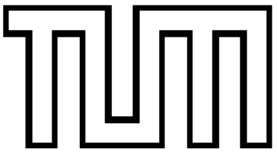

FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Masterarbeit in Wirtschaftsinformatik

A concept for the visual and interactive impact analysis and simulation of data changes to enterprise metrics

Ein Konzept für die visuelle und interaktive Analyse und Simulation der Auswirkungen von Datenänderungen auf Unternehmenskennzahlen

Author:	Matti Maier
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Thomas Reschenhofer
Submission Date:	August 21st, 2014

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Ich versichere, dass ich diese Master’s Thesis selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Garching, den 21. August 2014

Matti Maier

Abstract

Enterprise metrics in business accounting are based on a data model in one or more business information systems, while also consisting of multiple influencing variables. The calculation of a metric is often performed on a large data basis, for example on order receipts, accounting information and other metrics. When evaluating the calculation results on a large model, the single variable's impact on the metric is elusive. A simulation of the calculation with changed influencing variables empowers the user to recognize the impact of each influencing variable on all defined enterprise metrics and thus he/she can better react to environmental changes. The work in hand describes an interactive simulation of a changed influencing variable's impact on metrics and it answers the question how to visualize the changes and their impact.

At first the business and technical requirements for the solution are described, followed by a literature and market research for validated solutions to the problem. On the basis of these findings a new solution approach is developed. To validate its technical viability, a prototypical implementation is created. Examples from hospitality management and enterprise architecture management show the solution's applicability.

Index

1 Motivation and Problem Statement

"If you can't measure it, you can't manage it" is a famous statement from Robert Kaplan's book Balanced Scorecard (1996). Although some disagree with this simplified statement, it is a fact that all companies need to measure their performance, at least for law compliance. Parmenter (2010) describes four types of performance measures in organizations: key result indicators (KRIs), result indicators (RIs), performance indicators (PIs) and key performance indicators (KPIs). Many KRIs are the result of various actions and cover a longer period of time than, for example, KPIs. "The PIs help teams to align themselves with their organization's strategy." (Parmenter 2010, p. 3) "A performance indicator can be defined as an item of information collected at regular intervals to track the performance of a system. Performance Indicators (PIs) are collected in many complex systems which, like education, deliver a service." (Fitz-Gibbon 1990) In the context of Paramenter, KPIs need to be in the center of a business's attention since they aim is to increase the performance dramatically (Parmenter 2010, p. 1). They are nonfinancial measures, which should be monitored frequently and have a significant impact on the business.

To determine the most relevant influencing factors on a business, one option is to look at the business model, the performance measures currently being used and the relations between these measures. Since performance measures are based on the data present in the information systems a business uses, it is a relevant consequence to investigate their data and calculation models. Hao et al. (2006) developed a method to analyze and visualize the influencing factors of business processes based on process instance data. By looking at their examples, the complexity of the processes and their influencing factors becomes apparent. Similar holds true for the calculation models of performance measures. The calculation of each metric comprises usually more than two variables. By taking into account that some metrics rely on others and the amount of data the metrics are based on constantly increases, the impact of single variable and thus its importance is almost impossible for a human to assess without the help of a computational system.

In addition to the growth of available data, the environment in which firms operate constantly changes. In order to succeed under these circumstances, constant adjustments and monitoring of the performance measures is required (cf. Barjis, Verbraeck 2010). However, monitoring indicators based on data of the past is not enough to draw conclusions for the future. The identification of the most important influencing variables and the fictive adjustment of these variables simulate a changing environment and thus they allow one to forecast possible effects (cf. Currie, Hlupic 2000). With this information at hand, current and future actions can be adapted to prevent negative consequences.

In the following sections, two examples are described to illustrate the problem and to show possible use cases for a solution approach.

1.1 Example 1: Hospitality Management

According to Harris, Brander Brown (1998) a "wide range of topic coverage, particularly in relation to management accounting, financial management and decision-making techniques" (p. 174) has been researched. Accounting systems and performance measurements have been developed. On the basis of generated data and a direct costing approach typical questions are stated and solved using a simulation-based solution approach, for example:

1. How is the gross margin affected when the room rate is increased by 1.0%, considering 2.0% less guests?

2. In case energy costs increase by 3.0% and, loans and wages by 2.5% and the fixed costs by 2.0%, what are the effects on the profit?
3. Will earnings decrease significantly if the breakfast charge would be included in the room rate?

In order to answer these questions, an artificial and simplified extract of a hospitality information system's data model provides meta data for the analysis.

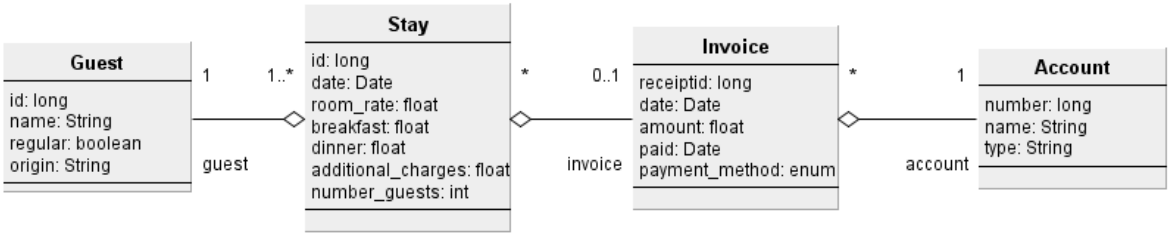

Illustration 1: Artificial data model of a hospitality information system (own diagram)

As displayed in Illustration 1 the data basis is categorized in four major types: *Guest*, *Stay*, *Invoice* and *Account*. The *Guest* relation comprises of a primary key – namely a numerical *id* – and additional fields such as a *name*, whether the guest is a *regular* guest and a string representation of the guest's home address (*origin*). In a real world scenario, the *origin* attribute would most likely be a foreign key to an address or contact relation, but for the exemplary purpose this association is not relevant. In the center of the data model the *Stay* relation references to the *Guest* relation with the semantic that each stay is associated with one guest. However, the actual number of guests (*number_guests*) for a stay can vary. For example, a couple checking in at a hotel has one room, one associated guest, but two actual guests. The attributes *breakfast* and *dinner* represent a rate per room for this *Stay*, dependent on the actual number of guests. In contrast, *additional_charges* are not based on the number of guests, neither is the *room_rate*, nor the *id* which serves as the primary key for this relation. Another reference in the *Stay* relation is targeted at the *Invoice* relation. A *Stay* represents one day and one night at the accommodation. Since guests can stay multiple days in a row, the invoice groups stays together. Furthermore the *amount* attribute of the *Invoice* relation is calculated in this example by applying this formula:

$$a_i = \sum_{s \in S | s_{invoice} = i} (r_s + b_s + d_s + c_s) \quad (1)$$

The amount a_i of an invoice *i* is the sum of *room_rate r*, *breakfast b*, *dinner d* and *additional_charges c* of all *Stays s* with the *invoice* value *i*. The *receiptid* contains the primary key of the invoice which is used to reference an invoice in the foreign key column of the *Stay* relation. The date-typed attributes *date* and *paid* represent the creation date of the invoice and the date when the invoice was paid. How the invoice was paid is stored in the *payment_method* attribute. Furthermore the *Invoice* relation comprises a foreign key to an account referencing the account *number*. This attribute is the primary key for the *Account* relation which otherwise contains only the *name* and *type* of an account. Possible types could be earnings, fixed or variable costs. Another way to look at the types is to view them as a categorization of the accounts.

Based on this simplified model one can calculate basic direct costing measures such as the turnover *u*, gross margin *m*, fixed costs *f*, variable costs *v* and earnings before interests and taxes *e* following (Haedrich et al. 1998, pp. 339–340)):

$$x_t = \sum_{i \in I \mid i_{account} = k_{number}} a_i \quad \forall k \in K \mid k_{type} = t \quad (2)$$

$$u = x_{turnover}$$
$$v = x_{variable}$$
$$f = x_{fixed}$$
$$m = u - v$$
$$e = m - f$$

In (1) the calculation of the invoice amounts is described. Equation (2) builds on (1) and sums all invoices' amounts which reference an account *k* of type *t*. x_t is used as a temporary variable to increase the readability of the equations. The gross margin *m* is computed as the turnover *u* minus the variable costs *v*. Likewise the earnings are the gross margin *m* minus the fixed costs *f*.

Although the example is simplified, the complexity to understand the system in a way that the impact of a single influencing factor such as the room rate can be calculated is obviously not trivial. With a spreadsheet or database application, however, it is easily possible to implement the formulas to calculate such a model. However, this is still insufficient as the user might still have little understanding of the system and he/she is likely only to be able to guess the impact of a single influencing factor/variable.

In addition, changes to the underlying data should not be made in order to guarantee the operative systems to work properly. A common way to avoid this problem is to copy the data to a data warehouse or another database (cf. van der Lans (2012), p. 167f). This way the data is eventually transformed, but it will still not solve the problem that the original data has to be compared to a changed set of data. Duplicating the same data multiple times adds further complexity such as dealing with consistency-gaps, higher storage requirements and an increased need for processing power and network performance. One possible solution to compare two sets is the consideration of a delta set. A delta set contains the differences of two sets. Combining the original data with the delta set generates a virtual set of data (*data virtualization* according to van der Lans (2012)). With this, changes to influencing factors can be simulated and the results can be compared.

To elicit the requirements for a technical solution, an approximation of the data amounts is helpful. According to the accommodation capacity and occupancy study of the federal office of statistics in Germany 34.8% was the average occupancy in 2013 (Statistisches Bundesamt 2014). The average number of beds per hotel is 63.57 and a guest stays an average of 2.7 nights (Deutschland123.de 2014). The official German account system for the gastronomy industry is called SKR70 and has between 130 and 10,000 accounts (DATEV eG 2014). For a range of hotel sizes the number of rows per year (with 365 days) is estimated and displayed in Table 1:

Table 1: Gastronomy industry relation size estimation

Number of Beds	50	100	300	500
Occupancy	34.8%			
Occupancy per Night in Beds	17.4	34.8	104.4	174
Number of Stays per Year	6351	12,702	38,106	63,510
Number of Guests per Year (Factor 2.7)	2,352.2	4,704.4	14,113.3	23,522.2
General Factor of Receipts per Year (10% of the Stays per Year)	635	1,270	3,811	6,351
Sum of Receipts per Year	2,987.2	5,974.4	17,924.3	29,873.2
Number of Accounts in SKR70	<10,000			

In total the number of entities for the *Guest* relation is between 2,352 and 23,522 (factor 10), likewise the *Stay* relation has a size of 6,351 to 63,510 entries and the *Invoice* relation approximately 2,987 to 29,873. From the SKR70 account system one can estimate a size of less than 10,000 accounts. These considerations only estimate the relations sizes for one year, but even over several years the total number of entries is likely not to succeed one million. Yet for a hotel chain with a leveraging factor of ten or more hotels, it is likely to enter large data amounts of at least one million entries for the *Guest*, *Invoice* and *Stay* relations.

1.2 Example 2: Metrics in Enterprise Architecture Management

One area of application is Enterprise Architecture (EA) Management where measures of complexity are applied to control the application landscape. A more precise definition of EA Management is:

> "*EA management is a continuous and iterative process controlling and improving the existing and planned IT support for an organization. The process not only considers the information technology (IT) of the enterprise, but also business processes, business goals, strategies, etc. are considered in order to build a holistic and integrated view on the enterprise.*
>
> *The goal is a common vision regarding the status quo of business and IT as well as of opportunities and problems arising from these fields, used as a basis for a continually aligned steering of IT and business.*" (Ernst et al., p. 13)

Recently the definition, verification and validation of complexity metrics among researchers and practitioners have gained attention and are therefore considered as an example for the application of the approach presented herein. So far various types of metrics have been developed, for example heterogeneity (Schütz et al. 2013), topology (Lagerström et al. 2013) and industry metrics as part of current related research. When regarding an application landscape as a set of interconnected applications (Addicks, Steffens 2008, p. 397) each dependency between business applications is considered an information flow. Business applications run on infrastructure components, for example data base systems and operating systems. Aside from the technical aspects, business applications comprise features categorizing an application into a functional domain. The diagram below shows an overview of the terms.

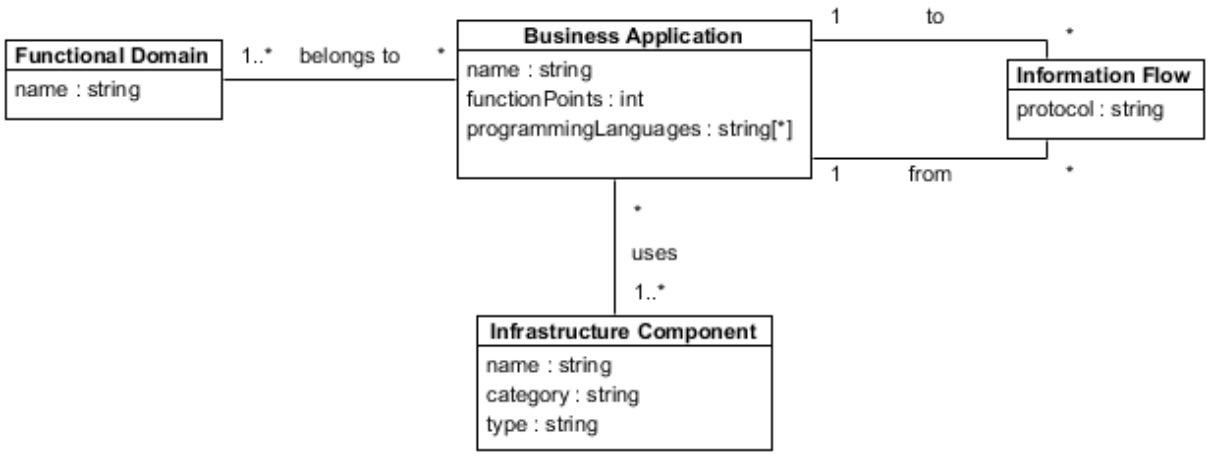

Illustration 2: EA Management - Examplary Data Model (extract from related research)

Based on different datasets comprising data on the application landscape of four banks, complexity measures are defined and modelled. One example of a complexity measure is the database heterogeneity which is a metric expressing a statistical property which refers to the diversity of

attributes of database management systems (cf. Widjaja et al. 2012). Another example of a complexity metric is the functional scope. It refers to the total number of function points of one business application or one domain. The total functional scope p_d for one domain d is defined as the sum of all function points p_i per application i that belong to the domain. Within the following equation, n describes the total number of applications per domain while I_d describes this set.

$$p_d = \sum_{i=1}^{n} p_i \mid i \in I_d$$

With these definitions at hand, the following questions are posed targeting the simulation of data change impact on application landscapes:

1. How is the complexity of an application landscape affected when a new application is added?
    a. How is the database heterogeneity of a domain affected, when a new application is added to the domain?
2. How is the complexity of an application landscape affected when the attributes of applications change?
    a. What is the impact of changing an application's function point measure by one point in regard to the total function scope?

The more tangible questions below each generic question will be answered with the help of the prototypical implementation presented herein. For this purpose the four datasets mentioned above contain data about application landscapes spanning from 234 to 1898 applications and respectively 539 to 8252 information flows between the applications. In contrast to the hospitality management example, one cannot expect such large quantities of data, thus the requirements in respect to performance and processing times are already covered.

1.3 Research Methodology

As illustrated with the examples above the key question of this thesis is the search for an appropriate concept to visualize the impact of data changes to enterprise metrics. An appropriate concept herein means a theoretically profound idea along with a technical and visualization concept. In the context of a visualization concept another question arises: How can interaction on this visualization help to improve the user's understanding of the calculation system? This question targets the user's ability to understand the impact of changes to the underlying data. Furthermore it aims at the creation and comparison of possible scenarios.

1.3.1 Design Science

Following the *Design Science in Information Systems Research* methodology by Hevner et al. (2004), the seven guidelines (p. 82-90) are applied. The main result of this thesis is a prototypical and documented implementation comprising of source code, configurations, models and compilations. These artefacts will be applied to solve the questions posed in the example sections above. With its visualization and interaction components it enables the user to improve his/her understanding of the enterprise metrics. In this, the proposed solution differs from current solutions of the problem. To demonstrate the functionality, applicability and purpose of the design artefacts automated unit tests along with an architectural discussion and typical use cases are presented. Especially the visualization technique and the interaction of the user with the visualization artefacts are the main contribution of this thesis to the field of study. In addition, various application programming interfaces (APIs) define the interaction between the modules of the architecture. All interface designs aim at simplicity, yet effectiveness and efficiency. To develop the architecture design patterns are studied and applied.

Furthermore the design is refined and extended during the development process. In particular using different data sources such as relational databases, file-based data stores and semi-structured data stores will provide an evidence for the stability of the proposed design. Finally the document in hand presents the research results.

1.3.2 Systematic Literature Review

On the basis of the questions stated at the beginning of this section, a review protocol was created. A systematic literature review (Kitchenham, Charters 2007) serves as a profound basis for the Design Science approach. Furthermore it reveals the existing solutions and approaches and it verifies that the proposed solution varies from existing approaches and applications.

In online search catalogues such as SpringerLink[1], ScienceDirect[2] and the ACM Digital Library[3] as well as on Google Scholar and the online search of the library of the Technische Universität München (TUM), the search terms "information visualization", "impact visualization" and "change visualization" were observed. Criteria for publications to be considered in the search are that the source must be either a published paper or book, or an online article of a major and recognized website such as ZDnet[4] with an author and publishing date. Community based articles, blog entries and advertisements are examples for sources excluded from the considerations due to the limited ability to verify a non-biased view and interests contrary to this research. However, for a market research application presentations (which could be considered advertisements) are regarded.

1.4 Outline

In order to apply the stated research methodology, the document in hand is structured as follows. At first, the business, functional and non-functional requirements are described. Functional requirements are further subdivided into user, data and technology requirements to illustrate the different perspectives on the topic. In the next chapter, findings from the systematic literature review are presented. The evaluation of available products and their fulfillment of the stated requirements complete the view of current approaches and practical solutions.

In the second part of this thesis, a new solution approach based on the afore-mentioned requirements is described. First, the technology selection is explained followed by a discussion on the architectural design and its evolution during the development and integration process. A user-guided view through the application and the implementation allow the reader to assess the details. Based on the examples and the solution to their questions, typical modelling patterns are explained to assist the user employing the application.

A conclusion with the findings from the literature and market research as well as the learnings from the implementation opens the discussion for future work, thus putting the work in perspective. In the appendixes the requirements fulfillment of one tool, a compatibility matrix of the components and performance measurements are attached.

[1] Springer, Part of Springer Science+Business Media; http://link.springer.com/, checked on 7/24/2014.
[2] ScienceDirect® by Elsevier B.V.; http://www.sciencedirect.com/, checked on 7/24/2014.
[3] ACM Digital Library by ACM Inc.; http://dl.acm.org/, checked on 7/24/2014.
[4] CBS Interactive; http://www.zdnet.com/, checked on 7/24/2014.

2 Requirements

As depicted in the chapter Motivation and Problem Statement, the simulation of metrics can be performed utilizing a software tool. In order to evaluate the feasibility of such a tool, the business, functional and non-functional requirements are outlined in advance.

2.1 Business Requirements

A short definition of business requirements is "What must be delivered to provide value" (Goldsmith 2004, pp. 2–3). According to the author this statement contains three major terms: *What* describes the characteristics of the product, *delivered* stands for the outcome and *value* "is achieved by meeting an objective", however, "value can also be in terms of intangibles" (Goldsmith 2004, p. 3). An extension of this definition is the various perspectives the stakeholders of a project bring to the table. "The business requirements describe the primary benefits that the new system will provide to its sponsors, buyers, and users." (Wiegers 2003, p. 83). Furthermore it is necessary to define the term sponsors/stakeholders as "individuals, groups, or organizations who are actively involved in a project, are affected by its outcome, or are able to influence its outcome." (Wiegers 2003, p. 88).

In the case of Example 1: Hospitality Management possible stakeholders for the tool in question are the hotel management, leading employees in the accounting department and consultants. In contrast, enterprise architects and management staff controlling enterprise architecture are potentials users as illustrated in the context of Example 2: Metrics in Enterprise Architecture Management.

Numerous authors describe the necessity to understand the essence (Robertson, Robertson 2006) or the business objectives (Wiegers 2003; Chemuturi 2013) in order to build the product needed from the business. Morato (2013) describes common business objectives in chapter 2:

1. Attainment of financial objectives such as increasing profits or net income in relation to sales, assets, investments or equity
2. Expanding market presence by increasing sales volume, pushing sales growth, enlarging market share and deepening market penetration
3. Satisfying customers with quality products and services, delivering the right quantities at the right time, and at acceptable prices
4. Attending to the needs of various stakeholders

Although these objectives are on a very high-level perspective, it is necessary to consider them building a product dealing with business metrics which measure the attainability of these goals. In addition, the environment in which a business operates is constantly changing as the financial crisis in Europe recently illustrated. The analysis of different scenarios for the future can help to make profound managerial decisions becoming especially valuable in changing environments. In the context of EA Management the business goals are closely related to the complexity of an enterprise architecture. Therefore the changing environment is the application landscape as a result of changing requirements in economics and from innovations. The following table presents a summary of the business requirements (BR) elicitation.

Table 2: Business Requirements

No.	Title	Short Description	Derived from Source
BR-01	Improve change management	A better predictability of changes can reduce costs and improve customer satisfaction by decreasing the adaption obstacles during a change process.	Currie, Hlupic (2000)
BR-02	Support decision making	Complex models and calculations challenge consultants and managers familiar with decision making on a daily basis. By increasing the understanding of the models, the decisions are likely to result in a better performing organization.	Morato (2013)
BR-03	Compare scenarios	Comparing various scenarios helps to identify their differences and the impact of influencing factors. Therefore either an optimization or adapted plans for each scenario lead to a better preparation and consequently to less issues during the change process.	Currie, Hlupic (2000)

2.2 Functional Requirements

According to Wiegers (2003, p. 10) *functional requirements* specify the software functionality that the developers must build into the product to enable users to accomplish their tasks, thereby satisfying the business requirements. They describe what the developer needs to implement. Another definition targeting the objectives of the user is: "A functional requirement is an action that the product must take if it is to be useful to its users. Functional requirements arise from the work that your stakeholders need to do." (Robertson, Robertson 2006, p. 9)

In the section on business requirements the stakeholders and their objectives have been described. On the basis of these objectives we can describe the stakeholders' actions and tasks to achieve their goals. Later the identified requirements are used to evaluate existing tools (see the chapter 4) and to develop a new tool. The requirements are further categorized in user, data and technology requirements to improve their readability and traceability.

2.2.1 User Requirements

In the context of this document user requirements (UR) are the product's functions a user needs to fulfill his/her tasks and actions. Since a product is used in an environment, it should be described at first. As management and financial staff are considered main users of the product, one observes that they spend their majority of time on a desk (British Psychological Society (BPS) 2012). Hence the tools they work with are desktop and notebook computers as well as books, printouts and other utilities around the desk. As a recent study (SoftWatch 2014) shows, one of the main productivity applications, namely *Microsoft® Office*, is excessively used in financial, law and marketing departments, but in general the usage is low (only 48 minutes a day in average). Despite this observation the need for productivity applications in business seems to be unabated as 27 million downloads of the tablet version of *Microsoft® Office 365™* shows (Crothers 2014). However, due to performance constraints and screen size limitations the mobile products are reduced in features and so far not suitable for large amounts of data. Therefore software targeted at notebook and desktop computers with their comparably large screens is to favor over their mobile competitors (cf. Gruman 2013, Bott 2014).

After the usage environment is outlined, the actual interaction of the user and the application needs to be addressed. Business metrics are calculated based on data retrieved from and stored in databases. Their metadata (data structure) is readable and accessible. Typical calculation operations, as shown in Example 1: Hospitality Management, include the addition, subtraction, multiplication and division of values as well as aggregation functions such as the sum of multiple values, their product, the minimum and maximum or the number of values (count). In return, calculation results can be part of other calculations. The user must have the possibility to enter these calculations, either in a visual or declarative way. Inputs should be specifiable with high precision and they must support large numbers (e.g. billions). Different input data types must be accepted, for example, numbers, dates, enumerations and strings. All references to data fields must be usable as variables when modelling the system.

When visually modelling the calculations, it must be possible to reorder the factors of an equation. Furthermore the user must be able to display the equation for review. Mathematical rules for computing such as multiplication and division first, then addition and subtraction must be followed. In addition, the system has to allow the user to alternate each factor of an equation absolutely and relatively. By alternating a factor of an equation the user is empowered to simulate changes to influencing factors. Setting an absolute simulated value results in a fictive replacement of the original value for the purpose of a simulation. Similarly a relative simulation value adapts the original value by a factor, for example, setting the relative simulation value to 0.05 or 5% the original value is multiplied by 1.05. The simulation values will neither affect the underlying data, nor will they affect the calculation of the values in dependent equations, however, they will affect all simulated values based on this influencing factor and equation.

Moreover it must be possible to save the model and its changes either locally or on a remote system. Different models on the same data basis can be created. The user has to be able to start the simulation of a model and display the results afterwards. Once the simulation has completed successfully, the results are displayed to the user. This includes the results based on the original values and the results of the alternated values by an absolute or relative simulation value. One principal goal is to present the user the impact of his/her simulated changes. Therefore the impact of the deviations must be displayed visually. As an optional feature, the comparison of different deviated values can be included in the product. In this case, a basic case of scenario comparison is supported.

In addition to the above-mentioned deviation of factors in an equation, the fictive manipulation of the underlying data enables the user to simulate the impact of additional entries in the data set. To support this feature, the user must be able to insert and remove virtual data for each underlying type and field. The virtual data affects the simulation result, but it does neither affect the production data, nor the calculation of the result which is based on the original values.

All requirements from the previous paragraphs are summarized in the following table:

Table 3: User Requirements

No.	Title	Short Description	Derived from Source
UR-01	Desktop Environment	Users interact with the application from their desktop or notebook computer, not from a mobile device.	SoftWatch (2014)
UR-02	Support for various screens	Arising from the requirement above is the need to support multiple screen sizes of notebook and desktop monitors.	SoftWatch (2014) UR-01
UR-03	Model calculations	Equations can be modelled graphically or entered in another graphical form.	Market Research, Nazemi et al. (2011)

UR-04	Calculation operations	Supported operations are at least: addition, subtraction, multiplication and division.	Examples: sections 1.1, 1.2
UR-05	Aggregations	Aggregation functions such as the sum and product, a count of the entries in a data set and the computation of the minimum and maximum of a numerical data set are supported.	Examples: sections 1.1, 1.2
UR-06	Input data type support	Simulated values and constants are of a numerical type. Data selection constraints can apply a number, date, enumeration or string typed value.	Kerzner (2013)
UR-07	Input precision	Results and simulated values must be displayed with a high precision. The input of simulated values must be supported with high precision as well.	Microsoft Corporation (2005), SAP SE
UR-08	Large number input	Numbers in the billions with a decimal fraction are common when calculating business metrics. Therefore the support for large numbers is required.	Microsoft Corporation (2005), SAP SE
UR-09	Metric dependencies	The result of a calculation can be comprised in another calculation; in other words: Metrics can be dependent on other metrics.	Examples: sections 1.1, 1.2
UR-10	Reorder factors in equation	The order of factors in an equation is essential for the computation of the result. Therefore the model supports to save the order of factors. Additionally, factors can be reordered when required.	Usability matter, cf. NFR-08 and section 2.3.1
UR-11	Display equations	When modelling equations visually, the equations are not visible as such. Therefore a simply view of the equation must be possible, e.g. a = b + c.	Spence (2007), cf. section 4.2
UR-12	Follow mathematical rules	The computation of an equation's result must adhere to mathematical rules, e.g. 1+2*3 = 7, not 9.	cf. NFR-09
UR-13	Set absolute factor deviation	To simulate the replacement of all values in a set with a constant, an absolute value can be entered as the simulated value.	Examples: sections 1.1, 1.2
UR-14	Set relative factor deviation	To simulate the deviation of all values in a set by a factor, e.g. by 5%, the relative simulated value (in the example 5% or 0.05) can be entered.	Examples: sections 1.1, 1.2
UR-15	Deviation factors do not affect production data	By setting and saving a simulated value, the deviations do not affect the original production data.	cf. NFR-06
UR-16	Simulated values affect dependent values and equations	By setting and saving a simulated value, the computation of the simulation results of dependent metrics regards these simulated values.	Recursive dependencies, cf. UR-08
UR-17	Save a model	The user can save a model and close the application or even shut down his/her computer and he/she can later reload the model and continue to work with it.	Persistency, Power-failure prevention (cf. NFR-01)
UR-18	Create multiple models	To support different algorithms to calculate a metric the creation of multiple models must be supported.	cf. BR-03, Currie, Hlupic (2000)

UR-19	User can start simulation	A simulation run can be started by the user and is not required to be started by an administrator or scheduled task on a server, for example.	Interaction requires this feature
UR-20	Display the simulation results	The results of the calculation as well as the results of the simulation must be visible for the user.	Hidden or restricted results defy interaction
UR-21	Display impact of deviations	The difference between the calculation result and the simulation result is the impact of the deviations. This impact needs to be displayed that the user can understand where the deviations origin is.	Core benefit and contribution of this thesis
UR-22	Scenario comparison	As an optional requirement, the scenario comparison allows to run simulations on the same model with different simulated values. Each run and its results can be saved either temporarily or persistently and compared afterwards.	cf. BR-03
UR-23	Virtual data manipulation	Entries can be added to a dataset virtually to be regarded in the simulation runs. This also includes the removal of virtual data.	EA Management example (section 1.2)
UR-24	Virtual data manipulation does not affect production data	Virtual data is stored separately from production data in order not to affect the operations of other applications.	cf. UR-15
UR-25	Virtual data manipulation does not affect the calculation of results	If virtual data is present, the calculation of the results on the original values is not affected.	cf. UR-21
UR-26	Virtual data manipulation affects simulation results	If virtual data is present, the calculation of the simulation results is affected.	cf. UR-16

2.2.2 Data Requirements

In contrast to user requirements, data requirements (DR) focus on the metadata and data quality as well as on the access to the data of the underlying production systems.

Since data sources of production systems can change over time and because of the generic modelling approach, the tool is supposed to provide a concept to support different data sources. However, multiple data sources do not need to be usable at once and in one model.

In order for the product to support the user as best as possible when modelling the metrics' calculations, the data source needs to provide appropriate metadata. For example, if a relational database management system (RDBMS) is connected, the tables and fields must be readable from the application as well as their relations, practically their references. The metadata can then be used to reduce the input necessary from the user to define constraints and values. By supporting different data providers an eligible abstraction must be found to increase the extensibility of the system for future data sources.

Table 4: Data Requirements

No.	Title	Short Description	Derived from Source
DR-	Support for different	The product must be able to handle	Examples: sections

01	data sources	different data sources, however, not simultaneously.	1.1 (e.g. RDMBS), 1.2 (e.g. T39, CSV)
DR-02	Extensibility for further data sources	An extension mechanism must be provided to support future data sources.	cf. section 10.1; technology change (Brock, Moore 2006)
DR-03	User support through metadata	To support a user best, metadata should be used to avoid unnecessary input and reduce sources of error.	cf. NFR-08

2.2.3 Technology Requirements

Since business applications are usually embedded in a landscape of other applications, the technology requirements (TR) target the environment of the software and its interoperability.

