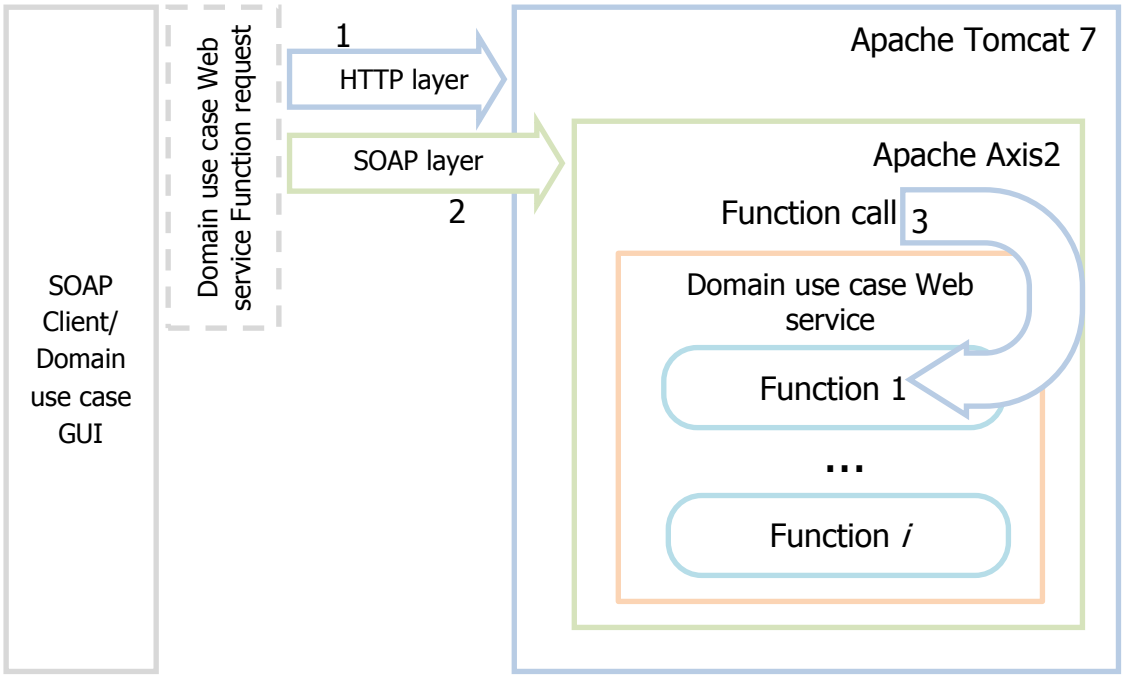


Figure 6.4: Domain use case Web service deployment architecture and SOAP request implementation

The request is further delivered to particular web applications, in this case Apache Axis2. The second step is to process SOAP request (see arrow 2 – SOAP Layer in Figure 6.4) which is done by Apache Axis2. It transforms received SOAP document to the Web services particular function call with all the necessary parameters. And the last step is Web services function call (see arrow 3 – Function call in Figure 6.4) where all function actions are defined. Web service may have defined several functions (SOAP request has the indications which particular Web services functions request). In this case there are two options for the Web service function – to execute domain use case simulation Model or to call domain use case Model utility library functions.

Domain use case Web service functions may vary by domain use cases. Each domain use case may have different Web service functions with various functionality in any case functions have interface descripted in WSDL. WSDL describes how the Web service can be called, what parameters it expects and what data structure it returns.

LUCC use case domain Web service WSDL example see Figure 6.5.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:ns1="http://org.apache.axis2/xsd" xmlns:ns="http://WS.fupol.ssii.org"
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:ax23="http://POJO.fupol.ssii.org/xsd"
xmlns:ax21="http://response.landuse.WS.fupol.ssii.org/xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
targetNamespace="http://WS.fupol.ssii.org">
  <wsdl:documentation></wsdl:documentation>
  <wsdl:types>
    <xs:schema attributeFormDefault="qualified"
elementFormDefault="qualified"
targetNamespace="http://POJO.fupol.ssii.org/xsd">
      <xs:complexType name="LandUseTypePOJO">
        <xs:sequence>
          <xs:element minOccurs="0" name="code" nillable="true"
type="xs:string"/>
          <xs:element minOccurs="0" name="title" nillable="true"
type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="LandDistributionItemPOJO">
        <xs:sequence>
          <xs:element minOccurs="0" name="area" type="xs:double"/>
          <xs:element minOccurs="0" name="areaPercentage"
type="xs:double"/>
          <xs:element minOccurs="0" name="landUseType" nillable="true"
type="ax23:LandUseTypePOJO"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
    <xs:schema xmlns:ax22="http://response.landuse.WS.fupol.ssii.org/xsd"
attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://WS.fupol.ssii.org">
      <xs:import
namespace="http://response.landuse.WS.fupol.ssii.org/xsd"/>
      <xs:element name="startSimulation">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" name="layerName" nillable="true"
type="xs:string"/>
            <xs:element minOccurs="0" name="bbox" nillable="true"
type="xs:string"/>
            <xs:element maxOccurs="unbounded" minOccurs="0"
name="selectedLandUseType" type="xs:int"/>
            <xs:element minOccurs="0" name="restrictionLayerName"
nillable="true" type="xs:string"/>
            <xs:element minOccurs="0" name="simulationTime" type="xs:int"/>
            <xs:element minOccurs="0" name="changeProbability"
type="xs:double"/>
            <xs:element minOccurs="0" name="cellViewDistance"
type="xs:int"/>
```

```
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="startSimulationResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" name="return" nillable="true"
type="ax21:StartSimulationResponse"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="getLandUseTypes">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" name="layer" nillable="true"
type="xs:string"/>
            <xs:element minOccurs="0" name="bbox" nillable="true"
type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="getLandUseTypesResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" name="return" nillable="true"
type="ax21:GetLandUseTypesResponse"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="getLandUseDistribution">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" name="layer" nillable="true"
type="xs:string"/>
            <xs:element minOccurs="0" name="bbox" nillable="true"
type="xs:string"/>
            <xs:element minOccurs="0" name="area" type="xs:double"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="getLandUseDistributionResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" name="return" nillable="true"
type="ax21:GetLandUseDistributionResponse"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:schema>
    <xs:schema xmlns:ax24="http://POJO.fupol.ssii.org/xsd"
attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://response.landuse.WS.fupol.ssii.org/xsd">
      <xs:import namespace="http://POJO.fupol.ssii.org/xsd"/>
      <xs:complexType name="StartSimulationResponse">
        <xs:sequence>
          <xs:element minOccurs="0" name="errorMessage" nillable="true"
type="xs:string"/>
```

```
          <xs:element minOccurs="0" name="simulationThreadID"
nillable="true" type="xs:string"/>
          <xs:element minOccurs="0" name="status" type="xs:int"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="GetLandUseTypesResponse">
        <xs:sequence>
          <xs:element maxOccurs="unbounded" minOccurs="0"
name="landUseTypes" nillable="true" type="ax23:LandUseTypePOJO"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="GetLandUseDistributionResponse">
        <xs:sequence>
          <xs:element maxOccurs="unbounded" minOccurs="0"
name="landDistributionList" nillable="true"
type="ax23:LandDistributionItemPOJO"/>
          <xs:element minOccurs="0" name="totalArea" type="xs:double"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
  </wsdl:types>
  <wsdl:message name="getLandUseDistributionRequest">
    <wsdl:part name="parameters" element="ns:getLandUseDistribution"/>
  </wsdl:message>
  <wsdl:message name="getLandUseDistributionResponse">
    <wsdl:part name="parameters"
element="ns:getLandUseDistributionResponse"/>
  </wsdl:message>
  <wsdl:message name="getLandUseTypesRequest">
    <wsdl:part name="parameters" element="ns:getLandUseTypes"/>
  </wsdl:message>
  <wsdl:message name="getLandUseTypesResponse">
    <wsdl:part name="parameters" element="ns:getLandUseTypesResponse"/>
  </wsdl:message>
  <wsdl:message name="startSimulationRequest">
    <wsdl:part name="parameters" element="ns:startSimulation"/>
  </wsdl:message>
  <wsdl:message name="startSimulationResponse">
    <wsdl:part name="parameters" element="ns:startSimulationResponse"/>
  </wsdl:message>
  <wsdl:portType name="LandUseDomainPortType">
    <wsdl:operation name="getLandUseDistribution">
      <wsdl:input message="ns:getLandUseDistributionRequest"
wsaw:Action="urn:getLandUseDistribution"/>
      <wsdl:output message="ns:getLandUseDistributionResponse"
wsaw:Action="urn:getLandUseDistributionResponse"/>
    </wsdl:operation>
    <wsdl:operation name="getLandUseTypes">
      <wsdl:input message="ns:getLandUseTypesRequest"
wsaw:Action="urn:getLandUseTypes"/>
      <wsdl:output message="ns:getLandUseTypesResponse"
wsaw:Action="urn:getLandUseTypesResponse"/>
    </wsdl:operation>
    <wsdl:operation name="startSimulation">
      <wsdl:input message="ns:startSimulationRequest"
wsaw:Action="urn:startSimulation"/>
      <wsdl:output message="ns:startSimulationResponse"
wsaw:Action="urn:startSimulationResponse"/>
```

