2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing

Controlling the Processing of Smart City Data in the Cloud with Domain-Specific Languages

Michel Krämer
Fraunhofer Institute for Computer Graphics Research IGD
Technische Universität Darmstadt
64283 Darmstadt, Germany
Email: michel.kraemer@igd.fraunhofer.de

***Abstract*—In this paper a new user interface for the processing of big geospatial data in the Cloud is presented. The paper focuses on the processing of Smart City data since this kind of data is inherently large and becomes more and more important for the maintenance and planning of sustainable future Smart Cities. The user interface is facilitated by Domain-Specific Languages (DSLs) that are tailored to the urban planning domain. In this paper a modelling method for DSLs is described and then applied to an example use case from the research project IQmulus funded by the European Commission. The resulting DSL is specified using a Parsing Expression Grammar (PEG) which is used to automatically generate a language parser. Furthermore, a technique to interpret scripts written in the Domain-Specific Language is presented. This technique makes use of mapping rules that specify how the interpreter has to translate terms in the DSL to processing services in the Cloud.**

I. Introduction

Smart City data will take on an important role in future urban planning and policy making. Modern algorithms and ICT tools can analyse large amounts of information and produce knowledge that can substantially improve the planning process. In this context intelligent data acquisition and preprocessing methods are key to the successful use of Smart City data. However, today we often see that spatial data such as orthophotos, digital terrain models, or 3D city models, for example, are collected at certain points in time but hardly kept up to date. City administrations order data acquisition regularly but not very often. In Europe a typical time range between two LiDAR (*Li*ght *D*etection *A*nd *R*anging) scans generating digital terrain models, for example, is three to four years. Due to this, changes happening in the meantime in a modern rapidly evolving city cannot be incorporated into the urban planning process.

Modern artificial satellites collect orthophotos and 3D point clouds constantly. Their products could help alleviate the problem of lack of information, but the sheer amount of data generated (several TiB or even PiB per day) just cannot be processed fast enough. The same is true for in-situ sensing and sensor networks that collect data about traffic, air quality, etc. Such data arrives in streams but current geospatial software applications used in public administrations are not yet ready to process them as needed.

The Cloud offers new possibilities since it provides city administrations not only with distributed storage of Smart City data but also with virtually unlimited processing power. There is an ongoing effort to utilise the Cloud for the processing of geospatial data [1], [2]. Current approaches, focus on software architectures and processing algorithms but often neglect the user interface. Although some work has been published in this context [3]–[5], previous approaches target expert users with IT background (most likely software developers) but not domain users such as urban planners or policy makers.

This paper is about a new approach of creating a user interface to control the processing of Smart City data in the Cloud. This user interface is based on Domain-Specific Languages (DSL) which are lightweight programming languages tailored to domain users with little to almost no IT background. DSLs use the vocabulary of a specific domain (in this case urban planning) and are hence easy to understand and learn for people working in this domain.

The remainder of this paper is structured as follows. First an overview over related work is given. Then the DSL modelling method is presented, followed by an example use case demonstrating how this modelling method can be applied to a practical use case. After that, a technique to map DSL expressions to processing services in the Cloud is described. The paper ends with a summary and conclusion.

II. Related work

According to Khan et al. Smart Cities can benefit from “big, and often real-time cross-thematic, data collection, processing, integration and sharing through inter-operable services deployed in a Cloud environment” [2]. Khan et al. therefore present a software architecture for a Cloud-based analysis service supporting planning and decision making in future Smart Cities. Their solution is based on well-known Cloud technologies, tools and open standards that allow their architecture to be easily integrated into existing Smart City environments. They conclude that “Cloud computing provides a great opportunity to manage, analyse and process Big data generated by cities but needs new tools and services”. One of these new tools is the user interface presented in this paper. The Cloud is ought to help the people working in the city administration but standardised service interfaces alone are not enough to satisfy their requirements. A well-designed user interface is necessary to bridge this gap. Domain-Specific Languages (DSLs) as they are used in this paper can act as such an interface.

The use of DSLs in the area of Cloud Computing to control distributed processing has been demonstrated by Olston et. al who present the software library Apache Pig and in particular its high-level processing language Pig Latin [6]. The language looks a lot like the database query language SQL—which is in fact also a Domain-Specific Language—so users who are familiar with SQL are able to quickly learn and use Pig Latin as well. The language drives distributed MapReduce jobs [7] executed in the Cloud by the Apache Hadoop framework. Pig Latin therefore simplifies the process of specifying complex parallel computing tasks which can sometimes be tedious even for experienced programmers. However, it is very generic and does not target a specific application domain like the urban planning DSL presented in this paper.