In a connected world many applications do not reside on local machines anymore. They are hosted in data centers and accessed via a network or the Internet. Since the application in question cannot expect to access all data locally, the support for network connectivity is required.

In order to reduce the dependency on a single software provider and to increase the interoperability, standards should be used where applicable and economically reasonable. For example, connecting to a database via a proprietary driver should be avoided in favour of a Java Database Connector (JDBC) or an Open Database Connectivity (ODBC). Another example is the selection of exchange formats. Formats like JavaScript Object Notation (JSON) and eXtensible Markup Language (XML) as well as Comma-Separated Values (CSV) are preferred over proprietary formats, because they are widely used, tested and encoding as well as decoding implementations exist for common programming languages.

Table 5: Technical Requirements

No.	Title	Short Description	Derived from Source
TR-01	Network connectivity	It is possible to connect network resources as data sources.	EA Management example (section 1.2), cf. DR-01
TR-02	Application of standards	Where economically feasible and applicable, standards should be used.	Standards contribute to the extensibility (DR-02); cf. NFR-09 to NFR-11

2.3 Non-Functional Requirements (NFR)

"Nonfunctional requirements are properties, or qualities, that the product must have. In some cases, nonfunctional requirements – these describe such properties as look and feel, usability, security, and legal restrictions – are critical to the product's success" (Robertson et al. 2008, p. 10). Another definition for non-functional requirements applying the term quality attributes is that "*Quality attributes* augment the description of the product's functionality by describing the product's characteristics in various dimensions that are important either to users or to developers. These characteristics include usability, portability, integrity, efficiency, and robustness." (Wiegers 2003, p. 11) Among the non-functional requirements as identified by Wiegers (2003, p. 178-180) are performance, safety (especially for controlling units) and security (integrity, privacy) requirements as well as other software quality attributes like availability, efficiency, flexibility, integrity, interoperability, reliability, robustness, usability, maintainability, portability, reusability and testability (p. 217). In contrast to the common classification into functional and non-functional requirements, Chemuturi (2013) describes core and ancillary functionality (pp. 13-17). "When we say 'non-functional' it has the connotation that the requirements may not serve business process functions

directly but they are serving a useful purpose in the software, They indirectly, perhaps, assist in the better functioning of business processes." (p. 14) He categorizes the ancillary functionalities in statutory, safety, security, usability, data integrity protection, response time, memory constraints, software footprint, fault tolerance, reliability, feel-good (e.g. look and feel), esteem (user's pride) and one-upmanship (more functionality than the competitors) functionality (pp. 15-17).

Combining the numerous non-functional requirement categories and selecting the relevant ones for the tool in question results in a comprehensive perspective of the quality requirements. The ancillary safety, memory constraint and software footprint functionalities seem to refer to embedded systems and thus are not regarded in the overview. The security matter is split into integrity and reliability requirements and the feel-good, esteem and one-upmanship requirements are condensed in the usability and flexibility categories. Robustness and fault tolerance as well as usability and response time are group together as well.

Table 6: Non-Functional Requirements

No.	Title	Short Description	Derived from Source
NFR-01	Availability	Access to the tool on their desktop and notebook computers must be available to all users authorized by their manager.	SoftWatch (2014)
NFR-02	Efficiency	Understanding the calculation system is improved drastically when loading and response times are kept so short that a direct interaction with the results is possible. However, depending on the operation environment it is very difficult to set a limit[5] for response or loading times. Finally the user will have to decide whether the response time is appropriate to fulfill his/her needs.	Nielsen (2010)
NFR-03	Flexibility	The connectivity support increases the flexibility to operate and use the application. From a user perspective having the possibility to model calculations generically (cf. UR-03 to UR-10) and in different models increases the flexibility of the system even further.	TR-02, DR-01, DR-02 and UR-02
NFR-04	Integrity	With a potential preview of data, as well as the opportunity to constrain data selections in a matter that single datasets can be identified, an access limitation is inevitable. Only authorized users are allowed to access the system.	Wiegers (2003), Chemuturi (2013)
NFR-05	Interoperability	On the basis of TR-01 and TR-02, the interoperability between systems should be possible. This requirement strongly depends on the tangible setup and concrete business case, thus it can be regarded as a supplementary requirement.	Wiegers (2003), Chemuturi (2013)
NFR-06	Reliability	When executing the same calculation, on the same model and data basis repeatedly, the results are always the same and thus predictable.	Wiegers (2003), Chemuturi (2013)
NFR-07	Robustness	A function to validate the model is required from the tool. The user can execute the validation to correct his/her mistakes made by modelling the calculations.	Wiegers (2003), Chemuturi (2013)

[5] As Nielsen (2010) suggests website response times of under one second, but at least under ten seconds before they think of something else. In addition, Kim/Stoel (2004) have shown that the user experience of a website is dependent on more dimensions, thus the response time should not gain too much attention.

NFR-08	Usability	Since usability is difficult to measure, one of the basic and comprehensible requirements for the tool is to visualize the calculations on a graphical user interface (GUI). Furthermore the metrics and their calculation must be in the center of the attention for the user.	cf. section 2.3.1
NFR-09	Maintainability	In case the tool is developed individually, the maintainability of the solution is of major importance. Therefore the source code must contain sufficient documentation, i.e. at least one sentence per module/class about the purpose and content of it. A modular architecture and formatted code will further increase the maintainability of the tool.	Wiegers (2003), Chemuturi (2013)
NFR-10	Portability	By supporting standards (cf. TR-02) and the use of different data sources (cf. DR-01) and well as the ability to run on several screen sizes (cf. UR-02) a basic portability is guaranteed. Leveraging the possibilities of a web application can further increase the portability for the end user. Adhering to standards like the Java EE will also enhance the portability of the application. This requirement is not mandatory.	Wiegers (2003), Chemuturi (2013)
NFR-11	Reusability	If the solution is comprised of components with documented interfaces, single components can be reused in the future. This is not a strict requirement, but it can prolong the application lifecycle and thus increase the value of the tool.	Wiegers (2003), Chemuturi (2013)
NFR-12	Testability	Test cases to verify NFR-06 are required as well as test cases to show the correct evaluation of the calculations. Every operator, aggregator and each type of input must be tested and documented at least once.	Wiegers (2003), Chemuturi (2013)

2.3.1 Usability

As recent research shows (cf. Fernandez et al. 2011; Letondal et al.; Marcus 2013), the specification and evaluation of *Usability* is a complex task. The ISO standard 13407: *Human centred design processes for interactive systems* was critically discussed and is not sufficient to specify tangible criteria, however, it provides a process to evaluate the usability of a system (cf. Jokela et al. 2003; Earthy et al. 2001). Due to time and resource restrictions, *User Testing* (Fernandez et al. 2011) is not part of this work. For the purpose of requirements engineering, the criteria specified in the table above are the guideline to evaluate the prototypical implementation.

2.4 Limitations

Besides the afore-mentioned requirements, the application in question does not need to fulfill any of the following features. The metric calculations do not include any cyclic dependencies, for example, metric *a* depends on metric *b*, which in turn depends on metric *a*. All value types are simple value types, not binary or complex types such as images or binary objects. Custom functions and aggregations are optional, however, all basic operations like addition, subtraction, multiplication and division (cf. UR-04) must be supported as well as the aggregators sum, product and count (cf. UR-05).

3 Literature Research

The previous chapter outlined the requirements for a product that solves the problem stated in the first chapter. In this chapter the results of the literature research is presented.

3.1 Key Performance Indicator Visualization

In this section the question how key performance indicators can be visualized is discussed. By its definition, a key performance indicator (KPI) is calculated based on quantitative data (cf. Fitz-Gibbon; 1990). The KPI is expressed as a figure of a numeric data type such as float or integer. The influencing variables to calculate the KPI thus need to be of a numeric format, too. In case the parameters are of another format like a character sequence or enumeration, aggregations such as counting values or mappings can be employed to transform the information to numeric values.

One visualization technique for KPIs is to present multiple KPIs at once on a dashboard. Each KPI is displayed using a graphical component. The five most common components are Alert Icons, Traffic Light Icons, Trend Icons, Progress Bars and Gauges. (cf. Kerzner 2013, pp. 6.6/3-5) To provide further context to rather simple dashboard components, *supporting analytics* can be displayed. „Often these supporting analytics take the form of more traditional data visualization representations such as charts, graphs, tables and, with more advanced data visualization packages, animated what-if or predictive analysis scenarios." (Kerzner 2013, pp. 6.6/3) In the context of the problem at hand, a graphical and interactive what-if analysis is subject to further discussion.

According to Nazemi et al. (2011, p. 578) "[…] two-dimensional graph-based layouts […] are the most adequate visualization techniques for the standard user: He can easily interact with them and most accurately interpret the displayed data." Based on the taxonomy they developed, one of the *classical*, *radial* or the *node+link* graph layouts are appropriate for the given use case. A reason for that is that mathematical relations in the form of y = f(x) "could be represented by a node and directed link diagram." (Spence 2007, p. 71) In case the concrete calculation of a metric system has no cyclic dependencies, a tree or hyperbolic representation is possible. In contrast, cyclic dependencies of metrics can be presented using a node and link representation with directed edges (cf. Spence 2007, pp. 71-91).

The results presented in this section help to assess the modelling of a metric system and its calculation in a graphical form. Based on the findings, the next question is how to visualize changes of influencing factors in graphs.

3.2 Impact Visualization of Influencing Factor Changes

In Hao et al. (2006) the impact of influencing factors on processes is evaluated and visualized. "The goal was to provide an appropriate visual layout, that is able to represent an abstraction of the underlying complex workflow, but at the same time shows the relationships between discovered relevant business impact factors for further visual analysis." (Hao et al. 2006, p. 16) With the categorization developed by Nazemi et al. (2011) and Spence (2007) they visualize the influencing factors and their relations with a hyperbolic graph layout using various visual parameters such as position, length, size and color. Different colors and different line weights show the impact of each factor. To display the delta of the original value versus the changed value, the size of a node can be employed like Spence (2007) illustrates in an example where the population of cities are displayed on a map: "By its use of circle size to encode each cities population, we can quickly gain an impression of how population is distributed." (Spence 2007, p. 46) Transferring these findings to represent the

impact of a single factor on the metric system, one concludes that the numerical deviation can be represented in the size of the shape a node is represented with and its sign, so whether the impact is positive, negative or null, can be represented by the color. In combination with the weight of a link between nodes, the impact of an influencing factor can be followed along the line. One example of this technique is the representation of the US federal budget 2013 using the graphic library D3.js as shown in Illustration 3. "D3.js is a JavaScript library for manipulating documents based on data." (Bostock 2014)

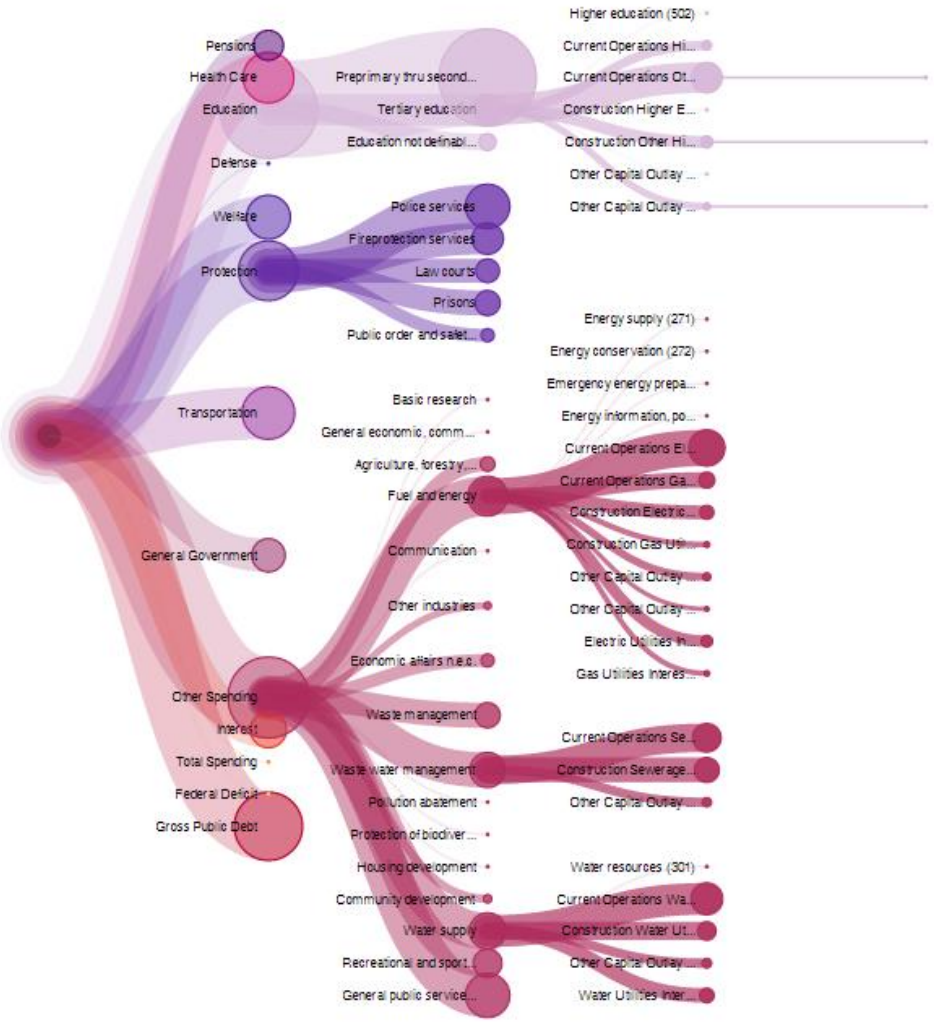

Illustration 3: US Local Budget 2013 (BrightPoint Consulting Inc. 2014)

Requirement UR-22 describes the system's ability to display different scenarios based on the changed including factors and their impact. Spence (2007) illustrates the change of a city's population over time with changing circle sizes depending on the point in time that is selected on bar containing all years that measurements are available (cf. p. 51). Conveyed to the comparison of different scenarios in a metric system, each scenario could be one measurement. By changing the measurement selection, the graph adapts to the impact of the selected measurement. This way the user can visually compare different scenarios in the model. The animation of the scenarios is also suggested in literature and it can be observed in many practical examples (Russell, Carter 2009).

3.3 Impact Calculation

In addition to the visualization of KPIs and changes of influencing factors, the calculation of the metric system needs to be observed as well. Beginning with the calculations the user models before they are performed, the internal representation of the model is crucial. One common way to represent composite structures is described by Gamma et al. (1995) as the *Composite Pattern*.

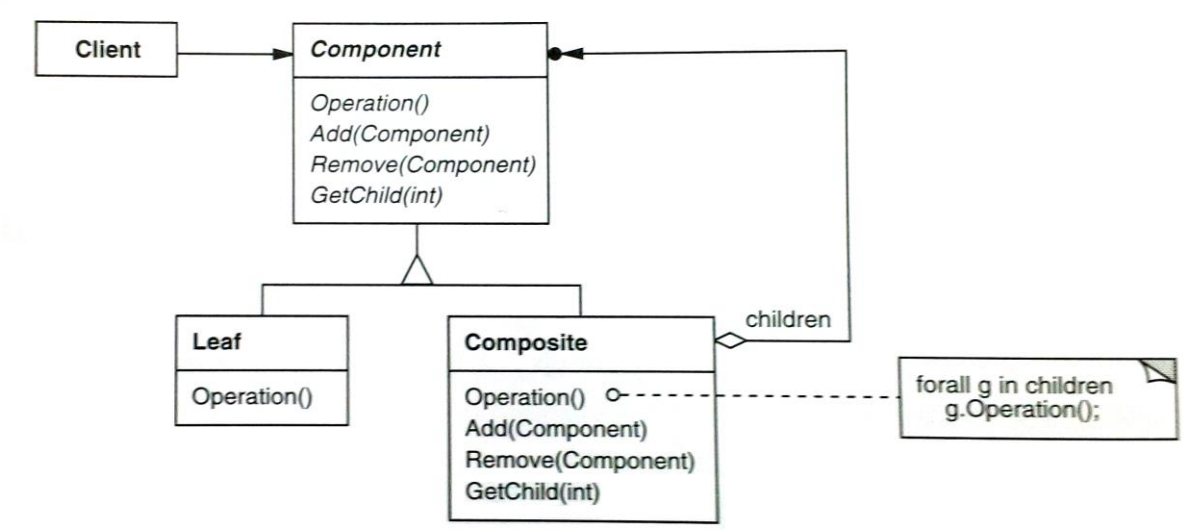

Illustration 4: Composite Pattern by Gamma et al. (1995, p. 164)

A *Component* interface or abstract class encapsulates the internal structure. *Composites* represent the inner nodes and *Leafs* the nodes without children. This structure is particularly suited to represent tree-like structures. Equations can be represented by such a structure, for example, the equation $a = b + c$ and $c = d + e$ could be illustrated as a graph (cf. Spence 2007, p. 71):

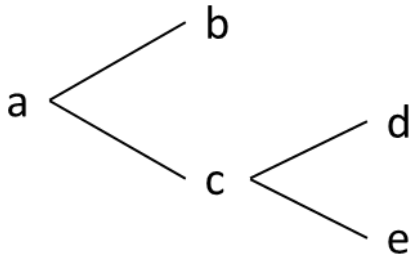

Illustration 5: Example for the Visualization of an Equation (own diagram)

In this case one possible object diagram for this structure applying the Composite pattern is displayed in the following illustration:

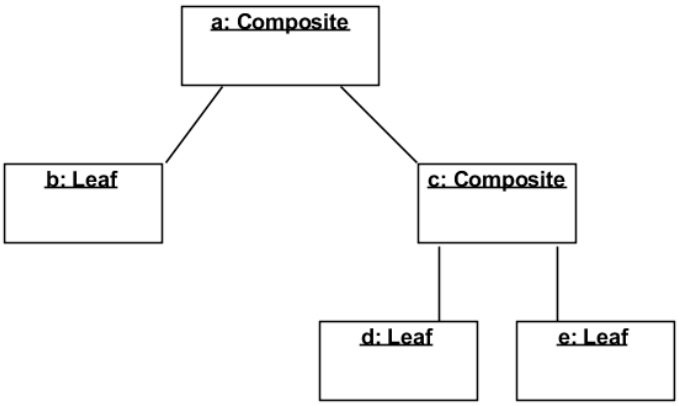

Illustration 6: UML Object Diagram of Example with Composite Pattern (own diagram)

To calculate the value of the equations' components one will have to iterate over the structure. An abstraction from the iteration algorithm is the *Iterator* pattern as described by Gamma et al. (1995, pp. 257–271). Such an abstraction can be helpful "to support multiple traversals of aggregate objects" (p. 259). In conjunction with the Composite pattern it serves as a solid basis for the calculation process.

Based on the depth-first search (DFS) algorithm (cf. Cormen 2009, p. 603) one can evaluate all components of an equation recursively and return the result if all *Leafs* (cf. Composite pattern) were reached. Theoretically a breadth-first search (BFS) (cf. Cormen 2009, pp. 594–600) could also be applied, however, the evaluation of a calculation would require to execute the algorithm twice – once to reach all *Leafs* to compute their value, then once again to calculate all inner nodes. Based on this finding, the DFS algorithm is preferred. In the example developed throughout this section, this means that the value of *d* and *e* are computed first. Once their computation is completed, their values can be used to compute *c*. In addition with the value *b*, the value of *a* or in other words the result of this metric system can be calculated.

4 Market Research

After the requirements elicitation and the literature research phases all information was gathered to evaluate existing products on the market. “Include a comparative evaluation of existing products and potential solutions, indicating why the proposed product is attractive and the advantages it provides” suggests Wiegers (2003, p. 83). An online search using the Google search engine revealed several products that could fulfill the requirements partly or completely. In this chapter, the tools are evaluated shortly for their common suitability for the task in question.

4.1 Simulation Software

The search term “KPI simulation” reveals two interesting software tools: Siml8[6] by Simul8 Corporation and an educational *Balanced Scorecard Simulation*[7] tool from the KPI Institute. The KPI Institute’s product is applied for educational purposes in the same-named course and does not seem to be available to the public. Furthermore the limited functionalities (displayed by a screenshot on the website) are not likely to fulfill the requirements stated in chapter 2.

Searching for other terms like “simulation software” or “process simulation software” reveals many more products such as Simio[8], SimuLink/MathWorks[9] or OptQuest[10] which are similar to Simul8. Another search for “open source simulation software” revealed tools like OpenSim[11] and OpenModelica[12]. All products target either process modelling for industrial and/or service sector applications or they are meant to model physical problems involving forces, friction, temperature, et cetera. Often a programming language is offered to model complex problems. The visualization of KPIs is, for example in case of Simul8, mostly used to present metrics on calculations such as the runtime, the resources used and statistical figures. When trying to model the problem of the first example (Hospitality Management) with the open source tools, the author was confronted with source code editors, terminals and complex runtime environments. Given the potential users of the evaluated systems are physicists, mathematicians, engineers and other domain experts, this seems an appropriate approach. However, considering the user group of economists, accountants and management staff, it is unlikely the tools will satisfy their needs and usability expectations (cf. BR-01 to BR-03). In the opinion of the author, all of the mentioned tools are not suitable to model KPIs as described in the requirements in chapter 2.

4.2 Microsoft® Excel

Most readers familiar with KPIs will ask themselves: Why not use *Microsoft® Excel*? According to the documentation (Microsoft Corporation 2014c) it supports multiple data sources including SQL-based relational database management systems, text files, web resources and application connectors. With the power of the integrated programming language Visual Basic (VBA) *Excel* becomes a platform for application development and data analysis. Furthermore *Excel* offers simple ways to filter data (only 2-3 steps (Microsoft Corporation 2014b)), it restricts access to data (Microsoft Corporation 2014d) and it is the de-facto standard for spreadsheets and thus present in most enterprises (Pelkmann 2010).

[6] Website: http://www.simul8.com/, checked on 7/24/2014.
[7] Website: http://kpiinstitute.org/education/simulations/balanced-scorecard-simulation/, checked on 7/24/2014.
[8] Website: http://www.simio.com/, checked on 7/24/2014.
[9] Website: http://www.mathworks.de/discovery/simulation-software.html, checked on 7/24/2014.
[10] Website: http://www.opttek.com/OptQuest, checked on 7/24/2014.
[11] Project website: http://opensimulator.org/wiki/Main_Page, checked on 7/24/2014.
[12] Project website: https://www.openmodelica.org, checked on 7/24/2014.

One of the core requirements outlined in chapter 2.2 is the graphical representation of a change's impact (UR-20/21). Another major requirement is the support for virtual data (UR-23 to UR-26). The latter could be achieved by creating an additional worksheet containing the virtual data. Then all calculations must factor in the virtual data. In addition, all formulas are hidden from the user by default. He or she needs to click on a cell to view the formula representing the tangible calculation. Since formulas can become complex and long – especially in regard to virtual data presence – a single line display is not very convenient to get a better understanding of its working as illustrated below.

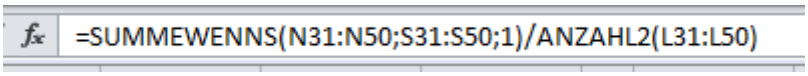

Illustration 7: Excel Formula

Besides the functional limitations of *Excel*, the non-functional requirements Maintainability (NFR-09) and Reusability (NFR-11) are challenged by the author. As discussed by practitioners (Altstaedt 2014) *Excel* spreadsheets are quickly distributed to many employees and often used as a template for other calculations. Refactoring the spreadsheets becomes a challenge, especially since semantic merges of two versions need to be manually applied. This also limits the Reusability. Once a spreadsheet is prepared for specific data, it must be copied in order to coexist. Afterwards both versions of the file need to receive updates when the base sheet/calculations will be changed.

4.3 Powersim Studio 10

One of the tools that were revealed by the online search mentioned in the beginning of this chapter deserves a closer look: Powersim Studio 10[13] by Powersim Software AS. It was evaluated based on the 64-bit Demo Version provided on the website of the manufacturer. A video introduction (Powersim Software AS 2012) ensured the correct application of the product for the given use cases and requirements and it helped to get started with the software after the installation. Once the application had started and the introduction guide was finished, a blank canvas in the center of the screen surrounded by many buttons appeared. The number of buttons can be reduced by turning the *Advanced Mode* off. As displayed in Illustration 8, the author created a simple example with variables and visualizations. Especially the sliders, graphical elements to change variables quickly, were perceived as convenient widgets. Changing a slider's value immediately resulted in an update of all values in the model and their visualization, in this case the gauge.

In the model simple constants and variables were created and used. To verify the use of this application for structured data the author tried to integrate another data source – an *Excel* spreadsheet. All sliders were replaced by applying an aggregation on the data behind each variable. For example, the variable "Number Guests" was replaced with a sum of all guests in the column "Number Guests". Now with data in the background, the values were not updated immediately anymore. The simulation had to be started using the play-symbol on the top-left of the screen. Consequently a change in the data also resulted in an update of the model's values and visualizations.

To verify the UR-20/21 requirements a combination of both approaches was evaluated. Adding a slider to the aggregated values only showed the result on the slider, but it was not changeable. By creating another variable to change the value of the aggregation, the original value is obviously changed as well (defying UR-15). Therefore the deviation could not be modelled within one element. A second graph besides the original one would enable graphical and numerical comparison; however, it would also lead to significant double work, the risk of asynchronous model updates and a lack of maintainability. In the context of this application the simulation changes the values, but it does not

[13] Product website: http://www.powersim.com/, checked on 7/24/2014.

compare the results or display deviations (UR-20/21). A problem linked to this matter is the simulation based on virtual data (UR-23 to UR-26). The application offers a tool to add structured data to the model (*Table Control*), but similar to modelling deviations in values, the deviations of data affect the entire model. Therefore a comparison of a scenario without the consideration of virtual data and with it is not easily possible.

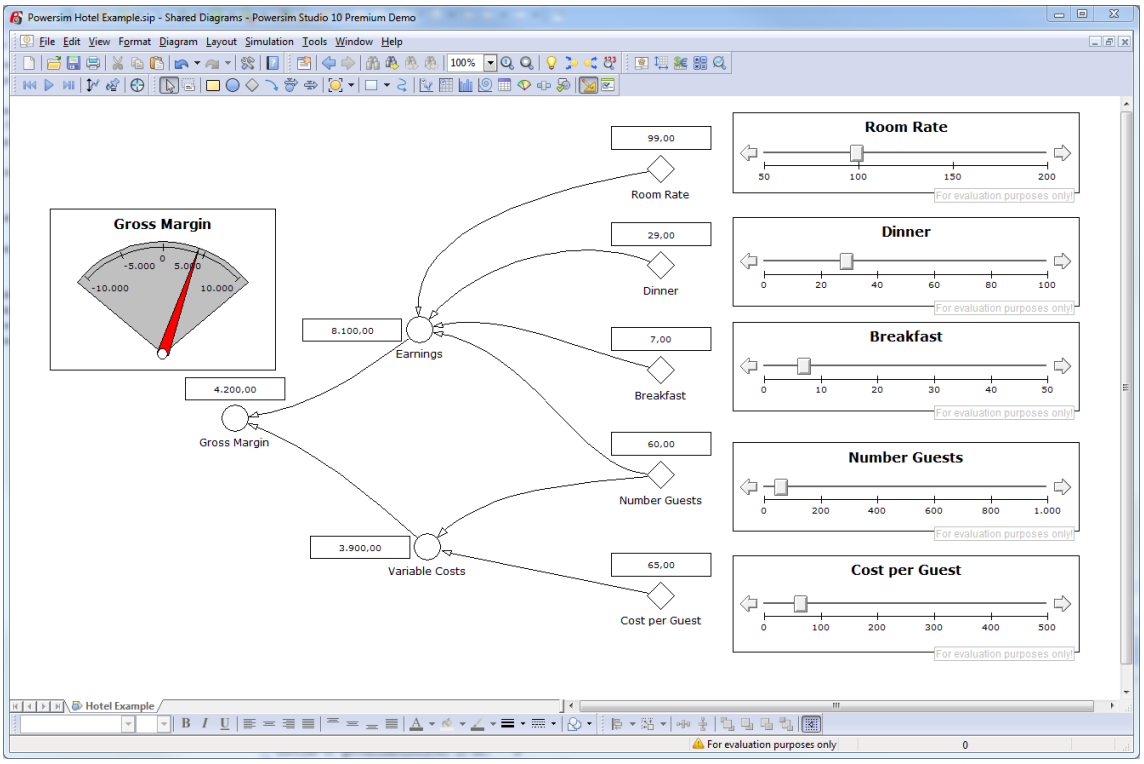

Illustration 8: Powersim Studio 10 Premium Demo - Hotel Example

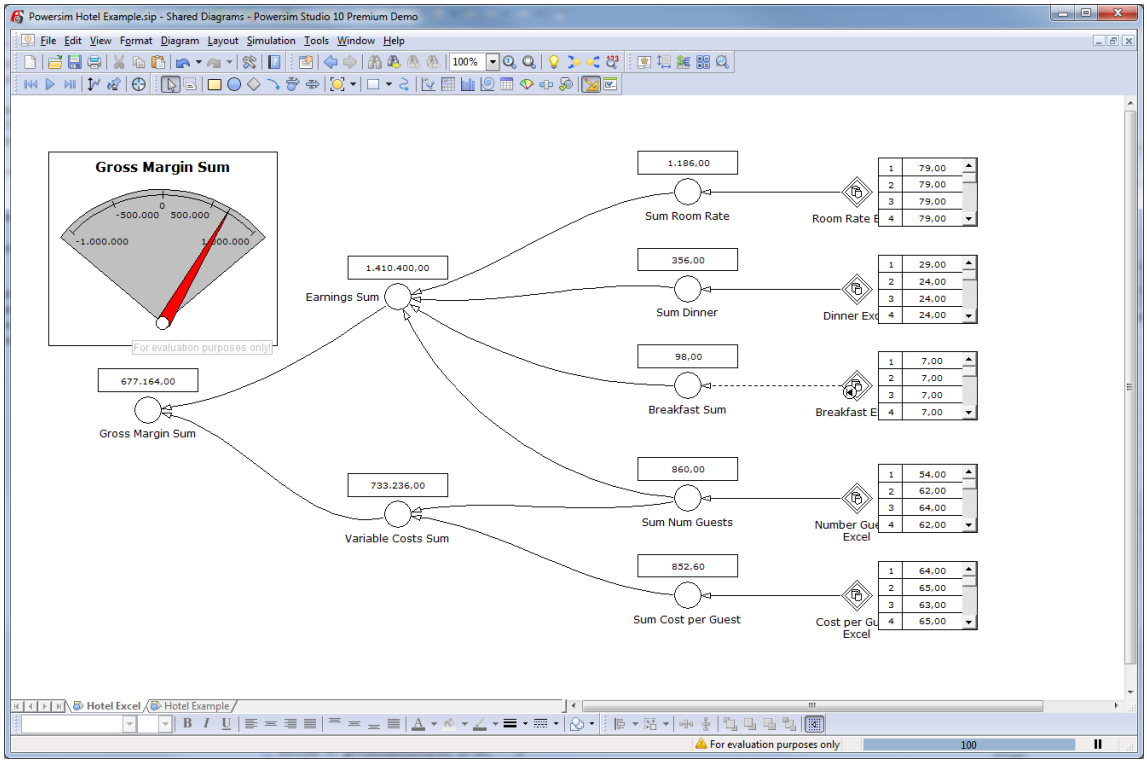

Illustration 9: Powersim Studio 10 Demo - Hotel Example with Excel Data

In regard to the data source requirement DR-01 and DR-02 various data sources are supported[14]. However, the user has to have the structure of the underlying data in mind when modelling since no metadata is available through the application (cf. DR-03). In regard to the non-functional requirements one aspect stood out: The large toolbars, the many different dialogs behind the toolbar actions and the different behaviours for clicks. For example, a double click on the circle of a variable opens the variable definition dialog, but a double click on the variable's name opens a dialog to format the name. Considering these aspects, the usability requirement (NFR-08) was marked as unfulfilled. A more detailed description of the fulfillment of all other requirements can be found in the Appendix 11.1.

