



PROCESSING DSL SPECIFICATION - BASELINE VERSION

Deliverable D2.4.1

Circulation:	PU: Public
Lead partner:	Fraunhofer
Contributing partners:	
Authors:	Michel Krämer, Ivo Senner
Quality Controllers:	Heino Rudolf, Norman Kießlich
Version:	1.0
Date:	30.04.2014

©Copyright 2014: The IQmulus Consortium

Consisting of

SINTEF	STIFTELSEN SINTEF, Department of Applied Mathematics, Oslo, Norway
Fraunhofer	Fraunhofer Institute for Computer Graphics Research, Darmstadt, Germany
CNR-IMATI-GE	Institute for Applied Mathematics and Information Technologies of the National Research Council (CNR-IMATI), Genova, Italy
MOSS	M.O.S.S. Computer Grafik Systeme GmbH (MOSS), Munich, Germany
HRW	HR Wallingford Ltd (HRW), Wallingford, UK
FOMI	Hungarian National Mapping and Cadastral Agency (FOMI), Institute of Geodesy, Cartography and Remote Sensing, Budapest, Hungary
UCL	University College London (UCL), Research centre for Photogrammetry, 3D Imaging and Metrology, London, UK
TU Delft	Delft University of Technology (TU Delft), Department of Earth and Climate Sciences & Man-Machine Interaction Group, Delft, The Netherlands
IGN	Institut National de l'Information Géographique et Forestière (IGN), Paris, France
UBO	Université de Bretagne Occidentale (UBO), European Institute for Marine Studies, Brest, France
Ifremer	L'Institut Français de Recherche pour l'Exploitation de la Mer (Ifremer), Brest, France
Liguria	Regione Liguria, Genova, Italy

This document may not be copied, reproduced, or modified in whole or in part for any purpose without written permission from the IQmulus Consortium. In addition to such written permission to copy, reproduce, or modify this document in whole or part, an acknowledgement of the authors of the document and all applicable portions of the copyright notice must be clearly referenced.

All rights reserved.

This document may change without notice.

DOCUMENT HISTORY

Version ¹	Issue Date	Stage	Content and Changes
0.1	2014-04-26	Draft	Early draft sent out for review
0.2	2014-04-27	Draft	Updated draft for review (urban showcase complete, marine showcase complete)
0.3	2014-04-28	Draft	Final draft for review (land showcase complete, document is now consistent)
0.4	2014-04-29	Draft	Review by Heino and Norman
1.0	2014-04-30	Final	Incorporated comments from review, submitted final version

¹ Integers correspond to submitted versions

EXECUTIVE SUMMARY

This document contains the baseline specification of a domain-specific language used to define geospatial processing workflows. We start with a description of our method to create domain-specific languages (DSLs). After that we go through each of the three IQmulus showcases (urban, marine, land) and perform the actual DSL modelling. We create a prototypical grammar that can be used in the control components described in deliverable D3.2 *Control Components – vertical prototype release*, in particular the workflow editor and DSL parser. This document concludes with additional notes on the rationale for the grammar/syntax we chose for our DSL.

‘Big Data handling’ considerations:

The DSL contains high-level and low-level statements.

Using high-level statements users can specify processing workflows in a way that is very easy to read and learn. These kinds of statements are targeted to decision makers or non-GIS experts, but expert users may use them as well if they, for example, need a quick way to perform queries. The high-level statements focus more on the end results instead of on the process. Users do not need to know specifics of the infrastructure the workflow is executed on. In addition, with high-level statements users do not have to know which data is available in the system and what processing algorithms have to be used to produce a certain result. This information/knowledge is generated/deduced by the system itself based on rules specifying declarative and procedural knowledge (see deliverable D3.2 *Control components – vertical prototype release*).

The low-level statements can be used by experts who want to specify exactly the algorithms that should be executed, in which order they should run, as well as on which data these algorithms should operate. Still these expert users do not need to know specifics of the infrastructure (number of cloud nodes, operating system, data location, process service location, etc.). This knowledge is encoded in procedural rules (see deliverable D3.2).

TABLE OF CONTENTS

Executive summary.....	2
1 Introduction.....	4
2 DSL Modelling.....	4
2.1 Related approach.....	5
3 Processing DSL - Baseline specification.....	6
3.1 Urban showcase	6
3.1.1 Vocabulary	6
3.1.2 Domain Model	7
3.1.3 Process Relations	7
3.1.4 Prototype DSL workflow.....	9
3.2 Marine showcase	12
3.2.1 Vocabulary	12
3.2.2 Domain Model	13
3.2.3 Process Relations	13
3.2.4 Prototype DSL workflow.....	14
3.3 Land showcase.....	15
3.3.1 Vocabulary	15
3.3.2 Domain Model	16
3.3.3 Process Relations	16
3.3.4 Prototype DSL workflow.....	18
4 Rationale for the chosen DSL Syntax.....	19
5 References.....	20
6 Appendix: Parsing Expression Grammar for the DSL	20

1 INTRODUCTION

Deliverable D3.2 *Control Components – vertical prototype release* addresses a tool chain that evaluates geospatial processing workflows specified by expert users. That deliverable describes a workflow editor that makes use of a Domain-Specific Language (DSL) to allow users to specify workflows at a very high level without requiring them to have deep knowledge of the specifics of the infrastructure the workflow will be executed on.

In this document we specify the syntax and grammar for the DSL used in the workflow editor. We first describe a method for DSL modelling and then apply it to the three IQmulus showcases (urban, marine, land).

Please note that the DSL specification presented here is a baseline version. This means that it may be subject to change as it is based on the current description of the IQmulus showcases (as of April 2014). An updated final version of this deliverable will be released later (see D2.4.2 *DSL specification – final version* due in April 2015).

