COMPUTING FEEDBACK FOR CITIZENS' PROPOSALS IN PARTICIPATIVE URBAN PLANNING

J. Dambruch [1,2]

[1] Technische Universität Darmstadt, Darmstadt, Germany
[2] Fraunhofer Institute for Computer Graphics Research IGD, Competence Center for Spatial Information Management
64283 Darmstadt, Germany – jens.dambruch@igd.fraunhofer.de

KEY WORDS: Public participation, co-creation, automated feedback, Domain-Specific Languages

ABSTRACT:

We show an approach how to provide computed feedback on citizens' proposals based on open data and expert knowledge in urban planning and public participation by using Domain-Specific Languages (DSL). We outline the process involving different stakeholders of engineering such a DSL and provide an architecture capable of executing the language and uploading new scripts at runtime. A real-world example of the city of Hamburg is used to show the principles and serves as input for development. A prototype has been implemented and evaluated at various events involving citizen and city representatives. We conclude that DSLs can be successfully applied to enable a new way to access data in a more convenient and understandable form, abstracting from technical details and focusing on domain aspects.

1. INTRODUCTION

Engaging individual stakeholders and citizens in urban planning is a growing trend. Besides meetings and written communication especially web-enabled tools can aid this participatory approach in various ways. As first step several open data initiatives have been started and major cities such as Hamburg[1], Rome[2] or London[3] have started publishing data online as open data.
Khan et al. (2017) report on the development of innovative ICT systems for public participation using open data. These platforms have the potential to transform city governance by facilitating both top-down and bottom-up decision making.
A common problem is that data is mostly published "as-is" and - without sound knowledge of the corresponding domain - hard to understand and use. Also, various digital formats necessitate the use of software tools targeted mainly at professional users. On the other hand, new technical developments such as web technology and improved 3D display hardware is available at low cost and can be used in this context successfully. Dambruch and Krämer (2014) show how a 3D interactive web portal can be used to visualize urban designs in interactive 3D and to gather citizens opinions and feedback at the same time. Also, Ruppert et al (2015) report on 3D visual analytics in an urban environment. Since data about planning and legal rules imposed on planning and development are often not visible or even formally defined it is necessary to explain such restrictions to citizens when making proposals. At best this should happen in an interactive fashion by a dialogue-based system.
The idea of such an interactive feedback facility is the motivation for this paper and in the following we report on the concept and prototypical implementation. The driving use cases are taken from the smarticipate[4] project. For both gathering requirements in the analysis phase and also for testing and evaluation in later phases, we adopted a simple use case elaborated with the city of Hamburg, where citizens should propose locations to plant new trees within the city. This use case should be implemented using a visual web-application, which gives direct feedback on the location selected by indicating possible obstacles blocking the location such as buildings, roads, existing trees or even legal regulations and city planning. Especially the reasons for declining such proposals should be made transparent by giving explanations such as: "There are gas pipes below the desired location and the roots could harm them".
For the design and implementation of these checks several aspects are to be considered. The rules to check have to be transformed to executable code from an informal description ranging from legislation text written by lawyers to orally inherited procedures involving details that nobody is aware of. So basically, programmers and experts are bound in an involved communication process any time they have to implement the changes on rules or even implement new rules. Another way would be to train administration to understand technical programming concepts such as rule-based systems or Semantic Web technology to implement those rules themselves.
Both options, training experts to be programmers or teaching programmers domain knowledge need a lot of engagement or are too expensive on the long run and especially the data to be used has to be considered carefully, since it has to be made available in a suitable format as well. This introduces a lot of overhead and leads to the idea that domain experts should be enabled to define rules themselves in a specialized language which is close to their abstraction level, without the need to care for technical details and we show how this can be accomplished with Domain-Specific Languages (DSL).
There is literature on Visual Analytics and Decision Making such as Ruppert (2018) and Kovalerchuck and Schwing (2004), but so far the concept of automated Feedback with DSL processing based on available data seems to be new.

The paper is structured as follows: Chapter 2 explains the concept of DSL for computing feedback on citizens proposals and how engineering of such DSLs is done. Chapter 3 elaborates on a prototypical implementation of such DSLs and a corresponding web-service with details on technology used and examples covering the use case mentioned before. Chapter

[1] http://www.hamburg.de/transparenzportal-hamburg/
[2] http://dati.comune.roma.it/
[3] https://data.london.gov.uk/
[4] http://www.smarticipate.eu

4 is about evaluation and demonstration of the prototype in the context of the smarticipate project. Chapter 5 is about future work regarding technology and further implementation of the prototype and chapter 6 is a discussion of the results and conclusion.