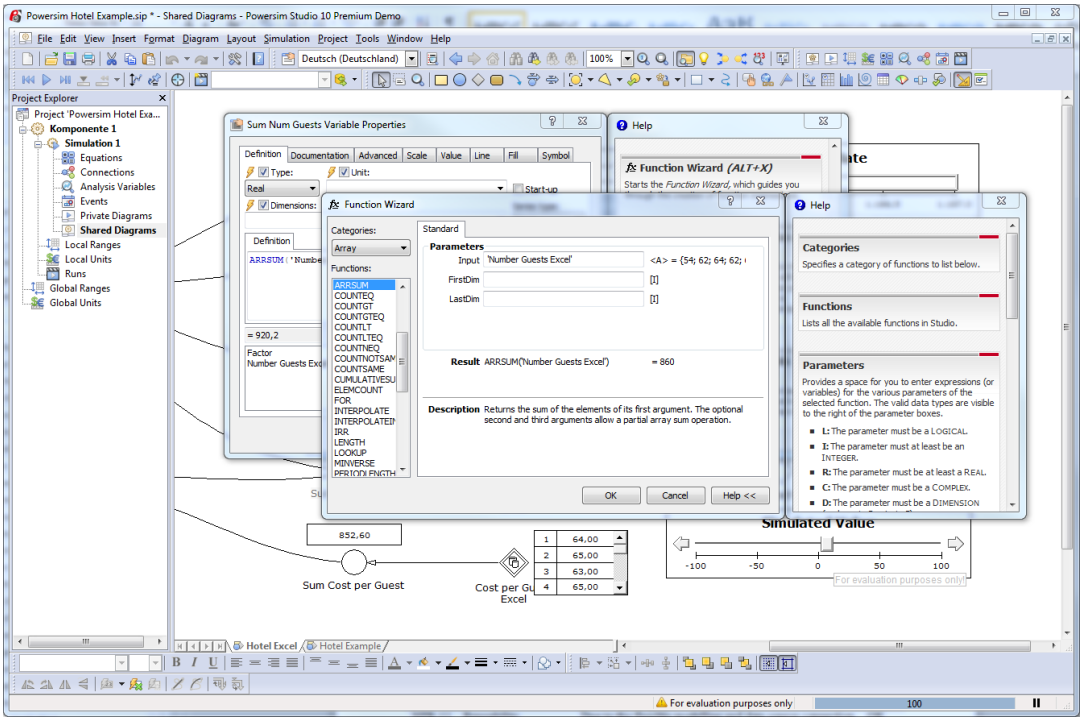

Illustration 10: Powersim Studio 10 Premium Demo - Advanced Mode with Dialogs

4.4 Summary of Market Research

Summarizing the market overview, a lot of the products are either not suited for the problem or too complex for the user group in question. *Microsoft® Excel* has its advantages on usability, prevalence and availability. However, it lacks to visualize changed impact factors of calculations. Furthermore the calculation scheme is often difficult to understand, virtual data structures need to be held separately and the maintainability of spreadsheets is poor. Powersim Studio 10 closes the gap between a data-centric *Excel* spreadsheet and the visualization of the calculations. The immediate feedback on calculations, the simple elements to create a model and the good integration with *Excel* are aspects to takeaway. However, the software does not fulfill approximately 30% of the requirements.

Based on the findings from the market research and literature review, the combination of both is the key to solve the problem in question. As Illustration 8 shows, this calculation visualization is in alignment with Spence (2007). Editing data easily and filtering data with a few clicks is the main takeaway from evaluating *Excel* on this problem. Since *Excel* and *Powersim Studio* lack to provide a unified way to apply models on different sets of data and compare various scenarios a new approach has to close this gap. Therefore the following chapters present a new solution approach and its prototypical implementation.

[14] Product Compatibility: http://www.powersim.com/main/products-services/modeling-tools/comparemodelingtools/

5 Solution Approach

In the previous paragraph the shortcomings of current solutions is summarized. In order to fill this gap and thereby fulfill all the requirements outlined in chapter 2, a new solution approach is presented in this chapter. The driving idea behind the solution is to visualize calculations, enable the user to change influencing factors (parameters) and finally displaying the user-defined deviations in comparison to the model without the deviations. This way, the user will be enabled to interact with the calculation, thus get a better understanding of it and ultimately see the impact of his/her deviations. In the following sections each requirement grouped by its category and its proposed solution is described.

5.1 Basics

The first business requirement is about improving change management with a higher predictability. In order to satisfy this requirement a deviation of the available data has to be considered. By that the original data can be regarded as such and compared to a value with the deviation factor Δ. Consequently the relation between the original value d in a data set and the simulated value s is defined as follows:

$$d * \Delta = s \qquad (1)$$

Furthermore additional objects have to be considered to simulate the impact of an additional invoice, another software application or another regular guest. Marginal cost calculations are just one example of a usage. In the context of this work the additional data is referred to as *Virtual Data*. For example, in a relational database rows should be added to a table virtually. All rows belonging to the table, but not original data are then considered virtual data. Formally defined the original data set D is extended with the virtual data V to form the data S on which the simulation is executed:

$$D \cup V = S \qquad (2)$$

With these definitions in hand, we have four different states to consider:

1) $\Delta = 0$ and $V = \emptyset$ (no deviation, no virtual data)
2) $\Delta = 0$ and $V \neq \emptyset$ (no deviation, virtual data present)
3) $\Delta \neq 0$ and $V = \emptyset$ (deviation, no virtual data)
4) $\Delta \neq 0$ and $V \neq \emptyset$ (deviation and virtual data present)

On the basis of these sets and values one can define complex calculations for a KPI as described in BR-02. The KPI is the evaluated once without the consideration of virtual data and the deviation factor Δ (see state 1) and once again taking the virtual data and Δ into account. At this point another question arises: How can one overwrite a value? In the context of hospitality management someone could ask what happens if a flat-fee is charged instead of a service or consume-based fee. To answer this question definition (1) is not sufficient and must therefore be extended with a simulated constant c:

$$d * \Delta + c = s \qquad (3)$$

With this formula in hand, the deviation factor Δ can be set to 0 and c can be set to a constant. A simulation run will now compare d and s. In order for this equation to solve the problem in question either the deviation factor Δ or the simulated constant c can be set. The value which is not set (unknown) results from the computation of equation (3).

Since BR-02 not only asks to model complex calculations, but it also requires enhancing a user's understanding of the model another solution is needed. If the calculations are modelled as a graph like in section 3.3 explained, a single node can represent a value in a data set on two levels. Either the node represents the concrete value, for example a scalar, or it can represent all values in a set. From a technical perspective the set could be a vector, a column in a database table or a collection of arbitrary type. In case the value is a scalar deviation, it can be displayed increasing the node's size. However, when the type of the value is a set, the deviation cannot be displayed easily except it is known, because it was defined manually. Since most KPIs are defined as numbers or on an ordinal scale which can be transposed to a numerical scale, the data sets have to be aggregated to a single value at some point during the calculation. From this point on it is possible to display the deviation graphically since the value is of a numerical type.

Alternatively many approaches exist to evaluate the deviation value of a set. For example, one could think of the arithmetic mean or median. Possibly, however, this estimation could lead the user to false assumptions. Therefore the author decided not to display deviation factors on sets, except the deviation factor is manually set.

Finally BR-03 requires the technical solution to compare different scenarios. In order not to defy the second business requirement, the comparison must be assisted visually. To solve this problem the calculation model is executed for each scenario individually. Afterwards all the scenarios are merged together for comparison. Thereby each node comprises d, Δ and s for each scenario. The value of c is irrelevant, because it is either s or 0 since the suggested solution approach prohibits setting Δ and c at once. Now the deltas of all scenarios can be compared and visualized. As Spence (2007) suggests a simple set of numerical values can be compared using a bar chart. The author adds the numerical values above all bars in order for small deviations not to be perceived as equal.

5.2 Functional Requirements

Based on the general solution approach the following sections describe the solution ideas for each requirement.

5.2.1 User Requirements

In the study of the British Psychological Society (BPS) (2012) the average office worker sits on his/her desk for more than five and a half hours a day. Based on this finding and the preference for *Microsoft® Excel* as a desktop application, the author suggests that a desktop web application would best suit the behaviour of the target user group. In contrast to a native desktop application, a desktop web application runs in the browser of the client and on a webserver. The author suggests that the webserver communicates with the data sources and executes the main calculations. Consequently the webserver can control the execution and parallelization of the calculation tasks. Furthermore the access to the application can be centrally controlled. Despite these aspects, the browser on the client has to render the results and display them on the screen. This puts load on the client machines and reduces the load on the webserver. Depending on the size of the calculation (model size), this load can be significant on the client machine (cf. chapter 8.8). Using the desktop web application is technically possible from desktop and mobile devices; however, as the reader will see in the following paragraphs, the interaction with the model is focused on desktop-use (cf. UR-01 and UR-02).

Derived from the findings in section 3.3 modelling calculations should be graphical (UR-03). Calculations can comprise constants and variables, aggregations and field values. Constants hold a numerical value entered by the user. Variables obtain their value as a result of its children's values. Field values represent a set of values, for example, one column in a relational database table.

Aggregations require at least one field value child. The logic behind this restriction is that the parent node aggregates all the values represented by the field value node to a single value. This is necessary due to the fact that single values can be compared easily and their difference can be plotted on a single dimension. In case multiple values are compared a one-dimensional scale is not sufficient. Thus by reducing the field values to a single value the difference (simulated delta) between the original value and the simulated value can be displayed with a single bar or sized circle, for example. If each factor of the calculation is displayed as a node in a graph, the size of the node indicates the user the relative impact of the node's simulated delta. In addition, all nodes are linked with edges. An edge's thickness represents the child's simulated delta. Following the thickest path in the graph, the user will be lead to the influencing factor with the highest impact on the target value (UR-20/21).

In order to aggregate the values represented by a field value node, different operations can be applied. Basic operations such as the sum, product, count, minimum and maximum of the values are included in the solution (UR-05). Custom operations depend on the tangible data source and the technical architecture of the solution (e.g. where the data is processed). Hence they are not considered in further detail. In case one aggregation has multiple children, these have to be set against each another. Therefore another set of operations is required (UR-04). Simple operations such as addition, subtraction, multiplication and division are supported. Custom operations are also related to technical questions and therefore subject to discussion in section 7.2.6. Taking both of these solutions in consideration, a combination enables a user to model complex calculations. For example, an aggregation can consist of a field value, a constant, a variable and another aggregated value. All of these child nodes are set against each other. To retrieve the result of the parent aggregation, the values have to be evaluated recursively (UR-09). To increase the performance each of the nodes can be calculated individually and in parallel. The solution assumes that all operations only read data and the values are not changed during a calculation. Finally the parallel operations need to be joined together and evaluated to receive the parent aggregation's value. During this process, the mathematical rules are followed (UR-12), for example 1+2*3 = 7 not 9. This applies to all calculations of the graph.

As the example shows, all values in a calculation graph are numerical (UR-06), except constraint values. If this requirement would be watered down to support nominal or ordinal value scales as well, calculations and comparisons become impossible. Obviously this is only true for non-transposed scales. If, for example, an ordinal scale is transposed to a numerical scale, then it is consequently supported – at least partly. To select only a subset of the values represented by a field value node, constraints can be applied. A constraint consists of a field, a comparator and a value. The field represents a column in the row of a relational database table, for example. Typical comparators include equal, unequal, greater than, less than and greater/less or equal. The value of a constraint can be either a numerical or a string value. In case multiple constraints are defined for a field value node, their values are combined with a logical *and*-operation.

New calculation graphs (models) are created starting with a single root node. Each node has a type (constant, variable, aggregation, field value), an operator, a value, a simulated value and a simulated delta. The user can either set the simulated value or the simulated delta (see section 5.1). Numerical values are handled internally with the highest precision supported by the selected technology (see chapter 0, UR-07 and UR-08). The user can assign a node to another to indicate a parent-child relationship. Furthermore the children of a node can be reordered (UR-10). Depending on a node's type, other input possibilities are displayed to the user. A constant node is the only one where the user can set the value and no other information but the basic information is required. Variables must have children in order to determine the value of the variable. The operation to set children against each other is specified in the node itself. For example, if a variable is defined as the addition of two constant children, the first constant must not have an operator, but the second constant must specify

the addition as its operator. The equation is visible to the user when selecting a variable node (UR-11). This also applies for aggregation nodes which potentially aggregate more than one field value node. In addition, aggregation nodes comprise an input method to specify the aggregation operation. Field value nodes are always children of aggregation nodes since they must be aggregated to a single value for the before-mentioned reasons (cf. UR-03). As described above, field values include constraints and the name of the field they target, for example a column name, in addition to the basic information for all nodes. Finally a user can set either an absolute value (UR-13) or a relative factor (UR-14) for each node. Based on equation (3) from section 5.1 the evaluation of the nodes is performed. Thereby the absolute value set by the user is the simulation constant *c* and the relative value is the simulated delta Δ. For example, the simulated value s_a of a sum-aggregation node *a* with the single field value node-child *x* is evaluated as:

$$\sum_{i=1}^{n}(d_i * \Delta_x + c_x) * \Delta_a + c_a = s_a \qquad (4)$$

$$\equiv (\sum_{i=1}^{n} s_i) * \Delta_a + c_a = s_a$$

In case the aggregation has multiple children $x_{1..n}$ and they are set against each other with the operation $\oplus$ then the simulated value *s* is evaluated as:

$$\left(s_{x_1} \oplus s_{x_2} \oplus \ldots \oplus s_{x_n}\right) * \Delta_a + c_a = s_a \qquad (5)$$

With these equations in mind, the user requirement UR-16 is fulfilled, because the deviated values are always considered recursively. To fulfill requirement UR-15, the simulated value *s* is calculated independent from the original value *d*. Therefore the calculation needs to be executed twice, once to consider only the original values and once shown by the equations above. Finally both results can be compared on a node level to determine their difference – which is by definition Δ. To make sure the calculation can always be repeated with the same result (cf. NFR-06) the original values are not changed, but intermediate results are only stored in memory.

Before the model can be simulated, it must be saved. Models are created on the client and sent to the webserver when the save-action is invoked. Therefore the client must serialize the model and the webserver must deserialize and store it. Each model has an individual identifier (model ID) and it is stored data source dependent (cf. UR-18). This is necessary since the model is created based on the meta information of the data source and this is likely to be different for each available data source. Furthermore models are user-account-specific. This way multiple users can create their own models based on the same data source.

Every user can start a model simulation run individually (UR-19). Therefore a button on the frontend will enable the user to send a request to the webserver to start the calculation. Depending on the model size, the resources available on the server(s) and on the network as well as the amount of data the model is based on, the user is confronted with a waiting dialogue until the model is simulated. Once the simulation is complete, the results are saved in the model and returned to the client. The client renders the model and displays the results (UR-20/21). In addition, the user can save the current result as a named scenario. The available scenarios (actually their names) are displayed immediately to the user in a dedicated area. He/she can click on a scenario name to update the model immediately with the result (UR-22). Thereby no interaction with the server is required since the results are stored in the client's browser memory. Although the solution does not require the simulation results to be structured equally, the merge algorithm will only compare the scenarios based on common nodes. Common nodes are defined as nodes with the same node identifier (node ID).

So far only original data and its numerical deviations were regarded. Virtual data represents additional data that is only considered to determine the simulated values and deltas of nodes. The equations (3) to (5) are applicable not only to *D*, but also to *S* as defined in equation (2). In order for the user to define the virtual data *V*, meta information of *D* is required (DR-03). Herein *V* is defined as structurally identical to *D*. From a technical perspective it would be tempting to store all additionally defined values of *V* in the data container of *D*, however, UR-24 prevents this. Therefore *V* is stored separately depending on the data source. In sequential order, the user requests to add virtual data to the simulation. Since the metadata of *D* is required, the webserver is queried for this data. Based on it, the application creates a form to insert, remove and save virtual values (UR-23). The serialized virtual values are then sent to the webserver. When the user starts the simulation of a model where *S* is regarded, the original values *d* will be evaluated with $V=\emptyset$, but the simulated values *s* and respectively their simulated deltas Δ will be based on *S with* $V\neq\emptyset$ (UR-25/26).

5.2.2 Data Requirements

A generic solution requires the support for several data sources since different enterprises employ different data sources (DR-01). Common to most enterprises is the storage of data in a structured, table-oriented way. Derived from this finding, the solution supports a file-based table storage format and the use of a relational database management system (RDBMS). In both cases field value nodes represent one column in a table. The selection of the values can be manipulated by adding constraints to a field value node. Since the file system does not offer a query interface for table-structured data by default, the constraints must be implemented manually. The RDBMS is queried using the Structured Query Language (SQL). In SQL the constraints are expressed using the `WHERE` clause. Often it is necessary to constrain a value selection based on values in tables related to the table of the field in question with foreign keys. If a constraint value targets another column in a foreign table then the first foreign key that matches the foreign table is taken to join both tables together. As mentioned above, all field value nodes are parents of aggregation nodes. Furthermore the model can comprise a recursion of arbitrary depth. To calculate the value of an aggregation node, views are used to temporarily store SQL queries on the data. Views thereby can be based on other views to model the recursive behaviour. Virtual data is inserted in a separate table with the same structure than the table it is based on. With another view the virtual data table is united with the original table. This view is then employed to calculate the simulated values.

In addition to structured data sources, enterprises can store semi-structured data in hybrid wikis (cf. Matthes et al. 2011). Reschenhofer (2013) defines the Model-based Expression Language (MxL) specifically for the calculation of KPIs on such a hybrid wiki. Similar to SQL MxL supports language constructs to query data from a hybrid wiki and to constrain the selection. Exemplary for hybrid wikis and semi-structured data sources T39 (also known as *Tricia* by *InfoAsset*[15]) is integrated in the proposed solution to verify the robustness and viability of the approach. In contrast to the SQL-based solution approach, MxL does not store virtual data persistently in T39. Virtual data is only kept in the user session and added to the execution environment when required by the querying entity. Furthermore T39 supports to generate models based on MxL queries.

In regard to the different data sources integrated in the proposed solution, the architecture presented in chapter 7 ensures the extensibility for further data sources (DR-02). As described above, meta information is required to edit virtual data and to define a node's properties (DR-03).

[15] Company website: http://www.infoasset.de/, checked on 7/24/2014.

5.2.3 Technology Requirements

Based on the requirements DR-01/02 the solution must connect to these data sources using a network. In addition, the proposed desktop web application requires a communication between the webserver and the browser client. Therefore the requirement TR-01 is automatically met. Similar applies the use of standards as requested by TR-02. The W3C defines a set of standards for an *Open Web Platform* (W3C 2012). Based on these standards, the architecture and exchange protocols are defined and documented as descripted in chapter 7.

5.3 Non-Functional Requirements

Complementary to the fulfillment of functional requirements are solutions to non-functional requirements. As NFR-01 states, the software solution needs to be accessible from the desktop and notebook computers. This is achieved by providing the application through a modern web browser. The application resides on the webserver and is delivered to the client upon request. One of the basic features is a login procedure to prevent unauthorized access to the application based on username and password. A certificate-based authorization mechanism could be implemented as well, however, due to the prototypical character of the application and the specific security guidelines in enterprises, this feature is not considered a proof of concept in the opinion of the author.

To provide a responsive, interactive and decision-supportive solution to simulate the impact of influencing factors on KPIs the efficiency of the solution has to be optimized (NFR-02). Rendering graphical elements such as the calculation graph has to be performed on the client respectively in the browser. Therefore a suitable technology has to be selected that enables the seamless integration in the application. Furthermore it has to allow interaction with the graph. In addition, the client should cache meta and model information to reduce the requests to the webserver. The same applies for the webserver which has to cache the meta information for a data source to reduce the load on the backend systems. For SQL data sources that support connection pooling, it should be applied to leverage the possibilities of parallel processing. By that the calculation of the independent parts of a calculation graph are executed individually and joined only when needed. This increases the performance of the application and reduces waiting times for the user.

In a fast changing environment such as the information technology world, the flexibility of a solution is essential for its sustainable success (cf. NFR-03). By supporting multiple data sources and the access from various computers via the Internet or a network a basic flexibility is guaranteed. However, flexibility is not limited to user-centric aspects. A modular architecture with the principles of object-oriented programming and the application of patterns enables exchangeable components on all levels. For example, storing models on the webserver's file system is an obvious solution. However, document-based data stores might increase the performance and abilities of a system. Therefore this component must be abstracted through a common interface allowing different implementations for the same functionality. Elaborating on this example, models are stored based on an identifier (model ID), a user and data source. Thereby multiple models can be stored per user and each user can store models individually. Additionally models depend on a data source since they reference fields and types. A type that is existent in one data source is not necessarily available in another data source. Furthermore their semantics might differ. For example, the column "date" in one invoice relation might refer to the date of creation while in another data source the "date" column refers to the payment or booking date.

As the Global Economic Crime Survey (PrincewaterhouseCoopers LLP 2014) emphasized, cybercrime has become one of the most important frauds in the industry. Considering the power of the solution in question, its abilities to performance calculations on productive data and to access confidential data, a proper security concept is required. At first the connection between client and

webserver must be established through a secure protocol such as the Hypertext Transfer Protocol Secure (HTTPS). The web application should be configured to reject non-secure HTTP requests. For user convenience the login page request can be forwarded to an HTTPS-secured resource. Although this protocol allows identifying a user with a certificate an additional identification layer is provided to allow users to access the application from computers which do not have the user's personal client certificate installed. Beginning with the authentication of the user through his/her username and password, the application changes its session identifiers after the user has successfully logged in to prevent an attack called *Session Fixation* (cf. Schadow 2014, pp. 81–92). Afterwards all input values are validated on the server-side to prevent injection and cross-site scripting attacks. Also the values retrieves from data sources are sanitized by rejecting values containing blacklisted keywords. In summary these measures identify the user, protect the user against manipulated and falsified information and thereby satisfy NFR-04.

Building on the web standard for secure transport, other standards enable the interoperability between systems (NFR-05). For example, relational database management systems (RDBMS) are often connected to Java applications using Java Database Connectivity (JDBC) (Oracle Corporation 2014b). All major RDBMS offer a JDBC driver and therefore a solution building on this technology is theoretically compatible with all the implementations. However, in practise the interface usage varies since the querying language, for example SQL, differs or some features are supported, others are not. Similar compatibility issues can be observed with other web standards such as Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), JavaScript and the JavaScript Object Notation (JSON). Although all modern browsers support these technologies in general, their implementation slightly differs and thus the application is not fully portable. Due to these differences solutions have been developed to abstract from the target execution environment. Conclusively the author suggests using an abstraction layer for the frontend web application technologies and the abstraction from the concrete data source to allow interoperability and portability (NFR-10).

Especially in environments with various different configuration possibilities the value of automated tests is high. To ensure the reliability of the solution (NFR-07), automated unit and integration tests must be implemented and executed for various configurations (regression tests). All critical aspects of the solution such as the validation of the correct calculation for all four states mentioned in section 5.1 must be tested individually. The integration of the various components must be tested with partly automated and manual tests. Combining multiple test cases to test suits ensures the complete execution of all test cases relevant for the scenario. Furthermore the test suits enable a faster execution, because single test cases and test classes do not have to be executed one-by-one. With faster execution, test cases can be executed more often and thus improve the quality of the product since defects are found sooner. Despite the positive effects of testability (NFR-12), some tests have to rely on a certain configuration or data in the backend and thus fail. To interpret the success or failure of a test case an automated test environment is not sufficient, but a human familiar with the requirements and goals must evaluate the results.

Another form of testing is the validation of the syntactic correctness of models (model validation). The user has to have the ability to validate his/her model to identify why a model cannot be simulated or fails to produce the expected results (NFR-07). Depending on the data source and the configuration, the correctness of the model can vary. Therefore three validation levels define whether the simulation should be blocked when the model is syntactically incorrect or whether a warning should be displayed, but the simulation should be executed or whether the validation results should be ignored and the simulation should be executed regardless of the results. In addition to the model validation, an input validation using a security framework prevents the user from false input and malicious intentions.

In section 0 the difficulties and limitations to define the usability of the solution are explained. Despite the odds, the solutions proposed so far targets usability is various ways. For example, due to improve processing performance and reduced waiting times the interaction with the model is more fluent. Furthermore graphical rendering on the client enables the user to visually model his/her problem and receive visual feedback. The visual aspects as well as the architecture and the security features are guided by best practises from industry (for example through patterns) and a systematic literature review harvesting the experience and finding of other authors. Additionally the syntactic validation and thorough documentation of the solution allows the user to prevent and correct mistakes. Finally section 9.3 provides useful patterns for modelling KPIs with the application. This helps the user to efficiently and effectively create calculation graphs for his/her needs. In combination with the features and improvements mentioned before, the basic usability requirements are fulfilled (NFR-08).

The requirements for maintainability (NFR-09) and reusability (NFR-11) are solved together. While the maintainability is improved by thorough documented code and the reusability is enhanced by a modular architecture both aspects profit from an interactive documentation in addition to comments in the code and the documentation of protocols and the architecture. One of the drivers behind this documentation style is the popularity of Cloud Application Programming Interface (API) Management Platforms (AMP) acting as a broker between API providers and consumers (Höß et al. 2014). For the solution in hand, the interfaces between the webserver and the client as well as the data source implementation and the T39 backend are documented using a structured, JSON-based format called Swagger (wordnik 2014). Please refer to section 8.9 for details.

5.4 Limitations

In contrast to the requirements fulfillment as described in this chapter so far, the solution has its limitations. A theoretical limitation of equation (2) is the lack of defining subsets of *D* as virtual data. This feature is needed to answer questions concerning scenarios where values should be omitted. With the given solution the comparison of scenarios is possible; however, the graphical support is limited. For example, one could define a subset of *D* by applying constraints to field values and executing the simulation. After the simulation is complete the results are saved in a scenario and the additional constraints are removed again. Then the simulation is executed and saved as another scenario. Now both scenarios can be compared on a numerical level.

Another conceptual limitation is the lack of defining cyclic dependencies of nodes. While one KPI can rely on another, two or more KPIs cannot rely on the same KPI. The reason for this limitation is the reduction of the modelling tool to the smallest viable and usable set of interaction possibilities for the user with the model. To implement this feature the user would have to define the relationships between nodes differently than dragging a child on its parent to mark a parent-child relationship. The user would have to define links between nodes and he/she must be enabled to remove links. In addition the data structure to store the model would have to be changed since a node can have multiple parents. In addition, processing the model would require a cycle check and solve cyclic dependencies. Finally custom operations can be defined with the solution presented in the following chapter; however, the reader will see that it requires an alternative computation approach.

6 Technologies

On the basis of the theoretical solution presented in the previous chapter, the technologies to implement the solution are selected. With a suitable programming language in hand, the tangible data source technologies are depicted. Finally the web technologies and frontend with the visualization of the models are explained.

6.1 Programming Language

According to four different studies and datasets, the most commonly used programming languages are C, Java, Objective-C, C++, C#, PHP, Python and JavaScript (Barzokas Vassilios (2014), Gautier de Montmollin (2013), Stephen O'Grady (2013), TIOBE Software BV (2014)). In respect to their usage for web applications, Java is selected as the primary language for server-side applications and JavaScript for front-end development.

6.2 Data Sources

Based on the DB-Engines Ranking for March 2014, eight of the ten most widely used and discussed database management systems are relational databases (solid IT 2014). Also Gartner acknowledges in its Magic Quadrant for Operational Database Management Systems that the industry leaders – namely Oracle, Microsoft, IBM and SAP – are focused on relational database management systems (Gartner 2013). "Most are beginning to add this [in-memory] functionality to the DBMS, some with an in-memory-only model. In-memory computing, with its inherent speed, is becoming necessary for the processing of interaction and observation data integrated into transactions." (Gartner 2013) Since the solution is targeted at enterprises, it should comprise at least one relational database management system. Exemplary for these systems the open source in-memory-capable database management systems *H2*[16] and *HyperSQL*[17] were selected. The reason for selecting these technologies over a classical *MySQL*[18], *PostgreSQL*[19], *Microsoft® SQL Server*[20] or alike was their claim to be lightweight, fully in-memory capable and open source. Furthermore generating test data with a custom application on a MySQL database was significantly slower (factor 2) and even crashed after a few hundred thousand inserts on a standard notebook computer while *HyperSQL* and *H2* still managed the load. However, the implementation uses JDBC and SQL, thus all of the mentioned database management systems can be employed as well, although with adaptations in the SQL syntax.

Based on the finding that *Microsoft® Excel* is widely used in enterprises today (cf. section 5.2.1), the user should be enabled to use datasets from *Excel*. With *Microsoft® Excel 2007* the default file format changed from xls to an XML-based xlsx (Microsoft Corporation 2014a). To support both formats and possible exports from other applications, the more generic format of Comma-Separated Values (CSV) was selected. CSV-files can be uploaded to the application and then used as a data basis. Beside SQL-based and file-based data sources, T39 was selected as a representative of an individual application. Prior to the realization of the solution in hand, the application had no API usable for external tools. Furthermore it allowed the validation of the proposed solution on non- or semi-structured data.

[16] H2 Database Engine: http://www.h2database.com/html/main.html, checked on 7/24/2014.
[17] HSQLDB: http://hsqldb.org/, checked on 7/24/2014.
[18] MySQL: http://www.mysql.com/, checked on 7/24/2014.
[19] PostgreSQL: http://www.postgresql.org/, checked on 7/24/2014.
[20] Microsoft SQL Server 2012-2014: http://www.microsoft.com/en-us/server-cloud/products/sql-server/, checked on 7/24/2014.

6.3 Web Technologies

On top of the data sources and with Java selected as an appropriate language for the development of a web application, the webserver had to be selected. Similar to the selection of relational databases were many possibilities such as the Eclipse project Jetty[21], Apache Tomcat[22], JBoss Application Server[23] and others available. Based on the author's experience with Apache Tomcat and the native integration in the integrated development environment (IDE) Eclipse[24], it was selected as the target platform for development. In addition, the Java Enterprise Edition (JEE) was selected as the main technology for the server-side application components. With its JSON interface (JSR 353, Java Community Process (2013)) and the JSONP implementation[25] as well as its Servlet and Java Server Pages (JSP) features, JEE is a powerful application platform. Additionally the following libraries were added during development: Apache Commons Lang[26], Apache Commons Validator[27], exp4j[28] and the Java drivers to connect to the backend systems.