Also note that this document only contains the language specification but does not give details on how the language is actually parsed or interpreted. This information can be found in deliverable D3.2 *Control components – vertical prototype release*.

2 DSL MODELLING

Domain-specific languages (DSLs) are languages with the following properties (Krämer & Stein, 2014):

- They are tailored to a specific application domain or even to a single use case;
- The language’s vocabulary contains words well known to the domain expert;
- The language’s expressiveness is rather limited.

The latter means the language cannot be used for general purposes—that is why it is indeed called a domain-specific language—but instead it is a lot easier to understand and to use for non-IT personnel.

The IQmulus workflow editor allows users to specify geospatial processing workflows via a domain-specific language. The DSLs used in IQmulus have the following properties (see deliverable D2.3.1 *IQmulus architecture – baseline version*):

- They are domain-specific in the sense that they are tailored to describing geospatial workflows.
- They are domain-specific in respect to the use case or scenario they are used in (i.e. the three IQmulus showcases for urban, marine and land applications).

In order to create an IQmulus processing DSL that meets these requirements, we use the following modelling method. We start with a typical approach from object-oriented software engineering, consisting of text analysis and domain modelling. From this we then derive DSL sample scripts and finally create a syntax and grammar. In detail, our method is as follows.

1. Analyse the application domain.
2. Create scenarios/storyboards (→ showcase descriptions).
3. Analyse storyboards and look for relevant subjects, objects, adjectives, and verbs. This analysis provides the basis for the domain vocabulary.

4. Create a domain model using subjects and objects found in the storyboards as classes.
5. Analyse process relations to identify relevant verbs which will become actions in the DSL.
6. Build sample DSL scripts based on the modelled domain.
7. Derive formalized grammar and syntax from the sample DSL scripts.
8. Review and reiterate if needed.

The first step (domain analysis) has already been done in the IQmulus project. The results are summarized in the deliverables D1.2.1 *Initial User Requirements* and D1.2.2 *Consolidated User Requirements*.

It is crucial that language modelling is performed in strong collaboration with domain users, in order to make sure the final language contains the vocabulary that is actually used in the targeted domain and can in fact be understood by the domain experts. In the IQmulus project we therefore created a showcase taskforce with the responsibility to create a detailed description of the three showcases urban, land and marine (step 2 of our process). The result of this effort was a document that we analysed in depth (step 3).

2.1 RELATED APPROACH

Krämer and Stein present another modelling method that helps create domain-specific languages which are easy to understand and use (Krämer & Stein, 2014). The method consists of the following steps (cited from the original paper):

1. Analyse the application domain.
2. Create scenarios/storyboards.
3. Analyse storyboards and look for subjects and objects. Create an ontology and use the subjects and objects as concepts.
4. Look for verbs. Use them in the ontology as relations to connect subjects and objects. Free verbs that are not related to concepts become actions in your language.
5. Build sample DSL scripts that use the created ontology and the free verbs.
6. Review and reiterate if needed.

Krämer and Stein use domain ontologies to identify the vocabulary of their DSLs. Such an ontology is a set of concepts (things that exist in that domain) as well as their classification and relations (Nicola, Missikoff, & Navigli, 2009).

The definition of a formal ontology is considered an essential step for domain-specific language design or even for any software project (Gašević, Djuric, & Devedžic, 2009). Ontologies can be very useful for the definition of domain-specific languages where they act as the basis from which the vocabulary and parts of the grammar are derived.

Analysing the IQmulus showcase descriptions results in a large number of concepts and relations. Creating ontologies for all showcases does not help model the IQmulus processing DSL as ontologies are typically rather generic and cannot be translated to actual software code as easy as domain models which are basically tailored for that purpose. The mismatch between the genericity of ontologies and the specificity of domain models has also been recognized by Spyns et al. (Spyns, Meersman, & Jarrar, 2002). They basically claim that, in order to be valuable and reusable, ontologies should be “*as much generic and task-independent as possible*”, which means they should be shareable, portable and interoperable. Ontologies are also often used in the area of the Semantic Web where they act as the basis for Semantic Reasoning and related techniques. This is where the genericity and portability comes in very handy. Domain models on the other

hand are rather specific to a use case or application. For the IQmulus processing DSL, domain models are more appropriate than ontologies as it is not planned to reuse the concepts and relations in any other software. Furthermore, the specificity of domain models allows a direct mapping of concepts (i.e. classes) to actual code (i.e. elements in the DSL).

3 PROCESSING DSL - BASELINE SPECIFICATION

This section describes the process of creating the DSL and the workflows related to the showcases. The DSL contains high-level and low-level statements. Using high-level statements users can specify processing workflows in a way that is very easy to read and learn. These kinds of statements are targeted to decision makers or non-GIS experts, but expert users may use them as well if they, for example, need a quick way to perform queries. Compared to the low-level statements, the high-level ones focus more on the end results instead of on the process. Users do not need to know specifics of the infrastructure the workflow is executed on. In addition, with high-level statements users do not have to know which data is available in the system and what processing algorithms have to be used to produce a certain result. This information/knowledge is generated/deduced by the system itself based on rules specifying declarative and procedural knowledge (see deliverable D3.2 *Control components – vertical prototype release*).

The low-level statements can be used by experts who want to specify exactly the algorithms that should be executed, in which order they should run, as well as on which data these algorithms should operate. Still users do not need to know specifics of the infrastructure (number of cloud nodes, operating system, data location, process service location, etc.). This knowledge is encoded in procedural rules (see deliverable D3.2).