```
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="LandUseDomainSoap11Binding"
type="ns:LandUseDomainPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="getLandUseDistribution">
      <soap:operation soapAction="urn:getLandUseDistribution"
style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getLandUseTypes">
      <soap:operation soapAction="urn:getLandUseTypes" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="startSimulation">
      <soap:operation soapAction="urn:startSimulation" style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="LandUseDomainSoap12Binding"
type="ns:LandUseDomainPortType">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <wsdl:operation name="getLandUseDistribution">
      <soap12:operation soapAction="urn:getLandUseDistribution"
style="document"/>
      <wsdl:input>
        <soap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getLandUseTypes">
      <soap12:operation soapAction="urn:getLandUseTypes" style="document"/>
      <wsdl:input>
        <soap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
```

```
    <wsdl:operation name="startSimulation">
      <soap12:operation soapAction="urn:startSimulation" style="document"/>
      <wsdl:input>
        <soap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="LandUseDomainHttpBinding"
type="ns:LandUseDomainPortType">
    <http:binding verb="POST"/>
    <wsdl:operation name="getLandUseDistribution">
      <http:operation location="getLandUseDistribution"/>
      <wsdl:input>
        <mime:content type="application/xml" part="parameters"/>
      </wsdl:input>
      <wsdl:output>
        <mime:content type="application/xml" part="parameters"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getLandUseTypes">
      <http:operation location="getLandUseTypes"/>
      <wsdl:input>
        <mime:content type="application/xml" part="parameters"/>
      </wsdl:input>
      <wsdl:output>
        <mime:content type="application/xml" part="parameters"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="startSimulation">
      <http:operation location="startSimulation"/>
      <wsdl:input>
        <mime:content type="application/xml" part="parameters"/>
      </wsdl:input>
      <wsdl:output>
        <mime:content type="application/xml" part="parameters"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="LandUseDomain">
    <wsdl:port name="LandUseDomainHttpSoap11Endpoint"
binding="ns:LandUseDomainSoap11Binding">
      <soap:address
location="http://localhost:8080/FupolWebService/services/LandUseDomain.Land
UseDomainHttpSoap11Endpoint/"/>
    </wsdl:port>
    <wsdl:port name="LandUseDomainHttpSoap12Endpoint"
binding="ns:LandUseDomainSoap12Binding">
      <soap12:address
location="http://localhost:8080/FupolWebService/services/LandUseDomain.Land
UseDomainHttpSoap12Endpoint/"/>
    </wsdl:port>
    <wsdl:port name="LandUseDomainHttpEndpoint"
binding="ns:LandUseDomainHttpBinding">
```

```
      <http:address
location="http://localhost:8080/FupolWebService/services/LandUseDomain.Land
UseDomainHttpEndpoint/"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Figure 6.5: LUCC use case Web service WSDL script

This WSDL example shows that LUCC use case Web service has three functions:

- startSimulation;
- getLandUseTypes;
- getLandUseDistribution.

WSDL defines what data these functions return and what parameters they expect. Based on this information SOAP requests may be made. Further every LUCC domain use case example Web service function with its request and response is explained.

6.2.2.1 Function "startSimulation"

In LUCC domain use case example this function is launching the model and after simulation returns simulation process ID. WSDL shows that this function requires seven parameters to be passed.

Functions "startSimulation" parameters definition in web services WSDL see Figure 6.6.

```
<xs:element name="startSimulation">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" name="layerName" nillable="true"
type="xs:string"/>
            <xs:element minOccurs="0" name="bbox" nillable="true"
type="xs:string"/>
            <xs:element maxOccurs="unbounded" minOccurs="0"
name="selectedLandUseType" type="xs:int"/>
            <xs:element minOccurs="0" name="restrictionLayerName"
nillable="true" type="xs:string"/>
            <xs:element minOccurs="0" name="simulationTime" type="xs:int"/>
            <xs:element minOccurs="0" name="changeProbability"
type="xs:double"/>
            <xs:element minOccurs="0" name="cellViewDistance"
type="xs:int"/>
```

```
          </xs:sequence>
        </xs:complexType>
      </xs:element>
```

Figure 6.6: Function „startSimulation" WSDL script

Function "startSimulation" expects these parameters to be passed:

- layerName – String data type;
- bbox – String data type;
- selectedLandUseType – Integer list data type;
- restrictionLayerName – String data type;
- simulationTime – Integer data type;
- changeProbability – Double data type;
- cellViewDistance – Integer data type.

Functions "startSimulation" request example see Figure 6.7.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:ws="http://WS.fupol.ssii.org">
 <soap:Header/>
 <soap:Body>
  <ws:startSimulation>
   <ws:layerName>fupol_simulator:corine_land_use</ws:layerName>
   <ws:bbox>100,200,100,200</ws:bbox>
   <ws:selectedLandUseType>111</ws:selectedLandUseType>
   <ws:restrictionLayerName>restriction_layer</ws:restrictionLayerName>
   <ws:simulationTime>5</ws:simulationTime>
   <ws:changeProbability>0.3</ws:changeProbability>
   <ws:cellViewDistance>2</ws:cellViewDistance>
  </ws:startSimulation>
 </soap:Body>
</soap:Envelope>
```

Figure 6.7: Function „startSimulation"request example

In this example Web services function "startSimulation" is requested with following parameters:

- layerName – fupol_simulator:corine_land_use;
- bbox – 100,200,100,200;
- selectedLandUseType – 111;

- restrictionLayerName – restriction_layer;
- simulationTime – 5;
- changeProbability – 0.2;
- cellViewDistance – 2.

Functions "startSimulation" response in Web services WSDL is defined as shown in Figure 6.8.

```
<xs:complexType name="StartSimulationResponse">
 <xs:sequence>
  <xs:element minOccurs="0" name="errorMessage" nillable="true"
type="xs:string"/>
  <xs:element minOccurs="0" name="simulationThreadID" nillable="true"
type="xs:string"/>
  <xs:element minOccurs="0" name="status" type="xs:int"/>
 </xs:sequence>
</xs:complexType>
```

Figure 6.8: Function „startSimulation" response in WSDL notation

The response definition shows that functions "startSimulation" response has three elements:

- errorMessage – String data type;
- simulationThreadID – String data type;
- status – Integer data type.

Functions "startSimulation" response example see Figure 6.9.

```
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
 <soapenv:Body>
  <ns:startSimulationResponse xmlns:ns="http://WS.fupol.ssii.org">
   <ns:return xsi:type="ax21:StartSimulationResponse"
xmlns:ax21="http://response.landuse.WS.fupol.ssii.org/xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <ax21:errorMessage xsi:nil="true"/>
   <ax21:simulationThreadID>908968573989634902398</ax21:simulationThreadID>
   <ax21:status>1</ax21:status>
   </ns:return>
  </ns:startSimulationResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

Figure 6.9: Function „startSimulation"response example

This functions "startSimulation" response example returns such values:

- errorMessage – this parameter is empty – no value;
- simulationThreadID – 908968573989634902398;
- status – 1.

6.2.2.2 Function "getLandUseTypes"

In LUCC domain use case example this function returns a list of land use types of specified map area. LUCC domain use case Web service WSDL shows that this function expects two parameters to be passed.

Functions "getLandUseTypes" parameters definitions in Web services WSDL see Figure 6.10.

```
<xs:element name="getLandUseTypes">
 <xs:complexType>
  <xs:sequence>
   <xs:element minOccurs="0" name="layer" nillable="true"
type="xs:string"/>
   <xs:element minOccurs="0" name="bbox" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Figure 6.10: Function „getLandUseTypes" definition in WSDL notation

Function "getLandUseTypes" expects such parameters:

- layer – String data type;
- bbox – String data type.

To make SOAP function request both parameters have to be passed.

Functions "getLandUseTypes" request example see Figure 6.11.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:ws="http://WS.fupol.ssii.org">
 <soap:Header/>
 <soap:Body>
  <ws:getLandUseTypes>
   <ws:layer>fupol_simulator:corine_land_use</ws:layer>
<ws:bbox>2702013.58716,7717021.855494,2703685.647153,7718216.18406</ws:bbox
>
  </ws:getLandUseTypes>
 </soap:Body>
</soap:Envelope>
```

Figure 6.11: Function „getLandUseTypes" request example

In this example Web services function "getLandUseTypes" is requested with following parameters:

- layer - fupol_simulator:corine_land_use;
- bbox - 2702013.58716,7717021.855494,2703685.647153,7718216.18406.

Functions "getLandUseTypes" response in Web services WSDL is defined as in Figure 6.12.

```
  <xs:complexType name="GetLandUseTypesResponse">
    <xs:sequence>
     <xs:element maxOccurs="unbounded" minOccurs="0" name="landUseTypes"
nillable="true" type="ax23:LandUseTypePOJO"/>
    </xs:sequence>
   </xs:complexType>
```

Figure 6.12: Function „getLandUseTypes"response in WSDL

The functions "getLandUseTypes" definition shows that response has a list of data type LandUseTypePOJO named "landUseTypes".

Data type LandUseTypePOJO is described in Web services WSDL as in Figure 6.13.

```
<xs:complexType name="LandUseTypePOJO">
 <xs:sequence>
  <xs:element minOccurs="0" name="code" nillable="true" type="xs:string"/>
  <xs:element minOccurs="0" name="title" nillable="true" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

Figure 6.13: Data type „LandUseTypePOJO" description in WSDL

The response list item of the function "getLandUseTypes" can be explained as:

- LandUseTypePOJO – List of "LandUseTypePOJO" object types:
    - code – String data type,
    - title – String data type.

Functions "getLandUseTypes" requests response data example see in Figure 6.14.

