
# PROCESSING DSL SPECIFICATION - FINAL VERSION

## Deliverable D2.4.2

| | |
|---|---|
| Circulation: | PU: Public |
| Lead partner: | Fraunhofer |
| Contributing partners: | |
| Authors: | Michel Krämer (Fraunhofer, v2.0, v1.0), Ivo Senner (Fraunhofer, v1.0) |
| Quality Controllers: | Stephan Plabst (MOSS, v2.0), Heino Rudolf (MOSS, v1.0), Norman Kießlich (MOSS, v1.0) |
| Version: | 2.0 |
| Date: | 30.04.2015 |

Consisting of

| | |
|---|---|
| SINTEF | STIFTELSEN SINTEF, Department of Applied Mathematics, Oslo, Norway |
| Fraunhofer | Fraunhofer Institute for Computer Graphics Research (IGD), 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 GrafikSysteme 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 |
| TUDelft | Delft University of Technology (TUDelft), Department of Geoscience and Remote Sensing & Man-Machine Interaction Group, Delft, The Netherlands |
| IGN | Institut National de l'InformationGéographique et Forestière (IGN), Paris, France |
| UBO | Université de Bretagne Occidentale (UBO), European Institute for Marine Studies, Brest, France |
| Ifremer | L'InstitutFrancais de Recherche pour l´Exploitation de la Mer (Ifremer), Brest, France |
| Liguria | Regione Liguria, Genova, Italy |

## DOCUMENT HISTORY (D2.4.1 PROCESSING DSL SPECIFICATION – BASELINE VERSION)

| 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 |

## DOCUMENT HISTORY (D2.4.1 PROCESSING DSL SPECIFICATION – FINAL VERSION)

| Version | Issue Date | Stage | Content and Changes |
|---|---|---|---|
| **2.0** | 2015-04-20 | Draft | D2.4.1 → D2.4.2 (see list of changes below) |
| **2.0** | 2015-04-29 | Draft | Review by Stephan Plabst (MOSS) |
| **2.0** | 2015-04-30 | Final | Final version |

## LIST OF CHANGES IN COMPARISON TO D2.4.1 PROCESSING DSL SPECIFICATION – BASELINE VERSION

| Section | Page | Change |
|---|---|---|
| | 3 | Updated executive summary. Added section about the differences between D2.4.1 and its update D2.4.2 |
| 1 **Introduction** | 5 | Updated introduction. Removed paragraph about baseline version of the specification and added reference to D3.4.1. |
| 3.1 **Integrated Urban showcase** | 7 | Renamed section to 'Integrated Urban Showcase' instead of 'Urban showcase' to use the same terminology as in deliverable D1.2.3. |
| 3.2 **Integrated Marine showcase** | 13 | Renamed section to 'Integrated Marine Showcase'. |
| 3.3 **Integrated Land showcase** | 16 | Renamed section to 'Integrated Land Showcase'. |
| 5 **Applying the language to individual showcase workflows** | 21 | Added new section summarizing how we applied the language to individual showcase workflows as described in deliverable D1.2.3. |
| 7 **Appendix: Parsing Expression Grammar for the DSL** | 23 | Updated grammar specification to the latest version implemented in the interpreter. |

[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.

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).

**Updated deliverable for the final language specification (PM30)**

The deliverable has been updated in PM30 to reflect the final specification of the IQmulus domain-specific language (DSL). The first version of the deliverable (i.e. D2.4.1) already provided a very useful specification. We were able to implement all showcase workflows that we were planning to implement up to PM30. For the update (D2.4.2) we therefore had to make only slight changes. The majority of the document, however, was left unchanged.

The complete list of changes compared to previous versions of this document can be found on page 2 above. In short, we revised the document to make it consistent to other deliverables that have been released since the previous version. We also added a new section showing how we apply the language specification to different showcase workflows to demonstrate the use of the DSL in the context of the project.