We decided to use the same domain specific language (DSL) for both kinds of statements. This should help to increase the user experience and reduce the learning time.

As described in section 2, we start the DSL modelling by first examining the showcase. After extracting the vocabulary which is used, we build a domain model containing all relevant concepts and have a look at the processing steps and their relations. Using this information we can start to build two workflows using the high-level and low-level statements, respectively. In the following we will use the terms high-level workflow and low-level workflow to differentiate them. When applying the same process to the other showcases, we keep the already created workflows in mind to extend the functionality rather than building everything from scratch again.

3.1 URBAN SHOWCASE

We will analyse the urban showcase first since it involves a couple of techniques which can be reused later on. To exclude moving objects (e.g. cars) and identify different kind of objects (e.g. trees and facades) feature extraction and classification needs to be used. Furthermore change detection is used to differentiate between added, removed and especially deformed objects (e.g. facades).

3.1.1 Vocabulary

Starting with the urban showcase we need to extract all information which helps us to describe the domain and the workflow, respectively. In particular, these are all subjects and objects (concepts), verbs and adjectives which are directly related to the processing scenario.

Concepts

3D Catalogue	Chimney	3D City Model
Urban Topographic Object	Facade Element	Point Cloud
Car	Roof	LMMS Data
Rubbish Bin	Antenna	Image
Non-Static Object	Charge Point	Color
Bike	Topographic Object	Inventory
People	Cable Network	Distance
Static Object	Street Edge	Intersection
Tree	Urban Furniture	Dataset
Bus Stop	Traffic Light	Facade
Phone Booth	Topographic Map	Change Detection
Change	Addition	Removal
Deformation		

Verbs

update	take out	identify
remove	compare	determine
include	store	co-register
exclude	give	colorize
monitor	save	apply
foresee	visualize	select

Adjectives

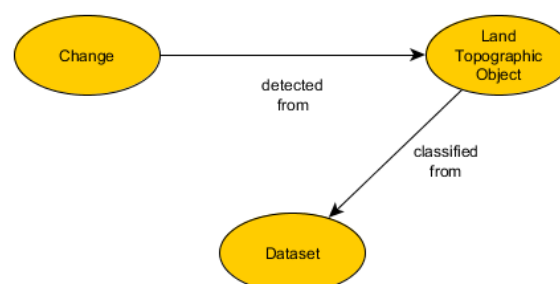
certain	uncertain	clean
stochastic	recent	added
deformed		

3.1.2 Domain Model

Using the above-created list of concepts allows us to model the domain with all its entities and relations (see Figure 1, next page). The result allows us to have a clear insight for all elements which might be involved in the processing workflow.

3.1.3 Process Relations

The examination of the urban showcase reveals two main processing issues, namely feature extraction and change detection. The first one is used to find and classify topographic objects from the source data, while change detection uses the result and a formerly detected set of objects from the same area to produce a list of the changes that occurred.



