DashThings
Project in Software Customization Ubiquitous Computing

Nikolaj Schademose Reibke, nirei12@student.sdu.dk
Emil Sebastian Rømer, emroe12@student.sdu.dk

May 24, 2016

1 Introduction

The first step towards automatization of buildings and homes with Internet of Things is to understand the data provided. Through this understanding it is then possible to decide actions to take based on one or more sensor input given. Making decisions and taking action for one building is not generalizable enough to apply to other buildings without customization. Further are actions rooted in a problem that needs to be solved. This means that customization is needed whenever one of the following is true:

- Problems that needs to be solved might differ from one building to another building
- The buildings infrastructure might not be the same between buildings meaning that the solution might differ
- Context of the environment have changed

In general there is a great need for customization or customizable application in building automatization through Internet of Things.

Using charts or graphs to visualize large amounts of complex data is easier than poring over the data itself. Data visualization is a quick, easy way to convey concepts in a universal manner. Data visualization can be used for different purposes. Some of these are:

- Decision making
- Identify areas that need attention or improvement
- Clarify which factors influence behaviuor
- Identify behaviour or patterns

The problem is that a visualisation is not universal in sense of the purposes it can be used for. Therefore, there are a great need for customized or customizable visualizations that supports the different purposes.

The purpose for the project is to have a website, with responsive design, which can be viewed on any browser like Mobile, Desktop, Laptop and tablet. The pages on the website will contain Links between the pages on the webpage. It will also be possible to link to external webpages. Additionally the pages will contain graphs and tables which shows data. The data will either be fetched from external sources or external sources will have the possibility to post data to a data source given a predefined data schema.

Data in data sources can be transformed using formula expressions and have to go through at least 1 formula expression in order to be displayed on a graph. This expression will be on the graph.

In order to enable the user to create the above in a easy customizable way three self made language will be used.

2 Tools

- Explain which tools and languages we have used (R, which R packages, Emils python/java code etc)

3 section1

4 The Domain Specific Language, DSL

4.1 Overview

The DSL is divided into three components, a arithmetic expressions language, a language for specifying a web site and a language for specifying input sources for the system. The expression language is a utility language which is included by the two others. This language defines how a user can create arithmetic expressions in either of the other two languages. The Web visualizer is a language to create dynamic web sites, like the sample page shown on the front page. This language lets the user create pages on a site, link different pages to each other

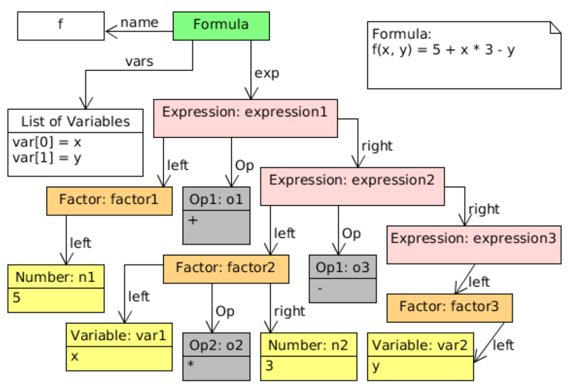

Figure 1: Instance of Formula

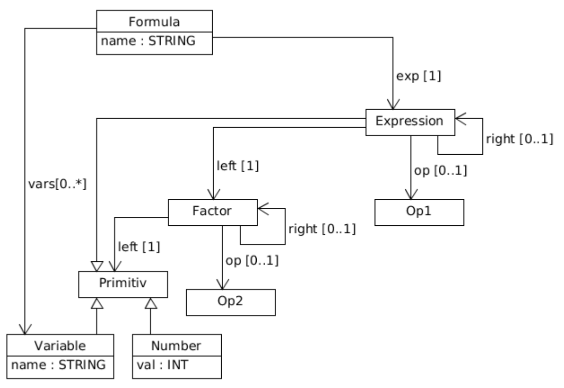

Figure 2: Model of the formula module

and visualize information through the use of graphs. The Datasources language, which is still to be developed, will be used to specify different data sources, like for instance a external databases or APIs. In addition this language will also be used to specify internal persistence and a API for other systems to use for posting data to the Dashk system.