Pig as well as Pig Latin lack support for geospatial processing. This gap is closed by the SpatialHadoop framework which adds spatial indexes and geometrical datatypes and operations to Apache Hadoop [4]. In addition to that, SpatialHadoop offers a DSL based on Pig Latin. Pigeon is a high-level query language that allows users to specify complex spatial queries in a readable and concise way [5].

DSLs have also been successfully applied to the area of urban planning. In a previous work we presented a Domain-Specific Language for urban policy modelling [8]. The language offers the possibility to specify policies and policy rules in a formalised and machine-readable way. At the same time the policy rules stay clearly readable and understandable for urban planners and decision makers. The idea has further been elaborated in another work where we present a graphical DSL which can be used to specify typical spatial processing workflows in the urban planning domain [9]. The graphical user interface has been integrated into a 3D GIS (Geospatial Information System). User evaluation carried out in the urbanAPI research project funded by the European Commission confirmed that such a DSL is indeed useful, but more work is needed, in particular since the language is not designed to operate in a Cloud Computing environment.

III. DSL MODELLING METHOD

In this paper the following incremental and iterative method to create Domain-Specific Languages is proposed:

1) Analyse the application domain.
2) Create user stories.
3) Analyse user stories and look for relevant subjects, objects, adjectives, and verbs.
4) Create a domain model using subjects and objects found in the user stories as classes.
5) Identify relevant verbs which become actions in the Domain-Specific Language.
6) Build sample DSL scripts based on the modelled domain.
7) Derive formalized grammar from the sample DSL scripts.
8) Review and reiterate if needed.

The first couple of steps are inspired by object-oriented software engineering [10]. Steps 1 and 2 are essential to every modern agile and lean software development project. They belong to the general requirements analysis phase where software developers and stakeholders (most likely domain users) work together to identify functional and non-functional requirements. This typically results in a number of user stories describing how the domain users would like to interact with the system. Ideally these stories are created by the users themselves. This ensures they are written in "the user's own words" which basically means they use the exact same vocabulary that the domain users are used to from their everyday work. Based on this a Domain-Specific Language can be created that contains terms from the application domain and that is hence easy to understand and learn.

The text analysis performed in step 3 of the DSL modelling process provides the basis for the domain vocabulary. The objects, subjects and verbs gathered so far have to be structured and classified so a machine-readable programming language can be defined. In step 4 a domain model is created describing the relations between subjects and objects. Additionally, in step 5 the relevant verbs are identified that will later become actions in the Domain-Specific Language.

In step 6 sample scripts written in the (not yet formalised) DSL are created. Just like in the first two steps, the domain users should be involved here to provide feedback on the sample scripts. This ensures the final language will look as expected.

After that a formalised grammar is created in step 7. This makes the language machine-readable and interpretable (see section V). The whole modelling process is iterative. So, finally, the result is reviewed and revised if necessary—of course in close collaboration with the domain users.

IV. EXAMPLE USE CASE

The IQmulus research project funded by the European Commission aims for creating a platform for the fusion and analysis of high-volume geospatial data such as point clouds, coverages and volumetric data sets. One of the major objectives is to automate geospatial processing as much as possible and therefore reduce the amount of human interaction with the platform. At the same time the project tries to exploit modern Cloud technology in terms of processing power and distributed storage. Three showcases related to marine, land, and urban applications were defined. These showcases drive the software development as well as the evaluation of the tools created within the project.

This paper focuses on the urban showcase in order to demonstrate how a Domain-Specific Language tailored to experts from the urban planning and policy making domain can be created. The showcase is about the typical work of an urban planner who needs to integrate and process geospatial data from different sources to create products such as topographic maps, orthophotos, and 3D city models. Suppose within the project's requirements analysis phase—i.e. the first two steps of the DSL modelling method presented in section III—the following user stories have been defined (cf. [11]).

- As an urban planner I want to capture topographic objects (such as cable networks, street edges, urban furniture, traffic lights, etc.) from data acquired by mobile mapping systems (LIDAR point clouds and images) so I can create or update topographic city maps.
- As an urban planner I want to automatically detect individual trees from a LIDAR point cloud in an urban

area, so I can monitor growth and foresee pruning works.