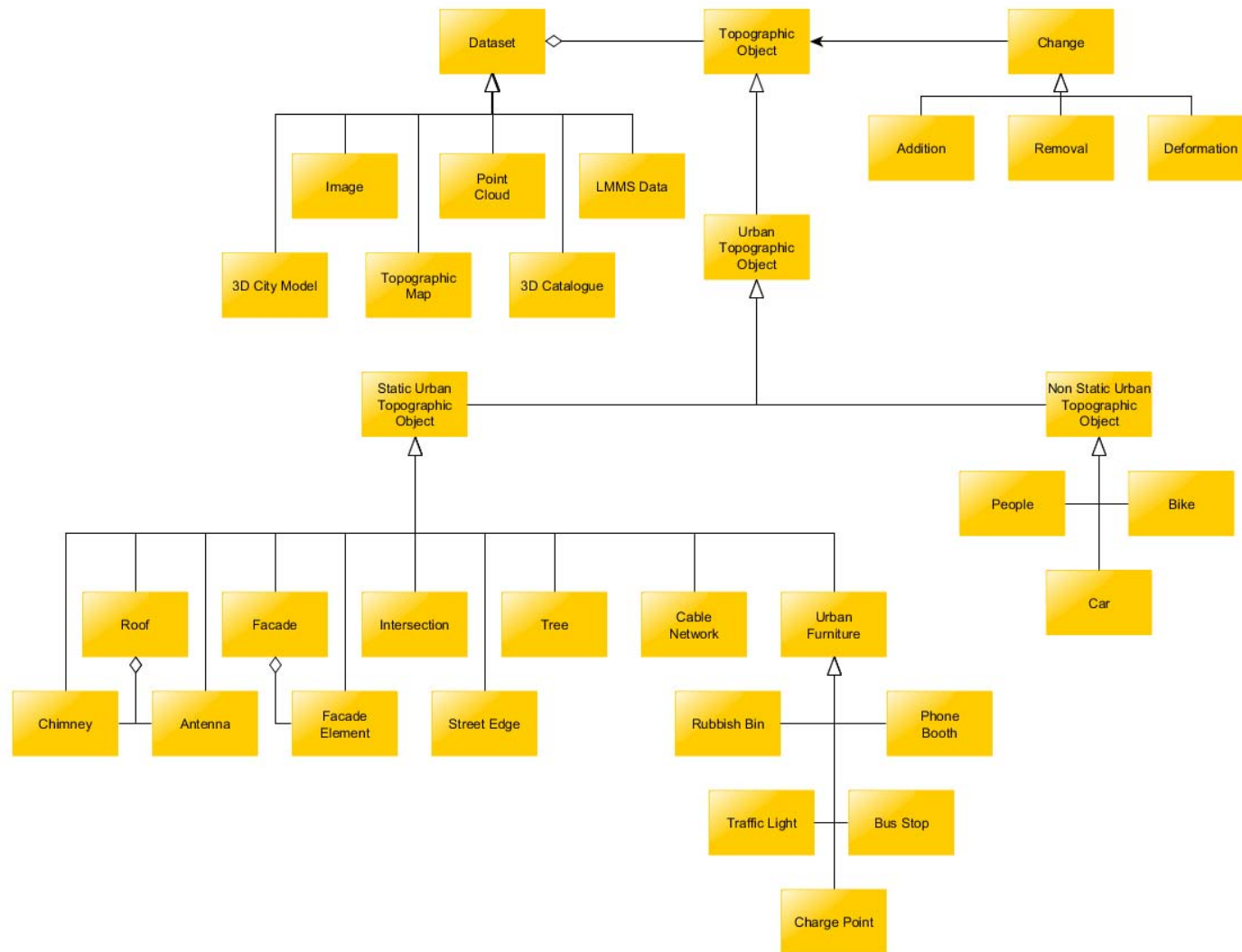


FIGURE 1 DOMAIN MODEL FOR THE URBAN SHOWCASE

3.1.4 Prototype DSL workflow

Starting with the high-level workflow, we can derive a couple of terms from the vocabulary and the domain model, which has been created earlier. Actually, there are three categories of words so far. The term *concept* is used to describe a group of entities or even a particular entity of the domain like *Topographic Object* or *Bike*, while an *adjective* is directly related to such an object and its properties, respectively. Obviously a *verb* describes an action which should be applied to a *concept*. At this point we introduce a fourth category of words containing *keywords* of the language itself. These keywords are used to make the language more readable and provide additional, domain-independent functionality.

To create the first workflow which aims to describe the processing of the urban showcase, we need to select a few concepts, adjectives and verbs, which are listed in the following table.

Concept	Adjective	Verb
Data	recent	exclude
CityModel	added	select
NonStaticObjects	deformed	add to
Trees		visualize
FacadeElements		
Antennas		

Furthermore we introduce the following keywords to increase usability and readability of the workflow.

Keyword	Description
with [...] [...]	Use the first term and apply it to the second.
do [...] end	A block of statements, where all of them are executed in the same context. A context might contain values which will be applied to all statements sequentially.
[...] and [...]	A simple enumeration of elements. If there are more than two elements, “and” is typically used in front of the last element only, while all prior elements are separated using a comma.

While *with* and *and* are simply terms to enhance the structure of the language, and hence write- and readability, *do [...] end* is providing more complex functionality. Every block statement created by these *keywords* relates to a context which might envelope a value, i.e. if used in combination with the keyword *with*. When the block is executed, the value stored in the context will be applied to the first statement if it expects a parameter of the same type. If this statement returns a result the content of the context will be overwritten by it. All subsequent statements will be processed in the same way using the updated context.

Using these terms we are able to create a simple workflow.

```
with recent Data do
    exclude NonStaticObjects
    select added Trees and deformed FacadeElements
    add to recent CityModel
end

with recent CityModel do
    exclude Antennas
    visualize
end
```

In this and all further listings, the terms are colored according to their category: Light blue is used for keywords, purple for concepts, orange for adjectives and red for verbs.

The first block has been defined with the keyword *with* to use the most *recent Data* as an implicit value. Therefore the first statement excludes all *NonStaticObjects* from that data and replaces the implicit value stored in the context with the result. Therefore all subsequent statements will use the manipulated one. The *select* statement accepts a list of concepts or to be precise *Topographic Objects* and selects only those from the data set, in this case given by the implicit value. The last statement will add the result of the prior statement, i.e. *select*, to the most *recent CityModel*.

The second block works similar to the first one. The first statement excludes all *Antennas* from the most *recent CityModel*, while the second visualizes the result.

Since a workflow should describe a process which should be applied not only once on a particular dataset, there has to be a way to design it more flexibly. In order to reapply a workflow to a number of different datasets, there has to be a dynamic way to address it. A simple way to do so is to introduce placeholders. These placeholders need to be replaced just before the workflow is executed. Hence these placeholders can be considered as parameters for the workflow itself.

Applying this to the former example, we end up with a workflow similar to the following.

```
with [Area] do
    exclude NonStaticObjects
    select added Trees and deformed FacadeElements
    add to [CityModel]
end

with [CityModel] do
    exclude Antennas
    visualize
end
```

The placeholder *Area* will be used to let the user specify an area of interest before executing this particular workflow. As a result, the JobManager from the processing chain (see deliverable D3.2 *Control components – vertical prototype release*) has to select all data which is needed to represent this area prior to execution. Analogously the placeholder *CityModel* can be used to specify a city model which is updated with the added trees and deformed facade elements.

To specify a low-level workflow there is another category of terms required which maps to the list of *services*. Therefore this category does not have a specific number of terms, but as many terms as services available in the IQmulus infrastructure. These terms are colored in green.