2. CONCEPT

2.1 Domain-Specific Languages (DSL)

DSL is a well-known concept in computer science and is widespread for example for descriptions of configurations of systems. According to Fowler (2011, p. 27) a DSL is a computer programming language of limited expressiveness focused on a particular domain. This means that no general-purpose programming is in mind rather a simple language design tailored to users' needs. In principle, this DSL is on the same or very close to the semantic level of domain users hiding unnecessary technical details. A concept of using DSL for urban analysis has been outlined by Malewski, Dambruch and Krämer (2015), which used DSLs as an instrument for analyzing and visualizing geospatial data in a web environment. A DSL was used to analyzed cycle paths and also for highlighting results of the analysis. One of the major drawbacks of this approach was that an expensive data preparation was needed to make the data fit for the rule execution system via annotating data and transforming it in special formats. This should be replaced by ad-hoc annotation. Mernick et al. (2005) report that DSLs can be enablers of reuse in that the domain analysis carried out is now available in an executable form – the DSL - for other developers as well. On the other hand, they consider DSL development as a hard task for both domain experts and developers since both domain and technical knowledge or a close dialogue between both is needed. A DSL will also enable domain experts to concentrate on aspects important and abstract from technical implementation aspects, which reduces development efforts.
In terms of the use case example of tree planting outlined before, we aim to implement a feedback service using a DSL as a technique to enable experts to define the rules and processing needed to deliver automated feedback. As Stein and Krämer (2014) point out, DSLs are especially designed keeping in mind that these users are typically domain experts, familiar with their domain model, but not with programming languages or technical data processing tools. Therefore, based on the analysis outlined next, we want to design a formal language as close as appropriate to the language used by experts, but can also be processed by computers in an effective and efficient way.

2.2 Engineering a DSL

The crucial part is to define the DSL and also how scripts of this language are executed in a computer environment. We conceptualize this as a collaborative 3 phase process between stakeholders, where phases are implemented as iterative sub-processes as well as the process itself can be applied iteratively.

Krämer (2014) outlines an approach based on Nicola et al. (2009), which we adopt as basis for modelling phases. It names the following aspects:

- Requirements Gathering
- Definition of Use Cases and User Stories
- Domain Analysis
- Definition of a Terminology and a Domain Model
- Mapping of Terminology to software artefacts and actions
- Building of sample DSL scripts based on the terminology and models defined above
- Derive formalized grammar from the sample DSL scripts
- Review and reiterate if needed.

2.2.1 Phase 1: Design

The first phase (Figure 2.1) is a co-creation session between domain and data experts and IT experts on the other side.

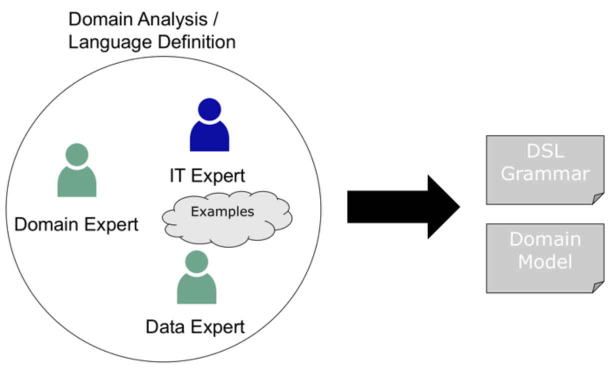

Figure 2.1 Phase 1

The domain experts are the persons that have the knowledge about the domain semantics and what rules are to be considered and decide how these rules look like from a formal perspective. They select and explain the use cases relevant for design and elaborates them in user stories. The vocabulary for the language is elaborated by text analysis and takes the keywords from user stories. Data entities can be identified by the nouns used and discussed whether they match existing data sets and also the relationships between them can be discussed. Verbs indicate possible actions on the data sets.
The data expert is responsible to provide data that enables rule processing. Typically, this can be some GIS expert or internal IT personnel.
The IT experts are responsible for defining formal grammar for DSL and mappings from data sets to the domain model. They come up with sample language sentences or fragments for further elaboration and also the data experts contribute to this by identifying data sets for the use cases and their relationship to the vocabulary. The outcome of this phase of co-creation is a formal grammar and domain model description driven by the use cases discussed.