# TABLE OF CONTENTS

# 1 INTRODUCTION

Deliverable D3.2 *Control Components – vertical prototype release* addresses a tool chain that evaluates geospatial processing workflows specified by expert users. It 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 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*. In addition this deliverable does not give information on how the workflows written with the specified language are actually executed. This process can be found in deliverable D3.4.1 *Integration components.*

# 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 (Gaševic, 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 INTEGRATED URBAN SHOWCASE

We will analyse the integrated 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

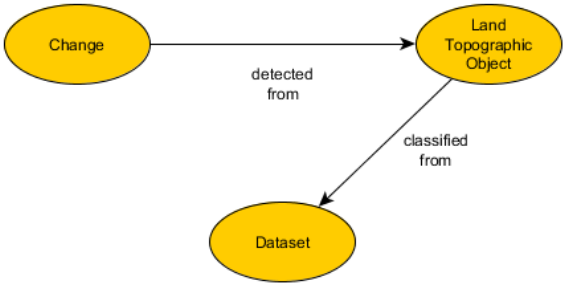
| | | |
|---|---|---|
| certain | uncertain | clean |
| stochastic | recent | added |
| deformed | | |

## 3.1.2 Domain Model

Using the above-created list of concepts we can 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 previously detected set of objects from the same area to produce a list of the changes that occurred.