Keyword	Description
[...] with [...]	The same like <i>with</i> [...] [...] but it applies the second statement to the first. If there is an applicable implicit value it will be combined with the result of the second statement.
[...] using [...]	Apply key-value pairs using the form <i>key: value</i> defined by the second statement to the first one. It is even possible to use the form { <i>key1, key2, ...: value1, value2, ...</i> } which might be more convenient in some situations.
[...] as [...]	The result of the first statement will be stored and accessible in the future using the specified term.
[...] of [...]	Create a subset of the second statement limited by the first one.

```

apply Intersection3D with recent DataSets
  using {x, y, z: [X], [Y], [Z]} and
    {notX, notY, notZ: [NX], [NY], [NZ]}
  as data
with [Area] of data do
  apply MultiObjectClassification
  exclude certain NonStaticObjects
  select certain Trees as trees
  select certain FacadeElements as facadeElements
  join trees and facadeElements as treesAndFacades
  apply StochasticChangeDetection
    with [CityModel] and treesAndFacades
    using cv: [CriticalValue]
    as changes
  select added Trees with changes
    as newTrees
  select deformed FacadeElements with changes
    as newFacadeElements
  apply Intersection3D with newTrees
    using {x, y, z: [X], [Y], [Z]} and
      {notX, notY, notZ: [NX], [NY], [NZ]}
  apply Intersection3D with newFacadeElements
    using {x, y, z: [X], [Y], [Z]} and
      {notX, notY, notZ: [NX], [NY], [NZ]}
  store
end

with [CityModel] do
  apply MultiObjectClassification
  exclude all Antennas
  visualize
end

```

There are just a few additional adjectives and verbs needed for this workflow: *certain* is used to limit a set of classified objects, while *all* allows it to select even the objects which are suspected to be of a certain kind. The verbs *store*, *join* and *apply* are used to store a particular dataset back to the distributed file system, join two sets and call a particular service, respectively.

Concept	Adjective	Verb
	certain	store
	all	join
		apply

The resulting low-level workflow actually provides the same functionality as the high-level workflow introduced earlier. An expert user, however, gets a lot more control on the processing.

3.2 MARINE SHOWCASE

The purpose of this showcase is to generate a digital terrain model of the sea ground, while dealing with a wide variety of different data sources. An important processing step in this scenario is deconfliction to merge the different datasets properly.

3.2.1 Vocabulary

To get the relevant concepts, verbs and adjectives used to describe the marine show case, we have to apply the same process like we did for the urban showcase.

Concept

Sea Bottom	Geological Process	Anthropic Impact
Water flow	Sand Dune	Surface
Human Intervention	Wreck	Digital Terrain Model (DTM)
Time Source	Space Source	Changes over time
Features	Sea Charts	Survey Quality
Survey Number	Deconfliction	Survey
Survey Coverage	3D Data	Digital Elevation Model
Interactive Deconfliction	Change detection	3D Data Acquisition
Point Cloud	Shape Representation	Data Set
Sea Bottom Data Set	Comparison Result	LR-Spline Surface
Visualization	Triangulation	Shape
Spline	Rock	Point Set
Outcrops	Crater	Structure
Glacier	Tile	Threshold
Surface Tolerance	Polynomial Cell Width	Tolerance
Uncertainty	Pattern	Feature Detection
Sand Bank	Stone	LR-Spline Approximation
Pipeline	Cluster of Points	Iceberg
Drag Marks	Outline	LiDAR Data
Sonar Data		

Verb

Track	Detect	Gather
Apply	Visualize	Perform

Validate	Compare	Restrict
Reproduce	Produce	Identify
Select		

Adjective

various	old	single
multiple	multi-temporal	smooth
human made	rocky	minimal
wave like	waviness	below
long	narrow	recent

3.2.2 Domain Model

When modelling the domain of the marine showcase we kept the domain model of the urban showcase in mind: Not to influence the entities and their relations, but to reuse objects of the former model if applicable. The result actually looks quite similar to the urban model which is reasonable as both showcases use datasets which contain information about an environment and the objects which can be found in it. The source of the data might be considered as a specific entity related to the general concept *Dataset*.

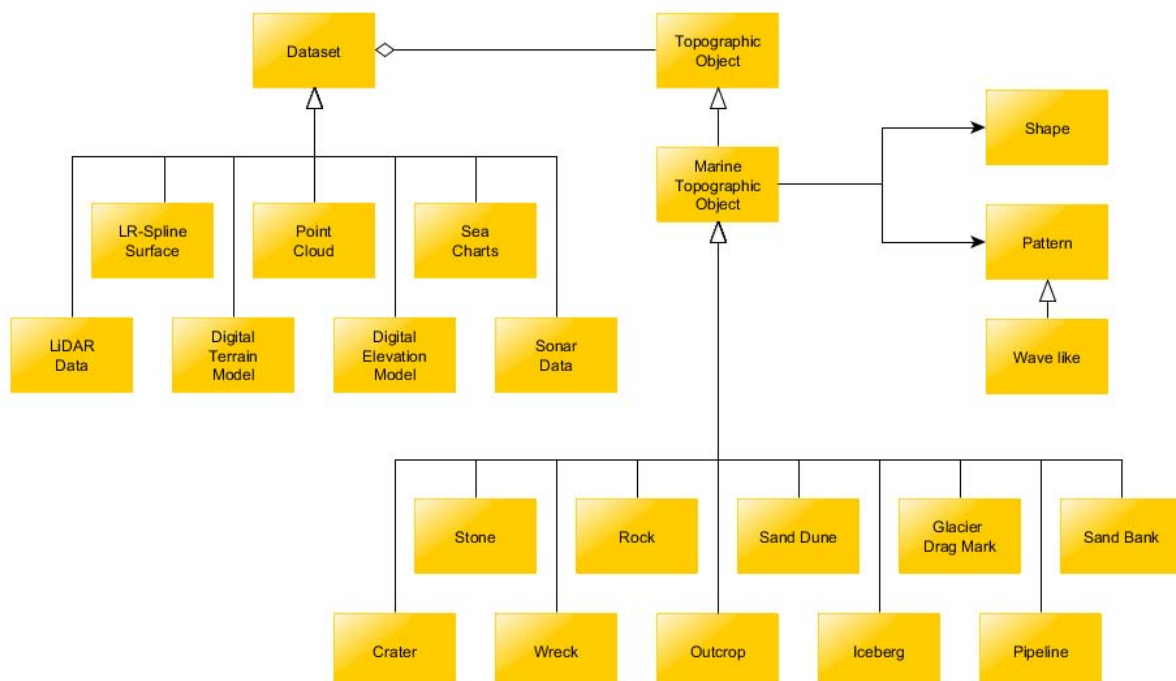
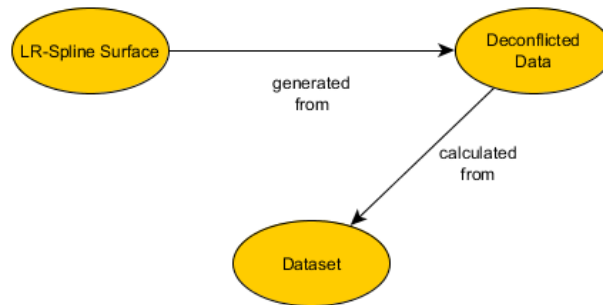