4.2 The Formula Expression Language

The Formula Expressions DSL is responsible for understanding mathematical formulas, which will be applied to tables of data.

Meta model The formula metamodel consist of multiple elements. The types include in the elements included in the model are:

1. Formula: The overall Structure Element
2. Variable: The varaible names included in the formula
3. Expression: The right side of the equal sign in $f(x) = x \cdot x$. The Expression collects the parts which is connected with plus($+$) and minus($-$).
4. Factor: A semi-structure of the expression, Factors collects the parts connected with multiplication($\cdot$) and division($/$).
5. Op1: Identifies a addition or substraction operation
6. Op2 Identifies a multiplication or divison a operation
7. Primitiv: Identifies as either a number or a Variable

Given the following the example:

$$f(x,y) = 5 + x \cdot 3 - y$$

The rules for the grammar is explained as below also referring to Figure 1:
A Formula Contains a name which is defined a the letter/word in front of the paranthesis also underlined in the following formula: $\underline{f}(x,y) = 5+x\cdot3-y$. It also contains a list of variables, defined within the paranthesis seperated by "," as underlined: $f\underline{(x,y)} = 5 + x \cdot 3 - y$. Finally the expression of the formula is everything on the right side of the equal-sign("="), here follows the rule that every variable used in the expression has to appear inside the paranthesis of the formula as well, as it works as a function, the expression part underlined: $f(x,y) = \underline{5 + x \cdot 3 - y}$.

Looking at the expression: $5 + x \cdot 3 - y$ it is broken down by taking continouisly working from the left to the right, from the expression perspective by taking one factor at a time. For the Expression the factors for the expression is 5, $x \cdot 3$ and y as underlined: $\underline{5} + \underline{x \cdot 3} - \underline{y}$. Once a factor is found. the rest of the expression is defined as a new expression which is the right side to the Expression. At first the expression would contain the factor 5 and the of the expression would be a nested expression for the overall expression.

Language Validation

Code Generation

4.3 The Web visualizer Language

For generating dashboards custom language have been created. This language specify:

- Which pages the web-interface consist of
- The navigation between pages
- Which data to display and where
- Where and how to obtain the data

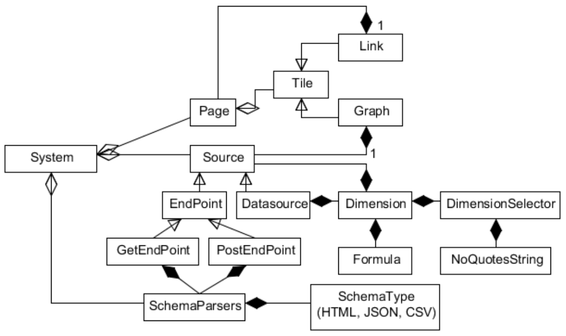

Figure 3: Model of the language

Internal data streams

External data streams

- How to manipulate the data streams

The following subsection will look into detail of how the language is structured and the syntax and grammar of the language.

The Grammar Figure 3 shows the structure of the language. This model shows that the language defines of a system containing three parts:

- Pages
- Sources
- Schemas

The Pages, is what the language in turns will use to create web pages in a system defined in the language. A page is a collection of tiles. Each tile is an extendible definition of a visual object on a page. In the current version, there are two types of Tiles, Links and Graphs. A Link definition contains a pointer to a page. A Graph however, has a reference data *source* from witch it needs to fetch data from. Figure 4 shows a sample code snippet from the grammar definition. The Figure shows how the grammar is defined for the Pages, Tiles, Links and Graphs. Similar does Figure 5 show and example use of this grammar. This example creates a system object with two pages inter connected by links and with a number of graphs.