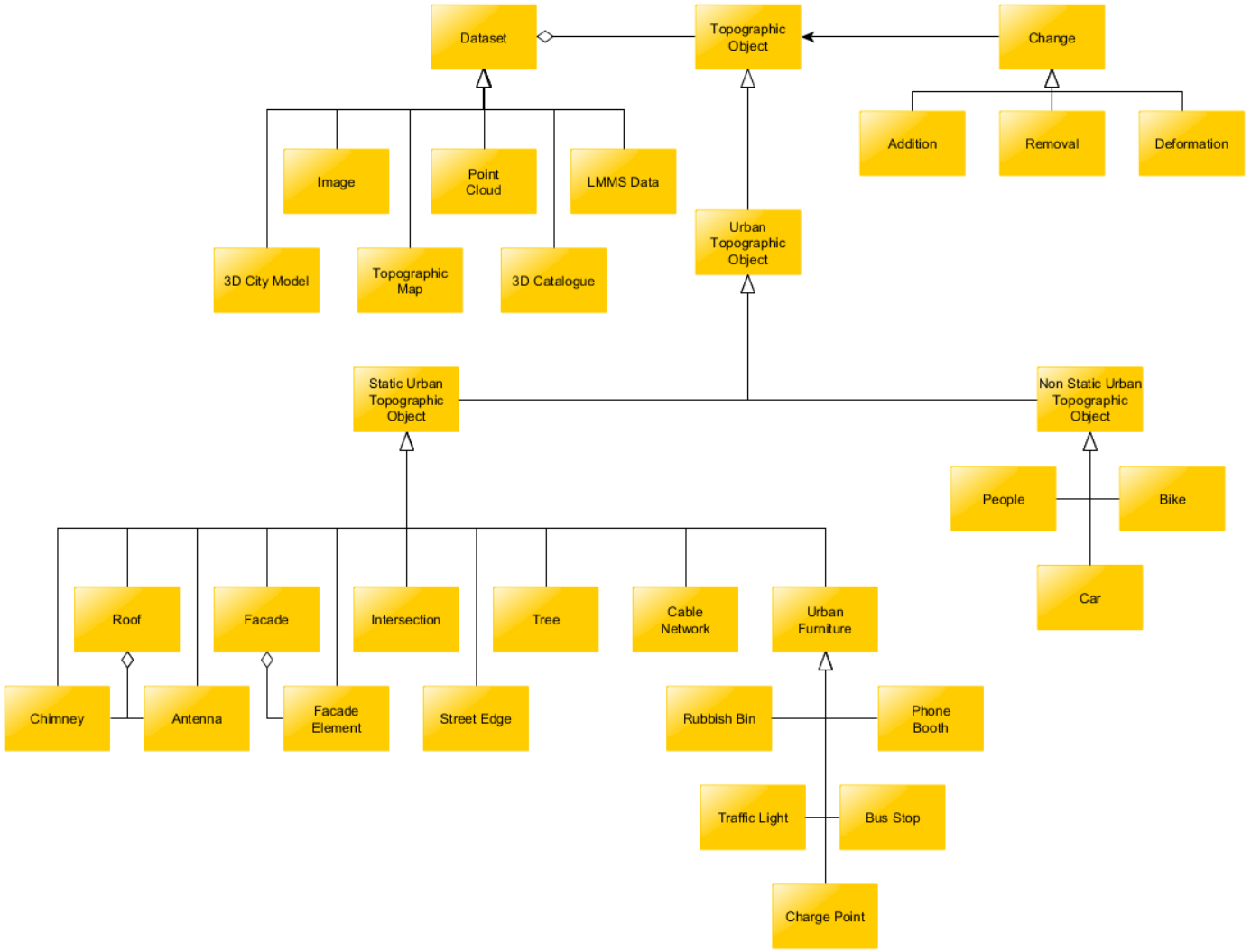



FIGURE 1 DOMAIN MODEL FOR THE URBAN SHOWCASE

### 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 in the previous section. 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. 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 writeability 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, e.g. when 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 are then 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 colorized 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. All subsequent statements will then 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 to not only one 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 viewed as parameters of the workflow itself.

Applying this to the previous 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 that 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 coloured in green.

| Keyword | Description |
|---|---|
| [...] with [...] | The same like *with [...] [...]* 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 *key: value* defined by the second statement to the first one. It is even possible to use the form *{key1, key2, ...: value1, value2, ...}* 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 write 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 INTEGRATED MARINE SHOWCASE

The purpose of this showcase is to generate a digital terrain model of the seabed, 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 | Antrophic 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*.

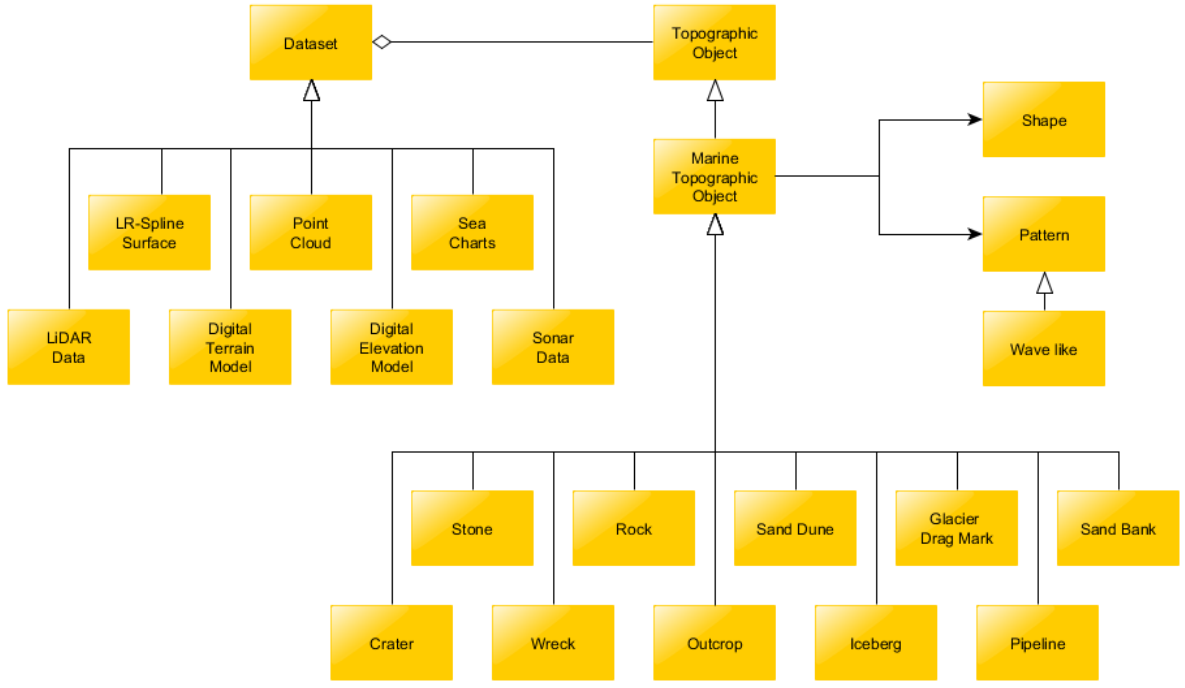
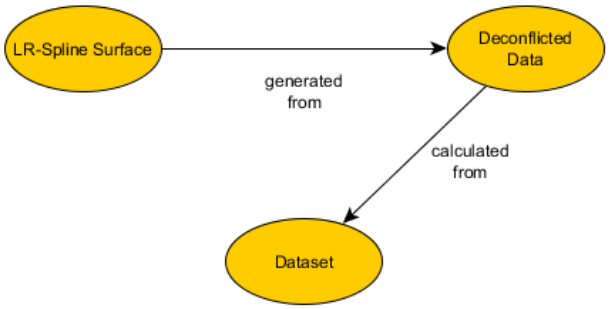