FIGURE 2 DOMAIN MODEL FOR THE MARINE SHOWCASE

3.2.3 Process Relations

The processing steps in the marine showcase, apart from visualization, can be described as deconfliction and surface generation. Therefore the overall workflow should use the source data for deconfliction first, which results in a set of deconflicted data. Hereupon, the result can be used to generate an *LR-Spline Surface*.



3.2.4 Prototype DSL workflow

Since the domain model of the marine showcase looks quite similar to the one of the urban showcase, we are even able to create a similar workflow by using the already introduced language elements. There are still a few different requirements compared to the urban workflow.

Working in a marine scenario usually requires taking more data sources into account. While an urban scenario is mainly based on *LiDARData* captured by car or airplane, the data which is used in marine scenarios can result amongst others from *SonarScans* or *SeaCharts*. As a consequence of the different nature of data, there are datasets which might be more up-to-date, while others are more accurate.

The workflow has to consider this diversity, in particular when it comes to deconfliction and surface generation, respectively. Using the keyword *of* in the high-level workflow, which has been introduced in Section 3.1.4 for the low-level workflow, allows the user to choose a particular kind of data for the further process. There are actually two ways to specify the data. Adjectives like *recent* or *high-quality* may be used to select data which fulfil these properties while not taking into account the source. On the other hand the use of concepts like *SeaCharts* or *SonarData* lets the user select data by source or type.

```

with [Area] of recent SeaCharts do
  generate Surface using threshold: 42
  and target: [SurfaceTarget]
  visualize
end
  
```

The workflow should proceed with the surface generation and all steps which are necessary for that. For this purpose the verb *generate* is introduced. *generate* is used to get a particular concept from a set of input data, where the processing exceeds a simple transformation. The keyword *using*, which has been defined in section 3.1.4, can be reused in the high-level workflow to append parameters to the generation process. In this case a threshold needs to be set in order to do deconfliction, and a target dataset has to be specified if the result is to be stored.

The following table lists concepts, adjectives and verbs which are used to build the high-level marine workflow so far.

Concept	Adjective	Verb
SeaCharts	recent	generate
Surface		visualize

The low-level workflow which describes the same processing steps might look like the following. There is no need to introduce additional terms or categories but we introduce a new parameter for the verb *store* which allows specifying the target which is used in this case to store the surface.

```
with recent SeaCharts do
  apply Deconfliction using threshold: [Threshold]
  apply Intersection3D
    using {x, y, z: [X], [Y], [Z]} and
      {notX, notY, notZ: [NX], [NY], [NZ]}
  as data
  apply SurfaceGeneration
  store using target: [SurfaceTarget]
  visualize
end
```

3.3 LAND SHOWCASE

The third showcase describes a scenario in which satellite images are processed to support flood and waterlogging analysis and prediction. After the images have been merged and pre-processed, waterlogging classification is applied to find and visualize the flooded areas.

3.3.1 Vocabulary

We extracted all needed terms for the urban and marine showcase earlier. The same investigation has to be done for the land showcase.

Concept

Event	Landslide	Model
Landslide Event	Flood	Simulation
Flooding Event	Precipitation	Reference Measurement
Critical Event	Images	Cloud mask
Slope deformation	Optical Images	Top of the atmosphere
Spectral indices	Satellite Images	ToA reflectance
NDVI	Cloud shadowing	Cloud
NDSI	Natural Water Mask	River
NDWI	Lake	Land Parcel Identification System (LPIS)
Natural Waters	Waterlog	Soil
Seriously Affected	Moderately Affected	Weakly Affected
Vegetation in waterlog	Dry areas clouds	No supported areas
Threshold	Surface Representation	Interpolation

Verb

analyze	predict	monitor
compare	obtain	produce
compute	preprocess	transform
calibrate	filter	reproject
calculate	process	classify

Adjective

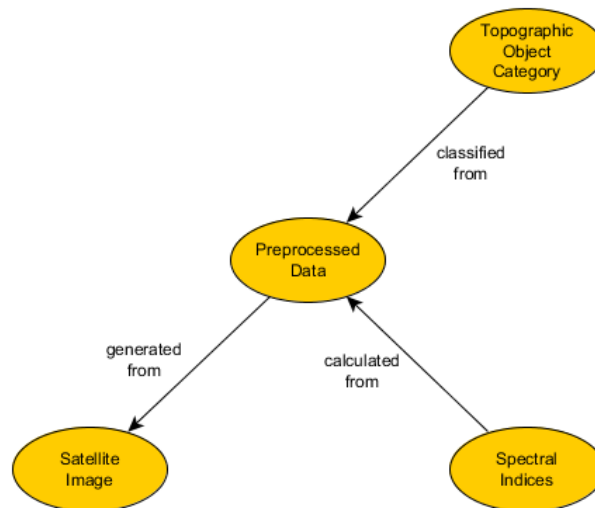
High-quality	landslide	flooding
mechanical	Hydrological	geometric
calibrated	Moderately affected	