```
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
```

```
 <soapenv:Body>
  <ns:getLandUseTypesResponse xmlns:ns="http://WS.fupol.ssii.org">
   <ns:return xsi:type="ax21:GetLandUseTypesResponse"
xmlns:ax21="http://response.landuse.WS.fupol.ssii.org/xsd"
xmlns:ax23="http://POJO.fupol.ssii.org/xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <ax21:landUseTypes xsi:type="ax23:LandUseTypePOJO">
    <ax23:code>-1</ax23:code>
    <ax23:title>none</ax23:title>
   </ax21:landUseTypes>
   <ax21:landUseTypes xsi:type="ax23:LandUseTypePOJO">
    <ax23:code>523</ax23:code>
    <ax23:title>Sea and ocean</ax23:title>
   </ax21:landUseTypes>
   <ax21:landUseTypes xsi:type="ax23:LandUseTypePOJO">
    <ax23:code>324</ax23:code>
    <ax23:title>Transitional woodland-shrub</ax23:title>
   </ax21:landUseTypes>
   <ax21:landUseTypes xsi:type="ax23:LandUseTypePOJO">
    <ax23:code>312</ax23:code>
    <ax23:title>Coniferous forest</ax23:title>
   </ax21:landUseTypes>
   </ns:return>
  </ns:getLandUseTypesResponse>
 </soapenv:Body>
</soapenv:Envelope>
```


Figure 6.14: Data type „GetLandUseTypesResponse"data example description

Function "getLandUseTypes" example response has such values:

- landUseTypes – LandUseTypePOJO data type object with child elements:
  - code – -1,
  - title – none;
- landUseTypes – LandUseTypePOJO data type object with child elements:
  - code – 523,
  - title - Sea and ocean;
- landUseTypes – LandUseTypePOJO data type object with child elements:
  - code – 324,
  - transitional woodland-shrub;
- landUseTypes – LandUseTypePOJO data type object with child elements:
  - code – 312,
  - title - Coniferous forest.

6.2.2.3 Function "getLandUseDistribution"

This function returns land use distribution in given area of map. Domain use case Web service WSDL shows that this function expects two parameters to be passed. Functions "getLandUseDistribution" parameters definitions in Web services WSDL see Figure 6.15.

```
<xs:element name="getLandUseDistribution">
 <xs:complexType>
  <xs:sequence>
   <xs:element minOccurs="0" name="layer" nillable="true"
type="xs:string"/>
   <xs:element minOccurs="0" name="bbox" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Figure 6.15: Functions "getLandUseDistribution" parameters definitions in WSDL

Function "getLandUseDistribution" expects such parameters:

- layer – String data type;
- bbox – String data type.

Functions "getLandUseDistribution" request example see Figure 6.16.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:ws="http://WS.fupol.ssii.org">
 <soap:Header/>
 <soap:Body>
  <ws:getLandUseDistribution>
   <ws:layer>fupol_simulator:corine_land_use</ws:layer>
<ws:bbox>2702013.58716,7717021.855494,2703685.647153,7718216.18406</ws:bbox
>
  </ws:getLandUseDistribution>
 </soap:Body>
</soap:Envelope>
```

Figure 6.16: Functions "getLandUseDistribution" requests example

In this request example Web services function "getLandUseDistribution" is requested with following parameters:

- layer - fupol_simulator:corine_land_use;

- bbox - 2702013.58716,7717021.855494,2703685.647153,7718216.18406.

Functions "getLandUseDistribution" response in Web services WSDL is defined in Figure 6.17.

```
<xs:element name="getLandUseDistributionResponse">
 <xs:complexType>
  <xs:sequence>
   <xs:element minOccurs="0" name="return" nillable="true"
type="ax21:GetLandUseDistributionResponse"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Figure 6.17: Functions "getLandUseDistribution" response in WSDL

The functions "getLandUseDistribution" definition shows that response has one parameter with data type GetLandUseDistributionResponse.

Data type GetLandUseDistributionResponse is described also in WSDL as in Figure 6.18.

```
<xs:complexType name="GetLandUseDistributionResponse">
 <xs:sequence>
  <xs:element maxOccurs="unbounded" minOccurs="0"
name="landDistributionList" nillable="true"
type="ax23:LandDistributionItemPOJO"/>
  <xs:element minOccurs="0" name="totalArea" type="xs:double"/>
 </xs:sequence>
</xs:complexType>
```

Figure 6.18: Data type "GetLandUseDistributionResponse" in WSDL

Data type GetLandUseDistributionResponse contains two elements:

- landDistributionList – LandDistributionItemPOJO data type;
- totalArea – Double data type.

Data type LandDistributionItemPOJO is described in WSDL as shown in Figure 6.19.

```
<xs:complexType name="LandDistributionItemPOJO">
 <xs:sequence>
  <xs:element minOccurs="0" name="area" type="xs:double"/>
  <xs:element minOccurs="0" name="areaPercentage" type="xs:double"/>
  <xs:element minOccurs="0" name="landUseType" nillable="true"
type="ax23:LandUseTypePOJO"/>
 </xs:sequence>
</xs:complexType>
```

Figure 6.19: Data type " LandUseDistributionItem POJO" in WSDL

Data type LandDistributionItemPOJO contains three elements:

- area – Double data type;
- areaPercentage – Double data type;
- landUseType – LandUseTypePOJO data type;

Data type LandUseTypePOJO is described in WSDL is shown in Figure 6.20.

```
<xs:complexType name="LandUseTypePOJO">
 <xs:sequence>
  <xs:element minOccurs="0" name="code" nillable="true" type="xs:string"/>
  <xs:element minOccurs="0" name="title" nillable="true" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

Figure 6.20: Data type "LandUseTypePOJO" in WSDL

Data type LandUseTypePOJO contains two elements:

- code – String data type;
- title – String data type.

Functions "getLandUseDistribution" requests response example see in Figure 6.21.

```
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
 <soapenv:Body>
  <ns:getLandUseDistributionResponse xmlns:ns="http://WS.fupol.ssii.org">
   <ns:return xsi:type="ax21:GetLandUseDistributionResponse"
xmlns:ax21="http://response.landuse.WS.fupol.ssii.org/xsd"
xmlns:ax23="http://POJO.fupol.ssii.org/xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <ax21:landDistributionList xsi:type="ax23:LandDistributionItemPOJO">
    <ax23:area>25.0</ax23:area>
    <ax23:areaPercentage>25.0</ax23:areaPercentage>
    <ax23:landUseType xsi:type="ax23:LandUseTypePOJO">
     <ax23:code>211</ax23:code>
     <ax23:title>Non-irrigated arable land</ax23:title>
    </ax23:landUseType>
   </ax21:landDistributionList>
   <ax21:landDistributionList xsi:type="ax23:LandDistributionItemPOJO">
    <ax23:area>50.0</ax23:area>
    <ax23:areaPercentage>50.0</ax23:areaPercentage>
    <ax23:landUseType xsi:type="ax23:LandUseTypePOJO">
     <ax23:code>231</ax23:code>
     <ax23:title>Pastures</ax23:title>
    </ax23:landUseType>
   </ax21:landDistributionList>
   <ax21:landDistributionList xsi:type="ax23:LandDistributionItemPOJO">
    <ax23:area>25.0</ax23:area>
    <ax23:areaPercentage>25.0</ax23:areaPercentage>
    <ax23:landUseType xsi:type="ax23:LandUseTypePOJO">
     <ax23:code>242</ax23:code>
```

```
      <ax23:title>Complex cultivation patterns</ax23:title>
     </ax23:landUseType>
    </ax21:landDistributionList>
    <ax21:totalArea>100.0</ax21:totalArea>
    </ns:return>
   </ns:getLandUseDistributionResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 6.21: Functions Functions "getLandUseDistribution" requests response example

Function "getLandUseDistribution" example response has such values:

- getLandUseDistributionResponse - GetLandUseDistributionResponse data type object with child elements:
    - landDistributionList – LandDistributionItemPOJO data type object with child elements:
        - area – 25.0,
        - areaPercentage – 25.0,
        - landUseType - LandUseTypePOJO data type object with child elements:
            - code – 211,
            - title – Non-irrigated arable land;
    - landDistributionList – LandDistributionItemPOJO data type object with child elements:
        - area – 50.0,
        - areaPercentage – 50.0,
        - landUseType - LandUseTypePOJO data type object with child elements:
            - code – 231,
            - title – Pastures;
    - landDistributionList – LandDistributionItemPOJO data type object with child elements:
        - area – 25.0,
        - areaPercentage – 25.0,

- landUseType - LandUseTypePOJO data type object with child elements;
    - code – 242,
    - title – Complex cultivation patterns;
- totalArea – 100.0

Figure 6.3 shows an arrow between domain use case Web service and domain use case model. This link represents that Web service can launch the simulation model and start simulation process within particular domain use case modeling. Web service implements domain use case model as Java library and calls its functions.

The links between domain use case Web service and domain use case simulation model utility library (see Figure 6.3) works the same way as link between Web service and model. Domain use case simulation model utility library is implemented as Java library in Web service and this service can call its functions.