- As an urban planner I would like to update my existing 3D city model based on analysing recent LIDAR point clouds.

The following sections demonstrate how the DSL modelling method can be applied to this example. The sections are ordered according to the steps in the modelling method.

Specific focus is put on the process of updating a 3D city model based on LIDAR point clouds. In order to complete this task the urban planner typically performs the following operations:

- Remove non-static objects (such as cars, rubbish bins, bikes, or people) from the point cloud,
- characterize changes of static objects (such as trees, bus stops, or facade elements),
- include new or exclude existing classified objects (e.g. roof tops or antennas),
- use the result to update the city model (i.e. apply the changes to the existing model).

Finally, the urban planner typically visualises the results in 3D to assess correctness and overall quality. In addition to that, such a 3D visualisation can be presented to decision makers, for example, if changes in the city such as new constructions should be discussed [12].

A. Vocabulary/Taxonomy

According to step 3 of the DSL modelling method, the user stories from the previous section now have to be analysed to find subjects and objects which can later be used as classes in the domain model. Verbs and adjectives are also important. They will become actions and parameters in the Domain-Specific Language in the end.

Table I depicts the results of this analysis. Please note that not all terms found in the user stories are actually relevant. For example, the verbs “monitor” and “foresee” as well as the objects “growth” and “pruning work” refer to something that happens *after* the urban planner has processed the data. They do not belong to the processing itself and hence do not appear in the Domain-Specific Language.

Also note, at certain points, the wording in the user stories is unclear. For example the expression “changes of static objects” is rather unspecific about what “changes” actually are. The only possibility to resolve this issue is to ask the users who might say that for them “changes” mean that objects are added to a dataset or removed from it—hence the adjectives “added” and “removed” in the table. That’s one of the reasons why users should be involved in each step of the DSL modelling.

B. Domain model

The next step in the modelling process is to create a domain model based on the text analysis and the subjects and objects found. Figure 1 shows the domain model for the example use case. `CityModel`, `PointCloud`, and `Image` are datasets containing topographic objects. Each `TopographicObject` is either a `StaticObject` (`Roof`, `Antenna`, `Facade`, `Tree`, etc.) or a `NonStaticObject` (`People`, `Bike`, `Car`, etc.).

TABLE I. RESULTS FROM THE TEXT ANALYSIS

Subjects and objects

topographic object	cable network	street edge
urban furniture	traffic light	image
LIDAR point cloud	mobile mapping system	topographic city map
tree	urban area	growth
pruning work	3D city model	object
car	rubbish bin	bike
people	bus stop	facade element
roof	antenna	

Verbs

capture	create	update
detect	monitor	foresee
remove	characterize	include
exclude	visualise	

Adjectives

non-static	static	recent
added	removed	

The domain model helps to structure the heap of terms found in the text analysis and to differentiate relevant and irrelevant words. The next step is to create sample DSL scripts based on the domain model.

C. Sample DSL scripts

As mentioned before each step of the DSL modelling process should happen in strong collaboration with the domain users. In step 6 (the formulation of sample DSL scripts) it is specifically important to talk to them to get feedback before the Domain-Specific Language is actually implemented.

In order to update the 3D city model and visualise the results the following sample is proposed.