The Sources, can be of two different types, an *EndPoint* or a *DataSource*. An EndPoint is intended to function as interface towards external systems or data sources. Whereas a Datasource is an internal definition intended for filtering, data manipulation or data grouping.

This Source, Endpoint, Datasource-Dimension structure have been defined as a compositional pattern, with source as the component, the Endpoint as a leaf and the Datasource as the composite. This

```
System:
    (pages += Page | sources += Source
         | schemas += SchemaParser)*
;
Page:
    'Page' name=ID
    '{'
    (tiles += Tile)*
    '}'
;
Tile:
    Link | Graph
;
Link:
    'Link' name=ID 'to' page=[Page]
;
Graph:
    'Graph' name=ID    source=[
        Datasource] ('label=' label=
        STRING)?
;
```

Figure 4: Sample grammar definition, For a complete view of the grammar, see Appendix

```
Page index
{
    Link toPageOne to pageOne
    Graph testGraph a
}

Page pageOne
{
    Link toHomePage to index
    Graph testGraph a
    Graph testGraph1 c

}
```

Figure 5: Example language use, defining pages

```
Datasource temperatureRaw {
    Dimensions:
        Formula x20508a1(x) = x using
            temperature[x20508a1] as x,
        Formula x205101(x) = x using
            temperature[x205101 ] as x,
        Formula x205111(x) = x using
            temperature[x205111 ] as x
}
```

Figure 6: Example language use, defining data sources

```
GetPoint temperature {
    url "http://10.123.3.12:8079/api/
        query"
    json "select\␣data␣in('{(datetime.
        datetime.fromtimestamp(time.
        time())␣-␣datetime.timedelta
        (1)).strftime('%m/%d/%Y')}',␣
        '{datetime.datetime.
        fromtimestamp(time.time()).
        strftime('%m/%d/%Y')}')␣where␣
        Path␣like␣'/1␣-␣Ground/%/
        Temperature'"
    Headers {
        "type" : "application/json"
    }
    use Schema schemaB
}
```

Figure 7: Example language use, defining endpoints

pattern makes the it possible to compose any number of different structures. In addition every defined dimension comes with a formula, which will be applied to the data. Since Source can have multiple dimensions (data streams) when specifying a how a Source an additional dimension selector can be specified to select a subset of the dimensions. Figure 6 shows and example of a Datasource created through the language. The first line creates a Datasource and the declares a list of dimensions. Each declaration of a dimension starts by declaring a formula. After the formula follows a declaration of Sources to use in the formula.

The Schemas, or SchemaParsers are a global way of defining how to parse date from an external source. This is both needed when defining an source to request data from and when external sources posts data to the system.

Meta model Below is the first meta model of the system. This model matches the language specification one to one. The System once again is a collection of pages, which again consists of a number of tiles which can be of the two previous described types.

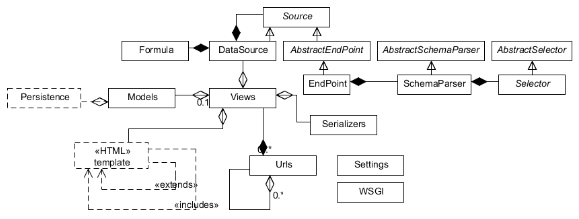

Figure 8: Model of the Website

Language Validation

Code generation To create the website the project uses the Django framework. This framework uses an MVC architektur which is extended with a template pattern with HTML documents. In order to make the output easy readable and customizable for a user, the first meta model needs to be mapped to a second meta model with more explicit information which matches the MVC architecture better. This second model can be seen in below diagram.

The first step of mapping the previous model into the second meta model, is to create a system with a configuration. Secondly a default controller is added with the default Django Admin configuration pages. This controller gets a number of URLs and default views. After this a second controller gets added, this controller is used for the pages defined in the language. Each page gets transformed into a url entity, view entity, model and a template HTML-file. The HTML file holds a easy customizable structure of the content added to the page. The view and model will in time be connected to the Data sources from the previous specified language. In addition will the view of a page be responsible of rendering the template with context and data from a datasource. After this model have been created each entry in the model is looped through and generated into a file.