6.2.2.4 Model utility library

Model utility library is a library that promotes functional operation of domain use case model. Functionality that is not directly connected with modeling itself is transferred from model to this library. Library includes functionality that is required by particular domain use case simulation model or other FUPOL software services, for example GUI or Visualization services. For example, in the case of LUCC domain use case, the library may include informative function that returns land use values.

Library is developed on programming language Java and is platform-independent and can run on almost all operating systems. Model utility library is stored as JAR (Java ARchive) file and other Java based programs can use it as library such as domain use case model.

In Figure 6.3 it can be seen that model utility library has four connecting flows – with Web service, with domain simulation Model, with PostgreSQL (PostGIS) DB and with Geoserver.

Model utility library is implemented as Java library by domain use case simulation model and domain use case Web service. Both of them also are written in programming language Java. Domain use case simulation model is built in ABM/MAS modeling software RePast Symphony which is written in programming language Java. Domain use case Web service is also written in programming language Java and uses domain use case model utility library as Java library.
Domain use case model utility library can have functionality that may require DB usage. Utility library will use ESB API (developed by WP3) to retrieve and store information related to particular domain use case modeling in Postgre (PostGIS) DB. All communication with database is done using ESB API (developed by WP3).

For example, LUCC domain use case model utility library has function that stores land use distribution in map form that was used in modeling. This function is called by domain use case model along each map generation process. At the same time utility library has function that retrieves land use distribution in particular map from database. This function is mainly used by particular domain use case Web service afterwards giving retrieved land use distribution information to the Web service user (SOAP client which made a request to this information).

6.2.3 Domain use case simulation model

Domain use case model is the main simulator element which does all the modeling for the particular policy domain use case. The model is developed based on theoretical models defined and verified by WP2. Depending on the pre-defined needs of the model it will use modeling techniques like ABM/MAS (Agent-based modeling/ multi-agent systems) and/or SD (System Dynamics).
Domain use case model is written in programing language Java using modeling software RePast Symphony. Model is executable Java JAR archive which can be used by other Java program or executed as stand-alone software. In domain use case, Web service is the only software supposed to execute model, in other words model is supposed to be launched only by web service functions. Figure 6.3 shows that

domain model has only two connecting data flows, one with domain use case Web service and the other one with Model utility library. Domain model implements Model utility library to use its functionality for its own purpose, for example, in LUCC example, domain use case model uses maps from Geoserver. In LUCC domain use case example all functionality related to maps (Geoserver) is devolved to Model utility library, so if the model needs a map from Geoserver it must use the Model utility library to retrieve it and the same is with the map saving process to Geoserver, it must be done with the help of the Model utility library. Model utility library uses ESB API to communication with Geoserver.

In the LUCC domain example, domain use case task is to model the land use in the certain area, at a certain period of time. The models result is land use maps for each modeling cycle. It stores a map about every modeling cycle (iteration) in Geoserver via domain use case Model utility library. At the beginning of modeling process the user defines the amount of cycles. One modeling cycle processes every cell (in LUCC domain use case example map is divided into small squares where each square represents specific land use type) and applies the land use type changes if necessary.

6.2.4 Collaboration with Geoserver

FUPOL Simulator policy domain use case models may use maps in modeling process, for example, in case of LUCC domain use case, model uses Geoserver maps for a modeling initial data. The connection to Geoserver is done using a Model utility library where all the necessary functionality to store and to retrieve data to/from Geoserver is implemented. As mentioned in previous sections, ESB will provide API to use Geoserver functionality. Fupol simulator will not have any straight access to Geoserver itself.

Maps that are created during the modeling process are stored in Geoserver using WMS functionality over ESB API.

Fupol software Geoserver supports these services:

- Web Feature Service (WFS) supports requests for geographical feature data (with vector geometry and attributes);
- Web Map Service (WMS) supports requests for map images (and other formats) generated from geographical data;
- Web Coverage Service (WCS) supports requests for coverage data (rasters).

These Geoserver services should be implemented by ESB API provided by WP3.

In LUCC domain use case example maps are organized as follows. Invariable maps and default maps are placed in Geoserver for public access. The maps that are generated during simulation are stored in Geoserver by particular users' parameters.

As FUPOL Simulator users will be authorized, modeled maps of each user will be stored separate from other user maps, meaning ESB API must provide functionality to restrict user access to other users' generated maps.

FUPOL Simulator requires such functionality from ESB API:

- Store vector polygon with parameters – user, simulations_id, iteration, layer_type;
- Read vector polygon by parameters – user, simulations_id, iteration, layer_type;
- Store raster layer with parameters – user, simulations_id, iteration, map_type;
- Read raster layer by parameters – user, simulations_id, iteration, map_type;

Where:

- user – is particular domain use case model users identification number;
- simulations_id – particular simulation session identification number;
- iteration – particular simulation sessions iteration number;
- layer_type – defines what data is stored in particular layer, i.e. urban restriction area in particular domain use case model for particular user;
- map_type – defines what map is stored in particular layer, i.e. CORINE Land use map.

6.2.5 Access to PostgreSQL (PostGIS) database

Domain use case simulation results and interim results are kept in PostgreSQL database mostly in SDMX format. These results may be useful for next modeling iteration or for other FUPOL software services, for example Visualization services (WP5). Fupol simulator defines business logical functions and in collaboration with WP3 implements these functions into ESB API. Business logic functions may vary among domain use case models but several domain use case models can use the same business logic functions in ESB API. For example business logic function may be "saveLandValue" which carries information about land value in particular area. This function can be uses by LUCC domain use case model as well as some inhabitant migration model, meaning this function is not bounded to any particular domain use case model. Data storage specification in more details can be made later when particular domain use case models will be made. As mentioned before other FUPOL software services will use domain use case modeling results as input data for their needs.

To write/read data, domain use case model utility library connects to PostgreSQL DB via ESB using its API, developed by WP3.

Figure 6.22 shows data flow in time from Domain use case model where data is generated to Fupol software service that requires such data. Domain use case simulation model stores data (in SDMX format) into PostgreSQL DB using particular ESB API functions. Afterwards Fupol software service uses ESB API to retrieve data *a* from PostgreSQL database.

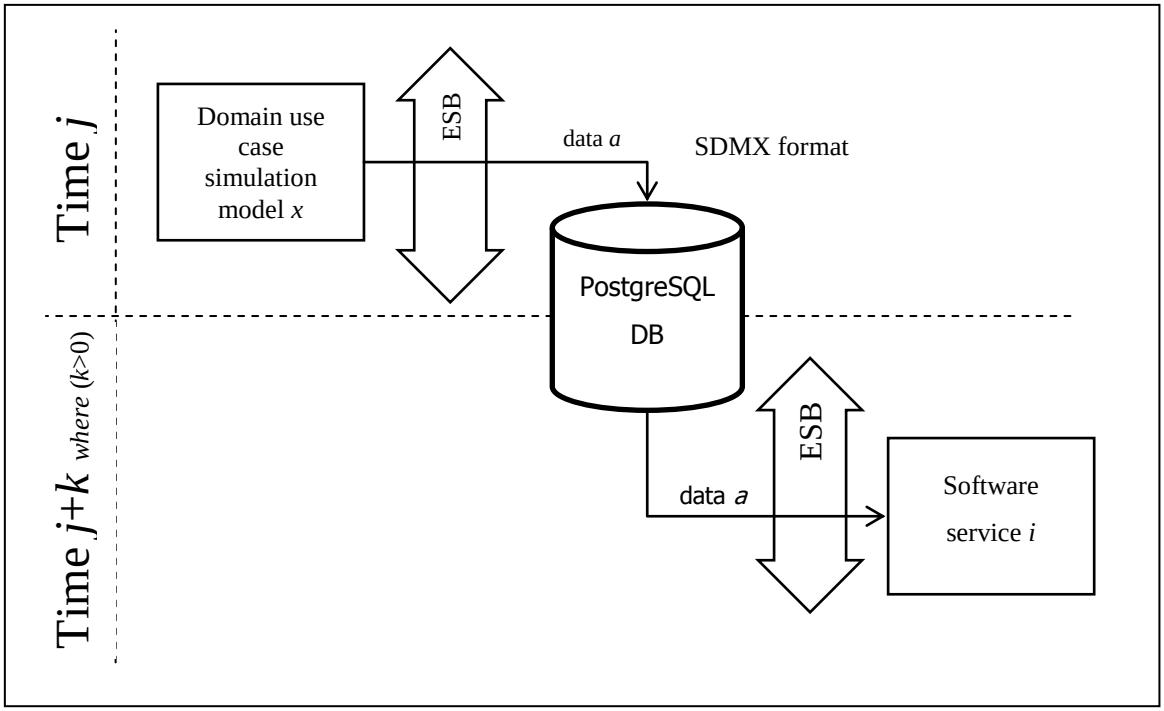


Figure 6.22: FUPOL software service data acquiring from database

6.2.6 Domain use case Simulation Initialization and Control block (GUI)

Domain use case GUI is built for each domain use case separately meaning – each GUI is built based on particular domain use case nature and FUPOL simulator users' expectations on this particular domain use case usage.

Each domain use case GUI is also a SOAP client and it makes SOAP requests to particular domain use case Web service. Domain use case GUI is made with HTML and JavaScript, but SOAP client is made with Java Servlet technologies (see Figure 6.23).