3.3.2 Domain Model

Again, using the concepts from the description of the land scenario allows us to create the domain model which is built considering the two models already created for the urban and marine showcase (see Figure). The entities of Dataset and Topographic Object, their descendants and their relation hardly changes from the former ones, except for the domain specific entities, namely the land topographic object and its children. On the other hand, the new entity *Topographic Object Category* is introduced which can be used to group different *Land Topographic Objects*. This is used in particular to create a *Thematic Map* in this showcase. Furthermore a *Topographic Object Category* might be associated with an Event which itself can be associated to a set of changes.

3.3.3 Process Relations

The land showcase basically contains three major processing steps. First, the selected satellite image needs to be pre-processed, which includes top of atmosphere reflection calibration and cloud masking. The result can be used to calculate spectral indices and consequently the topographic object categories can be found and classified.



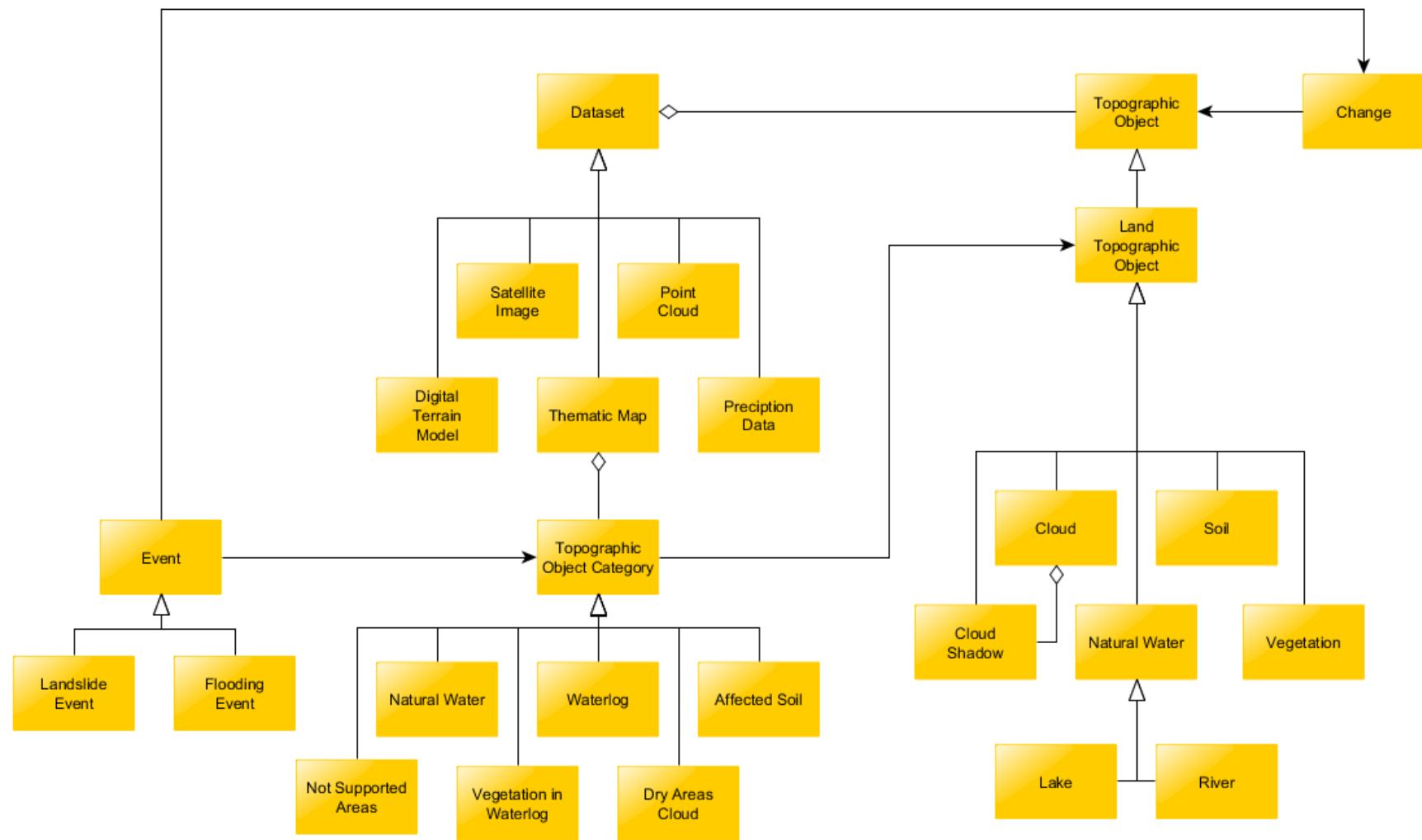


FIGURE 3 DOMAIN MODEL FOR THE LAND SHOWCASE

3.3.4 Prototype DSL workflow

Creating a simple high-level workflow from the showcase includes a lot of terms which have been introduced earlier. The data source is selected in the same way as in the former workflows and the verb *generate*, which has been introduced in section 3.2.4, will be used to generate the *ThematicMap*. A new verb which needs to be specified here is *colorize*. This one allows applying a user defined colour to an entity of the domain. In this scenario all objects associated with the category *Waterlog* will be colorized.

```
with [Area] of recent SatelliteImage do
  generate ThematicMap
  colorize Waterlog [WaterlogColor]
  visualize
end
```

All terms, apart from the language specific ones, used in this workflow are listed in the following table.