6.4 Frontend and Visualization

For the frontend displayed in the user's browser HTML5, CSS3, jQuery 2.1.0[29] and jQuery UI 1.10.4[30] were selected. One of the reasons for the selection is their wide support for browsers, their abstraction from implementation details of JavaScript and that they are partly prerequisites for using D3.js. Before D3.js was selected as the library to build the graphical representation of the models with, other frameworks such as Gephi[31] and GraphStream[32] were evaluated for their applicability for the problem. The following three illustrations show similar models developed with each of the frameworks:

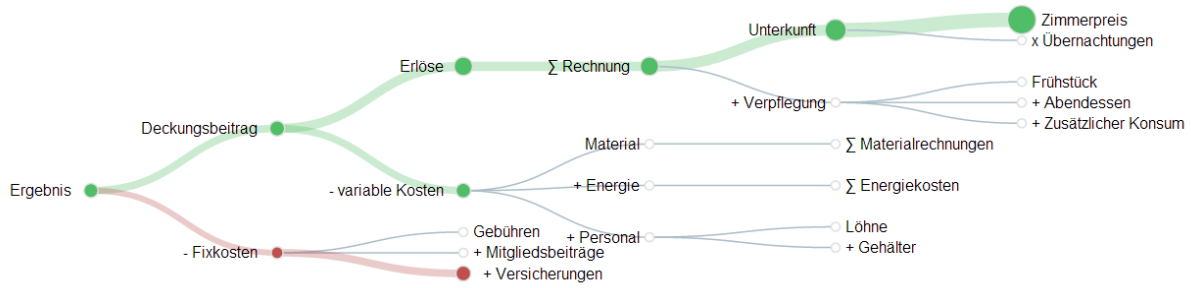

Illustration 11: Example Model with D3.js

[21] Servlet Engine and Http Server: http://www.eclipse.org/jetty/, checked on 7/24/2014.
[22] Apache Tomcat: http://tomcat.apache.org/, checked on 7/24/2014.
[23] JBoss Application Server: http://jbossas.jboss.org/, checked on 7/24/2014.
[24] Eclipse: http://www.eclipse.org/, checked on 7/24/2014.
[25] JSON Processing: https://jsonp.java.net/, checked on 7/24/2014.
[26] Apache Commons Lang: http://commons.apache.org/proper/commons-lang/, checked on 7/24/2014.
[27] Apache Commons Validator: http://commons.apache.org/proper/commons-validator/, checked on 7/24/2014.
[28] exp4j: http://www.objecthunter.net/exp4j/, checked on 7/24/2014.
[29] jQuery: http://jquery.com/, checked on 7/24/2014.
[30] jQuery UI: http://jqueryui.com/, checked on 7/24/2014.
[31] Gephi makes graphs handy: http://gephi.github.io/, checked on 7/24/2014.
[32] GraphStream – A Dynamic Graph Library: http://graphstream-project.org/, checked on 7/24/2014.

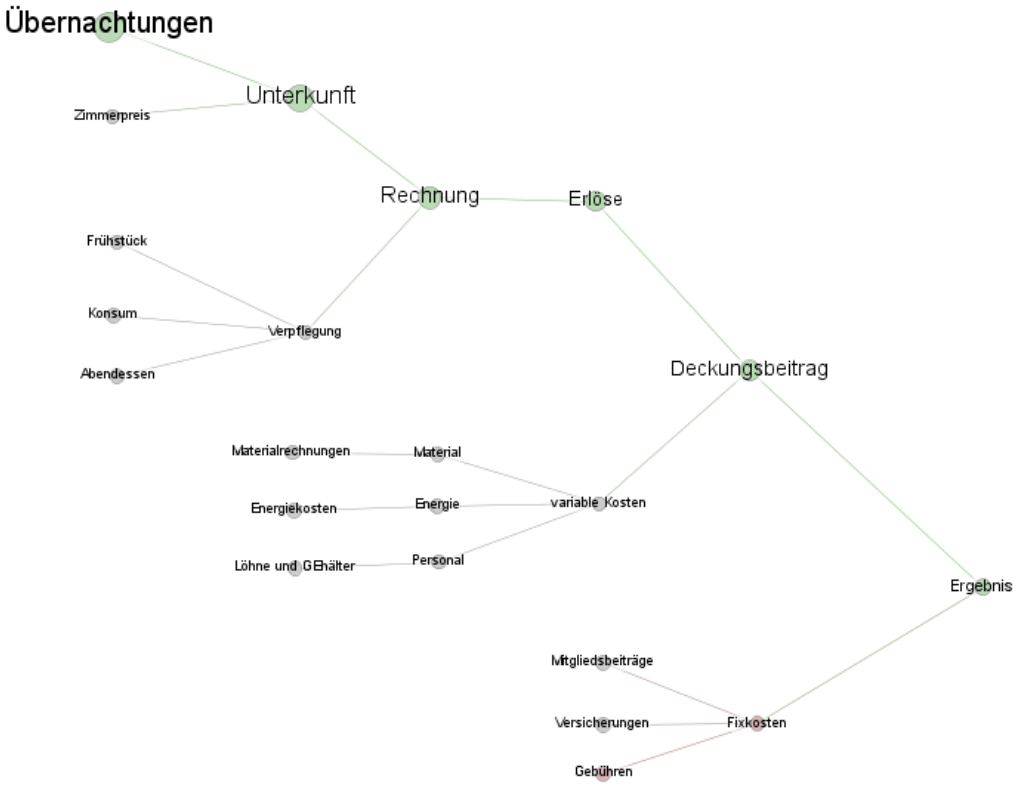

Illustration 12: Example Model with Gephi

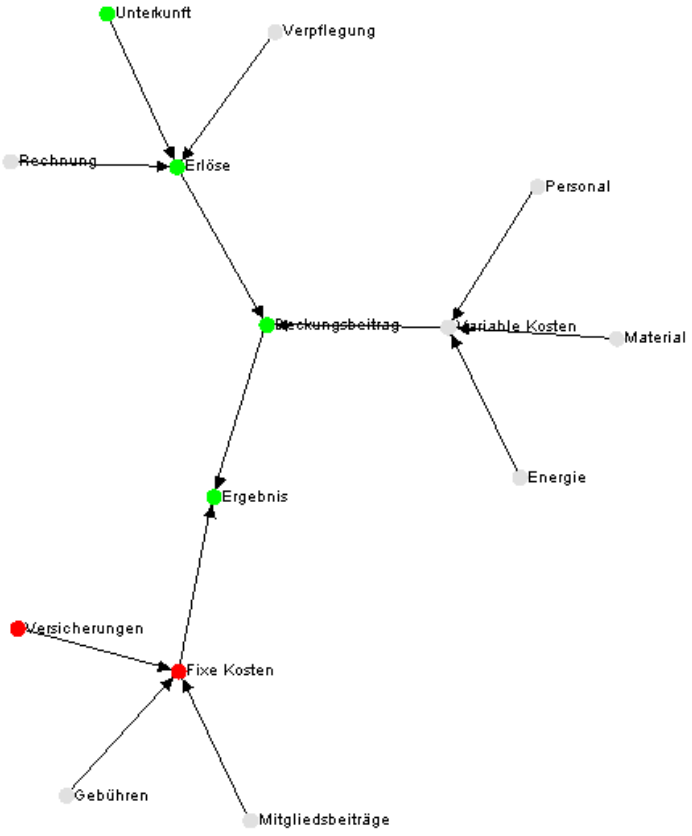

Illustration 13: Example Model with GraphStream

Technically all of the frameworks were suitable; however, based on their graphical representations of the models, the available documentation and user feedbacks on the Internet, the library D3.js was selected.

Finally the client application running in the browser and the server-side application had to be connected. Therefore two technological alternatives were regarded: a RESTful API and Simple Object Access Protocol (SOAP) web-services. Since a RESTful API with JSON as the data exchange format fitted to the D3.js library and could be processed easily with the selected Java technology, it was selected.

Beside the technologies presented in this section, *JavaDoc* (Oracle Corporation 2014a) is used to document the source code. The java.io file system API and the NoSQL document-based database management system MongoDB[33] are used to store models. The latter was selected due to its native Binary JSON (BSON) format which allows storing the models retrieved from the clients directly in a collection. JSON is also used to connect the server-side application to T39, but the connector itself offers a Java interface.

[33] MongoDB: http://www.mongodb.org/, checked on 7/24/2014.

7 Architecture

Based on the concrete technologies and the components outlined in the previous chapters, the interfaces and structure of the components have to be determined. At first, an overview of the architecture is given based on a Unified Modelling Language (UML) component diagram. Afterwards the architecture is described in further detail to prepare the implementation of the components. Finally the core data structures are presented.

7.1 Components

As shown in Illustration 14 below, the diagram comprises nineteen components and twenty-one interfaces. All components and interfaces related directly or indirectly to T39 are colored in blue. The component T39 itself refers to the platform component of T39. Since the platform abstracts from the MxL package of T39, this component is not explicitly modelled. Other components required by T39 are the *elasticsearch*[34] application in version 0.20.x and a JDBC compliant database management system such as a MySQL server (MySQL DB component). To access the platform a user utilizes the Web Frontend component related to T39. The contents for this component are assembled from the T39 platform and passed as HTML, CSS and JavaScript to the client application (browser). With the development of the MxL Driver component in the scope of this work, T39 receives another interface (JSON REST Client API). The MxL Driver component connects to the API and offers a Java interface to external applications. This component is employed by the T39DataSource component.

Data sources and their interfaces are colored in red. Their main purpose is to implement the data source specific parts of the MTCalcEngine component. Furthermore they handle the connections to their backend systems and the configurations necessary for those. In case a component fits in two categories, it is colored according to the category of higher importance in the context of the solution. For example, the T39DataSource component implements the DataSource and Authenticator interfaces. Authenticator implementations are used to identify and authorize a user. The T39DataSource component authenticates credentials against the T39 platform via the MxL Driver. Similarly the T39GraphGenerator Plugin implementation uses an MxL interface offered by the T39DataSource to connect to the T39 platform. Since the T39DataSource component can be configured as a DataSource implementation in the context of the MTCalcEngine component which is the central component in the architecture, it is colored red. In contrast, Plugin components and interface implementations are colored in green. As already mentioned, the T39DataSource component also implements the Authenticator interface which is colored in pink like its other implementation the SimpleAuth component.

White colored components are external systems connected to components of this solution using proprietary drivers or APIs provided by the JEE platform. The only exception is T39 which does not provide a driver. For this reason, a driver was developed for the purposes of this solution (cf. section 8.5.4.1). Other external components such as a MongoDB or the webserver's file system are used by the storage components colored in yellow. Their main purpose is to persist and retrieve models. Finally the orange-colored frontend component is connected via a JSON REST Client API interface to the MTCalcEngine component. Additionally a similar interface connecting the MxL Driver component to the T39 platform is documented dynamically as well (cf. section 8.9).

[34] Elastigsearch.org Open Source Distributed Real Time Search & Analytics: http://www.elasticsearch.org/, checked on 7/24/2014.

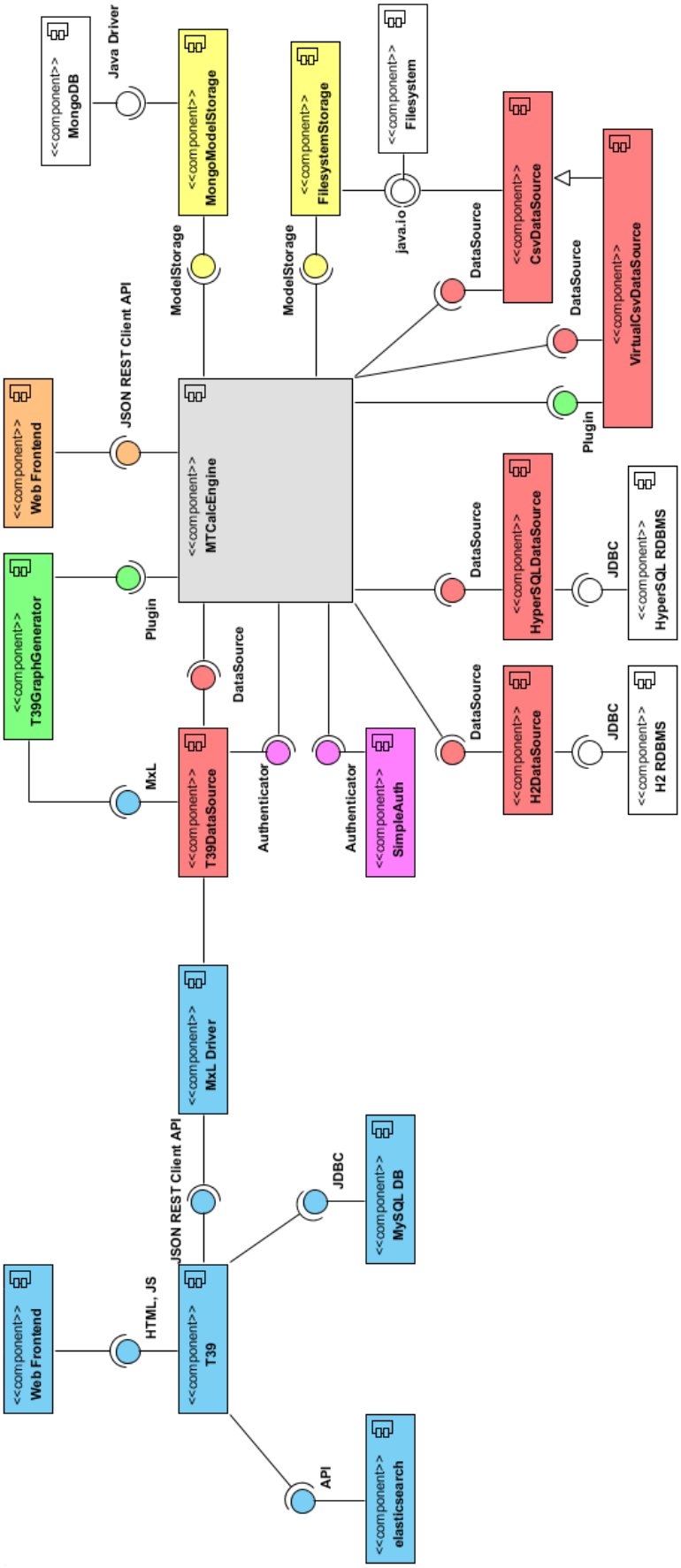

Illustration 14: UML Component Diagram (own diagram)

7.2 Detailed Design

Starting with the component diagram shown in the previous section, the detailed implementation design was created. It was extended and refined multiple times during the realization of the design; however, the core concepts and patterns remained. Therefore the following paragraphs describe these concepts and the patterns used to create a modular, scalable, flexible and maintainable architecture.

7.2.1 Interface Abstractions

Illustration 15 shows an UML class diagram with the structure of the MTCalcEngine component and its interfaces. Furthermore it contains stubs for adjacent components and it depicts the data structure for models and meta information. All classes are contained in packages targeted at a specific purpose. To begin with, the client package contains the `ClientAPI` interface the frontend part of the application communicates with. The implementation of the interface is the `ClientConnector` class. Since its inner workings are irrelevant for the structure as a whole, they were omitted in the diagram. Following this logic, most packages contain one major interface and different, often configurable, implementations of it. By this convention, the concrete implementation is hidden and therefore exchangeable. For example, the `ModelStorage` interface defines methods to store and retrieve models from their persistent storage. The `ClientConnector`, in this case the user of the interface, has no information on the class that is currently configured to be used behind the interface. Whether the models are stored and retrieved from the file system or from a MongoDB server is irrelevant for the `ClientConnector`. Similarly the `CalcGraphFactory`, `Plugin`, `Authenticator`, `DataSource` and `ClientAPI` are all interfaces abstracting from their implementation. Thereby a question arises: Where are the concrete implementations instantiated then? In order to encapsulate the configuration and instantiation process, the `CalcEngineConfig` class contains factory methods to retrieve instances implementing the interfaces. The `CalcEngineConfig` itself is created once in the `ClientConnector`. Using a *Singleton* pattern (Gamma et al. 1995) was refrained due to the fact that the webserver would instantiate the configuration only once for all sessions on the server. As it may sound performant since the configuration is only read once and the number of instances are reduced, it defies the fact that the instances itself have different inner states for each session. Therefore it was decided to reject the use of the *Singleton* pattern.

7.2.2 Value Guards

In respect to the security of the architecture, the `ClientConnector` uses the `VirtualDataGuard` in the security package to validate virtual and preview data. In turn, the `VirtualDataGuard` uses the `ValueGuard` class offering static public methods to validate the safety of single values such as strings. Also the `JsonGraphFactory`, the implementation of the `CalcGraphFactory` interface, utilizes the `ValueGuard` class to validate the harmlessness of the values provided from the user and external systems. The `JsonGraphFactory` plays an essential role in the architecture. Its main purpose is to convert Java objects to JSON and vice versa. This functionality is needed not only to communicate via the `ClientAPI`, it is also used by the `ModelStorage` implementations to persist objects. Even the `DataSource` implementations use the factory to handle virtual data and meta information. Despite the fact that the `JsonGraphFactory` is abstracted by its `CalcGraphFactory` interface, exchanging it for an implementation using another exchange format such as XML would result is a major refactoring of the whole application. As described in the previous chapter on technology, JSON was selected as the primary interchange format. Therefore the browser application builds on it and so do the MxL Driver component and the `MongoModelStorage` class.

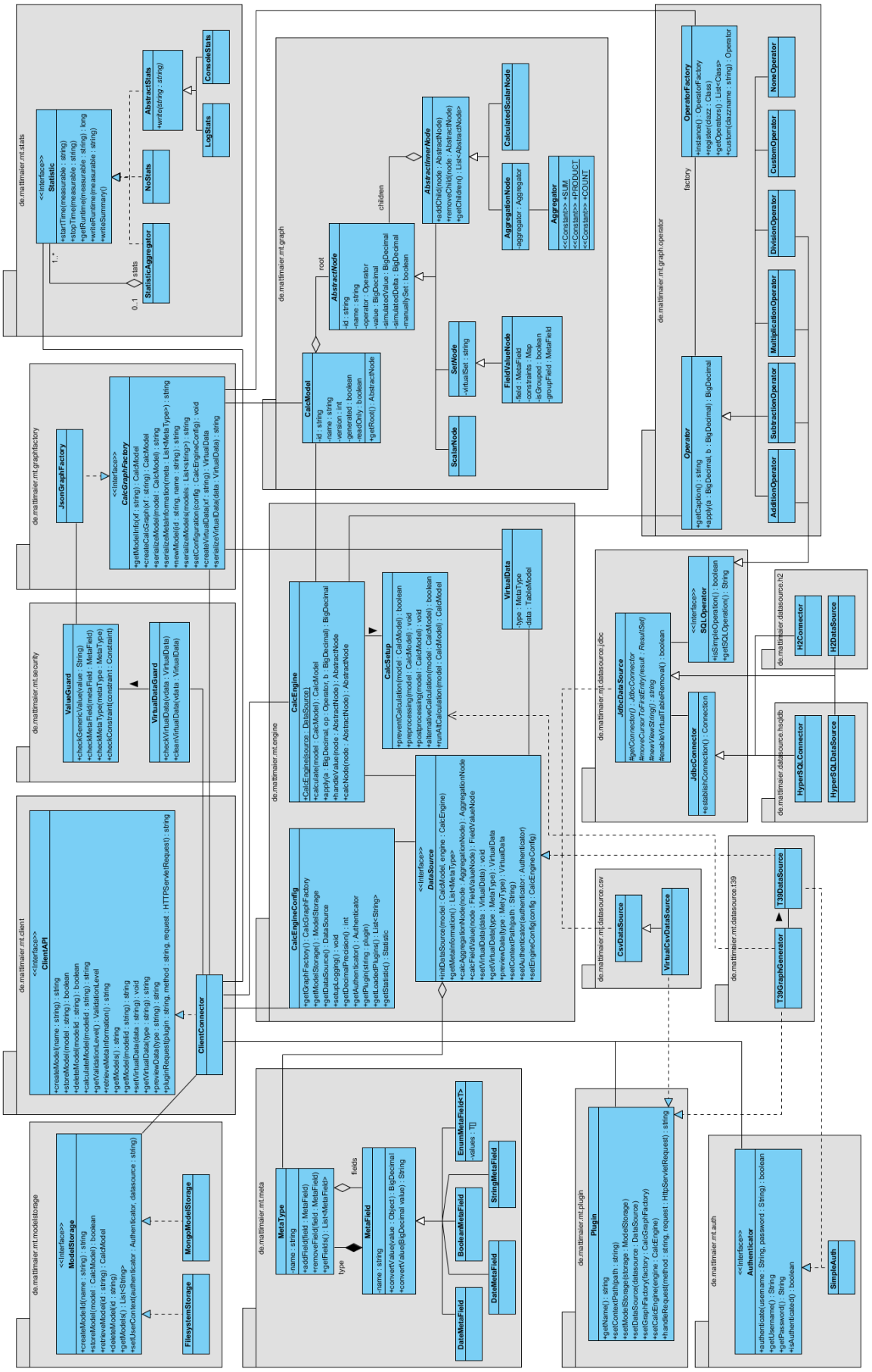

Illustration 15: UML Class Diagram - Detailed Design (own diagram)

7.2.3 Statistics

To complete the top row of packages displayed in the diagram, the stats package is left over. Its purpose is to save and eventually write statistical information on the runtime of operations. The `Statistic` interface allows the `ClientConnector` to start and stop a timer for an operation. In addition, the interface offers a method to print a summary of all registered timers. The `NoStats` class is implemented to simply ignore all timers while the `StatisticAggregator` contains multiple `Statistic` implementations handling their parallel use. For example, if statistics should be written to a log (`LogStats`) and displayed on the console (`ConsoleStats`), the `Statistic Aggregator` receives instances of both and forwards all requests to these instances.

7.2.4 Plugins

Plugins are instantiated by the `CalcEngineConfig`, similar to `Statistics`. However, `Plugin` implementation classes implement more than one interface in this case. Each plugin has a unique name and only the `handleRequest` method is critical for the implementation. Based on the return value of the `getName` method, a new plugin must create a folder under `WebContent/plugins` with this name. In this folder the JavaScript and CSS files contain the logic for the frontend. Additionally an icon in portable network graphics (PNG) format and a resolution of 16x16 pixels with the plugin name must be placed in this folder. The icon is displayed on the front end and when clicked by the user, the function with the name of the plugin (in the JavaScript file of the folder) is executed. Plugins can send requests to be processed by the `handleRequest` method of the Plugin implementation. Therefore the URL has the following format:

`CalcServer/plugins/<pluginname>/<methodname>?<parameters>`

The `ClientConnector` will automatically forward the request to the `Plugin` implementation with the name of the requested method as the first parameter of the `handleRequest` method's signature.

7.2.5 Authenticators

To protect the web application from unauthorized access, a login form with username and password fields is displayed to the user before he/she can use the application. In order to authenticate the user an implementation of the `Authenticator` interface is required. In addition, the `Authenticator` has a second purpose: Requesting credentials to authenticate itself against a backend. Most external systems such as T39 require a user to authenticate before access is granted. Therefore the given credentials can be used. From a security perspective, it is unsafe to keep the password after the authentication; however, for this case the password will have to be used to connect to a backend and therefore needs to reside in the memory of the server until the system is connected. To support this operation the `getPassword` method was not removed from the interface.

7.2.6 Operators

As described in section 5.2.1 in equation (5), single nodes are set against each other using operators. These operators are applied on two parameters and return one result. Each operator has a unique name that identifies itself for the user. These features are represented in the abstract `Operator` class. All subclasses represent a concrete operation such as an addition, subtraction, multiplication, et cetera. To register custom operators the `OperatorFactory` contains the register method. A new operator can be registered with its complete name. Afterwards it can be used on the front end.

7.2.7 Data Sources

Visually in the center of the diagram – thus also in the center of attention – is the `DataSource` interface. It abstracts from all data source dependent implementation details. From an architectural standpoint the core calculation algorithms should not contain any detail on how to calculate a node. This concerns the data source. Consequently the abstraction with an interface is suitable. To reduce unnecessary double implementation the hierarchy goes beyond the first level of inheritance. Direct implementations are the `T39DataSource` and the `CsvDataSource`, because both have little in common. Nevertheless the `VirtualCsvDataSource` almost completely reuses the `CsvDataSource`. Additionally the `VirtualCsvDataSource` implements the Plugin interface to manipulate the data the `CsvDataSource` uses. A similar amount of reuse can be observed with the `JdbcDataSource` and its subclasses the `HyperSQLDataSource` and the `H2DataSource`. Both classes resulted from a refactoring when the `JdbcDataSource` had to many branches for differing SQL dialects used in H2 and HyperSQL. From a conceptual view, this allows a fast integration of other SQL-based data sources as well.

Another minor difference between the operators used in Java and SQL is regarded by the introduction of the `SQLOperator` interface. All operators implementing this interface can be used in a `JdbcDataSource`. The `CustomOperator` and `NoneOperator` do not implement the `SQLOperator` interface, because of their nature. In SQL the link between two values/columns must be specified using an operator like "+". If no operator is specified, the operation cannot be executed successfully. Likewise the `CustomOperator` is a custom, because its behaviour is not standard. Although SQL supports the definition of custom functions and these functions might be available on the database server, a default custom function does not define which one to use and therefore it is not supported to be used in an SQL statement. As mentioned before, an application developer can implement a class inheriting from the `Operator` class and implementing the `SQLOperator` interface.

An additional refactoring helps to reduce double code to connect to a JDBC data source. The `JdbcConnector` class implements the default behaviour to open a JDBC connection. Since most JDBC database management systems enable connection pooling, but offer an individual connection pool implementation, the connector is extended by the `HyperSQLConnector` and the `H2Connector`. Both implement individual connection settings.

7.2.8 Calculation Engine

Finally the `CalcEngine` class brings all the ancillary implementations together. Triggered by its `calculate` method, it starts to evaluate the model. If the evaluation requires using a data source, the value is requested via the `DataSource` interface. In turn the `DataSource` implementations can use the `handleValue` method of the `CalcEngine` class to set the values of a node correctly. Especially for custom operators, the algorithms implemented in the `CalcEngine` class are not sufficient to produce the expected results. Therefore the `CalcSetup` class can be used to execute an alternative calculation. Furthermore the methods `preprocessing` and `postprocessing` offer the ability to alter the model before and after the default implementation is executed. In fact, the calculation can even be prevented. One example of an alternative model processing is the `T39GraphGenerator` class which defines an anonymous inner class extending `CalcSetup` to simulate generated graphs with unspecified operations.

One class in the `engine` package seems to be misplaced: the `VirtualData` class. It solely contains two fields, namely a type specifying the meta type the data refers to, and a data field comprising

virtual data in a table model. Independent from the data sources the `VirtualData` class is used by various classes depicted in the model. The class is essential to calculate the value of simulated values and therefore it was placed in this package.

7.3 Data Models

Two packages have not been considered so far: `meta` and `graph`. Both packages contain data structures to represent core parts of the solution. In the following two sections the details of each structure is described.

7.3.1 Meta Information

Technically speaking the data sources offer different concepts to structure data. While a CSV file allows arbitrary columns and rows with or without a header column, SQL-based data sources require the strict definition of tables with typed columns. In contrast T39 offers types and properties for each type. Properties and types can be constrained using definitions (cf. Matthes et al. 2011). In order to consolidate these concepts and form a common concept for meta information two main entities are defined based on Matthes et al. (2011): `MetaType` and `MetaField`. `MetaType`s refer to a table in a SQL database or to a type in T39. Since CSV files only contain the information of one – potentially large – table, the `MetaType` for this is solely created as "CSV". Each `MetaType` is identified by a unique name and contains `MetaField`s. `MetaField`s are typed; by default the `StringMetaField` subclass is used. `MetaFields` represent a column in a CSV or SQL table or a property of a type respectively type definition in T39. `MetaType`s and `MetaField`s are referred to as meta information hereafter. The structure is used to identify the type of virtual data, to show available fields and values for field value nodes and to define constraints.

7.3.2 Graph

In section 5.2.1 four different node types are described: constants, variables, aggregations and field value nodes. The `graph` package contains these types. Constants are implemented in the class `ScalarNode`, variables are represented by the class `CalculatedScalarNode`, aggregations by the `AggregationNode` class and field value nodes by the `FieldValueNode` class. Constants have been named `ScalarNode` to clarify the difference between a constant in a programmatic context and a single numeric value which is set by the user. The word scalar was chosen, because the value of the node could be multiplied by a set of values which could be considered a vector. To keep the naming concise throughout the implementation, a variable node is coined `CalculatedScalarNode` due to the fact that variables have a different meaning in programming and `CalculatedScalarNode`s are usually calculated based on `ScalarNode`s. Additionally, three more classes can be found in the node hierarchy. A `SetNode` is the general term for nodes representing a set of values. `AbstractInnerNodes` contain the coding for non-leaf nodes of a graph. Finally the `AbstractNode` class contains coding common to all nodes and abstracts for the concrete type of a node. Notably the classes form a specific structure named *Composite Pattern* as introduced by Gamma et al. (1995). An `AbstractInnerNode` might have arbitrary `AbstractNodes` as children. An `AbstractNode` cannot be instantiated but represents all node types. Furthermore objects instantiated from classes not inheriting from `AbstractInnerNode` are leaves in a graph and therefore do not contain children. The tree structure is referenced by a single node, the root node. In combination with a model ID, a name and version as well as the information whether it was generated and whether it is protected against simulation the root node makes up the `CalcModel`.

8 Prototypical Implementation

After all, the users of the implementation will have to decide whether the system is helpful and usable. Therefore the first section is described like a user manual. In the further course of this chapter the implementation details are described from a developer's perspective.

8.1 Frontend

To start the application, the user needs to open a modern browser supporting HTML5 such as Firefox 17 or newer. Enter the address of the webserver with the `/MTCalcEngine` path to navigate to the web application. Depending on the configuration of the webserver and the application, the browser might forward the user to the more secure HTTPS protocol and thus to a different port. The authentication of the user is not supported using certificates so far; however, it is recommended to install a certificate issued by a Certificate Authority (CA) on the webserver to prevent warnings or even the rejection of access by the browser. Finally a secure connection with an anonymous session has been established and the login page is displayed.

Technically speaking the HTML5, CSS and JavaScript content is delivered from the `WebContent` directory of the Web Application Archive (WAR) that is deployed on the Java Web Application Server (WAS). In addition, several Servlets like `de.mattimaier.mt.client.CalcServer` provide services to deliver data requested from the application. The main configuration of the application is located under `WebContent/WEB-INF/web.xml`. Besides the `WEB-INF` directory contains documentation, configuration, data, library and model resources.

In the following sections each screen of the application is described in further detail beginning with the login screen. Once the user has logged in, the model selection screen is shown. From there, the user can navigate to the modeller screen or the virtual data editor. The model selection and the modeller screen both share the model selection sub-screen, an overlay window displaying the models available for the user. Other overlay windows are the T39 Graph Generator and CSV Data Upload windows, both represented by plugins. Illustration 16 summarizes this navigation structure.

