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
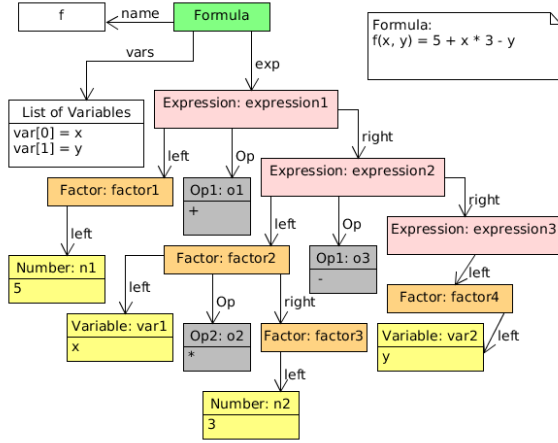
Figure 1: Instance of Formula

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 as the letter/word in front of the paranthesis also underlined in the following formula: $\underline{f}(x, y) = 5 + x \cdot 3 - y$. In Figure 1 "f" is defined as "name" seen as an attribute for the overall Formula. It also contains a list of variables, defined within the paranthesis seperated by "," as underlined: $f\underline{(x, y)} = 5 + x \cdot 3 - y$. The variables is saved as a list under formula as "vars" seen at List of Variables. 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: $f(x, y) = \underline{5 + x \cdot 3 - y}$ also saved as "exp" and seen as "expression1" in Figure 1.

Looking at the expression("expression1"): $5 + x \cdot 3 - y$ it is broken down by continouisly working from the left to the right. the left part of the expression is defined as a factor until it meets an addition or substraction sign. For the formula example this would put only the number "5" into the left side factor as "factor1" in Figure 1, futher containing "5" in the Number object "n1". Expression2 is then the rest of the equation after 5+. The new expression contains again one factor which is the left side($x \cdot 3$) seperated by the substraction sign The factor "factor2" contains on the left "var1" which is x, and on the right another factor is spawned seen as "factor3" in Figure 1, only containing a left side being the number seen in "n2" as 3. Coming back up to the expression. it contains the last part of y as "expression3" which only a left side being a factor(factor4) and that factor also only contains a left side which is the variable "var2" being the y.

For the variables "var1" and "var2", both is included in the List of Variables in as seen in Figure 1

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
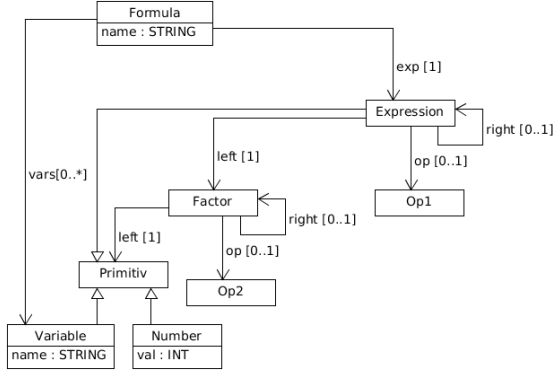
- Which data to display and where

Figure 2: Model of the formula module
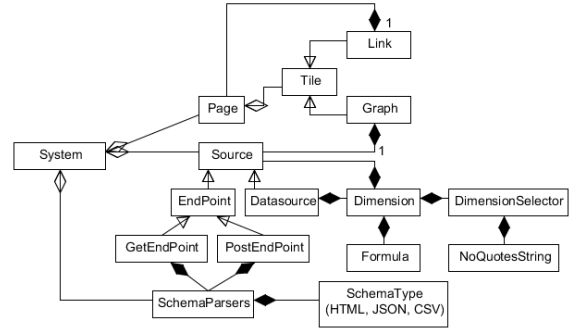
Figure 4: Model of the language

```
 5 System:
 6     (pages += Page | sources += Source | schemas += SchemaParser)*
 7 ;
 8 Page:
 9     'Page' name=ID
10     '{'
11     (tiles += Tile)*
12     '}'
13 ;
14 Tile:
15     Link | Graph
16 ;
17 Link:
18     'Link' name=ID 'to' page=[Page]
19 ;
20 Graph:
21     'Graph' name=ID source=[Datasource] ('label=' lael=STRING)?
22 ;
```

Figure 3: Sample grammar definition

```
 1 Page HomePage
 2 {
 3     Link GotoTemperatures to Temperatures
 4     Graph testGraph a
 5 }
 6
 7 Page Temperatures
 8 {
 9     Link GotoHomePage to HomePage
10     Graph temperatureGraph temperatureRaw
11 }
```

Figure 5: Example language use, defining pages

- Where and how to obtain the data

    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 4 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 3 shows how this grammar was specified. **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 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 pattern makes the it possible to compose any number of different structures. **Schemas**

Below table shows the grammar for the web visualizer language to the left and an example of use on the right. As seen from the left side the system consist of a number of pages which again consists of a number of tiles. One tile can, in current implementation be either a link to another page or a definition of a graph. Since the datasource language have not yet been implemented the web visualizer language currently work with a hard-coded testset. Further are only one graph type implemented (a line chart), however the system makes use of D3 framework meaning that extending the system with other graph types is a smaller task.

The example on the right site creates a System instance with three pages, each page with a number of Links and Graphs.

3

```
60⊖ Datasource temperatureRaw {
61      Dimensions:
62          Formula x20508a1(x) = x using temperature[x20508a1] as x,
63          Formula x205101(x) = x using temperature[x205101 ] as x,
64          Formula x205111(x) = x using temperature[x205111 ] as x,
65          Formula x20601b1(x) = x using temperature[x20601b1] as x ,
66          Formula x206031(x) = x using temperature[x206031 ] as x,
67          Formula x20604b1(x) = x using temperature[x20604b1] as x ,
68          Formula x225081(x) = x using temperature[x225081 ] as x,
69          Formula x225101(x) = x using temperature[x225101 ] as x,
70          Formula x225111(x) = x using temperature[x225111 ] as x,
71          Formula x22601b1(x) = x using temperature[x22601b1] as x ,
72          Formula x226031(x) = x using temperature[x226031 ] as x,
73          Formula x226041(x) = x using temperature[x226041 ] as x
74  }
75⊖ Datasource temperatureAvg {
76      Dimensions:
77          Formula average(x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12) = (
78  }
```

Figure 6: Example language use, defining data sources

```
51⊖ GetPoint temperature {  |
52      url "http://10.123.3.12:8079/api/query"
53      json "select data in('{(datetime.datetime
54      Headers {
55          "type" : "application/json"
56      }
57      use Schema schemaSmap
58  }
```

Figure 7: Example language use, defining endpoints

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.

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.

Figure 8: Model of the Website

Figure 9: Package diagram of generated sources

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

4

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 .