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

with CityModel do
    exclude Antennas
    visualize
end
```

The script consists of two parts. In the first one a recent point cloud dataset is processed. Non-static objects are removed and new trees and facade elements are detected. These new objects are added to the city model. In the next block all antennas are removed from the city model and the result is visualised on the screen.

The script is easy to read and shows exactly what processing steps are performed. The Domain-Specific Language proposed here makes use of terms from the domain model and

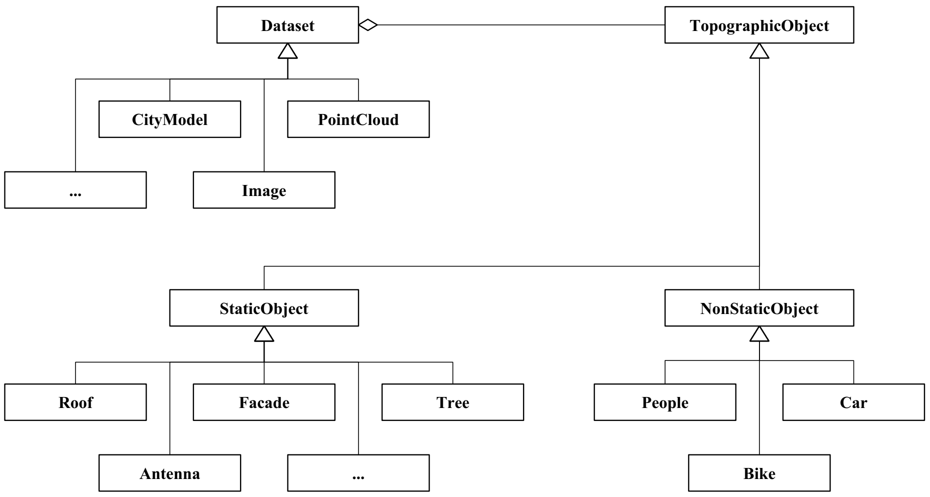

Fig. 1. Domain model of the example use case. For the sake of readability only an excerpt is shown here. Some classes have been left off.

from the results of the text analysis performed before. Domain users can therefore quickly learn the language and use it for their own needs.

D. DSL grammar

In order to make the Domain-Specific Language machine-readable its grammar and syntax have to be formalised. A common way to do that is the specification of either a context-free grammar (CFG) using EBNF (Extended BackusNaur Form) or a PEG (Parsing Expression Grammar). One of the benefits of PEGs is that they can never be ambiguous. They are therefore very easy to define and are often not as complex as CFGs, not least because they don't require an additional tokenization step. On the other hand, PEGs require more memory than CFGs, but for small languages such as DSLs this disadvantage can be neglected.

The PEG for the sample DSL scripts presented in the previous section is as follows.

```
start = SP* statements SP*

statements = statement ( SP+ statement )*

statement = block / process

block = with SP+ statements SP+ "end"

with = "with" SP+ dataset SP+ "do"

dataset = "recent" SP+ ID / ID

process = "visualize" / "exclude" ID /
    "select" SP+ param SP+ ID ( SP+ "and" SP+
    param SP+ ID )* / "add" SP+ "to" SP+
    dataset

param = "added"

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

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

The syntax used to the specify the PEG here is the one of the tool PEG.js, an open-source parser generator written in JavaScript[1]. Square brackets are used to define regular expressions. The plus character + means one or more occurrences whereas the asterisk * means zero or more occurrences. The slash character / is used to specify alternatives.

Please note that a grammar for the complete example use case would be much larger. The PEG shown here can only be used to parse the example script from the previous section. For the sake of readability additional grammar rules have been left off.

E. Reiteration

The final step of the modelling process is to review the Domain-Specific Language and to revise it if necessary. The language presented in this paper already is a result of several iterations in which software developers worked together with domain users to create to a reasonable and sensible DSL that meets the users' expectations.

F. Rationale for the chosen DSL syntax

The sample DSL presented before is based on the vocabulary from the domain model. Getting to the specific syntax was a matter of testing various alternatives and evaluating how they work in certain use cases. The following sample script was used as a starting point.

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

In this sample script the keyword `it` is used 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. It

[1]http://pegjs.majda.cz/

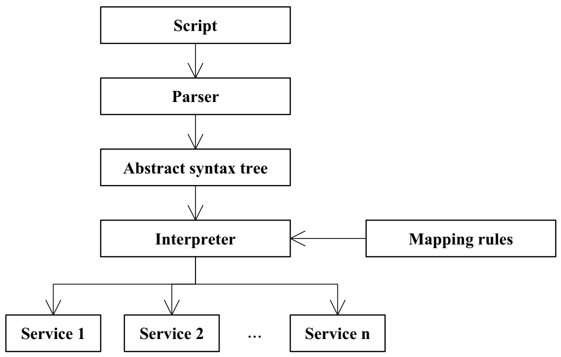

Fig. 2. Scripts are parsed to an abstract syntax tree. An interpreter traverses this tree and executes the processing services in the Cloud.

was recognized that human users have problems understanding such context-sensitive scripts. Instead the following syntax was tested.

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

In this case individual processing steps are connected with the **and** keyword. While this approach leads to unambiguous scripts, they still can quickly become hardly readable. The longer the scripts get, the harder it is to understand them as the sentences become longer and longer. On the other hand, blocks enclosed by **with ... do** and **end** (like they are used in section IV-C) make clear which processing steps affect which data set.