FIGURE 2 DOMAIN MODEL FOR THE MARINE SHOWCASE

### 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 previously introduced language elements. There are, however, 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. 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 processing. 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 INTEGRATED 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 was 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

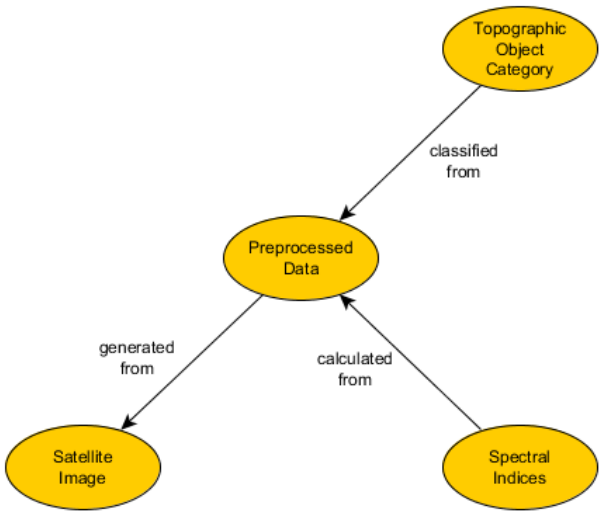
| | | |
|---|---|---|
| High-quality | landslide | flooding |
| mechanical | Hydrological | geometric |
| calibrated | Moderately affected | |

## 3.3.2 Domain Model

Again, using the concepts from the description of the integrated land scenario allows us to create the domain model which is built considering the two models already created for the integrated urban and marine showcases. The entities of Dataset and Topographic Object, their descendants and their relation hardly changes from the previous models, 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 with 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.