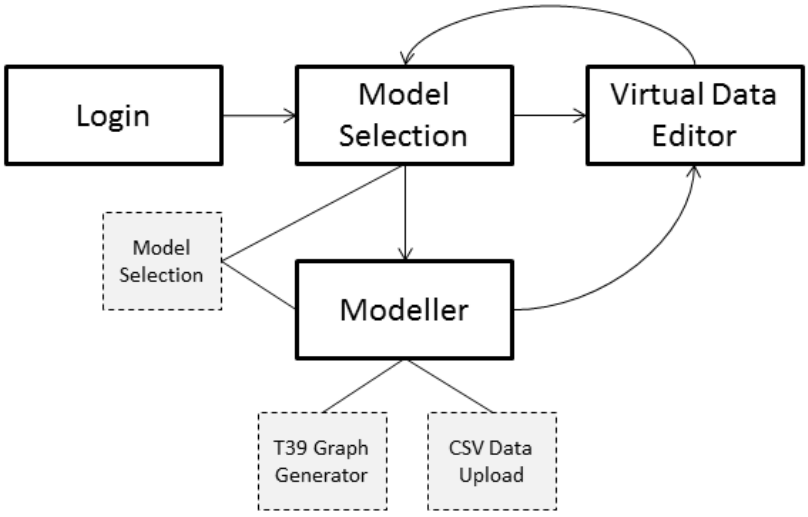

Illustration 16: Navigation Structure (own diagram)

8.1.1 Login

First, the user is greeted with a form to provide his/her credentials as username and password as shown in Illustration 17 below. Depending on the setup of the application, the username and password can be from another application, for example T39 credentials, individual credentials for each physical person or a commonly used username and password. By hitting the return key or by clicking on the Login button a JavaScript function is called which sends the username and password as HTTP POST form parameters to the `CalcServer/authenticate` service. There it is processed as described in section 8.2 and either success or an error message is returned. If the user has successfully logged in he/she is forwarded to the model selection screen. Otherwise the error message provided by the server is shown and the user can try again.

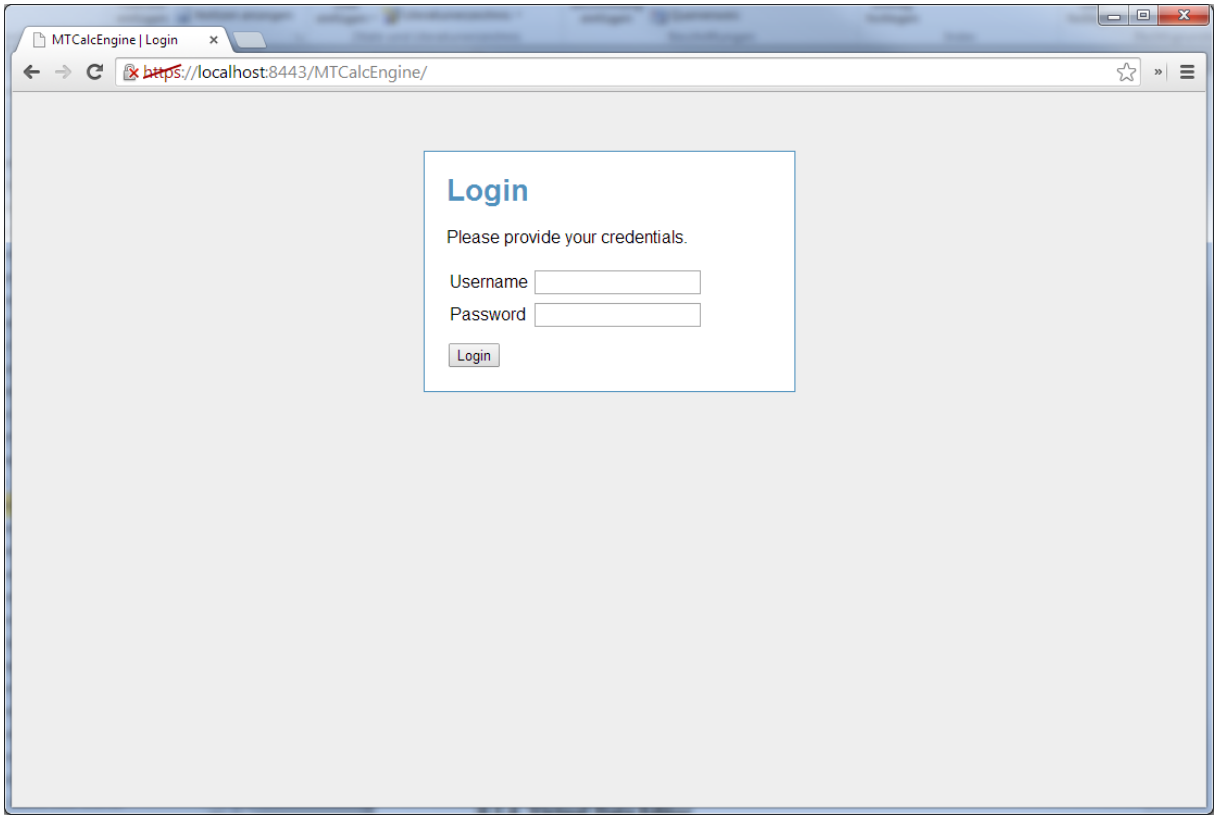

Illustration 17: Login Screen

8.1.2 Model Selection

In case someone requests the model selection or any other screen except for the login page and he/she is not logged in, then the application will automatically forward him/her to the login page. After successful login, he/she is forwarded to the model selection screen. Mainly the screen is blank with the exception of the top left corner where two icons are displayed: The left icon refers to the model selection and creation of new models, the right icon links to the virtual data editor. Technically speaking this screen is identical to the modeller screen with all model-specific controls hidden.

When the user clicks on the model selection icon an overlay window showing the available models will open (cf. Illustration 18). If no models are available only the form to create a new model is displayed. This form is displayed underneath the model list when models are present. Models can be deleted by pressing the trash can on the end of the model. A model can be selected and opened by clicking on its name. New models are created by entering a suitable name and clicking on the “Create Model” button. By doing that the server is asked to create a model with only one node called “root”, the version number one and a model ID. This identifier is created from the model name reduced by

special characters and white spaces. If a model with the same ID already exists, a counter will append an incrementing number at the end of the model ID. For example, if a model with the name “Example Model” and thus its ID “examplemodel” already exists, then the second model with the same name will have the ID “examplemodel0”. Before the server returns the newly created model to the client, it is persisted in the model storage (cf. section 8.3).

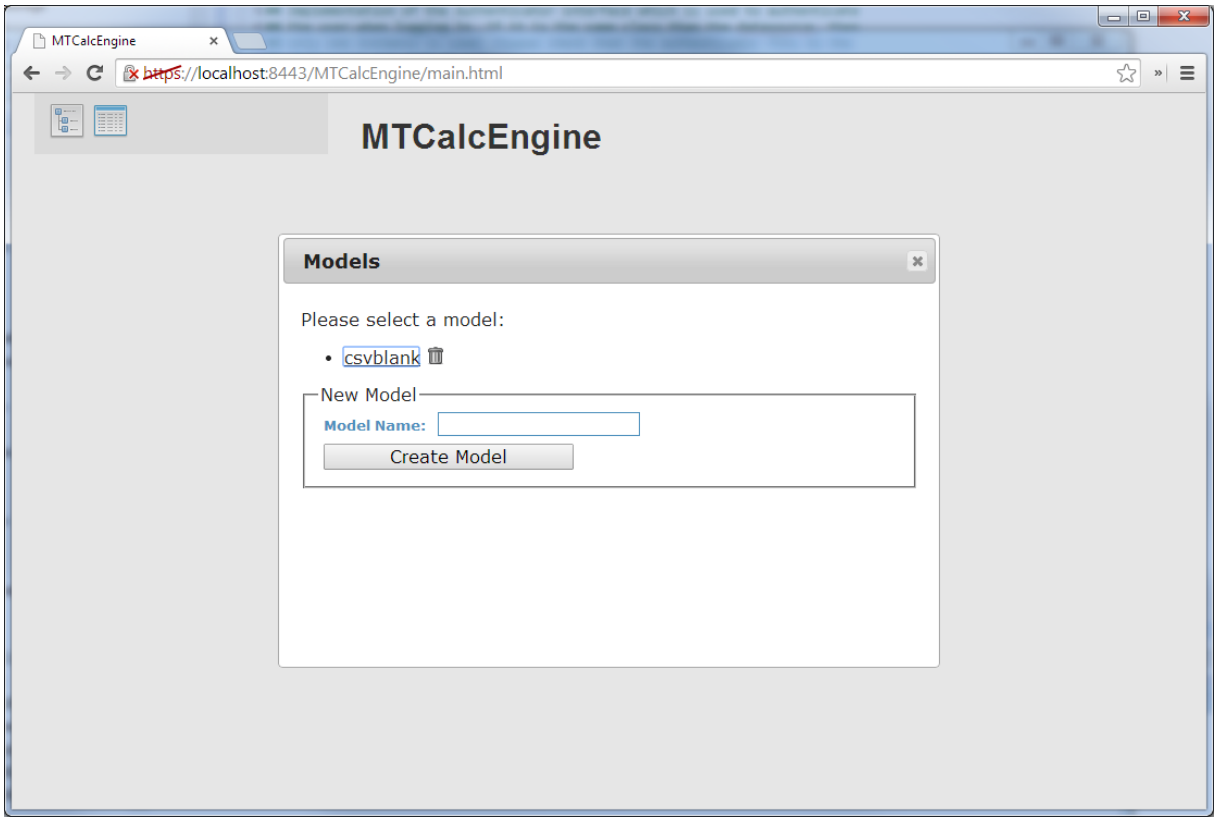

Illustration 18: Model Selection Screen

8.1.3 Modeller

8.1.3.1 Toolbar

An overview of the modeller is displayed in Illustration 20. The modeller screen has toolbar buttons in the top left corner of the screen similar to the model selection screen. In the first row one can find various buttons to perform actions on the model or temporarily save its state. While in the row below, the most needed functions to save and run a model are placed in front of the optional plugin buttons.

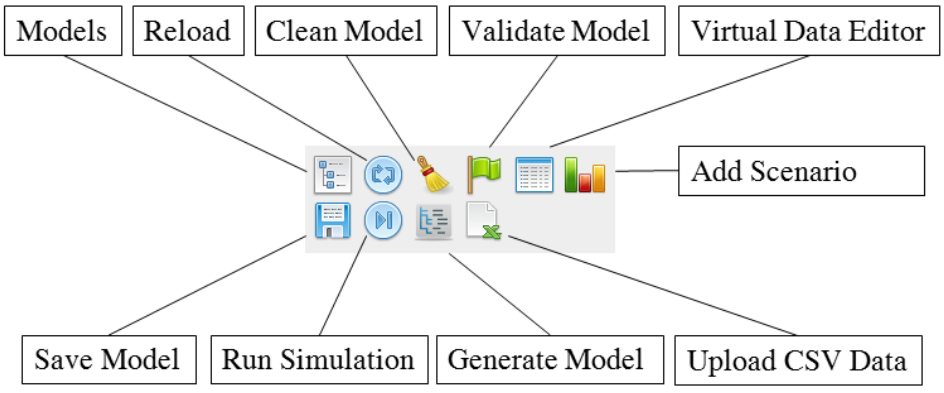

Illustration 19: Modeller Toolbar Description

On the far left of the first row, the “Models” button shows all available models for the user. When the icon is clicked, an overlay window as displayed in Illustration 18 opens. The user can choose the model which he/she likes to run or edit. All actions described hereafter are performed on the selected model. Beside this button on the right is a feature to reload the model as it is stored on the server. Essentially this button has the same effect than reloading the URL via the browser’s reload button, except that it checks if the current model has been edited and whether the user wants to discard these changes. Next in the upper row is the “Clean Model” button. When it is clicked, all simulated values and deltas are removed from the model and all values are cleared, except for the values set on scalar nodes. This feature is convenient to reset the model to an initial state.

Next to this feature is the “Validate Model” button. By clicking on it, the currently shown model is validated against a set of rules. For example, a leaf node must not have children; scalar nodes must have a value; aggregation nodes must have children; multiple children of a node need to be set against each other with operators other than the *NoneOperator* and so on. After the validation, a message showing success is displayed if the model is valid according to the rules. Otherwise the message comprises the first node where the model is not valid and the reason why it is not valid. With this information at hand, the user can correct his/her model and validate it again.

Second from right is the icon which links to the virtual data editor (see section below). Beside this icon is the “Add scenario” icon. A click on this icon opens a new dialog asking for a scenario name. When a scenario is added, the current model is saved with the given name in the browser’s memory. Furthermore the scenario’s name is displayed below the node details. A click on a scenario shows the saved model. In addition, each node’s details box contains a bar chart to compare the scenarios on the node. Using this feature, the combined impact of various influencing factors can be compared based on scenarios. The visualization of the scenario’s impact on each node lets a user assess the results faster, because the largest and smallest bar can be captured faster than the numerical values. Still each bar has the numerical values written on top of it to asses small differences as well.

On the left of the second row, the “Save Model” button is displayed. As the reader might guess its behaviour, this button saves the current model including the simulated values and deltas on the server. After a model is saved, a simulation run can be initiated or the model can be reloaded. A simulation run is executed when the icon beside the save icon is clicked. If the model has not been saved, a message reminds the user to do so. Depending on the validation level (cf. page 35) the model is validated. In case the validation fails and the validation level is set to “blocking”, the validation message is displayed and the simulation is not started until the user has resolved the issue. When the validation message is set to “warning” in the same case, the message is issued, but the simulation is executed. In case the validation level is set to “ignore”, no validation message is shown and the simulation is started right away.

Both buttons displayed to the right of the simulation run button in Illustration 19 are plugin dependent buttons. The first button starts the T39 Graph Generator plugin while the second button opens a window to upload a CSV formatted file. In combination, these buttons should not be activated as the compatibility matrix in the appendix shows. The details of each plugin are described in section 8.6.

Below the buttons is a field to search nodes and filter the graph. Especially in large models, searching for a specific node can be tedious. Therefore the search field provides a simple method to enter the name or parts of the name of a node to show only the node and the path from the root to this node. The filter/search is triggered when the value of the input field changes or the magnifier is clicked.

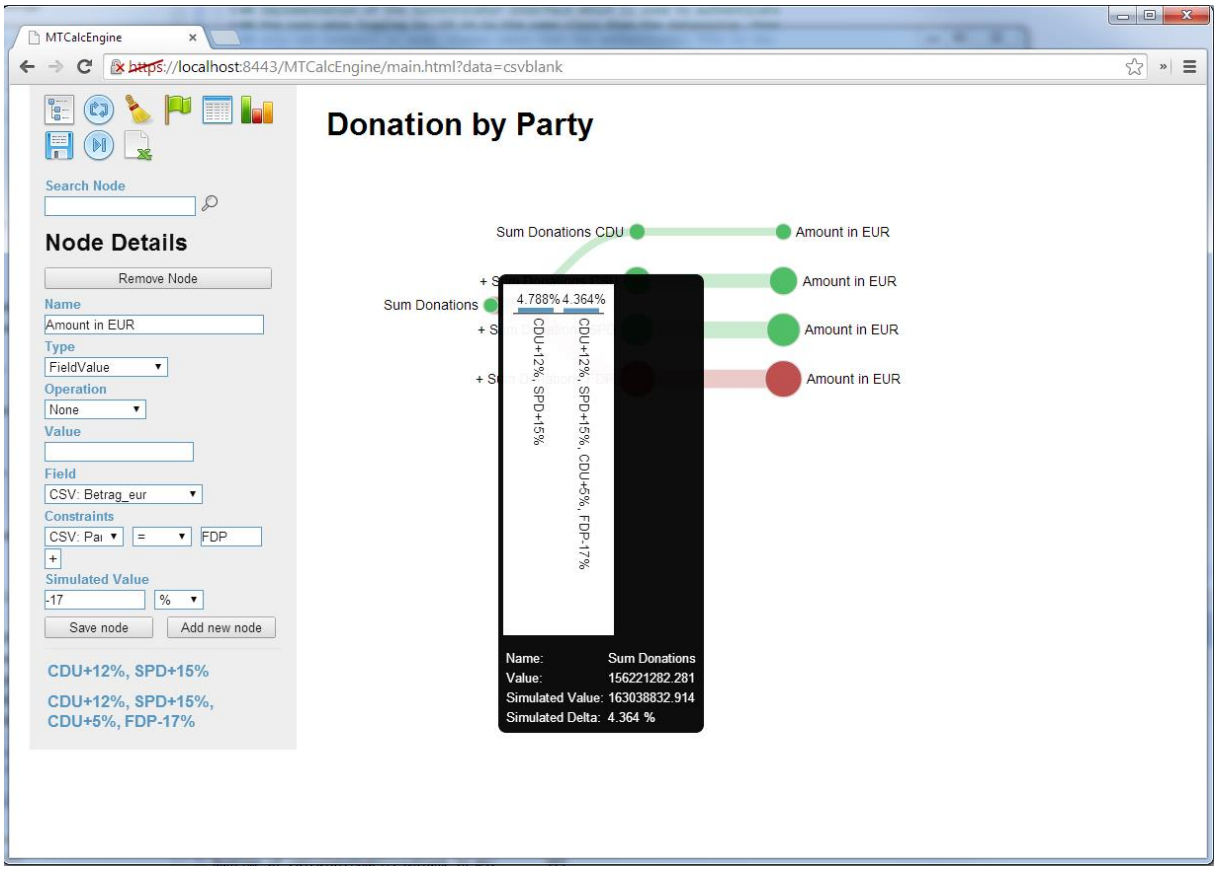

Illustration 20: Modeller with Simulation Result

8.1.3.2 Building a Model

As mentioned above, each model starts with a single node called "root". This node is typed as a calculated scalar node by default. A click on the node fills the "Node Details" section on the left of the screen. Depending on the type of the node, the fields in this section vary. A scalar node simply shows the node's name, it's "scalar" type, the operation with which it is set against other siblings, the value and the simulated value. All these fields can be edited. When a calculated scalar node is selected, the value cannot be edited and an additional non-editable field "Formula" is shown. Therein the children's names are concatenated in their order with the operations they are set against each other. The value of the calculated scalar node is determined according to this formula. Furthermore this field helps to identify missing operators for child nodes. In contrast, the formula field is replaced by a dropdown list to select the aggregation operation when an aggregation node is selected. Aggregation operations are: sum, product, count, min and max. The sum-aggregation adds all field values, the product-aggregation multiplies all field values, the count-aggregation simply counts all children and the min/max-aggregation selects the minimum/maximum of the field values. Field value nodes do not comprise an aggregation thus this field is not shown. However, the gap is filled with two more fields: A dropdown box to select the meta type and field this field value node refers to and the constraints. The dropdown box contains all meta types and their fields. A single entry has to be selected. By default only a plus-icon shows the possibility to add constraints to a field value node. Each constraint has a meta field it refers to, a comparator – for example, equals/not equals/greater than/smaller than – and a constraint value. An arbitrary number of constraints can be added to a node. To remove a constraint, the value is left empty. After the information for a node is inserted, the node can be saved with the "Save node" button. Alternatively the "Add new node" button creates a new node with the current information of the fields. Selecting a node and clicking on "Add new node" therefore copies a node.

All nodes added to the graph are inserted as children of the root node. From there, a node can be dragged on top of its new parent. While dragging a node, all other nodes receive a red circle around them to display that within this circle the dragged node is assigned to this parent. Adding the dragged node to a new parent inserts the node as the last child. With this rule, nodes can be sorted. For example, if the first node should become the last node, simply drag the node onto its parent again. This way the node is inserted at the end of the parent's children list. After all nodes are arranged, the user can validate his/her model for correctness. Thereafter the model must be saved. Only then it is written to the server's storage. A common mistake is that information on nodes are changed and allegedly saved, but the model has not been saved yet. For this reason, the save-before-simulate rule as described above was introduced.

8.1.3.3 Simulation Result Display

During the simulation a grey overlay with a white font reading "Please wait…" covers the screen to prevent input. After the simulation has finished, the cover is lifted to reveal the results. In case no simulated value/delta is set and no virtual data influenced the result, the graph is displayed as if only modelled. However, when hovering over a node, the value of it is displayed in a dark box beside the node. In case simulated values/deltas are set, the node representing the influencing factor is colored green when the deviation is greater than zero (positive) or red when it is less than zero (negative). Furthermore the size of the circle reflects the simulated delta. For example, a simulated delta of 5% results in a circle increase of 5px. The link between the influencing factor and its parent is colored like the node and its stroke width is also increased similarly. Each node is colored and sized in accordance with this scheme. When the simulated delta exceeds an absolute value of 20%, the size of a node increases so far that it might reach other nodes. To prevent this, sibling nodes are moved depending on the size of the large node. In general a result looks like the model displayed in Illustration 20. Without having to look at the numbers, a user can clearly see that the root has been affected positively by the changes to the influencing factors, although one of the influencing factors seems comparably large. Consequently the large red node has little impact on the root node.

When the layout of the screen was developed, the goal was to give the user as much space as possible to create, edit and view the model. Thus all controls outside the model are outlined on a single strip overlaying the model with slightly transparent background. Still the modelling space is not enough for small screens or large models. Therefore the user can click on the whitespace around a model and relocate the model. This way he/she can pan the model. By using the scroll wheel on the mouse, it is also possible to zoom in and out on the model. In combination, both features add to the interactive feeling of the model.

8.1.4 Virtual Data Editor

Adding virtual data to the original data enables to model scenarios such as what happens if we add another application to the application landscape or how much is our gross margin affected when we allow this group to stay in the hotel for the weekend. All virtual data entries are based on meta types and fields. Therefore the user selects a type first as displayed in Illustration 21. When the type is selected, the main table is automatically updated with the fields defined in the meta information and the virtual data that is already set for this type. In case there is no virtual data set, only the header is displayed. By clicking on "Get a preview" the backend system is queried for ten or less random entries of this type. The preview data helps the user to insert his/her virtual data in a format compliant with the original data. A click on "Add row" adds a row to the bottom of the table. The row can be removed if the "delete" button in the delete column of the table is pressed. Once the user is finished editing virtual data, "Save Virtual Data" sets the virtual data to be considered in all simulations in the current session from now on. "Save Virtual Data" must be executed to save the current virtual data. If a type is

changed without clicking on save, the table is refreshed and the inputs are discarded. Finally, when the user finished editing the virtual data, the "Go to Model Designer" brings the user back to the model selection screen as described in the beginning of this chapter.

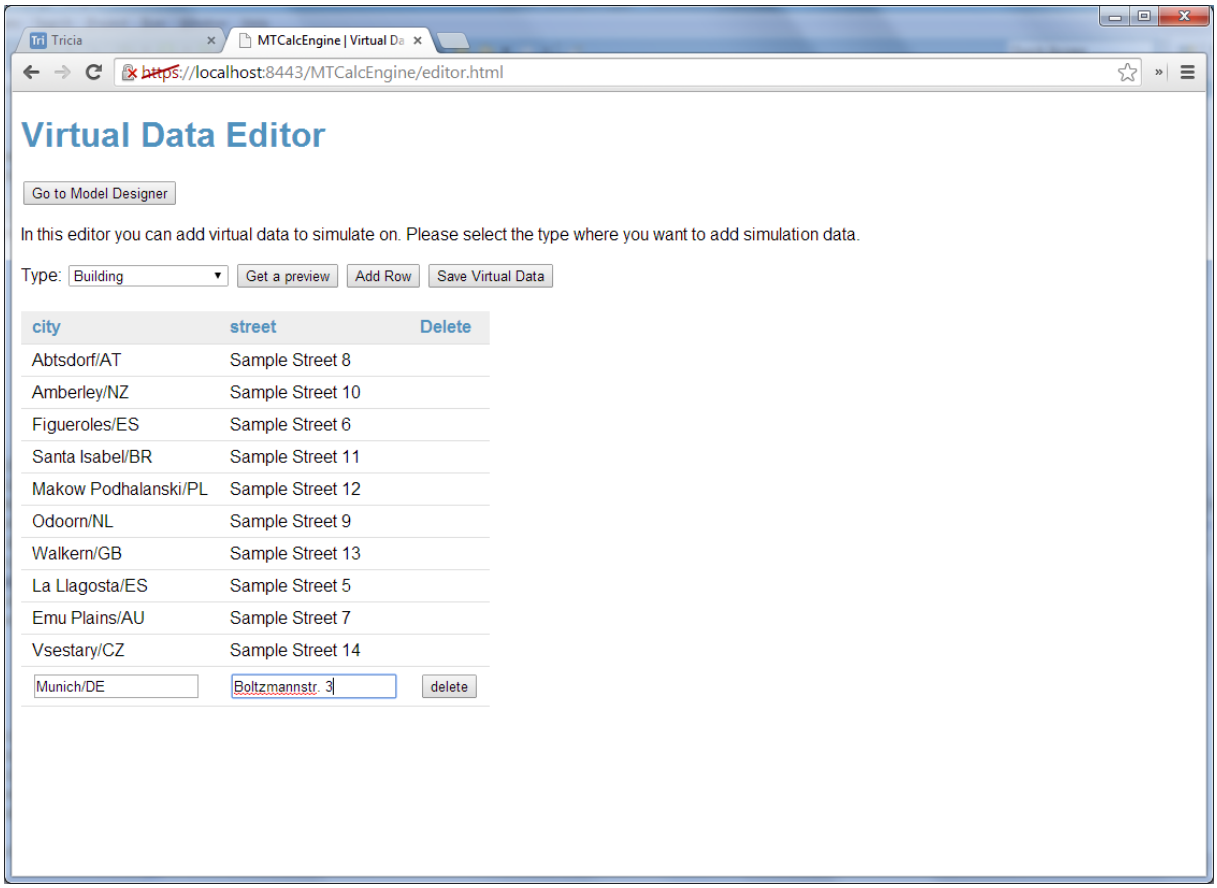

Illustration 21: Virtual Data Editor

8.2 Authenticator

As described in section 7.2.5 authenticators are used to authenticate the user against the application and the backend. When the user requests to access the application, he/she is forwarded to the `login.html` file by default. By submitting the login form, the username and password is sent to the server with an HTTPS POST request. There the application will read the configuration and initialize the `Authenticator` interface implementation. After the initialization, the method `authenticate(username, password)` is invoked. In the scope of this thesis two `Authenticator` implementations were realized: A `SimpleAuth` implementation with hardcoded credentials and the `T39DataSource`. In case the `SimpleAuth` implementation is configured, the username and password are checked against the credentials and true is returned when the username and password matches. Furthermore the `isAuthenticated` flag is set to true.

When the `T39DataSource` is configured as an authenticator, a new session between the server process and the T39 backend system is created by requesting a session token. The result of this request is a JSON formatted object with a `nounce`, potentially-null `userId`, a session ID and a session cookie name. The client calculates a digest based on the `nounce`, username and password in case the `userId` is not set (null). This digest is submitted to the T39 backend again and on success a valid `userId` is returned. In case the credentials were incorrect an error message is returned. Finally the T39DataSource also sets the `isAuthenticated` flag to indicate the successful authentication for

further requests. The calls and their replies are displayed in Illustration 21 as an UML sequence diagram.

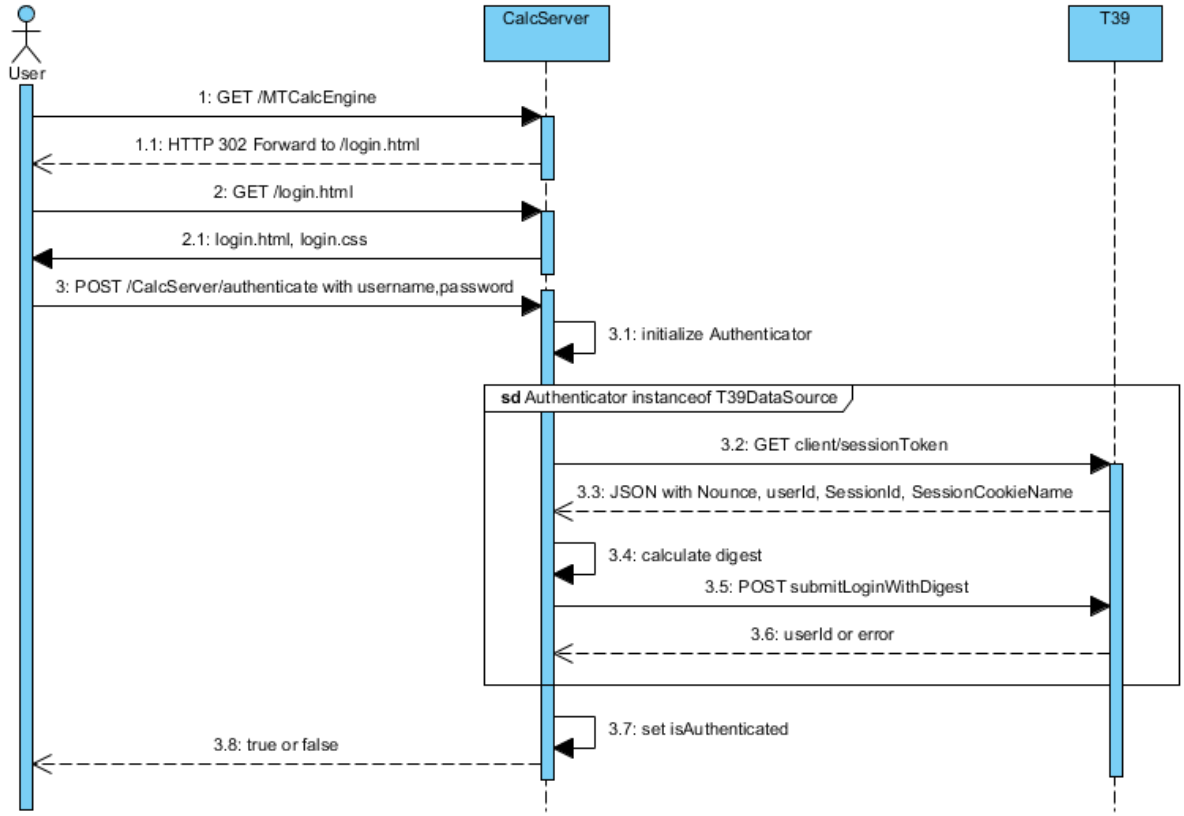

Illustration 22: Authentication Procedure (own diagram)

Once the user has successfully authenticated him- or herself, a new Java session is instantiated to prevent session fixation attacks. Another measure to increase the security on a session level would have been the limitation of the session duration. Due to the fact that simulations can run for an unforeseeable time, it was decided not to limit the session duration.

8.3 Model Storage

In previous sections the model storage was referred to as a place where models can be stored and retrieved. This is essentially the purpose of the model storage components in this implementation. The classes handling the storage implement the `ModelStorage` interface. The interface comprises methods to store a model based on its `CalcModel` representation or the exchange format – here JSON. Models are identified by a unique identifier (model ID). Additionally a list of all model ids for the given user and data source can be retrieved using the interface. Once a model is selected from the list it can be retrieved using its model ID. Finally the model ID can be used to delete the model from the storage, too.

Each `ModelStorage` implementation receives references to work with. For example, the `CalcGraphFactory` implementation is given in order to use it to serialize and deserialize models. Furthermore the context path of the application is given to read configuration files and eventually store models on the file system (see section below). All models are stored user and data source dependent. The reason therefore is that models refer to the meta information of a concrete data source. If the models could be used independent from the data source the application is set up with, a simulation will lead to errors that are not fixable by the user and consequently limit the usability. Thus storing models

data source dependent removes the need to identify the model's data source from the name and having to know which data source is connected at the moment. In addition, models are stored user dependent so users do not interfere with others. Furthermore nobody can edit another user's model. Although this limits collaboration, it improves the security.

8.3.1 File System Storage

The string based JSON format is predestined to be stored in a plain text file on the file system of the webserver. Therefore the `FilesystemStorage` class was the first implementation of the `ModelStorage` interface. Models are stored in the `WEB-INF/modelstore` directory with a name in the following format:

```
<datasource>_<reduced username>_<model id>
```

Where the data source name is the simple class name of the `DataSource` interface implementation and the reduced username is the username without special characters and spaces. This implementation assigns the user "max mustermann" and "maxmustermann" the same username, but this fact is ignored in favour of the simplicity and maintainability of this approach. Depending on the authenticator, the user ID could be used instead of the username. By that the implementation would deliver unique results. Finally the JSON structure is written to the file and the output stream is closed.