Pages

Data sources

Api

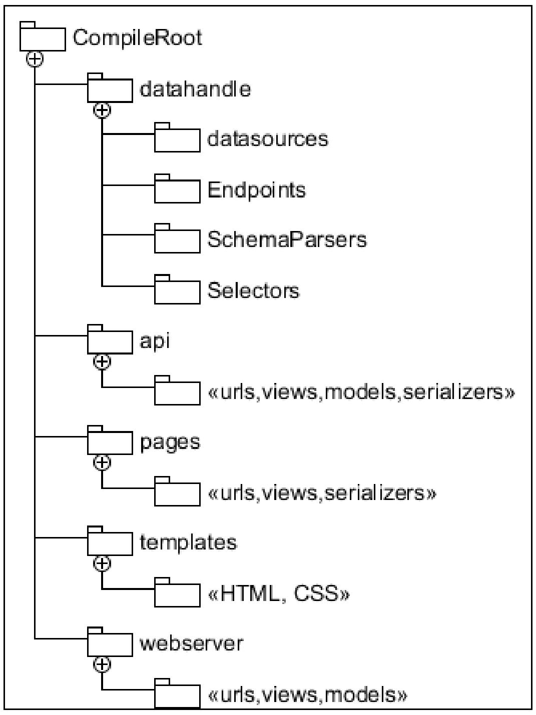

Figure 9: Package diagram of generated sources

5 section3

6 Conclusion

- Concussion

A Author Contributions

All participants of the project has contributed equally to the entire report, in addition to that all participants has additionally .

B The Grammar

```
grammar org.xtext.sdu.
    iotvizualizerlanguage.Vizualizer
    with org.xtext.sdu.
        formularzlanguage.Formular

generate vizualizer
    "http://www.xtext.org/sdu/
        iotvizualizerlanguage/
        Vizualizer"

System:
    (pages += Page | sources += Source
          | schemas += SchemaParser)*
;
Page:
    'Page' name=ID
    '{'
    (tiles += Tile)*
    '}'
;
Tile:
    Link | Graph
;
Link:
    'Link' name=ID 'to' page=[Page]
;
Graph:
    'Graph' name=ID   source=[
        Datasource] ('label=' lael=
        STRING)?
;

Datasource:
    'Datasource' name=ID
    '{'
    'Dimensions' ':'
    dimensions+=Dimension (','
        dimensions+=Dimension)*
    '}'
;

Dimension:
    'Formula' name=Formula
        sourceSelectors+=
        DimensionSelector ('and'
        sourceSelectors+=
        DimensionSelector)*
;

DimensionSelector:
    'using' source=[Source]'['
        selectVar=NoQuotesString ']' '
        as' name=ID
;

NoQuotesString:
    name=ID
;

Source:
    EndPoint | Datasource
;

EndPoint:
    GetEndPoint | PostEndPoint
;

PostEndPoint:
    'PostPoint' name=ID
    '{'
    'url' url=STRING
    'use␣Schema' parser=[SchemaParser]
    '}'
;

GetEndPoint:
    'GetPoint' name=ID
    '{'
    'url' url=STRING
    ('json' json = STRING)?
    'Headers' '{'
       headers+=Header (',' headers+=
           Header)*
    '}'
    'use␣Schema' parser=[SchemaParser]
    '}'
;

Header:
    keyword=STRING ':' value=STRING
;

// A SchemaParser is used to parse a
    schema(data structure) into a
    time series
SchemaParser:
    'Schema' name=ID
    '{'
    'SchemaType' '=' schemaType=
       SchemaType
    selectors+=Selector+
    '}'
;

enum SchemaType:
   XML='XML' | CSV='CSV' | JSON='JSON
        '

;

// Select the path to a specific
    dimension of the data
Selector:
    'Selector␣as␣' name=ID '{'
    steps+=STRING ('␣->␣' steps+=
       STRING)+
    '}'
;
```