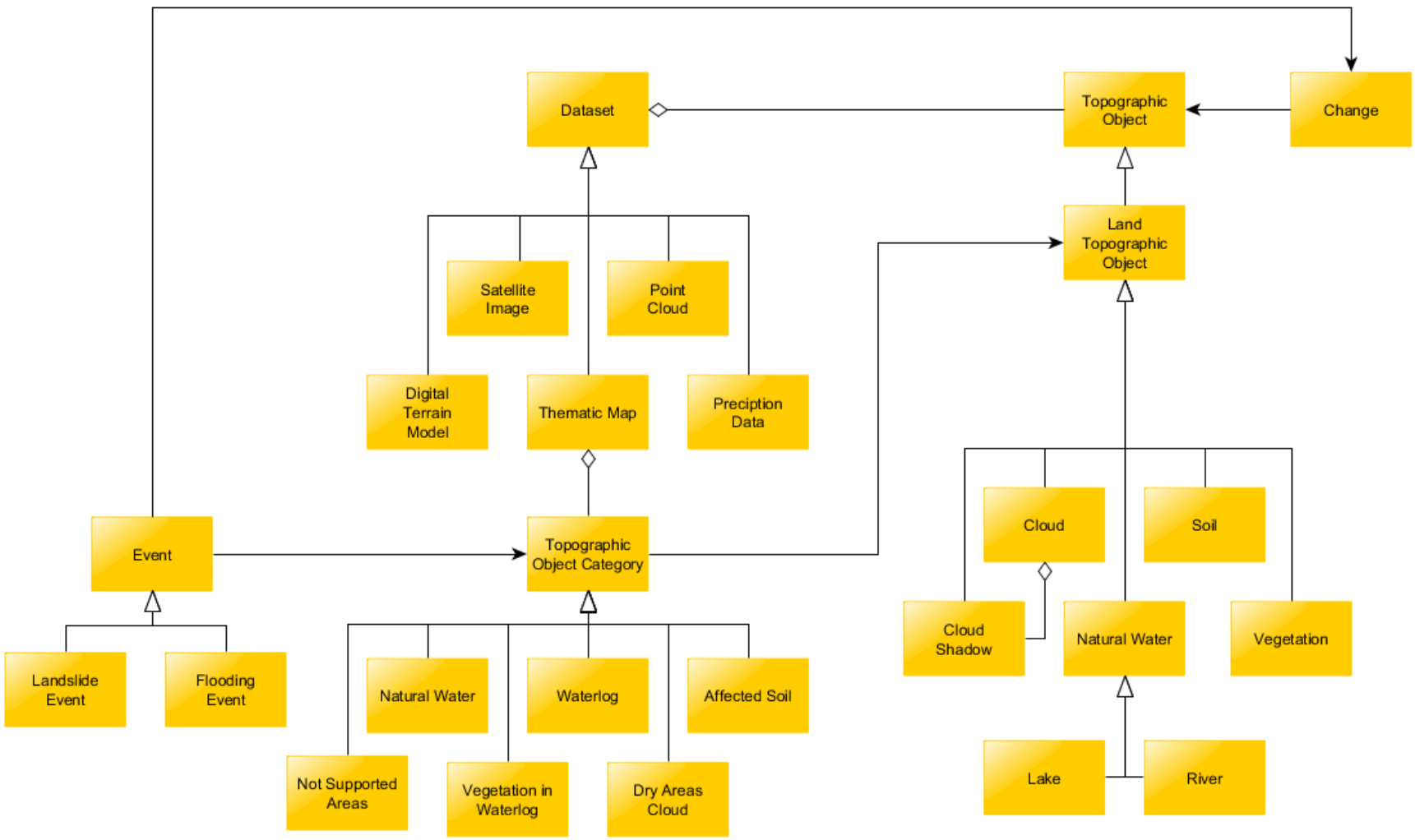



FIGURE 3 DOMAIN MODEL FOR THE LAND SHOWCASE

### 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 previous 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 previous 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 APPLYING THE LANGUAGE TO INDIVIDUAL SHOWCASE WORKFLOWS

In the following we describe how we applied the language specified in this document to individual showcase workflows to demonstrate its usage in practical examples. Please note, that we can only give examples here. The individual workflows can be written in the DSL in various ways. Giving DSL scripts for all showcases and all workflows is out of the scope of this specification document. We chose examples that cover most aspects of the specified language.

## 5.1 URBAN WORKFLOW US1

In this workflow three datasets should be analysed: a 3D city model, a digital terrain model (DTM) and a digital surface model (DSM). The aim is to identify changes (i.e. new buildings, changed buildings or deleted buildings) and to store these changes in three different files.

The following example demonstrates how this workflow can be implemented with the DSL:

```
with latest Buildings do
     apply BuildingChangeDetection
          with [DTM] and [DSM]
          using tolerance: 10
          as changes
     store changes.newBuildings
     store changes.deletedBuildings
     store changes.changedBuildings
end
```

The workflow selects the most recent dataset from the distributed file system containing buildings. At this point the user might also have inserted a placeholder to select a specific dataset instead of having the system figure that out. The workflow then applies the first step “BuildingChangeDetection” to the selected dataset and two other datasets specified through the placeholders [DTM] and [DSM]. The step has a parameter ‘tolerance’ with a value of ‘10’. The result of the step is saved in the variable ‘changes’. This variable is an object containing three items: newBuildings, deletedBuildings and changedBuildings. The workflow saves each item in a separate file in the distributed file system.

The term ‘BuildingChangeDetection’ is internally mapped to a number of different processing services. The mapping between DSL terms and processing services is described in deliverable D3.2. The example is considered high-level since the user does not need to know about the actual processing services involved.