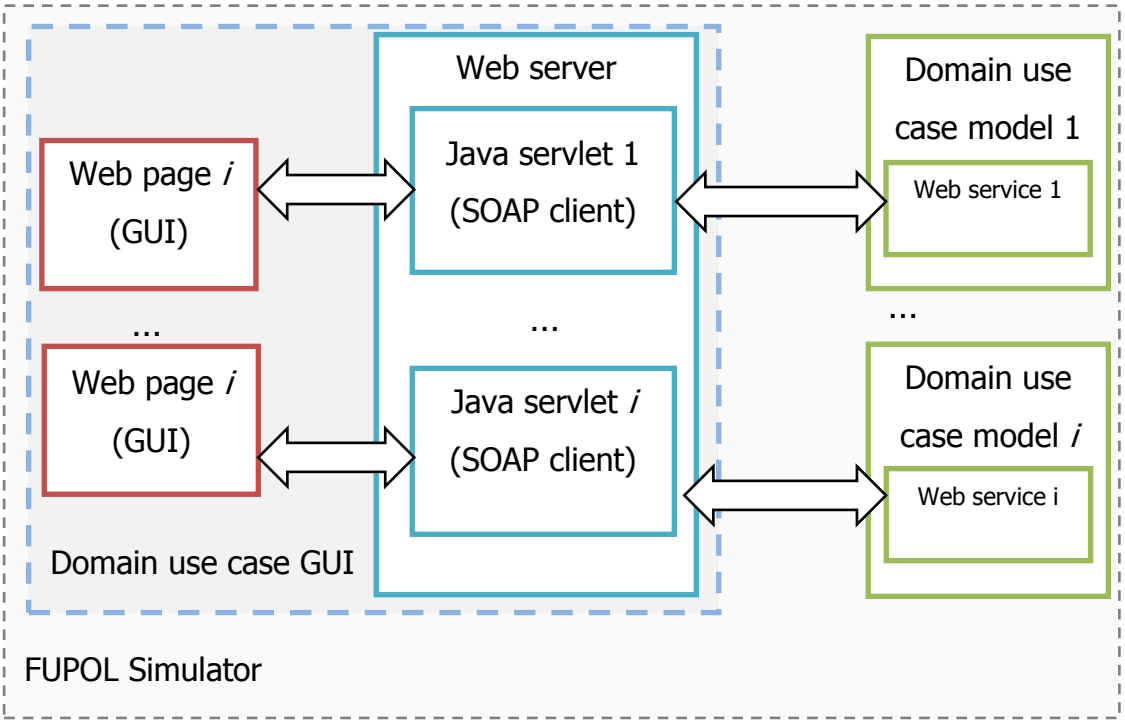


Figure 6.23: Domain use case GUI data flow

In Figure 6.23 one can see domain use case GUI data flow including all the main components that are implicated in data flow from user input to particular domain use case model execution.

Domain use case GUI consists of two main elements:

- Web page (GUI);
- Java Servlet (SOAP client).

Web page allows to set domain use case simulation model parameters and to view modeling results, but Java Servlet works as gateway between GUI and particular domain use case Web service.

6.2.6.1 Web page (GUI)

This element allows FUPOL software user to control particular domain use case simulation model parameters using HTML GUI elements. Domain use case GUI web

page is specifically made for each domain use case based on particular domain use case needs and domain use case users expectations on GUI. This must be done collaborating with WP2/7.

Web (HTML/JavaScript) is convenient technology to make GUI. This technology has following main advantages:

- Easy to develop – HTML and JavaScript are very popular therefore there are a lot of instructional materials and built libraries available;
- HTML based GUI is very accessible (software) – users would not need to install additional software (presuming that web browser is already installed);
- HTML based GUI is very accessible (hardware) – web page is cross-platform technology, every operating system has at least one web browser (excluding specific operating systems with narrow usage). And even cross-device technology, web page can be viewed with different devices, like desktop computers, tablets, mobile phones, etc.;
- Web technologies are low cost – there is no need to buy any licensed software to develop web based application (in this case domain use case GUI), all can be done with open source solutions.

GUI web page is built using HTML for the input/output elements which are crucial to control domain use case simulation model. To make it look more user-friendly CSS is used to style HTML elements.

JavaScript is used for two purposes – to make web page more dynamic (visual effects) and to make AJAX requests. To ease Javascript development jQuery and jQuery UI libraries are implemented. These libraries are distributed under MIT License (in other words they are free to use as long as the copyright header is left intact). jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and AJAX interactions for rapid web development [http://jquery.com/]. jQuery UI is a curated set of user interface interactions, effects, widgets, and themes built on top of the jQuery JavaScript Library [http://jqueryui.com/]. jQuery UI provides wholes set of varied GUI elements that are not available in HTML.

Through web page users sets modeling parameters and launches the modeling process and afterwards web page displays modeling results. The modeling process can be launched with SOAP request to particular domain use case Web service. Therefore Java Servlet is created that takes AJAX requests from web page and converts them to SOAP requests and sends them to domain use case Web service.

AJAX request example from web page to Java Servlet see Figure 6.24.

```
http://lucc_domain/start_simulation.html?
&BBOX=2711599.865401,7756827.275233,2712017.880399,7757125.857375&Layers=fu
pol_simulator:corine_land_use&selected_land_use=111&simulation_time=1&cell_
chnage_probability=5&cell_view=1&features=restriction_layer
```

Figure 6.24: LUCC domain use case GUI request example

Figure 6.25 shows example of the LUCC domain use case model GUI web page.

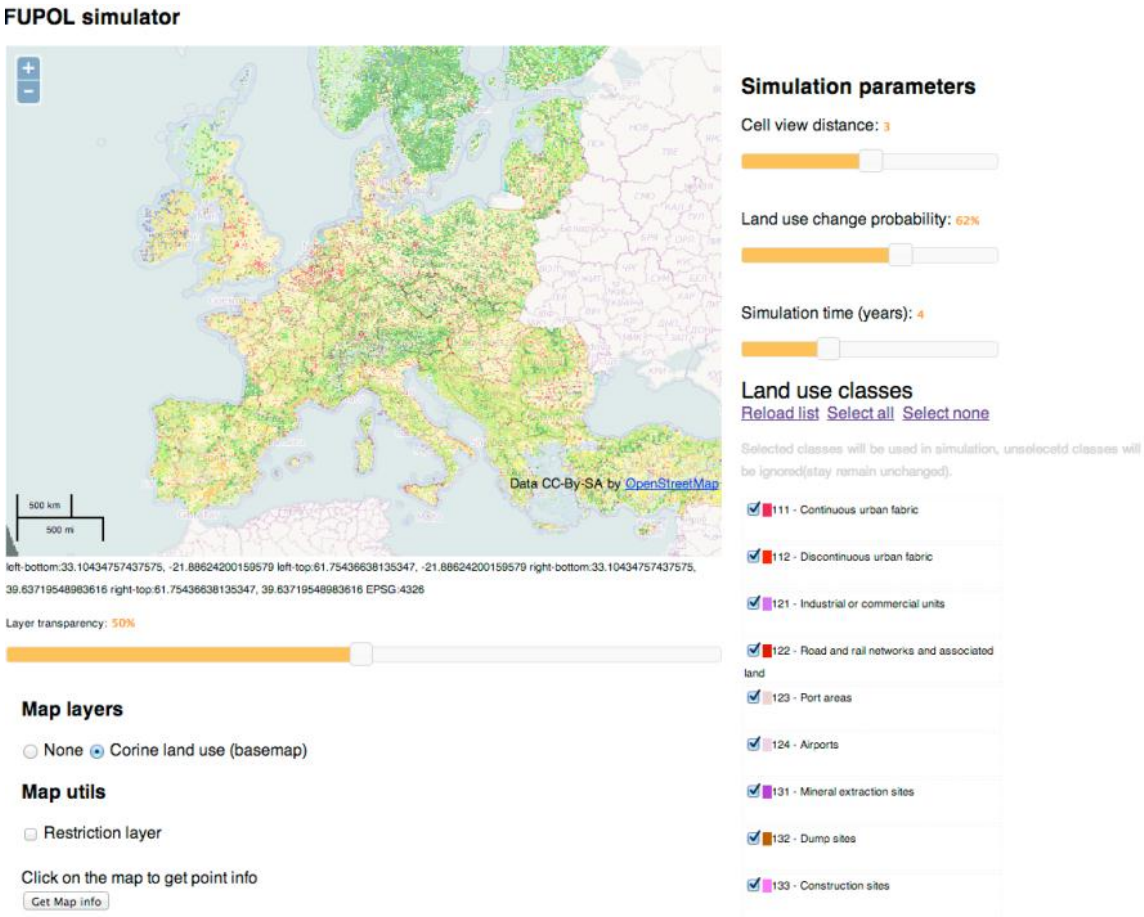


Figure 6.25: LUCC domain use case GUI example

6.2.6.2 Java Servlet (SOAP Client)

This element (see Figure 6.23) performs as gateway between domain use case GUI web page and domain use case Web service. It takes requests from GUI converts them to SOAP requests and connects to particular web service.

Java Servlet (SOAP client) works within web server as web application and is made in programming language Java, it implements Java Servlet specification to response to HTTP requests which are made by domain use case GUI web page. Apache Tomcat 7 is used as web server. It is an open source software implementation of the Java Servlet and JavaServer Pages technologies. Apache Tomcat is developed in an open and participatory environment and released under the Apache License version 2 [http://tomcat.apache.org/].