2.2.2 Phase 2: Implementation

For the second phase (Figure 2.2) we can split the role of the IT expert in roles such as analyst or programmer. The IT analyst is responsible for defining the grammar and mappings of the domain model to data while the programmer implements the grammar by mapping the DSL elaborated in phase 1 to a technical application programming interface (API) via code templates. The domain model can be mapped by standard tools such as style-sheet transformation.

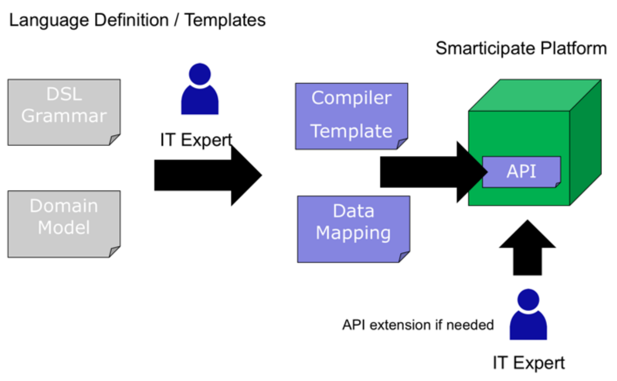

Fig 2.2 Phase 2

In most cases it is useful to introduce 2 DSL definitions: (1) the Domain Model: a model definition language naming and defining data types to model the domain and also how to access and map external data into the model and (2) the DSL Grammar: the actual domain specific language working on model entities. The entities defined in the domain model could also be reused more easily, but it needs to be assured that the semantics of the entities remains clear and bound to the use case.
In some cases, it might be necessary to extend the target API by programmers to keep the mapping simple or also to enable new functionality since the DSL is a restricted language by design and not computationally complete.
The mappings and templates created will be deployed by the IT expert on the technical target platform.

2.2.3 Phase 3: Application

In phase 3 (Figure 2.3) the domain expert can make use of the platform by creating scripts based on the DSL descriptions and mapping made in the previous phases checking constraints in planning on (geo-)data obtained from IT based services.
The Rule scripts can be deployed in self-service and also data is available via adapters configured to use the domain model created in phase 2.

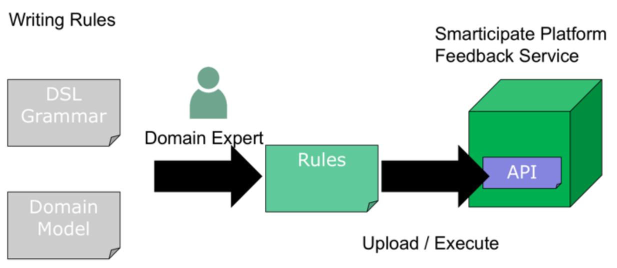

Figure 2.3 Phase 3

The whole process of language definition can be used iteratively to enable early access to the language and of course to amend mistakes or misunderstandings between stakeholders. These rather involved stops have to be carried out only once if a new language is elaborated. After this domain experts can use it to write rules on their own. Existing languages can be reused or adapted saving efforts, especially within the technology framework executing the programs and the libraries used to implement features in the DSL.

2.3 Feedback service design

The platform also needs to offer a runtime framework for management of such DSL scripts. In this case we propose a micro-service, which is responsible for managing data and network access and dispatch of DSL programs to generate feedback. Figure 2.4 shows how this generic service framework operates on an activity level.

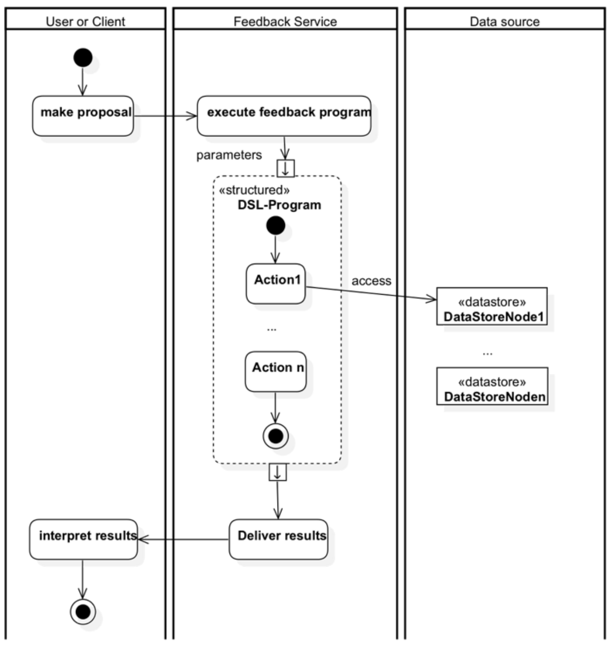

Figure 2.4 Generic operation of Feedback Service framework

A user or client triggers a standard web request via http on the feedback service. The DSL programs registered there will be triggered by a simple mapping of the request. Parameters provided will be mapped or transformed as needed and supplied to the program. Then the DSL program is executed and results will again be mapped and transferred to the client.
The handling of the DSL scripts is twofold: (1) installing the script in the service and (2) the actual execution of the script. The installation performs the following steps:

- Parsing of scripts
- Generating executable code module by mapping of language elements to functional modules
- Loading and registering of module as web service

The execution of the script triggers the generated code module and supplies the parameters to it. The steps involved depend on the actual script and can for example include other scripts, access to data source, mapping of data obtained and various calculations or string operations as in common programming languages. Informal rules and data source definitions are transcribed by Domain Experts to formal rule scripts and data source definition scripts using the DSL grammar provided by the programmers. These scripts get executed using the framework provided by programmers deployed as feedback service. Domain Experts access the services for example via web browsers and the feedback can be integrated in a visual application using maps or 3D city models.

3. FEEDBACK SERVICE IMPLEMENTATION AND RULE EXECUTION

In the following we describe how we implemented a runtime environment for the DSLs to compute feedback as a web-service. The concepts introduced before have been implemented, so that models and data access definitions can be generated with a generic DSL for that purpose. Artifacts created by this are handled as normal program code and get compiled and deployed with the service. The actual rule DSL can be changed at run-time by uploading scripts to the service by domain experts. The engineering steps such as requirements gathering, domain analysis and definition of the language have been carried out and documented in the context of the smarticipate project, which we use as input for our prototype implementation. It is also the goal to fully automate as many steps as possible, therefore generation of code, building and deployment are fully automated as scripts. Further technical details can be found in Dambruch 2016.

3.1 Technology Stack

We implemented to prototype using the commonly available platform Java, since there are already lots of mature toolkits and libraries available for implementing DSL and also common build and management tools for development.
The actual service is implemented using the Grizzly[5] server framework and the reference implementation of Java API for Restful Web Service named Jersey[6]. Jersey as reference implementation also adheres to a community driven open specification process and is not vendor specific.
Jersey allows to define web services based on plain java classes and annotations. Data interfaces are modelled as plain Java Classes with additional annotations for Java-XML[7] Binding. These annotations describe how Java objects are to be serialized as XML or in our case JSON data.
Jersey also allows to register services at run time, which offers a lot of flexibility. We use this possibility to generate and register service handlers from DSL rule programs at runtime. As tool for Grammar definition ANTLR[8] is used and as template mechanism we use Stringtemplate[9] which implement in combination the core of our DSL concept.

3.2 Framework parts

The code generated is using a java-based framework to simplify code templates. It should be avoided to have very complex code templates as this leads to complex error scenarios for code generation. Instead of this, libraries should be developed, which could be reused and tested separately, which reduces the complexity for fixing errors drastically. The development of the framework on the other hand can make use of all features available in modern software development.

3.2.1 Adapter

Adapters are responsible for encapsulating data sources. They make use of the defined entity and mapper artifacts and deliver Java objects with clear semantics. For example OGC compliant Web Feature Services are supported.

[5] https://javaee.github.io/grizzly/
[6] https://jersey.github.io
[7] https://github.com/javaee/jaxb-v2
[8] http://www.antlr.org, see also Parr (2012)
[9] http://www.stringtemplate.org

3.2.2 Selectors

Adapters can make use of query operators to select data based on spatial properties or other constraints. The results need to be post processed and mapped to entities.

3.2.3 Entities and Mappers

Entities are the model classes generated as mentioned before, resembling plain Java-Objects. Mappers contain information about the data source and how the data is mapped into entities. It is worth mentioning that no explicit relationships are modelled between entities, since this information is not available in most data sources. The relationships are to be modelled by the DSL and in most cases will be specific for the use case.