## 5.2 MARINE WORKFLOW MS4

In this workflow two seabed surveys from different years should be compared to each other. The difference should be saved in a separate file.

In the following example, we chose the low-level way of describing the workflow. Each service is directly addressed using the ‘apply’ keyword. Also the actual datasets are directly specified by the user through placeholders.

```
apply SplineInterpolation
      with [PointCloud2010]
      using tolerance: 0.3
            and iterations: 6
      as surface2010

apply TrimSurfaceWithPoints
      with surface2010 and [PointCloud2010]
      using tightness: 7
      as trimmedSurface2010

apply SplineInterpolation
      with [PointCloud2011]
      using tolerance: 0.3
            and iterations: 6
      as surface2011

apply TrimSurfaceWithPoints
      with surface2011 and [PointCloud2011]
      using tightness: 7
      as trimmedSurface2011

calculate Displacement
      with trimmedSurface2010 and trimmedSurface2011

store
```

At first, the workflow prepares the input datasets by applying the 'SplineInterpolation' and 'TrimSurfaceWithPoints' services. Finally the workflow calculates the 'Displacement' (i.e. the difference between the both surfaces) and stores the resulting displacement in the cloud.

Although this example is low-level (i.e. the services are addressed directly), internally there are more services involved. The service 'TrimSurfaceWithPoints' generates a spline surface. However, the 'Displacement' service can only handle grids. Internally, an additional service is inserted by the interpreter (see deliverable D3.2) converting the two spline surfaces to grids before they are passed to the 'Displacement' service. The example is therefore a mix of low-level and high-level DSL statements as the user does not need to take care of data formats.

## 5.3 LAND WORKFLOW LS3

The land workflow LS3 describes how satellite imagery needs to be processed in order to detect flooded areas. It is implemented as follows:

```
with [SatelliteImage] do
      apply ImageEnhancement
            using factor: 2.0 and offset: 0.0
                  and method: "toa-reflectance"
      apply FloodedAreaDetection
            using upperThresholdWaterlogging: 10
                  and lowerThresholdWaterlogging: 0
                  and upperThresholdSoilSerious: 11
                  and upperThresholdSoilModerate: 13
                  and upperThresholdSoil: 15
```

```
                and lowerThresholdSoilWeakly: 7
                and upperThresholdVegetation: 17
    store
end
```

The workflow starts with applying the 'ImageEnhancement' service to each selected satellite image. It then applies the 'FloodedAreaDetection' algorithm and stores the result to the cloud.

There are two important things to note about this workflow. First of all it uses low-level statements allowing the user to specify algorithm parameters such as thresholds for the detection of flooded areas. This workflow is therefore targeted to expert users who know how to process satellite imagery and who have experience with the used algorithms. On the other hand, the workflow hides some of the internal details of the processing services. The 'FloodedAreaDetection' service expects a spectral index as input data. The interpreter (see deliverable D3.2) therefore inserts the 'SpectralIndex' service internally. In addition, the interpreter renames parameters of both, the 'ImageEnhancement' service and the 'FloodedAreaDetection' service, because their actual names are not really readable by human beings. For example, the 'lowerThresholdWaterlogging' parameter is mapped to 'threshold-toa-4l-c2' and 'upperThresholdWaterlogging' is mapped to 'threshold-toa-4u-c2'. This makes the workflow a lot easier to read and demonstrates the strength of the DSL compared to executing the processing services directly.

# 6 REFERENCES

Gaševic, D., Djuric, D., & Devedžic, 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.*

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.