6.3 ESB API – FUPOL Simulator interface of data import/export and interoperability

This chapter defines the requirements to ESB API aimed to data import and export to/from FUPOL Simulator and ensurance of interoperability. Previous sections of this document describes FUPOL Simulator architecture, its main components and interoperability implementing. FUPOL Simulator architecture description allows other FUPOL software contributors to better understand functioanlity and dataflows related with FUPOL Simulator. WP3 and WP5 are direct collaboration partners to FUPOL Simulator (WP4) and it's crucial that WP4 partners fully understand FUPOL Simulator design and dataflows to achieve best solution of data import/export mechanism that satisfies all involved WPs – WP3, WP4 and WP5.

At this point of development on FUPOL project, WP4 can define hypothetical and high level data import/export functionality requirements for ESB, which is the main FUPOL software data router.

FUPOL Simulator defines necessary logical functions and in collaboration with WP3 implement these functions into ESB API. These functions must arrange with WP5 that these these fucntions have all the neccesery information that is crucial to WP5 (for data Visualisation). Defined functions may vary among domain use case simulation models, but several domain use case models can use the same business logic in ESB API.

This sections brings out all functionality which is requsted from ESB API by FUPOL Simulator. Based on these functionality requirements WP3 collaborating with WP4 will implement necessary functionality of ESB.

6.3.1 User authentication

FUPOL Simulator domain use case GUI is available only with authorized access. Domain use case GUI will use ESB API to authenticate simulator user. FUPOL Core platform has the user management service which keeps user data and handles live user sessions. WP3 will provide ESB functions to obtain user management functionality.

6.3.2 Geoserver functionality

In FUPOL Simulator maps will be one of the key resources as initial data and also as output data. Therefore Geoserver functionality is crucial in ESB. FUPOL Simulator defines such functionality requirements for ESB API:

- Store vector polygon with parameters – user, simulations_id, iteration, layer_type, [*some other parameters varying by particular domain use case*].;
- Read vector polygon by parameters – user, simulations_id, iteration, layer_type, [*some other parameters varying by particular domain use case*].;
- Store raster layer with parameters – user, simulations_id, iteration, map_type, [*some other parameters varying by particular domain use case*].;
- Read raster layer by parameters – user, simulations_id, iteration, map_type, [*some other parameters varying by particular domain use case*].;

where :

- user – is particular domain use case model users identification number;
- simulations_id – particular simulation session identification number;
- iteration – particular simulation sessions iteration number;
- layer_type – defines what data is stored in particular layer, i.e. urban restriction area in particular domain use case model for particular user;
- map_type – defines what map is stored in particular layer, i.e. CORINE Land use map.
- some other parameters varying by particular domain use case - at this point of developmenet, WP4 cannot define all of the parameters what will be used in each particular domain use case. These aditional parameters will be comprehend and described when particular domain use case model will be developed by WP2.

6.3.3 Database functionality

Domain use case simulation results and interim results are kept in PostgreSQL database mostly in SDMX format. These results may be useful for next modeling iteration or for other FUPOL software services, for example Visualization services (WP5). In this section is defined what database functionality is expected from ESB to satisfy FUPOL Simulator requirements. In FUPOL software architecture database is internal service that can be used only via ESB API. Based on FUPOL Simulator requirements WP3 will implement database functionality into ESB API.

Functionality requirements for ESB API:

- Store simulation configuration. Parameters could be: user, simulations_id, iteration, [*some other parameters varying by particular domain use case*].
- Read simulation configuration by these parameters: user, simulations_id, iteration, [*some other parameters varying by particular domain use case*].

where:

- user – is particular domain use case model users identification number;
- simulations_id – particular simulation session identification number;
- iteration – particular simulation sessions iteration number;
- some other parameters varying by particular domain use case - at this point of developmenet, WP4 cannot define all of the parameters what will be used in each particular domain use case. These aditional parameters will be comprehend and described when particular domain use case model will be developed by WP2.

6.4 FUPOL Simulator access to Visualisation services (WP5)

In the FUPOL project at least two kinds of visualisation of data must be distinguished:

- Visualisation of the modelling results (WP5);
- Visualisation of the simulation desktop (WP4).

More detailed requirements for desktop visualisation are described in D.4.1. It will be done for simulation management and internal Java written GUI will be used.
Interaction with general visualisation tools (WP5) will be based on asinhronous approach: reading and writing data in PostGIS data base. In data base will be stored intermediate and finalised simulation results in ASCII and/or SDMX formats.

The results of the simulation can be visualized in multiple ways. The planned forms are described in the following section. The major task is the visualization of the data, which will be calculated by the simulation systems and stored in Postgre DB in SDMX format. For a more dynamic and interactive inclusion of the simulation functionality, it is also planned to consider the calculated simulation data in a kind of animation within the geographical visualization, which will be optional to spatial data draft visualisation already done under the framework of FUPOL Simulator.

6.4.1 Interfacing to SemaVis (WP5)

WP5 uses for the visualization aspects the SemaVis-Framework. The SemaVis-Framework is a web-based technology that runs as web-client with the Flash technology. The system is embedded and directly connected on the FUPOL Core-Platform, the main development technology of WP3. For the exchange and sharing of data between the FUPOL Core Platform, which stores also self-generated content, and the visualization various data formats where be defined. The formats were chosen in dependence of the type of data (see Figure 6.26) and the contained aspects e.g. geographical or statistical information. The definition of just a limited number of exchange data-formats avoids developing parsers and converter on each FUPOL technology and technological environment (e.g. Java, Flash, PHP) for multiple times.

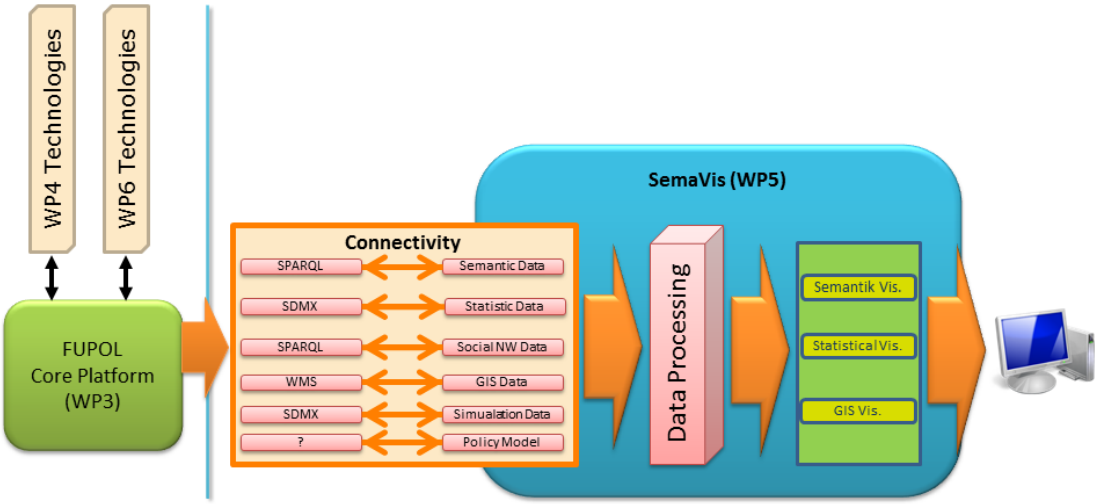


Figure 6.26: Integration and connection of SemaVis with the included process visualization model into the FUPOL Core Platform

The general data sharing for the visualization will be done through the FUPOL Core Platform who is Postgre DB holder. It organizes all available data-sources. Every technical system from the FUPOL partners is connected to the FUPOL Core Platform on Enterprise Service Bus (see Figure 6.1). The SemaVis is requesting required data for the visualization from the FUPOL Core Platform. Additionally, the SemaVis can also be influenced with external commands e.g. to show specific data-sets. The response data will then be visualized within the visualizations of SemaVis. The user has the ability to analyze the data and, if necessary, to initiate another request for further data from the Core Platform.

6.4.2 Visualization of Statistical Data

For the visualization of statistical data a number of statistical visualization is planned, which are designed to analyze the data from different perspectives. To request and access the data in a more sufficient way, SDMX[1] is defined as exchange format (see Figure 6.27). It allows a dynamic request for required indicators etc. on the one hand, and an extensive explanation about the data (metadata) on the other hand. This allows also an advanced access between different types of data, especially to allow a linking between the statics data to the semantics data.