3.3 Defining Models and Mapping

After analysis of the domain and the corresponding data sources a definition of the models and mappings with our DSL takes place. This requires the feedback service development environment and supplies tools to create the actual artefacts. After this, the generated sources are available for manual modification or amendment with special features within the development environment. A second tool will be used to build the server software, which makes the definitions available in the feedback service. Finally, a tool can be triggered to provide the software for deployment.
The grammar enables structured analysis of the user defined models and simplifies parsing by providing classes which have to be customized to extract the actual definitions. The next step is validation to deal with patterns which cannot be covered by the formal syntax. This should however be kept very small and at best the grammar structure should be designed to avoid such checks. Not every aspect can be checked by the grammar type supported by ANTLR, for example if an object referenced was defined before can only be modelled by a more complex class of grammars which are context sensitive. The next step is then the generation of the actual Java-artifacts. We use a template engine for this. The data extracted with the visitors is then put into placeholders inside the templates and saved to the development environment.

```
package "eu.smarticipate.hamburg.dsl"
define model Baum {
        "hausnummer" is string
        "kronendurchmesser" is realnumber
        "pflanzjahr" is number
        "strasse" is string
        "baumtyp" is string
}

define service Baum {
url "http://geodienste.hamburg.de/HH_WFS...
epsg 25832
type "app:strassenbaumkataster"
provides entity Baum {
        "hausnummer" from "hausnummer"
        "krone"from "kronendurchmesser"
        "pflanzjahr" from "pflanzjahr"
        "strasse" from "strasse"
        "baumtyp" from "sorte_deutsch"
        }
}
```

Fig. 3.3 Example of a model and data access definition

Figure 3.3 shows an example data and mapping definition for the Hamburg tree planting use case. The package statement defines an arbitrary scope for the definition and is needed to separate different use case implementations. As of the concept, definitions from other use cases are kept separate.
Each entity classified by a name has several attributes identified by a name and a type. As types numbers, strings and date are possible right now.
Next a service is defined having a name and properties for accessing data provided via a Web Feature Service (Vretanos 2014). Each of the services produces entities as result. Any entity defined before can be assigned, and also several services offering the same type of entity are possible. Next is the definition of how the data source attributes map to the entities. This is kept very simple as there are already lots of existing tools for data mapping, which can offer services tailored to the use cases. However, it is still possible to extend the DSL to enable more sophisticated mapping.
The next step is to generate technical artefacts representing the entities and services. For this we use an automated build script to generate java-based artefacts.
The build system in our case is gradle[10] which enables to define tasks in the build process. In our case the task starts a parser for the model and mapper definition and generates java artefacts. These artefacts are then available in the source code structure of the feedback service, filed under the package definition given in the aforementioned package statement. It is advisable to use the standard java naming conventions or a plain name without special characters.

3.4 Rule Language for tree checks

Based on the language elaborated for tree checks a program example as in Figure 3.4 can be defined. Principles and guidelines for a language design are adopted from Fowler (2010), Karsei (2009) and van Roy (2004). First a package statement is used to scope programs. Then the name of the program is defined as "TreeRule" which is also the name to register as endpoint for service execution. After this the data sources to use are defined and named. "input" is the data to be supplied by the caller of the service. Here an entity name "Baum" is expected.

```
package "eu.smarticipate.hamburg.dsl"
define TreeRule as
datasource Building buildings, Baum hhtrees,
Lichtsignalanlage lsa, Landesgrundbesitz lgb

input Baum atree

check atree within distance 5 meters of buildings
on fail "Tree within distance of 5 meters"

check atree within distance 10 meters of buildings
on fail "Building within distance of 10 meters"

check atree within distance 10 meters of lsa
on fail "Lichtsignalanlage within distance of 10 meters"

check atree within distance 1 meters of lgb
on fail "Landesgrundbesitz within distance of 1 meters"
```

Fig. 3.4 Example of check rules program for tree planting

[10] https://gradle.org

The actual data is to be supplied as JSON data. This data will be parsed according to the definition and is provided as input to the rule. Now the actual check clauses start: mostly they determine if some datasource has objects close to the input object. If this is the case the "fail" statements will emit a warning to the overall result of the rule. The results are provided to the technical clients also as JSON or XML.

4. EVALUATION

In the context of the smarticipate project a workshop in Hamburg together with domain experts of the city was conducted either to gather requirements and to discuss the possible usage scenarios and feasibility of automated feedback generation based on already available geodata. The main use case discussed was about citizens suggesting where to plant new trees with an online application. Citizens should get direct feedback about the suitability of the selected planting position. Table 4.1 shows the main criteria identified for automated feedback generation, considering data available. However, it got clear that a lot of topics cannot be covered due to missing or inaccurate data. Especially infrastructure that is maintained by private companies is not accessible as open data. Also, manual control due to the processes established within the city necessitate that the system cannot make final decisions: these have to be made by city employees also due to legal reasons.