V. Running DSL scripts in the Cloud

The Domain-Specific Language defined above allows urban planners to specify processing workflows. In order to run these workflows in the Cloud, scripts have to be parsed and interpreted. This section contains a description of the implementation of the DSL parser and interpreter from the IQmulus project. In IQmulus all data to be processed is stored in a distributed cloud storage—i.e. an object storage or a distributed file system. Processing services running on virtual machines in the Cloud access this distributed storage and process the geospatial data. In IQmulus a number of processing services have been implemented—e.g. feature extraction, corregistration, interpolation, etc.

The open-source library PEG.js is used to automatically generate a parser based on the grammar defined in section IV-D. The parser generates an abstract syntax tree (AST) which can be traversed by an interpreter. The interpreter executes the processing services in the Cloud. Figure 2 depicts this process.

The interpreter makes use of a set of pre-defined rules that map terms in the AST to processing services. Figure 3 shows mappings that appear in the example use case.

- **One-to-one mapping.** If a term such as `exclude` appears in the AST the interpreter executes exactly one processing service—in this case a filter removing objects that should be excluded.

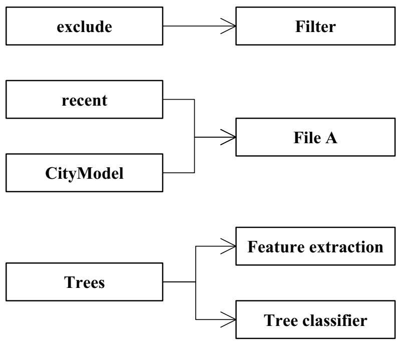

Fig. 3. Terms in the abstract syntax tree are mapped to data sets and processing services.

- **Many-to-one mapping.** The terms `recent` and `CityModel`, for example, are mapped to a specific data set (or file) which represents the latest version of the 3D city model kept in the distributed Cloud storage.
- **One-to-many mapping.** Terms such as `Trees` may need to be mapped to parametrised processing services. For example, the processing service for feature extraction is implemented using machine learning algorithms. The term `Trees` therefore needs to be mapped to both, the feature extraction service as well as a pre-trained classifier for trees.

In addition to that, **many-to-many** relations can also happen although they do not appear in the example.

The execution chain shown in figure 2 is modular. Individual parts can be replaced to allow for a wide range of applications. For example, the interpreter can be implemented using an existing scheduler such as Apache Hadoop YARN [13] or Apache Spark [14]. In IQmulus a dedicated scheduler is implemented which is able to access various backends [15].

By the use of mapping rules as it is proposed here the interpreter may be replaced without affecting the DSL and the parser. This means that even if the back-end (i.e. the interpreter, the mapping rules, and the processing services) are replaced by other implementations the scripts written by the domain users stay the same. In particular, this ensures the domain knowledge that the users put into the scripts remains valid for a long time even if the underlying Cloud infrastructure changes—e.g. if the infrastructure is transferred from one Cloud provider to another.

VI. Conclusion

In this paper a new user interface for the processing of big geospatial data in the Cloud was presented. The interface is facilitated by a Domain-Specific Language tailored to the urban planning domain. The DSL modelling method presented in this paper makes sure the language consists of terms that are known to the domain users. This makes the language easy to understand and learn and allows domain users to write their own processing scripts without a deep knowledge of the underlying Cloud infrastructure. In particular, the users do not have to care about on what specific hardware the processing is executed or what algorithms are exactly used and with what parameters they are called. The Domain-Specific Language offers just as much control as the users need. The rest is hidden in mapping rules that control how the language interpreter translates terms in a script to actual calls of processing services in the Cloud.

In addition to that, the rule-based approach presented in this paper allows the infrastructure as well as the technologies used to be replaced without requiring the users to rewrite their scripts. This is a great benefit compared to other approaches such as SpatialHadoop Pigeon [5] which is tightly coupled to a specific technology (i.e. Apache Hadoop). Furthermore, it allows scripts to be transferred to other Clouds and, thus, be shared amongst different departments in the same city administration or even between different cities.

Basically, the approach presented in this paper is intended to close the gap between the processing of geospatial data in the Cloud and the end user who typically is a GIS expert, but not an expert in Computer Science, and in particular not in Cloud Computing or Big Data. This paper specifically focused on the the processing of Smart City Data in the area of urban planning, but the approach presented can be applied to other use cases as well. In the future, Cloud technology will be used more and more often for the maintenance and planning of sustainable Smart Cities. Previous work has shown that this development has a lot of benefits for all stakeholders, but nonetheless a user interface that is easy to understand (like Domain-Specific Languages) may be key to the success of Cloud Computing in the area of urban planning.