[1] Information about the SDMX specification on official website: http://sdmx.org/

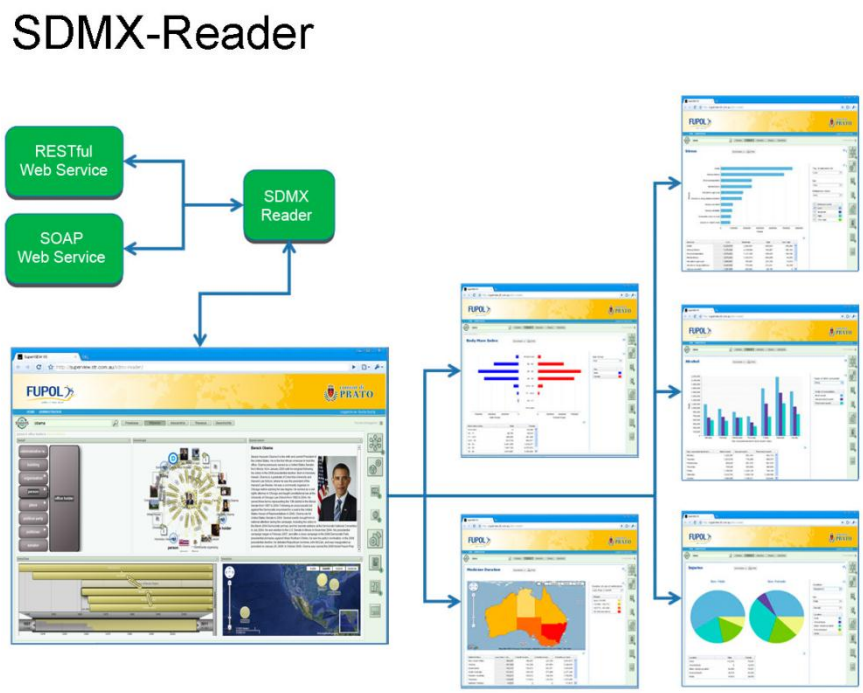


Figure 6.27: The Visualisation service of SemaVis access to SDMX data

The SemaVis includes SDMX-Reader to access (SDMX based) statics data from external sources. Through the integration of various statistic visualizations, the users will have facilitated view on the data from different perspectives.

The SDMX information will be requested and transformed for the visualization within the SDMX Reader of the SemaVis framework. The access on the server can be done over a RESTful Web Service or a SOAP Web Service. A SOAP Web Service allows additionally more complex request e.g. a selection of multiple indicators. After reading and parsing the statistical data, SemaVis can show the results in different kinds of visualization.

The main exchange will be done by an XML-based format named SDMX-ML. It consists of two parts. The first one contains meta-information about the data structure (e.g. used code lists, information about the concepts, and the data structures), see therefore Figure 6.28. This information is useful for the visualization to have, for instance, information about dependencies or about used units. Especially the conceptual meta-information are useful to have a kind of hierarchy and support data visualization with different granularity. The second part contains the data by

time, see therefore Figure 6.29. These are the main information for the statistics visualization which allows drawing e.g. a LineChart.


```
<?xml version="1.0" encoding="UTF-8" ?>

- <!--

 Copyright SDMX 2010  -   www.sdmx.org

-->

- <message:Structure
xmlns:message="http://www.sdmx.org/resources/sdmxml/schemas/v2_1/message"
xmlns:structure="http://www.sdmx.org/resources/sdmxml/schemas/v2_1/structure"
xmlns:common="http://www.sdmx.org/resources/sdmxml/schemas/v2_1/common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.sdmx.org/resources/sdmxml/schemas/v2_1/message
../../schemas/SDMXMessage.xsd">

- <message:Header>

<message:ID>DEMOGRAPHY</message:ID>

<message:Test>false</message:Test>

<message:Prepared>2010-11-13T08:00:33+08:00</message:Prepared>

<message:Sender id="ESTAT" />

</message:Header>

        - <message:Structures>

        - <structure:Codelists>

        + <structure:Codelist id="CL_DECIMALS" agencyID="SDMX" version="1.0"
        isExternalReference="true" structureURL="../common/common.xml">

        + <structure:Codelist id="CL_FREQ" agencyID="SDMX" version="1.0"
        isExternalReference="true" structureURL="../common/common.xml">

        + <structure:Codelist id="CL_CONF_STATUS" agencyID="SDMX" version="1.0"
        isExternalReference="true" structureURL="../common/common.xml">

        + <structure:Codelist id="CL_OBS_STATUS" agencyID="SDMX" version="1.0"
        isExternalReference="true" structureURL="../common/common.xml">

        + <structure:Codelist id="CL_UNIT_MULT" agencyID="SDMX" version="1.0"
        isExternalReference="true" structureURL="../common/common.xml">

        - <structure:Codelist id="CL_UNIT" agencyID="ESTAT" version="1.0" isPartial="true">

        <common:Name xml:lang="en">Unit code list</common:Name>

        - <structure:Code id="PERS">

        <common:Name xml:lang="en">Persons</common:Name>
```

```
</structure:Code>

- <structure:Code id="CPW">

<common:Name xml:lang="en">Children per woman (fertility
rate)</common:Name>

</structure:Code>

- <structure:Code id="YRS">

<common:Name xml:lang="en">Years</common:Name>

</structure:Code>

</structure:Codelist>

- <structure:Codelist id="CL_SEX" agencyID="ESTAT" version="1.0">

<common:Name xml:lang="en">Sex codelist</common:Name>

- <structure:Code id="F">

<common:Name xml:lang="en">Female</common:Name>

</structure:Code>

- <structure:Code id="M">

<common:Name xml:lang="en">Male</common:Name>

</structure:Code>

- <structure:Code id="T">

<common:Name xml:lang="en">Total</common:Name>

</structure:Code>

</structure:Codelist>

+ <structure:Codelist id="CL_COUNTRY" agencyID="ESTAT" version="1.0"
isPartial="true">

</structure:Codelists>

- <structure:Concepts>

+ <structure:ConceptScheme id="CROSS_DOMAIN_CONCEPTS" agencyID="SDMX"
version="1.0" isExternalReference="true" structureURL="../common/common.xml">

- <structure:ConceptScheme id="DEMO_CONCEPTS" agencyID="ESTAT"
version="1.0">

<common:Name xml:lang="en">Demography domain concept
scheme</common:Name>

- <structure:Concept id="COUNTRY">

<common:Name xml:lang="en">Reporting Country</common:Name>
```

```
        </structure:Concept>
        - <structure:Concept id="SEX">
        <common:Name xml:lang="en">Sex</common:Name>
        </structure:Concept>
        - <structure:Concept id="DEMO">
        <common:Name xml:lang="en">Demography</common:Name>
        </structure:Concept>
        </structure:ConceptScheme>
        + <structure:ConceptScheme id="DEMO_MEASURES" agencyID="ESTAT"
        version="1.0">
        </structure:Concepts>
        - <structure:DataStructures>
        - <structure:DataStructure id="DEMOGRAPHY" agencyID="ESTAT" version="1.0">
        <common:Name xml:lang="en">DEMOGRAPHY Data Structure</common:Name>
        + <structure:DataStructureComponents>
        - <structure:ConceptIdentity>
        <Ref agencyID="ESTAT" maintainableParentID="DEMO_CONCEPTS"
        maintainableParentVersion="1.0" id="DEMO" />
        </structure:ConceptIdentity>
        - <structure:LocalRepresentation>
        - <structure:Enumeration>
        <Ref agencyID="ESTAT" id="DEMO_MEASURES" version="1.0"
        class="ConceptScheme" />
        </structure:Enumeration>
        </structure:LocalRepresentation>
        </structure:MeasureDimension>
        </structure:DataStructure>
        </structure:DataStructures>
</message:Structures>
</message:Structure>
```


Figure 6.28: Example SDMX-ML responses of meta-information about a data request

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

- <message:StructureSpecificData
xmlns:demo="urn:sdmx:org.sdmx.infomodel.datastructure.DataStructure=ESTAT:DEMOGRAPHY(1.0):ObsLevelDim:DEMO:explicit"
xmlns:message="http://www.sdmx.org/resources/sdmxml/schemas/v2_1/message"
xmlns:data="http://www.sdmx.org/resources/sdmxml/schemas/v2_1/data/structurespecific"
xmlns:common="http://www.sdmx.org/resources/sdmxml/schemas/v2_1/common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.sdmx.org/resources/sdmxml/schemas/v2_1/message
../../schemas/SDMXMessage.xsd
urn:sdmx:org.sdmx.infomodel.datastructure.DataStructure=ESTAT:DEMOGRAPHY(1.0):ObsLevelDim:DEMO:explicit demography_xs_ex.xsd">

- <message:Header>

<message:ID>DEMO_XS_EX</message:ID>

<message:Test>true</message:Test>

<message:Prepared>2011-11-25T00:21:49-05:00</message:Prepared>

<message:Sender id="ESTSAT" />

- <message:Structure structureID="STR1" dimensionAtObservation="DEMO"
namespace="urn:sdmx:org.sdmx.infomodel.datastructure.DataStructure=ESTAT:DEMOGRAPHY(1.0):ObsLevelDim:DEMO:explicit" explicitMeasures="true">

- <common:Structure>

<Ref agencyID="ESTAT" id="DEMOGRAPHY" version="1.0" />

</common:Structure>

</message:Structure>

</message:Header>

- <message:DataSet data:structureRef="STR1" xsi:type="demo:DataSetType"
data:dataScope="DataStructure">

+ <Series FREQ="A" TIME_PERIOD="2007" SEX="T" COUNTRY="BE">

+ <Series FREQ="A" TIME_PERIOD="2008" SEX="T" COUNTRY="BE">

- <Series FREQ="A" TIME_PERIOD="2009" SEX="T" COUNTRY="BE">

<Obs xsi:type="demo:TFRNSI" OBS_VALUE="1.83" OBS_STATUS="P" UNIT_MEASURE="CPW"
UNIT_MULT="0" />

<Obs xsi:type="demo:LEXPNSIT" OBS_VALUE="80.6" OBS_STATUS="P"
UNIT_MEASURE="YRS" UNIT_MULT="0" />