Topic	Description
Land use and planned actions	Land already in use for buildings or streets is obviously not useable for planting trees. Also planned actions should be considered if data is available. For example, if construction is planned for a street no new trees should be planted until the construction has been finished.
Species is determined by neighbourhood of species	If for example an alley made up of all the same species of trees is given, a new tree should be of the same species, if the tree is reasonably close to the alley.
Species can be changed by definition	In contrast to the rule given above sometimes a tree species doesn't work out as desired on a certain location. Also, a possible climate change might influence the selection of trees to be planted. A rule should be implemented that overrides the rule of keeping the same species with a defined other species.
Distance to street lighting	Trees grow and possibly will mask street lights nearby. A minimum distance should be kept from such positions. Positions of street lights need to be given.
Distance to other trees	A certain distance to other trees is needed to avoid competition of both trees, for example sycamore trees need a distance of at least 8 meters, around 15 meters would be best.
Distance to traffic signs or traffic lights	Trees grow and possibly will mask traffic lights nearby. A minimum distance should be kept from such

	positions.
Flooding areas	Areas which can be flooded should be avoided in general or a species that can cope with these needs to be selected.
Condition of soil	Basically, every ground close to road works is denaturised and needs to be refurbished. Thou the surroundings of the potential tree position should be free of poisonous substances or demolition materials.
Privately owned land	Privately owned land is excluded in all cases

Table 4.1 Topics and rules suitable for automated checks

The actual service was used by several demo applications, for example a 3D visualization of the Hamburg city area together with a tree check rule program to demonstrate the possibilities of giving feedback in a 3D environment considering rules defined in Table 4.1. The architecture of the application is shown in Figure 4.1.

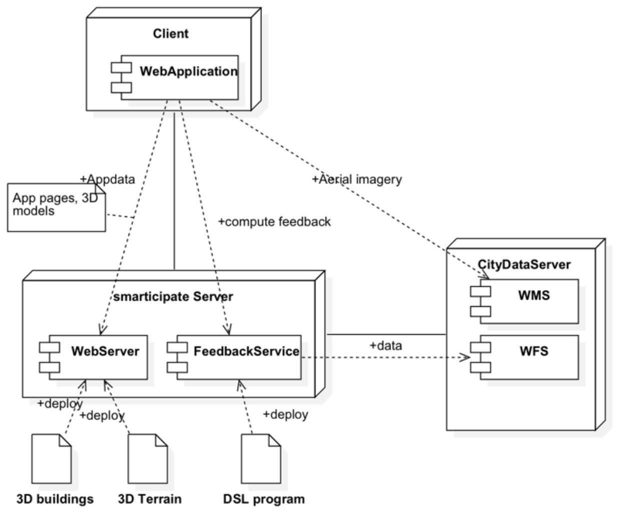

Fig. 4.1 Interactive 3D city visualization of Hamburg with integrated Feedback Service

The prototype has been shown at several events such as the smartathons in the context of the smarticipate project. A smartathon is special event similar to a smart hackathon where citizens gather with local administration personnel and discuss topics around urban planning and tools for planning. These events provided hands-on sessions and also a lot of direct interaction between planners, citizens and also IT experts. The demonstrations revealed great potential for such an automated feedback service. However, users stressed that the impact on the planning and the results along with the factors leading to an assessment should be more visual. This means that also the influences should be made visual for example by colouring them and not only showing a surrogate like a coloured cylinder as we did in the demonstration.
The following results were gathered during the events by observation and dialogues while using the system and are to be considered for improvements:

- A toolkit to visualize the results is needed: It is a hard task for visualization experts. Domain-Specific Languages can also here be applied as a means to apply proven visualisation patterns and stereotypes in the 3D application level by end-users (domain experts).
- The Feedback Service must be very clear regarding the quality of the results provided. It's not very likely that in all possible cases all details are given in a way that the feedback is deterministic and error-free and this has to be conveyed to observers. Missing aspects or missing data can lead to questionable results and to counter this the system need to give anytime an explanation which data was used and which rules have led to the result. For example, in the tree scenario, there may be circumstances beyond control, which hinders ultimately the planting of a new tree, which were not known due to a bad data situation, e.g. unknown pipes or hazardous ground below the area. It must be clear that a service works on models which make assumptions and those have to be transparent when interpreting the results.
- The Rules should be displayed and also the exact causes for the results should be emphasised visually. Also, additional information about the origins of the rules and further reading hints should be delivered.
- Missing data or missing rules are to be considered. There may be steps involving manual interaction by the city, searching archives for example or cases where simply no data is available and someone needs to do an on-site inspection. If this is the case the system should at least tell about those.
- Lots of negative responses from a feedback service when using online platforms to propose new developments can lead to frustration of users. It would be nice to have a proposal or guidance from the system where to place trees.
- On the other hand, discussions with domain expert show that it can complicated to find a suitable spot, for example due to regulations and the data situation. This can raise the awareness for the intrinsic problems in urban planning, even in such simple cases such as planting a tree and helps also explaining such situations to the public.