# 7 APPENDIX: PARSING EXPRESSION GRAMMAR FOR THE DSL

```
start
  = workflow / empty_workflow

empty_workflow
  = SP*

workflow
  = SP* statements SP*

statements
  = statement ( SP+ statement )*
```

```
statement
  = with / process_statement

process_statement
  = process datasets:process_statement_with?
    process_statement_using? process_statement_as?

process_statement_with
  = SP+ WITH SP+ dataset_expression (
    SP+ AND SP+ dataset_expression )*

process_statement_using
  = SP+ USING SP+ params

process_statement_as
  = SP+ AS SP+ varRef

with
  = WITH SP+ dataset_expression SP+ DO SP+ statements SP+ END

dataset_expression
  = dataset ( SP+ OF SP+ dataset )?

recent
  = RECENT SP+ NAME

latest
  = LATEST SP+ NAME

dataset
  = recent
  / latest
  / placeholder
  / varRef

params
  = param ( SP+ AND SP+ param )*

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

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

expression
  = placeholder
  / NUMBER
  / string
  / NAME

tuple_ids
  = NAME ( "," SP* NAME )*
```

```
placeholder
  = "[" NAME "]"

ref
  = objectRef
  / varRef

objectRef
  = NAME SP* "." SP* ref

varRef
  = NAME

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

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

store_process
  = STORE (SP+ ref)? (SP+ TO SP+ dataset)?

visualize_process
  = VISUALIZE

apply_process
  = ( APPLY / CALCULATE ) SP+ NAME

process
  = apply_process
  / store_process
  / visualize_process
  / urban_process
  / marine_process
  / land_process

add_process
  = ADD SP+ TO SP+ dataset
  / ADD SP+ ref SP+ TO SP+ dataset

exclude_process
  = EXCLUDE ( SP+ urban_dataset_param )? SP+ NAME

urban_process
  = add_process
  / exclude_process
  / JOIN SP+ NAME ( SP+ AND SP+ NAME )*
  / SELECT SP+ urban_dataset_param SP+ NAME (
    SP+ AND SP+ urban_dataset_param SP+ NAME )*
```

```
urban_dataset_param
  = ADDED
  / ALL
  / CERTAIN
  / DEFORMED

marine_process
  = GENERATE SP+ NAME

land_process
  = COLORIZE SP+ NAME SP+ dataset

/** NAMES/IDENTIFIERS
*********************************************************/

NAME
  = !ReservedWord [_a-zA-Z] NAME_MORE*

NAME_MORE
  = [_a-zA-Z0-9]

/** RESERVED WORDS
*************************************************************/

ReservedWord
  = Keyword

Keyword
  = ADD
  / ADDED
  / ALL
  / AND
  / APPLY
  / AS
  / CALCULATE
  / CERTAIN
  / COLORIZE
  / DEFORMED
  / DO
  / END
  / EXCLUDE
  / GENERATE
  / JOIN
  / LATEST
  / OF
  / RECENT
  / SELECT
  / STORE
  / TO
  / USING
  / VISUALIZE
  / WITH
```

```
/** TOKENS
**************************************************************************/

NUMBER
  = [0-9]+ ( "." [0-9]+ )?

ESCAPE_CHARACTER
  = ['"\\bfnrtv]

SP
  = [ \t\n\r]

/** KEYWORD TOKENS
**************************************************************/

ADD             = "add"             !NAME_MORE
ADDED           = "added"           !NAME_MORE
ALL             = "all"             !NAME_MORE
AND             = "and"             !NAME_MORE
APPLY           = "apply"           !NAME_MORE
AS              = "as"              !NAME_MORE
CALCULATE       = "calculate"       !NAME_MORE
CERTAIN         = "certain"         !NAME_MORE
COLORIZE        = "colorize"        !NAME_MORE
                / "colourise"       !NAME_MORE
DEFORMED        = "deformed"        !NAME_MORE
DO              = "do"              !NAME_MORE
END             = "end"             !NAME_MORE
EXCLUDE         = "exclude"         !NAME_MORE
GENERATE        = "generate"        !NAME_MORE
JOIN            = "join"            !NAME_MORE
LATEST          = "latest"          !NAME_MORE
OF              = "of"              !NAME_MORE
RECENT          = "recent"          !NAME_MORE
SELECT          = "select"          !NAME_MORE
STORE           = "store"           !NAME_MORE
TO              = "to"              !NAME_MORE
USING           = "using"           !NAME_MORE
VISUALIZE       = "visualize"       !NAME_MORE
                / "visualise"       !NAME_MORE
WITH            = "with"            !NAME_MORE
```