<Obs xsi:type="demo:LBIRTHST" OBS_VALUE="126000" OBS_STATUS="P"
UNIT_MEASURE="PERS" UNIT_MULT="0" />

<Obs xsi:type="demo:DEATHST" OBS_VALUE="104000" OBS_STATUS="P"
UNIT_MEASURE="PERS" UNIT_MULT="0" />

</Series>
```

```
- <Series FREQ="A" TIME_PERIOD="2007" SEX="M" COUNTRY="BE">

<Obs xsi:type="demo:LEXPNSIT" OBS_VALUE="77.3" OBS_STATUS="A"
UNIT_MEASURE="YRS" UNIT_MULT="0" />

<Obs xsi:type="demo:LBIRTHST" OBS_VALUE="63481" OBS_STATUS="A"
UNIT_MEASURE="PERS" UNIT_MULT="0" />

<Obs xsi:type="demo:DEATHST" OBS_VALUE="49804" OBS_STATUS="A"
UNIT_MEASURE="PERS" UNIT_MULT="0" />

</Series>

- <Series FREQ="A" TIME_PERIOD="2008" SEX="M" COUNTRY="BE">

<Obs xsi:type="demo:LEXPNSIT" OBS_VALUE="77.5" OBS_STATUS="A"
UNIT_MEASURE="YRS" UNIT_MULT="0" />

<Obs xsi:type="demo:LBIRTHST" OBS_VALUE="63926" OBS_STATUS="P"
UNIT_MEASURE="PERS" UNIT_MULT="0" />

<Obs xsi:type="demo:DEATHST" OBS_VALUE="50270" OBS_STATUS="P"
UNIT_MEASURE="PERS" UNIT_MULT="0" />

</Series>

- <Series FREQ="A" TIME_PERIOD="2009" SEX="M" COUNTRY="BE">

<Obs xsi:type="demo:LEXPNSIT" OBS_VALUE="77.7" OBS_STATUS="P"
UNIT_MEASURE="YRS" UNIT_MULT="0" />

<Obs xsi:type="demo:LBIRTHST" OBS_STATUS="M" UNIT_MEASURE="PERS" UNIT_MULT="0"
/>

<Obs xsi:type="demo:DEATHST" OBS_STATUS="M" UNIT_MEASURE="PERS" UNIT_MULT="0"
/>

</Series>

- <Series FREQ="A" TIME_PERIOD="2007" SEX="F" COUNTRY="BE">

<Obs xsi:type="demo:LEXPNSIT" OBS_VALUE="83.3" OBS_STATUS="A"
UNIT_MEASURE="YRS" UNIT_MULT="0" />

<Obs xsi:type="demo:LBIRTHST" OBS_VALUE="60614" OBS_STATUS="A"
UNIT_MEASURE="PERS" UNIT_MULT="0" />

<Obs xsi:type="demo:DEATHST" OBS_VALUE="50854" OBS_STATUS="A"
UNIT_MEASURE="PERS" UNIT_MULT="0" />

</Series>

+ <Series FREQ="A" TIME_PERIOD="2008" SEX="F" COUNTRY="BE">

+ <Series FREQ="A" TIME_PERIOD="2009" SEX="F" COUNTRY="BE">

+ <Series FREQ="A" TIME_PERIOD="2007" SEX="T" COUNTRY="EL">

+ <Series FREQ="A" TIME_PERIOD="2008" SEX="T" COUNTRY="EL">

+ <Series FREQ="A" TIME_PERIOD="2009" SEX="T" COUNTRY="EL">
```

```
+ <Series FREQ="A" TIME_PERIOD="2007" SEX="M" COUNTRY="EL">
+ <Series FREQ="A" TIME_PERIOD="2008" SEX="M" COUNTRY="EL">
+ <Series FREQ="A" TIME_PERIOD="2009" SEX="M" COUNTRY="EL">
+ <Series FREQ="A" TIME_PERIOD="2007" SEX="F" COUNTRY="EL">
+ <Series FREQ="A" TIME_PERIOD="2008" SEX="F" COUNTRY="EL">
+ <Series FREQ="A" TIME_PERIOD="2009" SEX="F" COUNTRY="EL">
</message:DataSet>
</message:StructureSpecificData>
```

Figure 6.29: Example SDMX-ML data responses that contains statistical data

To visualize a geographical or land use change for a city, it is also plan to simulate the development of a research object under specific circumstances. Therefore the simulation results will flow into the geographical visualization e.g. maps. To ensure this, it is essential to commit the visualization geo-positions and meta-information about the changes. For sharing these kinds of information, the WMF format is appropriated. Usually the simulator(s) will generate map layers for every aspect/variable under investigation, possibly even as a timeline of map snapshots. The FUPOL Core Platform supports both raster data (using WMS - web map service) and vector data (using WFS - web feature service). The maps layers will be generated by the FUPOL simulator handed over to the FUPOL Core Platform (using a web service provided by WP3) and finally get stored in the FUPOL Core platform's GIS system (GeoServer). The map layers will be logically linked to the simulation runs and to the campaign. As soon as the data is in the GIS system it can be pulled into SemaVis for visualization or be used as an input for additional simulation runs.

An important point for users is the ability to define in a feasible form, what should be simulated. This is determined by policy use case model elaborated by WP2 and simulated on FUPOL Simulator by WP4.

7 Conclusions

The result of the document is description of logical interface with WP2 aimed to policy use case model specification in form that allows simulation software designing for simulation model running on the FUPOL Simulator. Other result is elaboration of the preliminary requirements to FUPOL Simulator architecture and other interfaces, and definition functionality requirements of ESB (WP3) requested by FUPOL Simulator. Because simulator is part of FUPOL software platform the interaction with Core platform (WP3) and Visualisation part (WP5) must be ensured to import data that are source for FUPOL Simulator and export data that are FUPOL Simulator output data and the same time are source data for other FUPOL software services, i.e. Visualisation (WP5). ESB API functionality will be used to import/export data. As FUPOL Simulator output data is crucial for other FUPOL services it is important to describe how output data will be saved. This document describes what FUPOL Core platform services will be used and what data will be transmitted between FUPOL Simulator and FUPOL Core platform service. FUPOL Simulator export data will be described (in collaboration with WP5 and WP3) in more detailed in FUPOL Simulator software designing documents D4.4 and further when each particular domain use case model will be implemented.

The FUPOL Simulator will be organised in kind of Web service, however respecting complexity of potential use cases the architecture will remain relatively free and open. FUPOL Simulator architecture will be oriented to simulation of two-level and also distributed models, where on first-micro level ABM/MAS models are simulated, but upper-macro level systems dynamics can be simulated as optional. Data integration will be realised through Postgre DB storing the data mostly in SDMX format through ESB API, but interoperability between levels and models inside FUPOL Simulator separate level will be ensured by Easy Communication Environment (ECE). Communication with Visualisation services will be organized through Postgre database accessible through ESB API and provided by WP3. Current document is only initial point for FUPOL Simulator interfaces specification and comprises general

requirements only. Further updates will be involved in D4.4, D4.5, D4.6, D4.7 and D4.8.

8 References

Artis Aizstrauts, Egils Ginters, Dace Aizstrauta, Peter Sonntagbauer, 2012. Easy Communication Environment on the Cloud as Distributed Simulation Infrastructure. //In: Proceedings of the 5th WSEAS World Congress on Applied Computing Conference (ACC '12), Recent Advances in Computing Engineering Series 2, ISBN: 978-1-61804-089-3, ISSN: ISSN: 1790-5109, 2-4 May, 2012, Faro, Portugal, pp. 173-179.

Perumalla, K., 2009. Tutorial. Handling Time Management under the High Level Architecture, http://www.ornl.gov/~2ip/doc/perumalla-tutorialslides-iitsec06.pdf, (2009), 15.11.2011

Susanne Sonntagbauer, Anna Hassapi, Silvana Tomic-Rotim, Haris Neophytou, Miquel Angel Piera Eroles, Miguel Antonio Mujica Mota, Elena Palmisano. Deliverable 2.1 – FUPOL Guidelines on Policy for Cities and Municipalities, 2012, 325 p.

Miquel Angel Piera, Miguel Mújica, Maria Moise, Haris Neofytou , Deliverable 2.2 – FUPOL Cognitive and Causal Models for Prototype, 2012, 271 p.

Nikolaus Rumm, Hakan Kagitcioglu, Alexander Kamenicky, Alexander Krock, Peter Sonntagbauer, Susanne Sonntagbauer, Alexander Uhlir Deliverable 3.1 - Software Requirements Specification and Use Cases, 2012, 319 p.

Peter Sonntagbauer, Susanne Sonntagbauer, Mario Neumann. Deliverable 3.2 - Preliminary Software Design Description, 2012, 241 p.

Susanne Sonntagbauer, Egils Ginters, Dace Aizstrauta, Artis Aizstrauts, Andris Lapans, Visvaldis Valtenbergs, Inita Sakne, Girts Dreija, Sergejs Stepucevs, Miquel

Angel Piera Eroles, Wang Boyong, Mujica-Mota Miguel-Antonio, Shaun Topham, Gary Jones, Gary Simpson, Jonathan Gay, Nikolaus Rumm, Elizabeta Knorr, Silvana Tomic-Rotim, Roman Buil. Deliverable 4.1 – FUPOL Simulator Software Requirements Report, 2012, 231 p.