5. FUTURE WORK

The results from evaluation show that it is vital that rules and the outcomes of the feedback service are made visual and explain also the reasons why a rule holds. The way the rules are now transformed to code are too static for such a flexible processing and other ways of execution will be researched such as decision tables, production rule systems or dependency networks (Fowler 2011).
Also further evaluation sessions with experts from other domains will be conducted in upcoming events or with interested cities to investigate the practical applicability of the concept.
Another major aspect is the engineering process mentioned in chapter 2. These processes are well-established in IT-industry but not in city administration. There should be an overall process architecture as sketched in Krämer, Khan and Ludlow (2013) which could be further extended to serve as a platform for participatory processes. Another more technical aspect for research is about the DSL engineering itself: Since language design is an involved task is there a convenient way to involve end-users beyond capturing requirements or does this remain

in the realm of technicians? It is clear that such a language needs not be complete in the sense of computability theory as it always will be specific for a domain. The open research question is what the borderlines of such a domain are, so that it can be applied effectively and efficiently.

6. CONCLUSION

We have shown that DSLs are a suitable tool for providing feedback to citizens based on actual data and engineered knowledge in cities. DSL can be a convenient way to work with geospatial data also for non-technical experts focusing on domain problems rather than special technical implementations.
The concept has the potential to mitigate problems which typically arise when laymen are confronted with complex planning. It can help to make reasons for decisions visible, based on open data thus fostering transparency and mutual understanding of problems. Also, by exchanging the DSL scripts different perspectives on a scenario can be taken and evaluated, enabling different stakeholders to share their point of view. In comparison with the visual editor conceived from Malewski, Dambruch and Krämer (2015) in which they present a concept of a combination of 3D visualization and Domain-Specific Languages as means for interaction and analysis, we adopted a micro service approach and eliminated the need for annotated data. The data will be transformed by using the language while accessing the data, which reduces efforts for set up of data drastically. Since there is already geodata about various topics available this data can be used to automate routine inquiries from citizens to some extent. It is expected that simple cases can be automated quite well, e.g. checking for obstacles like buildings or ownership of the ground so that city employees can concentrate their work on special cases, ambiguous data or data only available on paper, thus offloading them with such tedious tasks.
Considering experiences from past projects the approach to utilize DSLs is more appropriate as, for example use of Semantic Web technology due to the following aspects:

- Data availability – usable annotated data or even RDF is not available in the participating cities.
- Additional work overhead for annotating data and redundant data storage is not feasible for cities.
- Expert users are typically not familiar with complex IT-concepts not belonging to their domain and would need support on long term basis.
- Definition of dynamic aspects, actions and visualization is more important than reasoning.
- Deriving results from data should be as easy as possible regarding the skills of users, which means that there should be means for representing expert knowledge on different abstraction levels.

The DSL based feedback service offers a lot of possibilities to develop custom DSLs, which are small and easy to use. On the other hand, the engineering of a DSL, syntax definition and mapping to executable code is still a challenging task. By having clear separations of roles, Domain Experts can make use of technology through a language façade tailored to their needs by IT experts. As a consequence, developers can focus on technical challenges while Domain Experts can focus on their domain challenges. The key is here to collaborate in a managed fashion between both worlds. Also, the DSL can offer a stable interface to data and processing, while the technology to implement domain knowledge can be exchanged easily to leverage new technology such as cloud computing or also using other implementations offering better performance. The trade-off is to find a language that is close to the domain and not too abstract, but also reusable.
From the IT perspective, it is rather questionable if every city needs to have a particular DSL of her own for the same topic as in other cities. One could rather think of a community driven engineering process, where an experts panel agree on standardized elements, which could serve as blueprints. A zoo of DSLs dealing with the very same topic can be avoided, but if special features are needed they can be implemented right away. In comparison to a generic model as Simple Feature model (Herring et al. 2010) the possibilities to model the semantics of rules and model elements is a major advantage in that the rules can be self-explanatory and readable even by laymen. So, in principle we can think of the DSL development as an annotation process to data, which we consider as a major step in making open data not just accessible but usable.