Another task of the model storage implementation is the creation of new model IDs. The combination of model ID, data source and username must be unique across the storage system. To prevent using the same name twice, the model ID checks the file system for existing files with the potentially same name. If a file with the same name is found, a numeric value is added to the name. For example, if a model with the ID "example" already exists, the second model with the name "example" gets the ID "example0", then "example1" and so on. Model name and model ID are not identical, but often the model ID is only the model name reduced by special characters and whitespaces.

Based on this scheme, retrieving models from the file system is straight forward: At first the filename where the model is stored is built, then the file is read and finally the `CalcGraphFactory` is utilized to build a `CalcModel`. To retrieve all model IDs stored in the file system, an iterator loops over the `modelstore`-folder and parses the model IDs. These are then returned as a list of strings.

8.3.2 MongoDB Storage

According to the documentation "MongoDB stores data in JSON documents (which we serialize to BSON)." (MongoDB, Inc. 2014) Therefore JSON documents can be stored, queried and retrieved using MongoDB. This native support for JSON documents was the primary reason for choosing this database management system as an alternative to file system storage.

In order to perform any action on a MongoDB via a driver, the application has to connect using a language specific driver first (here the Java driver in version 2.12.2 is used). The information to which server and on which port to connect, the user credentials to authenticate with as well as the database to use are stored in a configuration file. This configuration file is read when an action is requested that requires access to the database. If the application is not yet connected to the MongoDB a connection with the configured parameters is established and the "models" collection is retrieved and assigned to a private field of the implementation class. All operations necessary to be performed on the database are invoked on the collection object.

One of the method implementations not requiring access to the database is the creation of the model ID. The implementation is analog to the one described in the section above. In contrast to the file

system storage implementation, the MongoDB `ModelStorage` implementation saves models with their complete username and datasource name. In order to imply this context on all queries and data manipulation operations, it was refactored to a separate method called `addCredentials`. One of the use cases is retrieving a model based on its ID from the database. Therefore a query – also in JSON format – is sent to the database with one key called "model.id" and the model ID as a value. When the model is found, the result is returned and the `CalcModel` is created by the `CalcGraphFactory`. Storing a model in the collection is similar, except that the implementation needs to check whether to insert or update the model. While on the file system the file is simply overwritten, the implementation needs to call either update or insert on the collection reference. To decide which method to call, a query is initiated to search models with the given ID. If no model is found the insert method is called, otherwise the update method is invoked. All models can be retrieved by a simple empty query object. To return a list of the model IDs, only those are extracted from the result and added to the result.

8.4 Calculation Engine

The core of the application is the calculation engine. It is built to perform calculations generically. Only when needed, the calculation engine asks the data source to supply information or partial results. As the core of the implementation, the `CalcEngine` class does not implement any interfaces. However, the algorithm can be altered with a different setup (cf. section 8.4.2). By default the method `calculate(CalcModel model)` is called when the simulation run is triggered. Besides handling the specific setup, this method essentially calls the recursive method `calcNode(AbstractNode node)` to calculate each node separately starting with the model's root node. In turn, `calcNode` checks the type of the node and distributes the tasks to other methods individual to the node's type. Calculated scalar nodes are processed in the engine application itself, because the implementation should be the same across all data sources. Aggregation nodes and field value nodes are redirected to be calculated by the configured data source and scalar nodes do not have to be calculated, thus their values are handled internally. Handling a node's values internally means that the calculated value based on the original data is set against the simulated value and simulated delta as described in equation (3) in section 5.1. To distinguish between simulated values and simulated deltas set by the user and the application, the flag `manuallySet` in the `AbstractNode` class is used. When the flag shows true, the simulated delta or value has been set manually, otherwise the value/delta was set by the application.

To fulfill equation (5) from section 5.2.1 aggregation and calculated scalar nodes have to consider their children. While aggregation node values are computed by the data source implementation, the calculated scalar node sets its children against each other using the exp4j framework. The author decided not to implement the evaluation of expressions according to all mathematical rules himself to reduce potential bugs and security vulnerabilities. Furthermore the use of a framework increases the readability of the code and thus contributes positively to the maintainability of the application.

Since a child might be another inner node, the evaluation of each child is performed before their values are set against each other. To increase the calculation performance, the evaluation of each child is executed in a separate thread. Before the children's values are combined to an expression, all threads are joined. After the expression is evaluated for the value and the simulated value, the `handleValue` method is called for the calculated scalar node as well. By multithreading the computation of a calculated scalar node recursively many threads are created. Thus a data source might retrieve multiple requests at once. This requires the data source to execute all requests atomically and isolated from each other. When multiprocessor hardware or a distributed environment

operates the application, the processing resources can be used in parallel as much as possible and thus improve the application's performance, reduce the user's waiting times and increase the responsiveness and thus usability of the application.

8.4.1 Operators

Operators are used to set sibling nodes against each other. All operators extend the abstract class Operator that defines that two operators are equal when their class names are the same. As a result all instances of the same class are equal. Additionally the abstract class has three abstract methods:

- `BigDecimal apply(BigDecimal a, BigDecimal b)`
- `String getCaption()`
- `String getOperationSign()`

These methods are overwritten by the subclasses listed in the Operator-column in Table 7. The `apply` method sets two values against each other and is called by the calculation engine. The `getCaption` method returns the name of the operator as displayed to the user while the `getOperationSign` method contains the operator sign to be used by the exp4j or data source. One of the data sources making use of the operation sign is the `JdbcDataSource`. Operators implementing the `SQLOperator` interface return the operation sign through the `getSQLOperation` method. Separating these two methods has the background that individual operations defined on the database could be different in the exp4j framework and the SQL implementation. Therefore the implementation was designed to support both.

Table 7: Operator Implementations

Operator	Caption	Sign	Apply-implementation: a (operation) b	Implements SQLOperator?
`NoneOperator`	None		When a is null or zero, return b otherwise a	No
`CustomOperator`	Custom	o	Return the other if one is null, otherwise return the average	No
`AdditionOperator`	Addition	+	a+b	Yes
`SubtractionOperator`	Subtraction	-	a-b	Yes
`MultiplicationOperator`	Multiplication	*	a*b	Yes
`DivisionOperator`	Division	/	a/b	Yes

As mentioned above Table 7 shows the default operators and their implementation details. The NoneOperator is meant to be used as the first operator in a formula, for example, when a calculated scalar node has three children *a*, *b* and *c* then node *a* should have the `NoneOperator` set. Usually the `NoneOperator` is never applied, but in case it is, the second parameter is returned in case the first is null or zero, otherwise the first is returned. In the next row, the `CustomerOperator` represents custom operations. Neither the `CustomOperator`, nor the `NoneOperator` implement the `SQLOperator` interface. However, all other operators implement the `SQLOperator` interface. The `CustomOperator` has as a default implementation the average of both *a* and *b* as a result. This behaviour should be overwritten by a custom operator inheriting from the `CustomOperator` class. The `AdditionOperator`, `SubtractionOperator`, `MultiplicationOperator` and `DivisionOperator` are implemented as a reader expects it. For example, when the `AdditionOperator` is applied to two nodes, both nodes are added and the result is returned by the

`apply` method. The user chooses the `AdditionOperator` when selecting "Addition" in the operator dropdown selection box in the node details section on the UI. And the plus sign is displayed in the formula of a calculated scalar node parent as well as in front of the node's name in the graph. Finally the nodes are set against each other using this plus sign when evaluated as an SQL or exp4J expression. Similar applies to all remaining operators, except for the division operator which will throw an exception when the second argument is null or zero.

8.4.2 Calculation Engine Setup

Although the algorithm to compute the values of a node is held as generic as possible, it can be necessary to perform preparing or post processing operations on the model. Furthermore some data sources should be empowered to prevent the computation and to specify an alternative implementation. For this purpose the `CalcSetup` class offers a set of methods that can be overwritten. By default the setup does not perform any pre or post processing operations. It does not prevent the execution or specify an alternative implementation. One example of a setup overwriting the alternative calculation method is the `T39GraphGenerator` class which contains an anonymous implementation of an individual `CalcSetup`.

The setup is applied in the `CalcEngine`'s `calculate` method. Therein the `preprocessing` method of the setup class is executed before the default algorithm and the `postprocessing` method after the algorithm has finished. The execution of the default algorithm is prevented when the setup's `preventCalculation` returns true. If the `runAltCalculation` method also returns true, the `alternativeCalculation` method is executed instead of the default algorithm. Therein a custom implementation of the calculation engine can be specified. To enhance the reusability of the application, most of the methods from the `CalcEngine` class are public and should be utilized when implementing a custom algorithm.

8.5 Data Sources

In the previous section it is mentioned that the calculation engine passes the evaluation of field value and aggregation nodes to data sources. During the design phase, loading all data from the data source to the application server was discussed, but soon dismissed, because burdening large data sets on the memory of the webserver increases the resource requirements drastically. In addition, the transfer process would take too much time thus reducing the responsiveness. To resolve the problem, the aggregation operation was pushed closer to the data. In the following sections the `DataSource` interface is described first, then each data source implementation and its connection to the backend system is outlined.

8.5.1 DataSource Interface

All data source implementations are encapsulated by a single interface called `DataSource`. The first part of the interface comprises methods to set the context, for example, the path to the application resources which is needed for configuration files. Furthermore the authenticator and the engine configuration reference are given. Before other methods are invoked on the interface, the `initDataSource(CalcModel model, CalcEngine engine)` method is called. Therein the data source is initialized with the model to perform model-dependent operations on and the engine which performs the computations. The core consist of only six methods: One to get meta information from the data source, two to calculate the aggregation and field value nodes, two to get and set virtual data and finally one to retrieve preview data. Implementing this rather simple interface is sufficient to serve as a data source for the calculation engine. Since the `DataSource` abstraction was designed as

an interface, some data sources implement additional interfaces to serve as an `Authenticator` or `Plugin` at the same time.

8.5.2 JDBC-based Data Sources

In section 6.2 the relevance of relational database management systems was depicted. To take this consideration into account, a generic JDBC-based data source was developed. Furthermore a generic JDBC connector provides basic functionalities to connect to a relational database using the JDBC driver of the vendor. Both the JDBC data source represented by the `JdbcDataSource` class and the JDBC connector represented by the `JdbcConnector` class are extended to suit a concrete database's SQL dialect and connection procedures. In the scope of this thesis the H2 and HyperSQL database connectors are described hereafter.

8.5.2.1 JDBC Connectors

Connecting via JDBC to a RDBMS is straight forward: The driver is loaded and a connection is established using a connection string, username and password. In order not to hardcode the credentials, a configuration file named `jdbc.properties` located in the `WEB-INF/config` directory is employed. Therein the "URL" parameter denotes the connection string and the "USER" and "PASSWORD" parameters the obvious. To support both, the connection reading from a configuration file and hardcoded credentials, two constructors are offered by the `JdbcConnector` class. Other than the protected getters and setters for the fields, only one publically available method is essential in this class. Its signature is: `Connection connection() throws SQLException, UninitializedException`. By default it will call the static method `getConnection` on the `DriverManager` to retrieve a `Connection` object and throw an exception when the connection parameters are invalid, the database system is not available or another error occurs.

In case of the H2 database connector, the connector loads the H2 JDBC driver from the h2-1.3.175.jar library. Additionally the connection method is overwritten to optionally return a pooled connection. During the simulation tests within the scope of this thesis the H2 data revealed an insignificantly better performance when queried simultaneously over pooled connections than with a single connection. Furthermore the pool size must be fixed and lead to connection errors due to too much load on the database. To resolve these issues the pooled connection is turned off by default, but can be activated when the `setUseConnectionPool` method is called with the parameter value true.

In contrast, the HyperSQL connector uses a pooled connection. The driver and database seem to handle the requests well. In addition, the performance of simultaneously queried connections was much better than sequential queries. Therefore the author decided not to implement opening only a single connection. Connections to a HyperSQL database are established using the hsqldb.jar JDBC driver. Despite both H2 and HyperSQL connector's connection methods return an object with the same `java.sql.Connection` interface, the use of it slightly differs. Therefore the `HyperSQLDataSource` and the `H2DataSource` differ as outlined in the section below.

8.5.2.2 JDBC Data Source

The abstract `JdbcDataSource` class cannot be instantiated, but it still provides a constructor with no parameters. Thereby users of the subclasses can use the default constructor to create an instance of the data source. To use an instance of the connector internally the method `con()` returns an instance of an implementation of the `java.sql.Connector` interface. Therefore it initialized the connector if that has not been done so far and it calls the `connection` method upon it. At this point the

subclasses differ the first time. Each subclass has to overwrite the method `getConnector` method and return an instance of the `JdbcConnector` class or one of its subclasses.

In essence the JDBC data source implements the `calcAggregationNode` method. The implementation of the `calcFieldValue` method is less important since field value nodes are always children of aggregation nodes, thus can be calculated in their context. To retrieve the value and the simulated value of an aggregation node, all children are evaluated first and recursively. In case the child is not a field value node, the value of a child is used to compute the value of the node while a child's simulated value is used to compute the node's simulated value. If the child is a field value node, a view is created. This view has two columns and one row; therefore two values as a result – as with the others one value and one simulated value. The SQL select statement is generated by the application and comprises the node's field which refers to a column in a table, eventually a join condition if the table is not the same than the table of the other aggregation node's children and all constraints in the where-clause. If tables need to be joined together, the foreign keys respectively the defined references of the tables are regarded and the first value that enables to join the tables is used as a join condition. Thereby a user does not have to specify join-conditions; rather the application automatically joins tables. Constraints are and-linked and the type of the value (and thus its use in the SQL statement) is taken from the meta information. In addition, virtual data is added to be used to calculate the simulated by a union-construct. Having created the SQL statement, it is not possible to use a more secure prepared statement anymore, thus all values are validated before by the `CalcGraphFactory`. As a result the computation of one aggregated value can comprise a hierarchy of nested views. In the end the root of the hierarchy is queried with a simple select to retrieve the value and the simulated value. To keep all views in sync, the each view has a name dependent on the node id, its model id, the model's version and the user id. In case the user changes the model, the views have to be rebuilt and the old views can be dropped. In case the same model is executed more than once, the views are not rebuilt; the aggregation node's select-query is just executed again. By using non-materialized views, some database query-optimizers can identify and optimize the statement as much as possible and thus reduce the response time.

Beside the calculation of aggregation nodes, handling virtual data is an essential part of the implementation. In general virtual data is stored in a separate table with the same name than its original, but with a prefix to denote it as virtual data and to separate it from other production data. This is especially important since virtual data should not be regarded by any other application. This is also the reason why the structure of the original table is copied, but without references. In case the copy had references, other applications could follow the reference and refer to virtual data that should not be referred to. As mentioned above, the virtual data is united with the original data only for the purpose of computing the simulated value of a node. When united, the references and joins hold true also for the virtual data. In case no virtual data is defined, the virtual data table could be deleted and the views could be updated. However, in H2 version 1.3.175 a bug (cf. H2 Open-Source Project 2014) prevents the deletion of a virtual table since the complete database would be corrupted, even when all views are deleted at first. The author realized that queries considering virtual data are much slower than queries without it, thus the deletion of this unnecessary table would improve the performance further. In case future versions fixed this bug, the `enableVirtualTableRemoval` method in the `H2DataSource` class can return true instead of false at the moment. Other than that two more differences between the H2 and the HyperSQL JDBC usage arose: The cursor on the result set from H2 is moved to the first row with the `first`-method, but on a result set from HyperSQL it is moved with the `next` method. Furthermore the "create or replace view" statement which worked with H2 did not work in H2, thus it had to be replaced with "create view" in HyperSQL. All these minor changes are reflected in the subclasses of `JdbcDataSource`.

8.5.3 CSV-based Data Sources

Another table-oriented data source is a CSV file. As the acronym states, it consists of values separated by comma. Rows are separated by a newline character. Usually the first line of a CSV file contains the names of each column. This line is referred to as the header hereafter. In the `CsvDataSource` class, the `initDataSource` method loads the configuration file `csvsource.properties` in the `WEB-INF/config` folder. The configuration file specifies the format of the CSV file. For example, if a header is present, which character is used to separate values and the indices of the numeric columns. Although the format is called comma-separated, often values are semicolon or tab-separated. Thus it is necessary to specify the separation character. Since the CSV file format does not provide the ability to specify the type of a column, the indices of the columns with numeric values should be configured as a comma separated list in the configuration file. Finally the name of the file to read the table-structured data from is stated in the "filename" parameter. A reader might ask why a simple csv file on the webserver is interesting to simulate on and why the user cannot upload it him/herself. At first another application could replace the file periodically with new data, for example. Additionally the `VirtualCsvDataSource` class inheriting from this `CsvDataSource` implements the `Plugin` and `DataSource` interface and enables the user to upload his/her own files. These are evaluated automatically for numeric columns (cf. section 8.6.3).

After the configuration has been loaded, the file is read line by line and inserted into a `javax.swing.table.DefaultTableModel`. Although the application does not utilize swing components, the table model is easy to use and it offers a tested data structure to store table-structured data in-memory. To prevent stale data when the contents of the underlying CSV file changes, the data is reloaded from the file for each session. After the session is finished, the data is dismissed from the webserver's memory. In addition to the data from the CSV file, virtual data is also stored in the same field. Another field pointing at the first row with virtual data marks the end of the original data and the beginning of the virtual data. In case no virtual data is set, the pointer has the value of the total number of rows in the table.

Similar to JDBC data sources, only the evaluation of aggregation nodes is essential and the calculation of field values nodes can be disregarded. To aggregate the data of a column, the implementation iterates over each row and checks whether the row fulfills the constraints set by the field value nodes. From the index of the virtual data on, the rows are only considered for the aggregation of the simulated value. In case a row matches the constraints an expression is built comprising all children's values and the values from the data row. Actually two expression are built, one for the value and one for the simulated value. Afterwards each expression is evaluated and the results are aggregated. Finally the aggregated values are assigned to the aggregation node's value and simulated value before the calculation engine's `handleValue` method is called to calculate the simulated delta.

Inheriting from the `CsvDataSource` is the `VirtualCsvDataSource` as mentioned before. Besides implementing the `Plugin` interface, it overwrites the protected `getContents` method of the super-class. The method returns a list of strings containing the file contents. When the upload feature of the plugin's `handleRequest` method is called, the application reads the uploaded file and writes its contents to the list. When the end of file is reached, the content is loaded into the table model as described above. On the frontend the user is not required to define the numeric column indices, but the application tries to cast the values of a numeric column automatically. If the value is not numeric, but a numeric value is expected an exception is thrown. Since the `VirtualCsvDataSource` builds on the `CsvDataSource`, a file is still loaded initially. Therefore a blank plain text file is located in the `WEB-INF/csv` directory by default which is then overwritten by the data uploaded from the user.

8.5.4 T39 Data Source

The most advanced data source is the implementation in the class `T39DataSource`. Not only does the class implement the `DataSource` and `Authenticator` interface, it also uses a MxL Driver developed in the scope of this thesis. In addition, the data source can handle the T39 Graph Generator plugin and it caches meta information for a defined number of minutes to reduce network load and to increase the application's performance.

One of the first methods invoked on the data source is the `initDataSource` method. In the `T39DataSource` class this method checks whether the T39 Graph Generator plugin is activated and sets a reference to the data source on the plugin. Thereby the plugin is initialized as well. Afterwards the `T39Connector` class from the MxLDriver.jar is instantiated and assigned to the `t39` field. The connection parameters are read from the `t39.properties` configuration file in the `WEB-INF/config` folder. Configuration properties are the server's address and port, the context path on the server and optionally the page space id or name. The parameter "debug" can be set to "on" and "off". When set to "on" all the requests from and to the server are written to `System.out`, except the authentication request. However, when this debugging feature is turned on, the *nounce* and session ID is logged. This information can be used to hijack the session. Therefore this feature should be turned off in a production environment. In addition to the parameters configured in a file, the username and password are stored internally by the `Authenticator` interface implementation of this class to authenticate against a T39 system.

After the data source is authenticated with the help of the `T39Connector`, meta information can be retrieved from the platform. Therefore a request for the "getMetaInformation" handler is sent and when successful, a JSON formatted structure representing the type definitions is returned. This structure is then parsed and converted to `MetaField` and `MetaType` instances as outlined in section 7.3.1. Additionally the current timestamp is assigned to the `metadataTimestamp` field in order to tell the age of the meta information. In case caching is turned on – when the constant `CACHE_METADATA` is set to true – the meta information will not be reloaded from T39 unless it is more than the value of the constant `MAX_CACHE_AGE` minutes old. By default the maximum age is five minutes.

Analog to other data sources, the `calcAggregationNode` method is the most important one while the `calcFieldValue` method is less important. Similar to statements generated in SQL, the statements in this data source implementation are generated in MxL. Despite the syntactic differences, the structure is similar. Find-queries substitute the select queries and where-clauses exist in both languages. Thus constraints can be represented easily by where-clauses. In addition, the use of complex joins is omitted since MxL handles linked data itself. Furthermore the query handler offers a `virtual` parameter which denotes whether virtual data should be considered or not. These language features simplify the implementation of the graph computation a lot.

To compute the value and the simulated value two separate queries are submitted. While the first neither comprises simulated values and deltas, nor virtual data, the query computing the simulation results considers these aspects. Virtual data is added to a MxLScenario in the session on T39. To exchange virtual data, it is serialized in JSON in the same format than it is sent to the frontend. Even the preview data has the same format when sent over to the client, but it is retrieved using an MxL query with a select-clause on all fields of the type definition as it is stored in the meta information.

In order to illustrate the computation of a graph, let's look at a simple example from hospitality management. One type definition is called "Guest" and comprises a numeric attribute "age". If

someone wants to know how many guests are in the database and their average age to target the next advertisement campaign correctly, he/she could create a model like this:

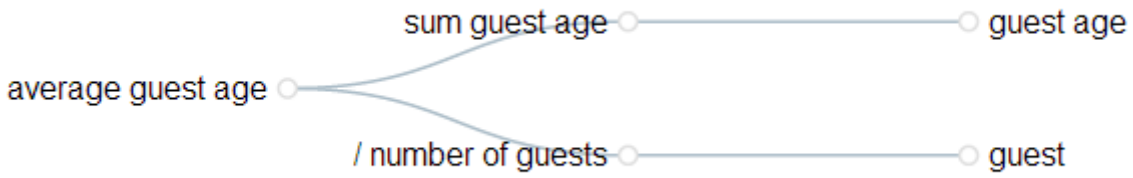

Illustration 23: Hospitality Management Example to Illustrate Graph Computation in MxL

The calculated scalar value "average guest age" is calculated by the sum of the guests' age divided by the total number of guests. In turn, the sum of all ages is an aggregation on the field age while the number of all guests is a count-aggregation on the same field. The application automatically splits the calculation of the root node into two threads to evaluate the aggregation nodes. Afterwards their values are combined to the following expression:

```
<sum guest age> / <number of guests>
```

Each statement in arrow brackets is replaced by the value from the evaluation of the queries. When the values are retrieved and the expression has been built, it is evaluated and the results are stored in the node. Then the node is serialized and returned to the client UI.

To continue with the example, the "sum guest age" node is evaluated with the following MxL query:

```
find(guest).sum(age)
```

The number of guests is evaluated with a shorter query:

```
find(guest).count()
```

Both queries return exactly one number that is saved in the value of the aggregation node. The features shown have little to do with simulation, therefore the example needs to be modified a bit. Let's say the owner realizes the average age of the guests, but he/she wants to know how an additional younger than average guest would impact his metric. Therefore the hotel owner inserts a virtual guest and runs the simulation again.

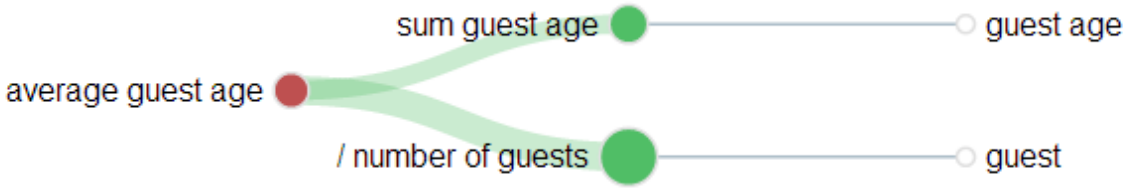

Illustration 24: Additional Virtual Entry Has an Impact on the Metric

In Illustration 24 the result is displayed. Although the number of guests (here a very small number of original guests) is affected positively, the sum is increased as well. However, the average age decreases. From the size of the circles, the reader recognizes that the number of guests has a stronger impact than the sum of the age. As this might seem unnatural, it makes sense when absolute numbers

are considered. The “number of guests” is much smaller than the “sum guest age”. Therefore adding little to the “number of guests” has a stronger impact than adding less than average to the “sum guest age”. Reasoning from this finding, the hotel owner could start advertising in media speaking to a younger audience.

8.5.4.1 MxL Driver

In order to connect to relation database management systems, Java provides a JDBC API which can be implemented by database vendors to enable developers to use a common interface to connect to different databases. Although this abstraction is not perfect as described in the beginning of this chapter, it still provides a high level of abstraction from the inner protocols of the databases. Therefore a similar approach was implemented to connect to a T39 system. To make sure this connector is reusable in other applications as well, the MxL Driver is developed in a separated project. When packaged as a Java Archive (jar) it can be integrated in any Java application. Only an implementation of the JSR 353 must be available and usable by the MxL Driver.

Connecting with the MxL Driver is easy. The main class named `T39Connector` has two constructors: One with a single string parameter representing the path to the properties file and one connector with the properties as parameters. To connect to a T39 server the following properties are required: A server address and port, the T39 context and a username and password. Optionally the space ID or name can be specified to limit the queries to a specific page space. At first the username and password must be used to authenticate the connector’s user against T39. Therefore the `authenticate(String username, String password)` method is provided. If the method returns true, the `isAuthenticated` method also returns true hereafter. Once this is the case other methods can be invoked on the connector. Listing 1 shows this connection procedure by employing the configuration file `t39.properties`.

```
private boolean connect() {
    t39 = new T39Connector("t39.properties");
    boolean res = t39.authenticate(username, password);
    assertTrue(res);
    return res;
}
```

Listing 1: Connecting to T39 using the MxL Driver

One of the most important methods is `createTypeDefinition`, which creates a type definition on the T39 system if it is not already present. Illustration 25 shows how the single parameter’s type is structured. The `TypeDefinition` class comprises a singular and plural name. Furthermore it has one method to serialize the object structure and a list of attributes of type `TypeDefAttribute`. Each attribute has a name, a multiplicity and a type. The multiplicity specifies the cardinality of the values for the attribute while the type denotes the data type of the values. A type can be a simple type such as a boolean, string or number or it can be an `EnumerationType` or `LinkType`. To serialize the structure and send it to the processing handler on T39, the `stringify` method on the `TypeDefinition` is called. In turn, this method relies on the `jsonify` and `addToJson` method of the referenced classes. All definitions are derived from the research related to the T39 system, especially Matthes et al. (2011).

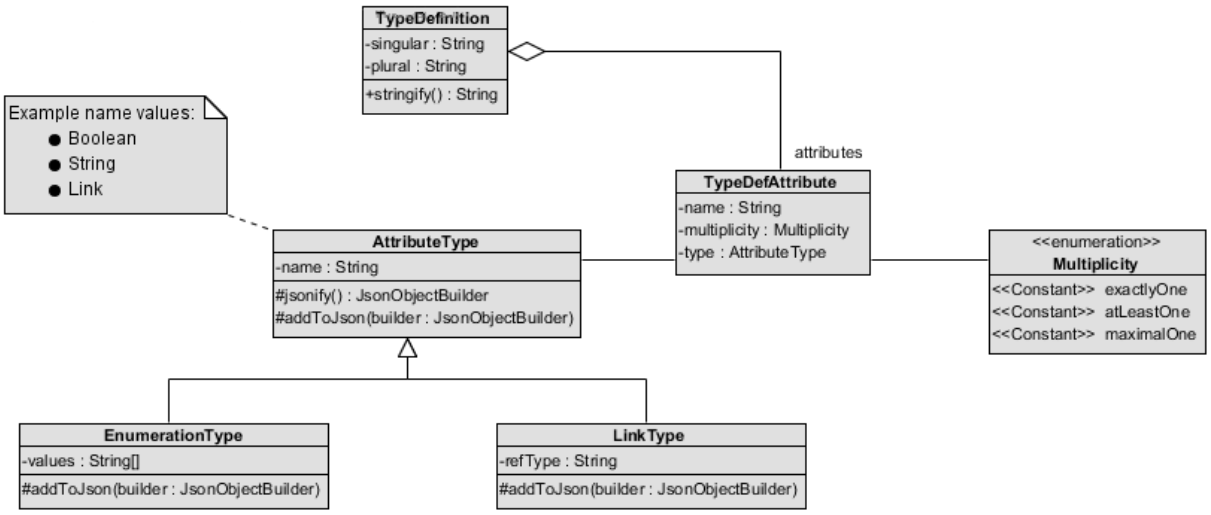

Illustration 25: Type Definition Structure in MxL Driver (own diagram)

With this structure at hand, Listing 2 shows its utilization in a Java application. At first, a new `TypeDefinition` object is created with the singular and plural name of the type definition. Afterwards attributes are added. Finally the `createTypeDefinition` method is invoked and its result is evaluated. In case the response is not successful, an error message is returned in addition to the `success` attribute.

```
// define new type
testObj = new TypeDefinition("Account", "Accounts");

// number
TypeDefAttribute numberAttrib = new TypeDefAttribute("number");
numberAttrib.setMultiplicity(Multiplicity.exactlyOne);
numberAttrib.setType(new AttributeType("Number"));
testObj.addAttribute(numberAttrib);

// type
TypeDefAttribute typeAttrib = new TypeDefAttribute("type");
typeAttrib.setMultiplicity(Multiplicity.exactlyOne);
typeAttrib.setType(new AttributeType("Text"));
testObj.addAttribute(typeAttrib);

JsonObject response = t39.createTypeDefinition(testObj);
assertNotNull(response);
assertTrue(response.getBoolean("success"));
```

Listing 2: Type Definition with MxL Driver

Essentially only one more method is core to the MxL Driver – the `sendRequest` method. As a first parameter either a `GET` or `POST` constant of the `HttpMethod` enumeration is required. Then the complete path excluding the context has to be handed as a second parameter and finally a string with the parameters and their values is the third parameter. In case a HTTP POST request is sent, the parameters string is used as the request's body. If a GET request is sent, the parameters must be encoded as URL parameters. One convenient wrapper method is the `query` method. It requires an MxL expression as the first parameter and true or false whether to consider virtual data or not. The result is a JSON object with the data retrieved from T39. In general, the `JsonObject` response of the `sendRequest` method comprises the reply string and methods to parse a JSON formatted string.