Concept	Adjective	Verb
SatelliteImage	recent	generate
ThematicMap		colorize
Waterlog		visualize

As we have seen in section 3.1.4 and section 3.2.4, a low-level workflow is mainly built from a chain of services, with the proper selection of parameters, which are called consecutively. All needed terms and keywords for this purpose have been introduced in the former sections already.

```
with [Area] of recent SatelliteImage do
  apply RasterDataPreprocessing
  apply SpectralIndicesComputation
    using si: [SpectralIndices] and wl: [WaveLengthInfo]
    as indices
  apply ThematicClassification
    using indices: indices
  apply Coloring
    using category: Waterlog and color: [WaterlogColor]
  visualize
end
```

Basically, this workflow selects the most recent satellite image which can be found for a given area and uses the service *RasterDataPreprocessing* to do pre-processing. After calculating the spectral indices and using them to create the thematic map the category *Waterlog* will be colorized in the same way like in the high-level workflow.

4 RATIONALE FOR THE CHOSEN DSL SYNTAX

In the previous sections we presented sample DSL workflows based on the vocabulary from our domain models. Even though the three showcases use different domain models and hence different terms in the DSL workflows, the syntax of all sample scripts looks quite similar. Identifying this common syntax was an additional task that we performed in order to create a DSL that looks similar in all showcases and that hence is easier to learn.

We tested various syntaxes, starting with the following:

```
with DataSet
exclude NonStaticObjects from it
select added Trees and deformed FacadeElements from it
add it to CityModel
```

In this sample script we used the keyword 'it' to refer to an object from the previous line or to refer to the result of the process performed in the previous line. This context-sensitive approach requires an intelligent parser that is able to clearly identify what 'it' means in the respective context. However, we realized that even human users have problems understanding what 'it' actually means, so we decided to remove this keyword from our language.

Next we created the following syntax:

```
exclude NonStaticObjects from DataSet
and select added Trees and deformed FacadeElements
and add to CityModel
```

In this case individual processing steps are connected with the 'and' keyword. While this approach leads to unambiguous workflows, the scripts can very quickly become hardly readable. The longer the workflows get, the harder it is to understand them as the sentences become longer and longer. We hence decided to introduce blocks using the 'with ... do' and 'end' keywords, to make clear which processing steps affect which data set.

```
with recent Data do
    exclude NonStaticObjects
    select added Trees and deformed FacadeElements
    add to recent CityModel
end
```

This syntax is very readable and can be applied to all three showcases as shown in the previous sections.

5 REFERENCES

Gašević, D., Djuric, D., & Devedžić, V. (2009). *Model Driven Engineering and Ontology Development* (2nd ed.). Springer.

Krämer, M., & Stein, A. (2014). Automated urban management processes: integrating a graphical editor for modular domain-specific languages into a 3D GIS. *Proceedings of the 19th international conference on urban planning and regional development in the information society GeoMultimedia*.

De Nicola A., Missikoff M., & Navigli R. (2009): A software engineering approach to ontology building. *Information Systems, Volume 34 Issue 2* (pp. 258-275).

Spyns, P., Meersman, R., & Jarrar, M. (2002). Data modelling versus ontology engineering. *ACM SIGMOD Record, Volume 31 Issue 4* (pp. 12-17). New York, NY, USA: ACM.

6 APPENDIX: PARSING EXPRESSION GRAMMAR FOR THE DSL

```

start
  = SP* statements SP*

statements
  = statement ( SP+ statement )*

statement
  = block
  / process
    ( SP+ "with" SP+ dataset_expression
      ( SP+ "and" SP+ dataset_expression )*
    )?
    ( SP+ "using" SP+ params )? ( SP+ "as" SP+ ID )?

block
  = with SP+ statements SP+ "end"

with
  = "with" SP+ dataset_expression SP+ "do"

dataset_expression
  = dataset ( SP+ "of" SP+ dataset )?

dataset
  = "recent" SP+ ID
  / "latest" SP+ ID
  / placeholder
  / ID

params
  = param ( SP+ "and" SP+ param )*
```

```

param
  = "{" SP* tuple_ids SP* ":" SP* tuple_expression SP* "}"
  / ID SP* ":" SP* expression

tuple_expression
  = expression ( "," SP* expression ) *

expression
  = placeholder
  / INTEGER
  / string
  / ID

tuple_ids
  = ID ( "," SP* ID ) *

placeholder
  = "[" ID "]"

string
  = "'" string_character* "'"
  / '"' string_character* '"'

string_character
  = !["\\r\\n"] .
  / "\\\" ESCAPE_CHARACTER

INTEGER
  = [0-9]+

ESCAPE_CHARACTER
  = ['"\\bfnrtv]

ID
  = [_a-zA-Z][_a-zA-Z0-9]*

SP
  = [ \\t\\n\\r]

process
  = "apply" SP+ ID
  / "store"
  / "visualize"
  / urban_process
  / marine_process
  / land_process

urban_process
  = "exclude" ( SP+ urban_dataset_param )? SP+ ID
  / "join" SP+ ID ( SP+ "and" SP+ ID ) *
  / "select" SP+ urban_dataset_param SP+ ID ( SP+ "and" SP+
urban_dataset_param SP+ ID ) *
  / "add" SP+ "to" SP+ dataset

```

```
urban_dataset_param
  = "added"
  / "all"
  / "certain"
  / "deformed"

marine_process
  = "generate" SP+ ID

land_process
  = "colorize" SP+ ID SP+ dataset
```