ACKNOWLEDGEMENTS

Research presented in this paper is partly funded by the European Commission's H2020 Framework Program Project *smarticipate*, Grant # 693729.

REFERENCES

Dambruch, J.; Krämer, M. (2014): Leveraging public participation in urban planning with 3D web technology. In Proceedings of the 19th International ACM Conference on 3D Web Technologies, Vancouver, BC, Canada, 8–10 August 2014; ACM: New York, NY, USA; pp. 117–124.

Dambruch J., Stein A., Ivanova V. (2016): Innovative Approaches to Urban Data Management using Emerging Technologies. In: Proceedings of REAL CORP Conference; pp. 375-384.

De Nicola A., Missikoff M., Navigli R. (2009): A software engineering approach to on-tology building. Information Systems, volume 34 Issue 2; pp. 258-275.

Dambruch, J. (2016): Semantic data integration software and semantic representation concept. Deliverable 3.2 of the smarticipate project. https://www.smarticipate.eu/resources/

Dambruch, J. (2016): Interface and Toolkit Redbook. Deliverable 5.2 of the smarticipate project. https://www.smarticipate.eu/resources/

Fowler M. (2011): Domain-Specific Languages, No. ISBN 0-321-71294-3, Addison Wesley.

Herring J. (ed.) et al. (2010): OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture, Open Geospatial Con-sortium Inc., http://portal.opengeospatial.org/files/?artifact_id=25355.

Karsai G., Krahn H., Pinkernell C., Rumpe B., Schindler M., &Völkel S. (2009): Design Guidelines for Domain Specific Languages. In: Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM' 09).

Khan, Z., Dambruch, J., Peters-Anders, J., Sackl, A., Strasser, A., Fröhlich, P., Templer, S., Soomro, K. (2017): Developing Knowledge based Citizen Participation Platform to Support

Smart City Decision Making: The Smarticipate Case Study. In: MDPI Information Journal 2017, 8(2), 47, Special Issue on smart City Technologies, Systems and Applications. doi:10.3390/info8020047 http://www.mdpi.com/2078-2489/8/2/47 .

Kovalerchuck, B., Schwing, J. (2004): Visual and Spatial Analysis – Advances in Data Mining, Reasoning and Problem Solving. Springer ISBN 1-4020-2939-X. Dordrecht, The Netherlands.

Krämer M. (2014): Controlling the Processing of Smart City Data in the Cloud with Do-main-Specific Languages. In: IEEE/ACM 7th International Conference on Utility and Cloud Computing, pp. 824-829.

Kramer, M.; Khan, Z.; Ludlow, D. (2013): Domain-specific languages for agile urban policy modelling. In Proceedings of the 27th European Conference on Modelling and Simulation (ECMS), Alesund, Norway; pp. 673–680.

Krämer M., Stein A. (2014): 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 and regional development in the information society GeoMultimedia.

Malewski C., Dambruch J., Krämer M. (2015): Towards Interactive Geodata Analysis through a Combination of Domain-Specific Languages and 3D Geo Applications in a Web Portal Environment. In: Proceedings of REAL CORP Conference 2015, pp. 609-616.

Mernik M., Heering J., Sloane A. (2005): When and How to Develop Domain-Specific Languages. ACM Computing Surveys, Vol. 37, No. 4, pp. 316-344.

Parr, T. (2012): The Definitive ANTLR 4 Reference, The Pragmatic Bookshelf, Dallas, Texas and Raleigh, North Carolina, USA.

Ruppert, T. (2018): Visual Analytics to Support Evidence-Based Decision Making. Technische Unniversität Darmstadt, Darmstadt, http://tuprints.ulb.tu-darmstadt.de/7045/.

Ruppert, T., Dambruch, J., Krämer, M., Kohlhammer, J., Balke, T., Gavanelli, M., Bragaglia, S., Chesani, F., Milano (2015): M. Visual Decision Support for Policy Making: Advancing Policy Analysis with Visualization. In Public Administration and Information Technology; Janssen, M., Ed.; International Publishers Association: Geneva, Switzerland.

Van Roy, P., Haridi, S. (2004): Concepts, Techniques, and Models of Computer Programming. Massachusetts Institute of Technology.

Vretanos P. A. (ed.) et al. (2014): OGC® Web Feature Service 2.0 Interface Standard, Open Geospatial Consortium Inc., http://docs.opengeospatial.org/is/09-025r2/09-025r2.html.