Listing 3 illustrates the implementation of the `query` method. During the implementation, many MxL requests were sent and retrieved similarly, thus this method was created. At first, the parameter string for the request is built using URL encoding. Afterwards the request is sent to T39 and processes. If the response was successful and not exception occurred, the parsed JSON result is returned. In addition, the `query` method is overloaded with the parameter `virtual` set to `false` by default. Therefore the `query` method can be invoked without the `virtual` parameter.

```
/**
 * Convenience method to query T39 with MxL.
 * @param mxl MxL expression.
 * @param virtual True when virtual data should be used, otherwise false (usually use false).
 * @return Parsed JSON result.
 */
public JsonObject query(String mxl, boolean virtual) {
    try {
        String virt = virtual ? "true" : "false";
        String parameters = "mxl=" + URLEncoder.encode(mxl, ENCODING);
        parameters += "&virtual=" + virt;

        if(isSpaceSet()) {
            parameters += "&t39_space=" + getUrlEncodedSpace();
        }

        T39Response response = sendRequest(HttpMethod.GET, MXL_DRIVER_PATH + "query", parameters);
        if(response != null) {
            return response.getJsonResponse();
        }

    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
    return null;
}
```

Listing 3: Query method implementation in the MxL Driver Package

8.6 Plugins

Extending the functionality of the system can cause the core to become hardly maintainable. Therefore a plugin-architecture is provided as section 7.2.4 described. The following section shows how to create a plugin and thereby how a plugin works. Afterwards two already implemented plugins are presented.

8.6.1 Create a Plugin

Basically five resources need to be created to produce a working plugin. First, a class which implements the `Plugin` interface. Within the interface two methods are essential: `getName` and `handleRequest`. While `getName` returns a unique identifier of the plugin, `handleRequest` must be implemented to process HTTP requests from the client. For a request to reach the plugin, it must be sent using the following address scheme:

`CalcServer/plugins/<plugin name>/<method name>[?<parameters>]`

Therein `<plugin name>` is replaced with the name of the plugin as returned by the `getName` method, `<method name>` is substituted by an identifier of the operation to be performed by the plugin and optionally `<parameters>` can be replaced with URL-encoded parameters. Within the `handleRequest` method the first parameter is used to identify which operation to execute. The second parameter is the request itself. From this request the parameters can be extracted on the server side in order to process the request and return a result. Results are returned as a JSON formatted string.

Second, a plugin-folder with the name that is returned by the `getName` method of the implementing class has to be created in the `plugins` sub-directory of the `WebContent` folder. In order to make this work, the `getName` method has to return a unique plugin name. Since the `plugins` directory contains all plugin folders, the uniqueness of a name can be checked there. Third a potentially empty cascading style sheet (CSS) with the same name as the plugin-folder must be created in the plugin-folder. Fourth, a JavaScript file named as the plugin is placed in the folder. Within the JavaScript file, a function with the name of the plugin must be declared. This method is called when the user clicks on the toolbar icon that is consequently provided by a portable network graphics (PNG) file – also named as the plugin – in that folder. All resources in the folder are loaded from the client when the plugin is activated. To activate a plugin, add it to the list of plugins in the engine's configuration file (cf. section 8.7.1).

8.6.2 T39 Graph Generator

Continuing with the example on page 63, the T39 Graph Generator plugin is presented. The owner of the hotel decides to analyze his guests in further detail. Therefore he/she needs to see how each guest's age affects the age structure. It would be tedious to manually add single field value nodes with constraints on guest names aggregated with pseudo aggregation nodes to fulfill this task. Thus the T39 Graph Generator plugin helps him to create this graph. As displayed in Illustration 19 he/she needs to click on the plugin's icon to open the following overlay window:

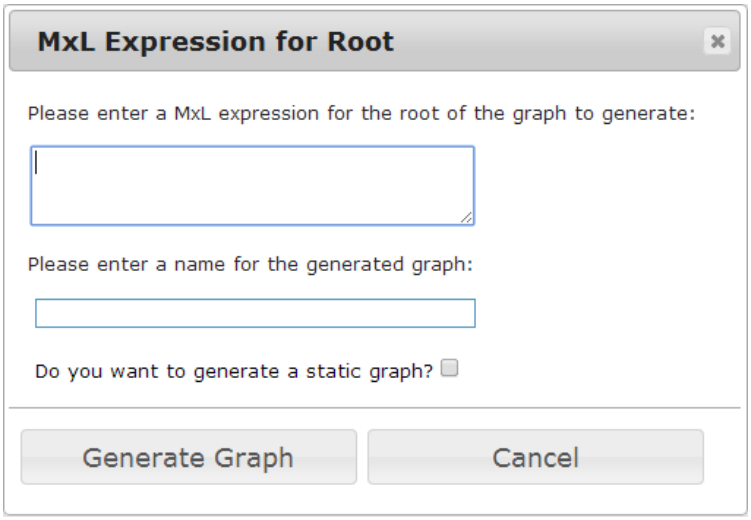

Illustration 26: T39GenGraph Window

In this example the user relies on derived attributes with the same name and definition as explained before. Derived attributes are the basis to generate the graph. In theory no derived attributes are needed to generate a graph, however, it enables the generation of complex, deep graphs. Continuing with the example, the owner recognized that one of his guests was a lot older than average. Therefore he/she decides to analyze how his average age metric would have been affected if the guest would have been rejected to stay at the hotel. Therefore the age of the guest is set to a number much lower than his/her real age, but still above average. Running the simulation again on the generated graph returns the result as displayed below:

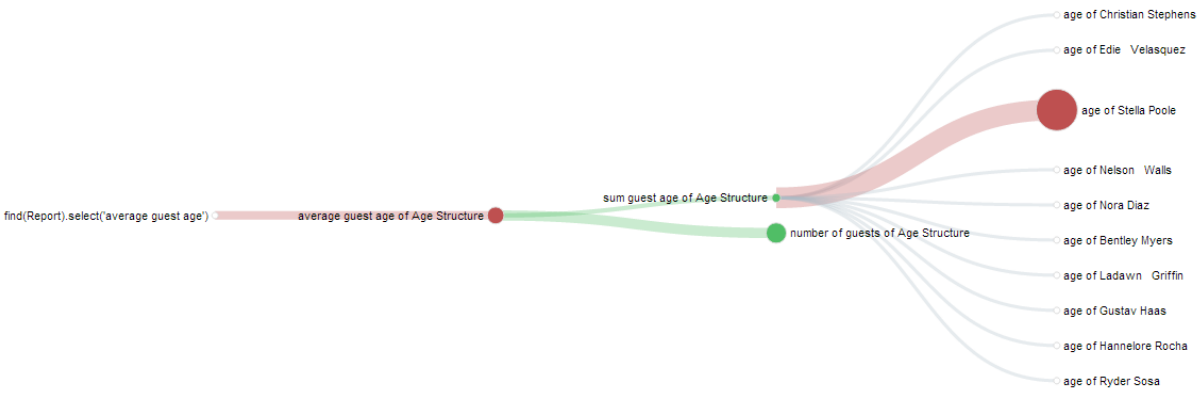

Illustration 27: Generated Graph with Simulated Value and Virtual Data

Clearly lowering the age of the guest did not have as much impact as the additional virtual guest. Still the overall average was reduced further as the red colored "average guest age of Age Structure" node shows. In Illustration 27 the size of the circle at the guest changed more than 20% which is why the other nodes were automatically moved in order not to overlap.

Technically speaking, the original algorithm to compute the graph is bypassed. An anonymous class inheriting from `CalcSetup` simply executes the graph generation again, but with a delta set. A delta set is represented by the `DeltaSet` class which contains the nodes and their simulated delta/value for a model. This delta set is serialized and sent to T39. There it is extracted and regarded when generating the graph. Then the newly generated graph and the original graph are merged together on the basis of the node ids. Finally the result is serialized and sent to the UI client.

8.6.3 Virtual CSV

In addition to the T39 Graph Generator plugin, the class `VirtualCsvDataSource` implements the `Plugin` interface as well. Its purpose is to enable the user to upload a CSV file to the server in order to use it as a data source for calculations. When the plugin is activated, it can be started as Illustration 19 shows. When no model is present, create a blank plugin and click on the plugin's icon then. Then a new overlay window appears:

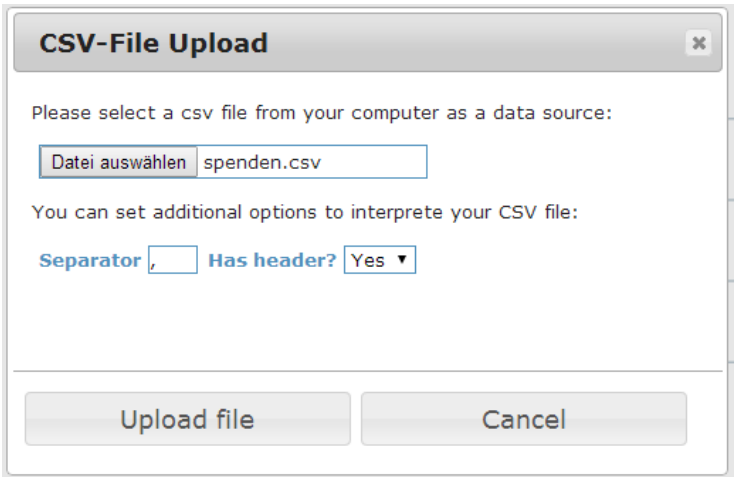

Illustration 28: Virtual CSV Upload Dialog

Since the default upload widget of the browser is used, the button to select a file from the file system of the user is called "Datei auswählen". Once it has been selected, the name of the file is displayed next to this button. Below, the separator can be specified. Not all CSV files have a header row,

therefore the dropdown menu right to the separator input box denotes whether to start with the first or second line of the file as data. A click on "Upload file" will send the file to the server where it is processed and stored in-memory for the duration of the session. A preview of the data interpretation can be viewed using the virtual data editor. Therein also virtual data can be specified for CSV file-based data sources. Other than that the Virtual CSV plugin inherits from the CSV data source and thus behaves as described in section 8.5.3.

One question the reader might ask is how to generate a dialog when a plugin essentially consists of only a cascading style sheet, a JavaScript file and a request processor on the server. All plugins are included in the main HTML document after the library and main application files. Therefore the functions and structures can be reused. Furthermore the dialogs are appended to the HTML body with a unique ID. Every time the dialog is started, the plugins check whether to recreate or reuse the dialog. Then it is displayed and closed after the user interaction is complete.

8.7 Configuration

So far many different and partly incompatible components have been presented within this chapter. To configure various aspects of the applications properties files are available. In addition, classes often employ constants to reuse static values. Logging is related to the configuration of the application since it can be configured itself and uses constants. Therefore a section is devoted to the topic. At the end of this section possible configuration options are presented to assist the configuration efforts when deploying an application.

8.7.1 File-based Configuration

By default, all configuration files are located in the `WEB-INF/config` directory. Essential configuration files for the application to function are: `engine.properties` and `logging.properties`. Furthermore all data sources presented within this work rely on configuration files and so do the documentation system (cf. section 8.9) and the MongoDB-based model storage component.

8.7.1.1 engine.properties

The `engine.properties` configuration file is the main configuration file of the application. In Table 8 each property and its value is described. Furthermore the Empty-column shows whether the property can be left without a value (empty) and if the value can be a comma separated list of values. All values referring to classes are referenced by their fully qualified class name which comprises the class's package.

Table 8: Engine Property Details

Property	Empty	List	Description of value
authenticator	No	No	Implementation of the `Authenticator` interface. If it is the same class than the dataSource property, only one instance is used.
calcGraphFactory	No	No	Implementation of the `CalcGraphFactory` interface. Please make sure that the exchange format implemented fits to the client UI (currently working with JSON).
modelStorage	No	No	Component implementing the `ModelStorage` interface.
dataSource	No	No	Data provider for the computation. The class name of the `DataSource` interface implementation must be specified.

decimalPrecision	No	No	Number of decimal figures to be displayed. This setting has no effect on the internal precision, only on the exchange format and thus on the UI.
validationLevel	No	No	Three possible values: `blocking`, `warning` and `ignore`. Please see section 8.1.3.1 for details.
plugins	Yes	Yes	Comma separated list of class names of `Plugin` interface implementations to be loaded with the application.
statistics	No	Yes	Use `de.mattimaier.mt.stats.NoStats` if you wish to use no statistics, otherwise define a comma separated list of class names of `Statistic` interface implementations.

8.7.1.2 logging.properties

To simplify exception-handling within this application, the class `de.mattimaier.mt.exception.ExceptionHandler` was created. It uses a default implementation of `java.util.Logger` in order to write exceptions to a file or the standard out. To configure the format of the log messages, the `logging.properties` file has many options as described in detail in the documentation of the Java Standard Edition by Oracle (2014). The configuration of the exception and application logging is independent from the statistics output and debugging settings in other configuration files described herein.

8.7.1.3 restapi.properties

In order to document the web interfaces as outlined in section 8.9, this configuration file needs to specify the endpoints and available APIs. The first property (`apiVersion`) contains the internal version of the API while the second property (`swaggerVersion`) contains the minimal version required for Swagger. The `apis` property is a comma separated list of identifiers for each available API. Following this parameter is the `title`, `description` and `contact` address of the administrator for the generated documentation. Further below each value of the `apis` property must specify three additional properties: `description`, `basePath` and `resourcePath`. Each property is prefixed with `api.<name of API as stated in the apis value>`. The description property contains the general purpose of the API. All operations and methods documented must be available on the base path, for example, "https://localhost:8443/MTCalcEngine". The resource path contains the path relative to the base path to where the API is reachable, not just the application in general.

8.7.1.4 mongo.properties

One of the configuration files specific in the context of MongoDB's usage as a model storage is `mongo.properties`. In Table 9 each property with its description and whether it is optional or not is displayed.

Table 9: MongoDB Model Storage Configuration

Property	Optional	Description of value
server	No	Name or IP of the MongoDB server
port	No	Port on which the MongoDB server is available, by default 27017
database	No	Name of the database where to store models
authentication	No	"true" when the MongoDB server needs authentication, otherwise "false". It is strongly recommended to use authentication.

username	Yes	The username with which the system will authenticate against the MongoDB server.
password	Yes	The password for the username.

8.7.1.5 jdbc.properties

The JDBC connectors mentioned in section 8.5.2.1 use the credentials and connection string specified in this properties file. Based on the concrete database the connection string assigned to the `URL` parameter differs. The username (`USER`) and password (`PASSWORD`) are commonly used to identify the system against the relational database system.

8.7.1.6 csvsource.properties

All settings for a CSV file to be loaded by the `CsvDataSource` are stored in this configuration file. The `folder` property contains the name of the folder in which the CSV file is located based on the `WEB-INF` directory. In the `filename` property is the name of the CSV file saved. Next is the boolean `headerline` property which denotes whether the data contains a header (`true`) or not (`false`). The `separator` property specifies the character which separated values in the file. Finally the `numericColumns` property specifies the indices that contain numeric values starting with zero. This setting is deprecated since the `CsvDataSource` implementation tries to cast all values to numbers when needed automatically and omits a value when it cannot be interpreted as a number. Furthermore values must be formatted without a comma as a group or decimal separator.

8.7.1.7 t39.properties

Connections to a T39 system are established based on the configuration properties set in the `t39.properties` file. The `protocol` contains the exchange protocol with which to communicate with the server. Usually this is set to `http`; however, `https` can be used as well. The `server` and `httpPort` properties describe the network address where the server is reachable. When T39 has no context to run in, then `/` must be written to this property, otherwise the context path on the server. Additionally a page space can be set to limit the MxL statements to this `space`. By default this property can be omitted (empty), but when a space is required the page space ID or name must be specified in this property. Finally a `debug` property enables a function that prints all queries and responses from the MxL Driver to the T39 system and vice versa. This property is mainly used for application development, but should not be used in a production environment, because critical data such as the *nounce* is logged as well. All requests to the `authentication` method are not logged for security reasons.

8.7.2 Constants

Within the application constants are used for different purposes. In the following table each value and its purpose is described. Some values are not constants in a strict sense, but values never changed by another class or hardcoded return values of methods.

Table 10: Constants

Constant	Description
`ValueGuard.KEYWORD_BLACKLIST`	List of blacklisted values that are not allowed as an input from anywhere – neither the user nor another application.
`FilesystemStorage.STORAGE_DIR`	The directory relative to the `WEB-INF` folder where models are stored, by default the `modelstore` directory.

CustomOperator.getOperationSign()	A UTF-8 character representing a custom operator.
CalcUtil.DATE_FORMAT	The format in which dates are represented in the application.
CalcUtil.APPLICATION_LOCALE	Dates and number parsers rely on the Locale specified in this constant.
T39DataSource. NUMBER_OF_PREVIEW_ENTRIES	In case the user wants to retrieve more or less than the default ten entries as a preview, please change it here.
JdbcDataSource.VIRTUAL_DATA_PREFIX	Virtual data is inserted in a separate database table with a prefix. Please change this prefix if it collides with your database setup or guidelines.
HyperSQLConnector.POOL_SIZE	Connection pool size for HyperSQL connections.
H2DataSource. enableVirtualTableRemoval()	When the before-mentioned bug in H2 is fixed so that views and tables can be removed without corrupting the database, the set this parameter to true.
FileByLineReader.FILE_ENCODING	By default CSV files are read in UTF-8.
SimpleAuth.username	Username for hardcoded credentials.
SimpleAuth.password	Password for hardcoded credentials.

8.7.3 Component Compatibility

In section 11.2 the compatibility of the application's components is depicted. As described above, some interfaces such as the `DataSource` interface only accept one implementation class to be configured. Other configuration properties can be set independent from others. For example, the `ModelStorage` implementation can be either the `FilesystemStorage` or the `MongoModelStorage` class no matter how other configuration properties are set. Similar applies to the `Statistics` implementations which can be either a combination of `LogStats` and `ConsoleStats` or just one of them. Other independent configuration properties are the validation level and the decimal precision. In contrast, the `DataSource`, `Plugin` and `Authenticator` properties must be compatible.

Since the data source configuration is the most important parameter, it is recommended choosing the preferred data source first. Based on this decision, the plugins and the authenticator are set. Hence several possible configurations apply; for example, the `T39DataSource` class as a data source and authenticator – optionally the `T39GraphGenenerator` plugin. For all other data sources the `T39DataSource` could be used as an authenticator, for example, if a T39 backend is used for identity management purposes, however, the `SimpleAuth` or custom implementations are more suitable. With this authenticator either a JDBC-based data source or a CSV-based data source can be configured. In case the `VirtualCsvDataSource` is selected as a plugin, it must be configured as a data source as well. Currently there are no plugins available for JDBC-based data sources.

8.8 Performance

Implicitly the non-functional requirement NFR-02 targets performance as described in section 5.3. While testing the application the author recognized that the performance mainly depends on backend systems like the database and the browser. Therefore these two aspects are discussed in further detail in the following two paragraphs.

8.8.1 Performance Dependent on Data Source

To demonstrate the dependency on the data source performance a simple model was created as displayed in Illustration 29. The model sums up a field value, then counts the number of fields and divides these aggregation values. Both fields target the same meta type.

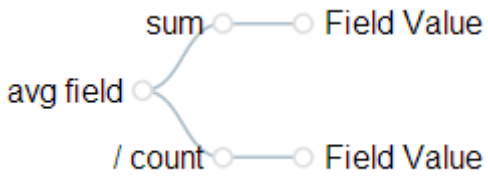

Illustration 29: Performance Measuring Model

Ten measurements were conducted using the `ConsoleLog` statistics implementation. The result and details of each measurement is shown in appendix 11.3. Clearly, the H2 data source implementation performs much better than using the application server to read a CSV file and compute the model. When compared with the T39 data source, the difference becomes even more evident. This data source implementation is approximately 280 times slower than the Virtual CSV data source as Table 11 shows. A question remains, however: Do the data source implementations differ widely or the performance of the backend systems? In case of the Virtual CSV data source the answer is simple since there is no backend. Therefore the implementation can be improved. In respect to T39, the data source implementation does not differ much. Both, the H2 and T39 data source implementations generate statements to calculate the value of a node on the backend. Since no virtual data or simulated values were set, each data source only sent one statement to retrieve a node's value. Therefore the influence of other factors is reduced. When running both data sources in debugging mode, the statements generated to compute the values are similar. Consequently none of the data sources loads large amounts of data into the application server and processes these there. Therefore the implementations do not differ much, but the performance of the backend systems.

Table 11: Data Source Performance Comparison

Data Source	Entries in Type	Average Runtime in ms	Runtime per Entry in ms	Compare with H2	Compare with Virtual CSV
H2	104,905	56.7	0.000540		
Virtual CSV	7,353	149.3	0.020305	37.57	
T39	1,710	9,742.2	5.697193	10,540.81	280.59

Why does this matter after all? The reason is that the usability, efficiency and interactive character of the application suffer when the computation takes too long. Hence optimizing the backend is beneficial for the application performance.

8.8.2 Model Generation and Model Size Limitations

Another aspect to look at is the visualization of the model. When defining small calculation models as shown in the section above the size of a model rarely exceeds hundred nodes and thus the visualization performance in the browser is of lower importance. However, when models are generated, for example, by using the T39 Graph Generator plugin, the size of the model can easily exceed 10,000 nodes. The question is, does the browser render the models fast enough to give the user a responsive and interactive feeling?

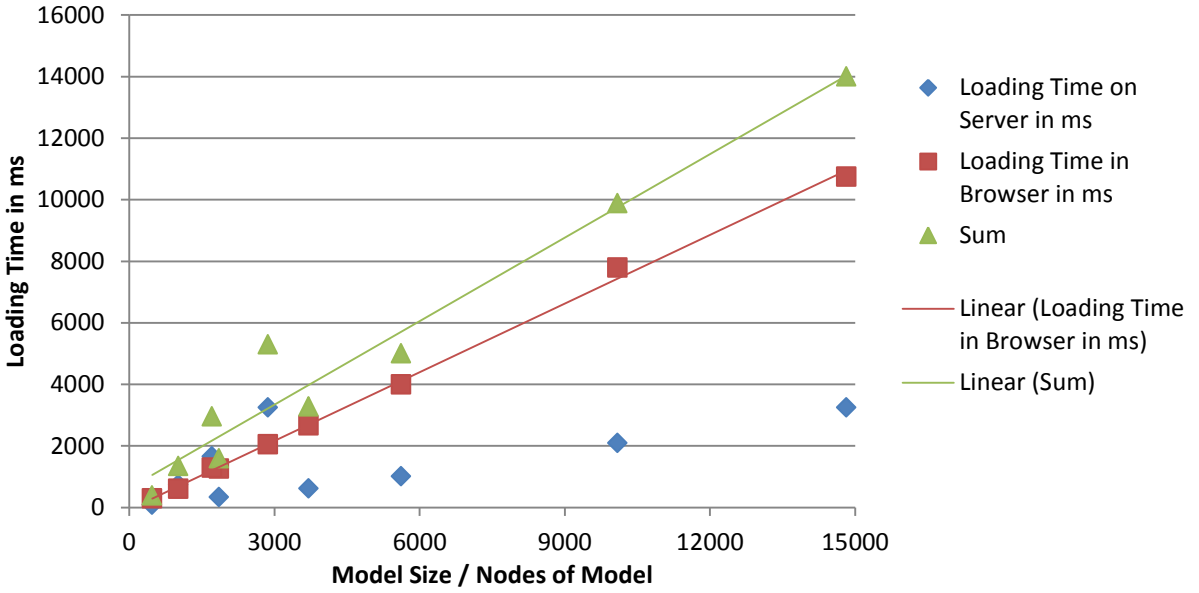

Illustration 30: Loading Time Measurements in Comparison

Table 15 and Table 16 in appendix 11.3 show the loading times measured on the client and server on a notebook with a Core i5-450M processor, no dedicated graphic processing unit, 6GB DDR3-10600 of memory and a solid state disk. For better readability the table is separated in two, thus the "Sum" column refers to the sum of both measurement averages per model. Each time was measured with the help of the statistics implementation for server times and the Google Chrome (Version 35.0.1916.153) developer tools for browser times. At first a model was generated using the T39 Graph Generator plugin. Afterwards the model was loaded several times. Each time the runtime in the browser and on the server was recorded.

In Illustration 30 these times are visualized. In addition to the data points, two linear regression lines are displayed. From these few measurements the loading time in the browser seems to make up the majority of the time until a model is displayed to the user. But, the loading time appears to linearly depend on the model size. In contrast, the loading time on the server has outliners that question a linear relationship. With respect to the outliners, the sum of loading times still seems to fulfill a linear relationship. In addition to the backend system performance that has been shown to be essential for the performance of the application, in average approximately two thirds of the model loading time is spent to render a model in the browser. Thus the hardware and software components of the client are relevant for the overall performance of the application, too.

After all, the size of the model – herein defined as nodes per model – has a significant influence on the performance of the application. Nielsen (2010) claims that response times should be ten seconds or less until a user thinks of something else (cf. NFR-02). Considering this, the maximum size of a model should be around 10,000 nodes. However, the author experienced that models with a size of approximately 3,000 nodes are the perceived maximum the Google Chrome browser could handle on the machine described above. All larger models lead to delayed mouse gestures, a juddering zoom and pan function as well as lagging responses from other UI elements such as the toolbar buttons.

8.9 Documentation

Additional to the documentation provided within this document targeting the requirements, solution concepts and architecture, the prototypical implementation and its interfaces are described in three

different styles. First, the code contains simple Java comments with information on difficult parts of the code. Furthermore the comments contain explanations whenever the author thought necessary. Moreover *JavaDoc* comments enrich the code and document the interfaces. Especially the usage of the various parameters of a method and the purpose of a class or method are reflected therein (cf. Listing 3). Whenever no *JavaDoc* comment is present, the author tried to give the method, variable or class a speaking name.

JavaDoc is only suitable for Java artefacts. Interfaces exposed over a network, for example, cannot be documented well with *JavaDoc*. Therefore the documentation language *Swagger* seems more appropriate. According to Reverb Technologies, Inc. (2014) "*Swagger™ is a specification and complete framework implementation for describing, producing, consuming, and visualizing RESTful web services.*" In Illustration 14 two RESTful JSON APIs are displayed: One between the calculation engine and the web frontend and the other between the MxL Driver and the T39 backend system. In order to document both interfaces properly, documentation artefacts were created according to the Swagger 1.2 specification (wordnik 2014). These artefacts are the `t39api.json` and `t39models.json` files in the swagger package of the `T39MxLDriver` project. To illustrate their content, an example of retrieving meta information from T39 is shown in Listing 4 and Listing 5.

```
"MetaType": {
    "id": "MetaType",
    "description": "A type definition.",
    "properties": {
        "id": { "type": "string" },
        "name": { "type": "string" },
        "fields": {
            "type": "array",
            "items": { "$ref": "MetaField" }
        }
    }
},

"MetaField": {
    "id": "MetaField",
    "description": "A hybrid property definition.",
    "properties": {
        "id": { "type": "string" },
        "name": { "type": "string" },
        "type": { "type": "string" }
    }
},
```

Listing 4: Extract from Model Definition in Swagger

The model with the name and `id`-attribute `MetaType` has three properties: `id`, `name` and `fields`. While the first two are of the simple string-type, the `fields` property describes an array with objects of type `MetaField`. A `MetaField` comprises the properties `id`, `name` and `type` where the latter refers to an instance property denoting the type of the field, not the type of the properties definition. Using these definitions, the operation `getMetaInformation` on the interface is depicted in Listing 5. By sending an HTTP-GET request to the `/mxldriver/getMetaInformation` path on the server, the processing webserver does not expect any parameters, but returns a message with the HTTP code 200 and JSON in the format of the `MetaInformation` model containing `MetaType` and `MetaField` objects. In case the server cannot process the requests, for example because the user is not authenticated, a 404 error message is returned.

```
{
    "path": "/mxldriver/getMetaInformation",
    "operations": [
        {
            "method": "GET",
            "summary": "Gets all types from the default workspace and returns them including the attributes and their types as defined.",
            "notes": "Please make sure to create a T39 session first.",
            "nickname": "t39mxldriverGetMetaInformation",
            "type": "MetaInformation",
            "parameters": [],
            "responseMessages": [
                {
                    "code": 200,
                    "message": "Success",
                    "responseModel": "MetaInformation"
                },
                {
                    "code": 404,
                    "message": "Not found, because of network restrictions."
                }
            ]
        }
    ]
},
```

Listing 5: Extract from Operation Definition in Swagger

Additionally one class offers access to the Swagger definitions in order for another Java class to extract and provide the documentation when the artefacts are encapsulated by a JAR. In the main project, the documentation file is located at `WEB-INF/api-docs/CalcServer.json`. Furthermore this project comprises three Servlets, one configuration class and the swagger-core files in the `WebContent/swagger` folder to provide an interactive user interface (UI) and to offer the documentation as an API, too.

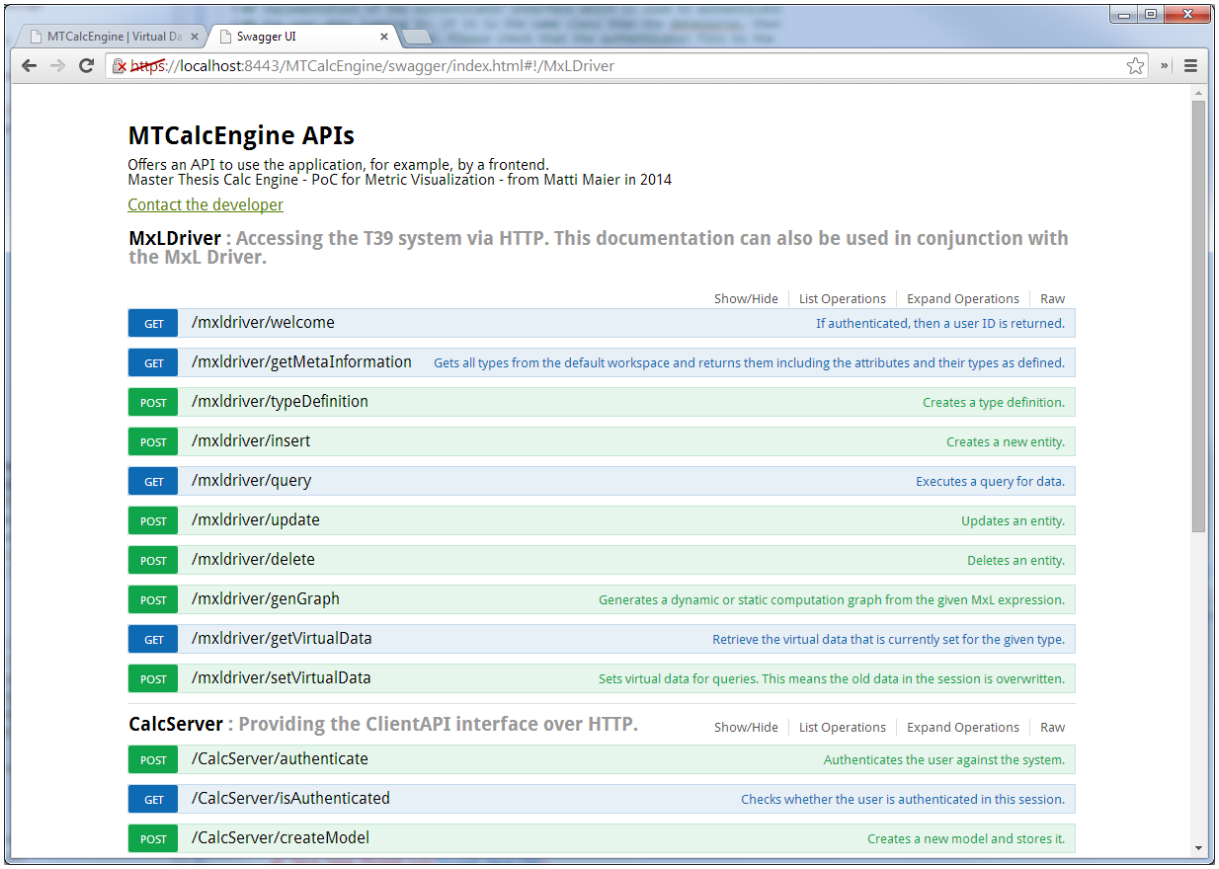

Illustration 31: Swagger UI - Method Overview

Illustration 31 displays an overview of all methods available on both RESTful APIs using in the integrated *Swagger UI*. In the upper part of the screen is the name of the documentation and a short introduction together with an e-mail address to contact the developer. Each API with its operations to be shown and hidden is displayed below. When an operation is selected by clicking on it, the details

are displayed. In Illustration 32the details of the `/CalcServer/authenticate` operation is exemplary for any other method. Now the UI user can insert his/her username and password and press "Try it out!" to send a request to the server and evaluate the response. Despite this interactive way to learn an API, a Servlet container is needed and the API providers must available. When these prerequisites are not met, a static documentation is more helpful. Therefore the Tempo (Olafsson 2013) JavaScript template engine is employed by the application to generate a static documentation based on the *Swagger* artefacts. To open the static documentation simply navigate to the following URL:

`http(s)://<server>:<port>/MTCalcEngine/static-docu.html`

The page that is loaded shows the available APIs as configured in the `restapi.properties` (cf. section 8.7.1.3). Each API contains a link to another static HTML document with the operations and models for the API. To make the contents available offline, save the first page with its linked resources.