The aim of this paper was to demonstrate the use of a Domain-Specific Language in the area of Smart City Data processing. In the future, the modelling method presented in section III will be used to create further languages enabling users from various application domains to describe geo processing workflows (not limited to the urban planning domain or Smart City Data processing). In addition to that, the technique to run DSL scripts in the Cloud (described in section V) will be improved and applied to larger processing workflows. It will be particularly interesting to see how this approach scales if very large and complex geospatial datasets should be processed. Domain-Specific Languages will provide the interface for this, allowing domain users to tackle the complex technical details of Cloud-based Big Data processing.

Acknowledgments

I would like to thank Ivo Senner for the valuable input and the discussions on this topic. Research presented here is carried out within the project "IQmulus" (A High-volume Fusion and Analysis Platform for Geospatial Point Clouds, Coverages and Volumetric Data Sets) funded from the 7$^{\text{th}}$ Framework Programme of the European Commission, call identifier FP7-ICT-2011-8, under the grant agreement no. 318787, started in November 2012. This paper is partly based on the public project deliverables D2.4.1 "Processing DSL Specification – baseline version" [16] and D3.2 "Control components – vertical prototype release" [15].

References

[1] Z. Khan and S. L. Kiani, "A cloud-based architecture for citizen services in smart cities," in *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, ser. UCC '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 315–320.

[2] Z. Khan, A. Anjum, and S. L. Kiani, "Cloud based big data analytics for smart future cities," in *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, ser. UCC '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 381–386.

[3] A. K. Sujeeth, H. Lee, K. J. Brown, T. Rompf, H. Chafi, M. Wu, A. R. Atreya, M. Odersky, and K. Olukotun, "OptiML: An Implicitly Parallel Domain-Specific Language for Machine Learning," in *ICML*, L. Getoor and T. Scheffer, Eds. Omnipress, 2011, pp. 609–616.

[4] A. Eldawy and M. Mokbel, "A Demonstration of SpatialHadoop: An Efficient Mapreduce Framework for Spatial Data," in *Proceedings of the 39th International Conference on Very Large Data Bases (VLDB)*, vol. 6, no. 12. VLDB Endowment, Aug. 2013, pp. 1230–1233.

[5] A. Eldawy and M. Mokbel, "Pigeon: A Spatial MapReduce Language," in *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 2014.

[6] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: A Not-so-foreign Language for Data Processing," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '08. New York, NY, USA: ACM, 2008, pp. 1099–1110.

[7] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[8] M. Krämer, D. Ludlow, and Z. Khan, "Domain-Specific Languages for Agile Urban Policy Modelling," in *Proceedings of the 27th European Conference on Modelling and Simulation (ECMS)*, W. Rekdalsbakken, R. Bye, and H. Zhang, Eds. Ålesund, Norway: European Council for Modelling and Simulation, 2013, pp. 673–680.

[9] M. Krämer and A. Stein, "Automated Urban Management Processes: Integrating a Graphical Editor for Modular Domain-Specific Languages into a 3D GIS," in *Proceedings of the 19th International Conference on Urban Planning, Regional Development and Information Society REAL CORP 2014*, M. Schrenk, V. V. Popovich, P. Zeile, and P. Elisei, Eds. Schwechat, Austria: CORP – Competence Center of Urban and Regional Planning, 2014, pp. 99–108.

[10] A. D. Nicola, M. Missikoff, and R. Navigli, "A software engineering approach to ontology building," *Information Systems*, vol. 34, no. 2, pp. 258–275, 2009.

[11] M. Belényesi and D. Kristóf (eds.), "IQmulus public project deliverable D1.2.3 – Revised User Requirements," Tech. Rep., 2014.

[12] J. Dambruch and M. Krämer, "Leveraging public participation in urban planning with 3D web technology," in *Proceedings of the 19th International Conference on 3D Web Technology (Web3D)*, Aug. 2014, to be published.

[13] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache Hadoop YARN: Yet Another Resource Negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: ACM, 2013.

[14] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*. Berkeley, CA, USA: USENIX Association, 2010.

[15] M. Krämer, M. Zulkowski, S. Plabst, and N. Kießlich, "IQmulus public project deliverable D3.2 – Control Components – vertical prototype release," Tech. Rep., 2014.

[16] M. Krämer and I. Senner, "IQmulus public project deliverable D2.4.1 – Processing DSL Specification – baseline version," Tech. Rep., 2014.