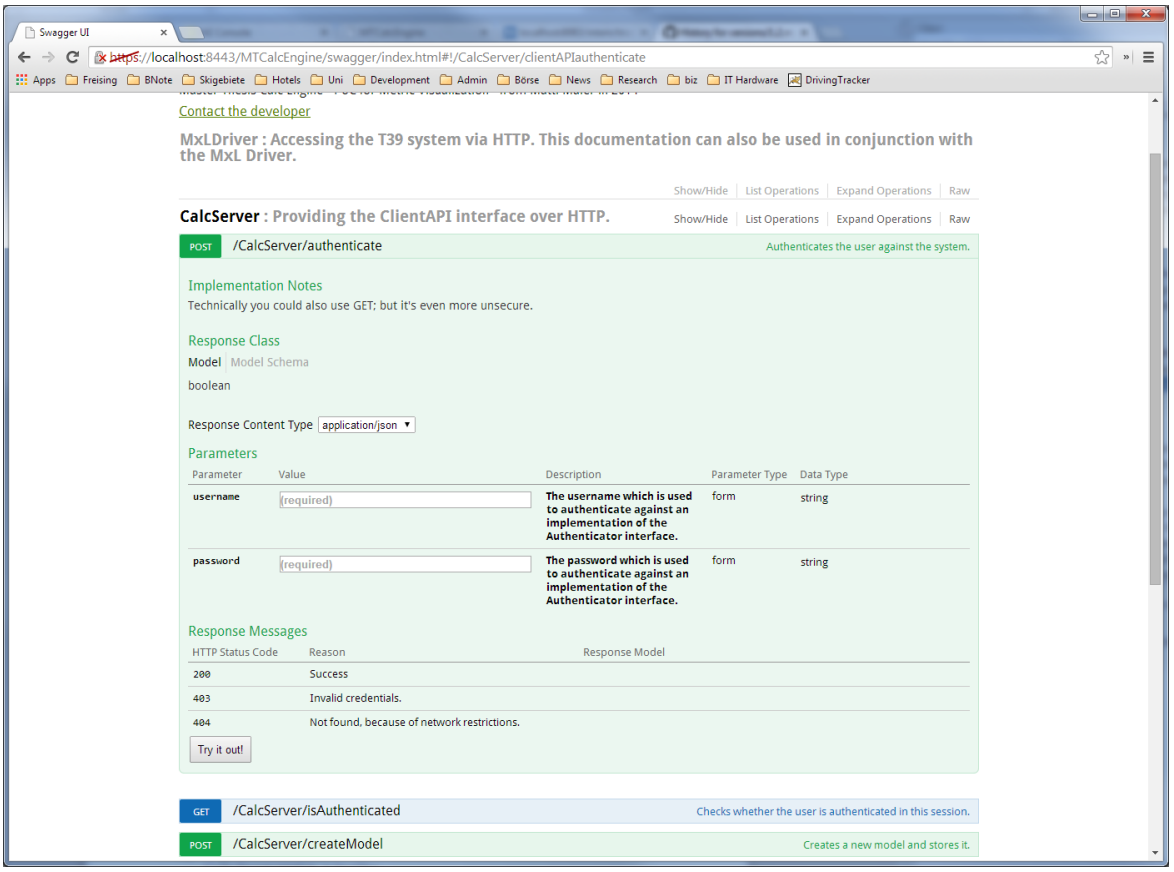

Illustration 32: Swagger UI - Operation Details

Summarizing the documentation efforts, a developer has various possibilities to seek help. If he/she needs to learn the use of the API, for example, to build an API consumer, the static or dynamic *Swagger* documentation will help. If a new data source or plugin is planned to be developed, the *JavaDoc* annotations will improve the learning curve. Finally if a bug has to be fixed or the current code basis should be extended, then the inline comments will assist the developer to complete his/her work effectively. The most important architecture-related diagrams are described in chapter 7. Users find help in sections 8.1 and 9.3.

9 Applying the Prototype

In the previous chapters the solution approach and the prototypical implementation of the approach is described. To validate the applicability of the prototype two examples were stated in chapter 1. The questions stated therein are answered with the prototypical implementation. In addition, modelling patterns found during evaluation complete this chapter.

9.1 Example 1: Hospitality Management

To validate the application in a business scenario, a simple gross margin calculation in hospitality management is regarded. Since structured data is mostly stored in RDMBS, the H2 database is used as an example in this circumstance. Table 12 shows the number of entries for each of the tables presented in section 1.1.

Table 12: Entries per Table in Hospitality Management Example

Table	Number of Entries
account	8
invoice	150,864
guest	100,000
stay	968,657
Sum	**1,219,529**

The model to solve the posed questions is displayed in Illustration 33. A fictive root combined the gross margin calculation with a metric on the occupancy. The latter is calculated as the number of guests divided by the total number of stays within the period of the data. In the upper part of the model the profit is calculated as the gross margin minus the sum of all fixed costs. Earnings are a combination of room rate and fare. In this example, the room rate is split even further in room rates paid by regular guests and room rates paid by new guests.

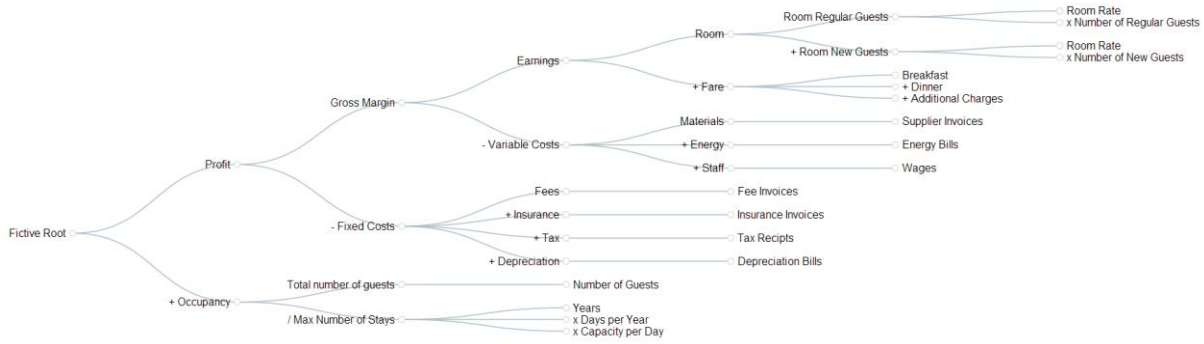

Illustration 33: Hospitality Management Example Model

First, the question is how is the gross margin affected when the room rate is increased by 1.0%, considering 2.0% less guests? Therefore a simulated delta of 1.0% is set on the "Room Rate" nodes while the "Number of New Guests" and "Number of Regular Guests" nodes have a simulated delta of -2.0% set. In 1859ms the "Room" node was calculated with a negative influence of -1.02%, earnings fell by -.0672%, the gross margin by -1.103% and the profit decreased by -1.256%. If the number of guests had decreased only by 1%, the impact on the gross margin would have been around -0.1% which another run in 2034ms showed.

In order to solve the second question, the “Energy Bills” node got a simulated delta of 3% and the “Wages” nodes of 2.5%. This increased the variable costs by 1.908% and the gross margin by -1.224%. Furthermore the “Fixed Costs” node received a simulated delta of 2% which resulted in a profit loss of 1.671% which was calculated in 1493ms. Given a new fee is introduced that has to be paid every year, it can be virtually inserted. In this example four new invoices are inserted as virtual data. Rerunning the model in 1795ms without any simulated delta reveals that fees increased by 17.805%, but the “Profit” is only affected by -0.055%. A user perceives the large impact on the fees and the tiny impact on the profit visually as displayed in Illustration 34.

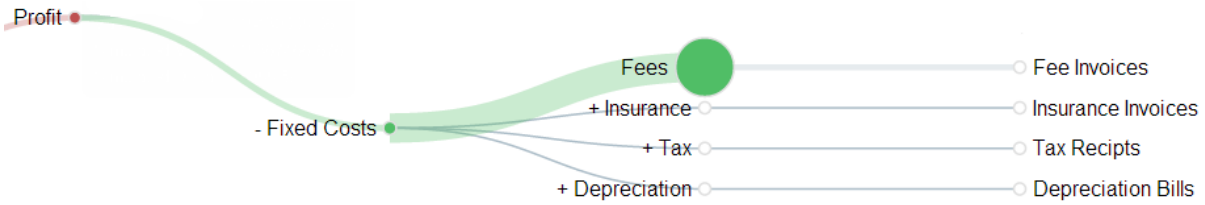

Illustration 34: Hospitality Management - Virtual Fee Impact

Third, the question is will earnings decrease significantly if the breakfast charge would be included in the room rate? The question does not state whether the room rate is added to the room rate or just spared without compensation. In case the room rate increases at the price of the breakfast, fewer new guests are expected. To solve this question with its possibilities scenarios are created. At first, the breakfast rate received as simulated value of 0 while the room rate remains the same. The 2196ms lasting calculation revealed a significant drop in gross main by 7.724%. Thus the option to increase the room rate with a breakfast charge has to be evaluated. As expected, there is no impact if the number of guests does not change. Now the number of new guests decreases by 5% and the number of regular guests decreases by 3% with the introduction of this measure. In 2559ms the application evaluated the “Room” node to increase by 2.364%, but the missing breakfast charge still reflects a negative impact of 7.724% and therefore the earnings decreased by 3.149%. Illustration 35 demonstrates the results for the last calculation.

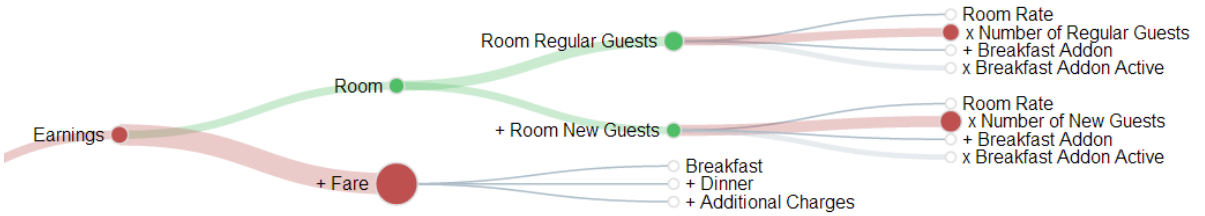

Illustration 35: Hospitality Management Model - Breakfast Compensation

In summary, all simulation runs were completed within a few seconds, although over 1.2M entries had to be evaluated. The first question targeted simple increases and decreases of values set by the simulated deltas. In the second question simulated values and virtual data was added. Finally the third question used conditions and scenarios to model a more complex decision. The interactive character of the model allows the user to assess various facets of a question and thereby get a better understanding of the system. In turn, the better understanding helps to make more profound decisions which adds business value because the likelihood of choosing the best alternative is increased.

9.2 Example 2: Metrics in Enterprise Architecture Management

Section 1.2 emphasizes the importance of metrics in enterprise architecture management. On the example of approximately 250 business applications and 100 functional domains, the two tangible questions stated in chapter 1 are evaluated.

In contrast to the example shown above, solving these questions required the use of the T39 Graph Generator plugin. At first, a graph comprising all entities related to the measure was generated based on the complexity metrics as defined by derived attributes in T39. Then a virtual application was added with a linked value to the functional domain. Rerunning the simulation with added virtual data revealed that the heterogeneity of a domain increased. Since the virtual domain is not displayed in the graph, the color and size of the heterogeneity metric changed.

To solve the second question, the number of function points was increased for one application as shown in Illustration 36. Relatively the size of the application's circle grew by over 100%, but the total functional scope was affected only minimally. Since the circle size would have expanded into other nodes, a new algorithm was implemented to move other nodes in order not to overlap the increased circle. This algorithm works well if applied as illustrated, however, with many enlarged nodes close by, overlaps are possible.

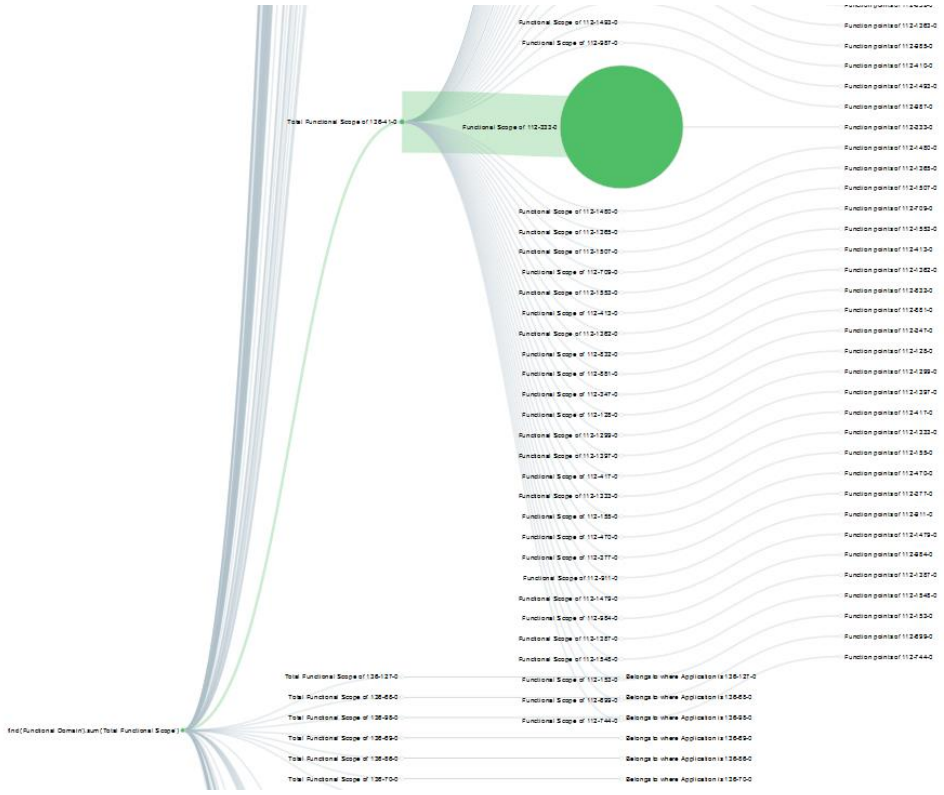

Illustration 36: Enterprise Architecture Management - Functional Complexity Example

Concluding the example, the application in question can be used to simulated changes to the enterprise architecture and its complexity metrics. Furthermore it visualizes the applications regarded by a complexity metric and it displays the impact of changes on the metric. Thus the tool helps to improve the manageability of enterprise architectures by enabling the architect to decide which applications to focus on.

9.3 Modelling Patterns

When solving the exemplary questions the author experienced similar modelling patterns for common problems. Based on these findings, the following pattern collection emerged.

9.3.1 Column Aggregation

One of the most basic tasks is to aggregate a column. Possible aggregations are the sum, product, minimum or maximum and simply counting entries. Illustration 37 shows an example graph with the node details. The aggregation node only describes the aggregation operation, within the example it is "SUM". The field to aggregate is defined by the children. Thus multiple children can be aggregated at once. Each child contains the name of the field, here "Renter: Rent" and optionally constraints. Please be careful with constraints since all constraints are AND-linked and thus might be contradictory. In this case model two separate aggregations and link them with a numeric operation.

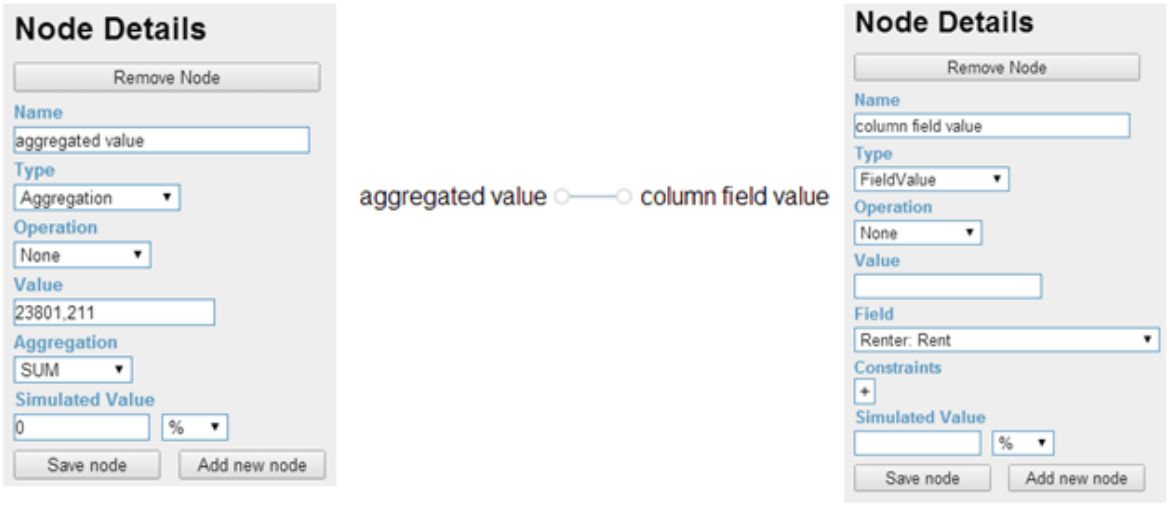

Illustration 37: Column Aggregation Pattern

9.3.2 Calculating the Average

Another common use case is modelling averages. Often data sources provide functions out of the box. Since not every data source has the function, it was not included in the aggregation operation set. However, the use case can still be modelled as displayed in Illustration 38. The formula to calculate the arithmetic mean (average) is the sum divided by the number of entries. This formula is represented within the root node. Both, the sum and the number of entries are both aggregations as described in the pattern above. "Column Value" and "Entry" are field value nodes pointing at the same meta field. To compute this model the application evaluates the "sum column value" node and the "number of entries" node and then divides these values for the "average column value".

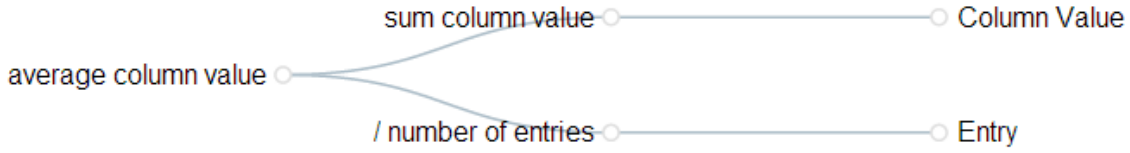

Illustration 38: Calculating the Average Pattern

9.3.3 Condition

Decisions often require choosing one option or another. Creating scenarios is difficult, since the models could structurally differ. By applying this pattern, one can model all options and turn each one on or off. In Illustration 39 a model with two options “Either this” and “Or that” is displayed along with their hover information boxes and the formula from the result node details. In the beginning the node “Activate Either this” has the value 0 and the “Activate Or that” has the value 1. Thus the result is calculated as *“Either this” x 0 + “Or that” x 1*. Now the alternative would be to set the simulated value of “Activate Either this” to 1 and the simulated value of “Activate Or that” to 0. Running the simulation reveals the value of “Either this”. Now the difference between both values is displayed in the “Simulated Delta” field of the result node. An alternative approach would be to set the simulated value of “Either this” to 0 respectively for “Or that”. Thereby other simulated values influencing the simulated value of these nodes would be eliminated. Thus this is the less preferred way.

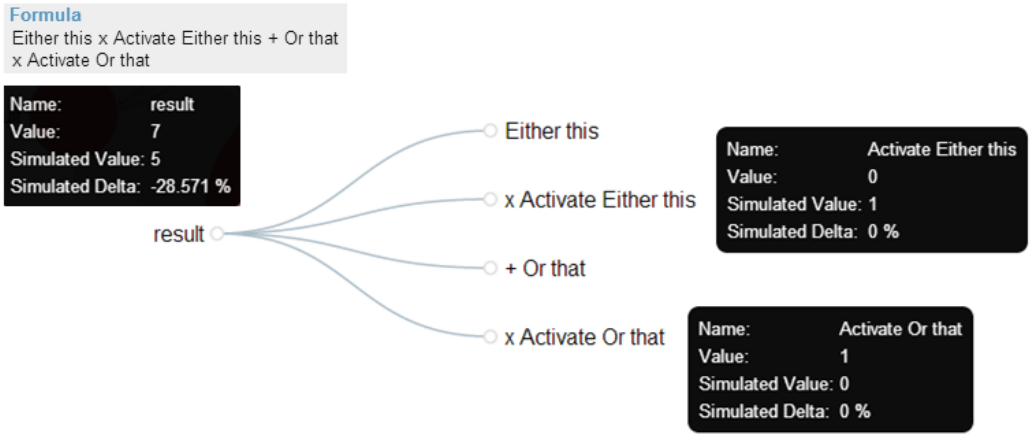

Illustration 39: Condition Pattern

9.3.4 Implicit Join

The last pattern presented within this document enables the user to implicitly join tables and follow references. Similar to the “Column Aggregation” pattern presented above, this pattern is depicted along an aggregation example as displayed in Illustration **40**. The goal is to compute only the sum of all field values fulfilling a constraint which contains a field in a foreign table or referenced type. For JDBC-based data sources foreign keys are used to join tables. If no direct foreign key between the table of the field value and the constraint field exists, the condition is omitted. In case of a T39 backend system, the MxL query always contains the constraint. If it cannot be resolved, an error message is displayed. In the example, the sum of all rents belonging to a building located in Austin is calculated.

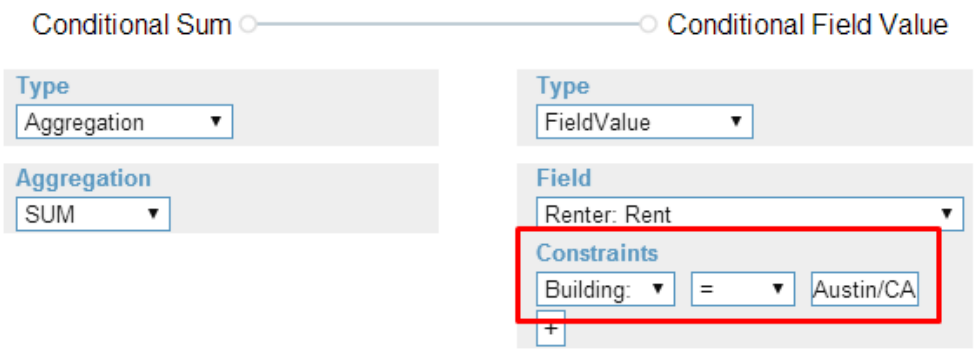

Illustration 40**: Implicit Join Pattern**

10 Conclusion

The quest of the work in hand is to develop an appropriate concept to visualize the impact of data changes in enterprise metrics. Appropriate thereby means a technically viable and theoretically profound concept. In addition, the question is how interaction on this visualization can help to improve the user's understanding of the calculation system. Therein the calculation system refers to the computation of the enterprise metrics. Helping the user to improve his/her understanding of the calculation system thus means to empower him/her to analyse the underlying information, display impacts of data changes and compare scenarios.

At first the term "enterprise metrics" is refined and subdivided into four categories: key result indicators, result indicators, performance indicators and key performance indicators. While each metric in these categories has a different purpose and expressiveness, all metrics are calculated based on data. In many cases a metric builds on one or more metrics, thus creating a hierarchy. Based on this finding a systematic literature review was conducted. As a result, two-dimensional graphs in a node-and-link-style are best assessable by humans. To evaluate the visualization of impacts, possible data changes are evaluated. Either the data underlying the calculation system can be adapted (virtual data) to lead to a deviated result or the influencing factors can be changed. An influencing factor can be replaced by another factor or altered by a factor (delta). Consequently all changes have an impact on the result. In this thesis numerical results and calculations are regarded only. Values on an ordinal scale could be mapped to numerical values; however, their interpretation is questionable since the results might be of a non-integer type. Taking these considerations into account, the result deviations are linear for each metric calculation. Following the findings from literature review, the extent of the deviation on a calculation is displayed as the circle size of a node. The stroke width of the link towards this node has the same width than the circle size. Thus a path is created leading to the node with changes resulting in the largest impact on the calculation. In addition, the color of a node and stroke represents the sign of the impact. A deviation of zero will be displayed as a thin gray line and circle while a positive impact is represented by a green circle and stroke. Respectively a negative impact is shown in red. Positive and negative hereby refer to the numeric interpretation, not the semantics of the metric. Detailed information is shown to the user when hovering over a node. Furthermore panning and zooming is made available to assess models of different sizes. Another aspect resulting from the main questions is the display of scenarios. In literature a selector for each scenario is suggested. Selecting one value of the selector results in an instant update of the graph showing the results of the referenced scenario.

Based on common business goals, requirement engineering literature and the findings depicted above, a short market research was conducted. As a result the tools presented within this document did not fulfill the requirements. However, some aspects were beneficial for the development of a new solution. For example, *Powersim Studio* enabled the user to define different types of nodes. Depending on the type, a dialog is shown to change the parameters of a node. Moreover nodes represent aggregated values, field values, constants and variables. In addition, other non-supported data sources can be added using a software development kit (SDK) with documented APIs.

With these findings at hand, a new solution approach fulfilling all requirements was created within this thesis. From a software engineering perspective the modularity, maintainability and extensibility are essential for the development of the architecture. A core principle is the separation between the calculation engine and its data source pushing the calculations as far to the data as possible. Therefore the model is translated into SQL, MxL or Java statements and executed on the backend systems to compute the results. Joining relations together is handled implicitly whenever possible. Thereby the

user does not define the link between tables/differently typed entries. In addition, all components are encapsulated by thorough documented interfaces and exchangeable by configuration. Especially for the documentation of the RESTful APIs with JSON as the exchange format three different documentation forms exist: The *JavaDoc* comments in the Java interfaces representing the interfaces on the server side, a dynamic documentation in *Swagger* and a static documentation for offline availability rendered from the *Swagger* artefacts.

Native Java interfaces include the data source as well as the model storage, authenticator, plugins, statistics and operators. Each has more than one implementation to demonstrate its exchangeability. However, not all components are compatible, thus working configurations are described and a compatibility matrix is provided in the appendix. Complementary to the functional features, non-functional requirements are fulfilled, for example: performance and security. Models are computed in parallel by default to empower the system to make use of multi-processor systems. Moreover the JDBC-based data sources use connection pooling to enable parallel computation on backend systems. Caching meta data is additionally used to reduce backend system load and request processing times. For the latter the lightweight JSON format is used as an exchange format over network connections and the amount of data transferred is reduced by sending delta sets instead of whole models were possible and by pushing the computation as far onto the data as possible. Thereby the application relies on the performance of the backend system and on the browser. On average two-thirds of the waiting time for large models is spent to render the model in the browser compared with only one-third spent on calculating the results on the server.

To prove the theoretical concept and its technical viability, two examples were evaluated on two different data sources. First, a gross margin calculation in the context of hospitality management shows the application of manually modelled calculations and different simulation approaches. Second, an example from enterprise architecture management depicts generated graphs in particular. While working with the application to solve the exemplary questions, patterns were discovered. Hence these patterns and their applicability are presented.

In summary an appropriate concept to visualize the impact of data changes to enterprise metrics is presented hereby. The solution approach is based on literature and market research while the prototypical implementation shows the technical viability of the concept. Empowering the user to interact with the model when setting simulated values and deltas, defining virtual data, comparing scenarios and managing different models helps to improve his/her understanding of calculation system and the underlying data. Hence the solution allows making profound decisions and thus reduces the risk to come to suboptimal conclusions. Consequently an organization can improve its performance and thus create more value.

10.1 Work in Perspective

In the context of related EA management research, the tool can help analyzing the effects of additional resources and application on EA metrics. Although the examples above show the solution's applicability, especially for EA metrics, a verification of the requirements as well as a validation of the usability should be anticipated in the future. Additionally, proprietary data sources can be integrated to apply the prototype in a tangible enterprise scenario. Similarly the authenticator can be extended to support single-sign-on mechanisms and network authentication services. Finally more use cases can proof or disproof the concept's viability in areas other than the ones presented within this work.

11 Appendixes

11.1 Requirements Fulfillment of Powersim Studio 10

Table 13: Requirements Fulfillment of Powersim Studio 10

No.	Title	Fulfillment	Check
BR-01	Improve change management	By using sliders, change can be simulated.	OK
BR-02	Support decision making	An instant update of the data helps to understand the system and its dependencies.	OK
BR-03	Compare scenarios	Scenario comparison is not directly possible.	-
UR-01	Desktop Environment	Powersim Studio 10 Premium is a desktop application.	OK
UR-02	Support for various screens	It is possible to resize the application for various screen resolutions, however, on high-resolution screens the toolbar buttons become very small.	OK
UR-03	Model calculations	Equations consisting of constants, operations and variables can be modelled graphically or entered in another form.	OK
UR-04	Calculation operations	Formulas can be defined with a wide set of functions.	OK
UR-05	Aggregations	Aggregation functions such as the sum and product, a count of the entries in a data set and the computation of the minimum and maximum of a numerical data set are supported.	OK
UR-06	Input data type support	No data selection criteria and no simulated values are supported. Only one value is shown.	-
UR-07	Input precision	The input and visualization supports the specification of the precision.	OK
UR-08	Large number input	The internal representation is not apparent, however, the data types seem to support large numbers.	OK
UR-09	Metric dependencies	Dependencies between variables and constants can be modelled.	OK
UR-10	Reorder factors in equation	An order of the factors can be achieved by specifying the formula which is evaluated during a simulation run.	OK
UR-11	Display equations	A simple view is shown in the formula editor	OK
UR-12	Follow mathematical rules	All mathematical rules are followed.	OK
UR-13	Set absolute factor deviation	Since the application only supports one value per variable or constant element, all the values are absolute.	OK
UR-14	Set relative factor deviation	Deviations in percent can only be simulated with the help of another element holding the simulated value, but not within one simulation run.	-
UR-15	Deviation factors do not affect production data	By setting and saving a simulated value, the deviations do not affect the original production data.	OK
UR-16	Simulated values affect dependent values and equations	Value changes are immediately pushed to all dependent nodes.	OK
UR-17	Save a model	The user can save a model and close the application or even shut down his/her computer and he/she can later reload the model and continue to work with it.	OK
UR-18	Create multiple models	Models are stored in projects which in turn are saved on the file system.	OK

UR-19	User can start simulation	A simulation run can be started by the user and is not required to be started by an administrator or scheduled task on a server, for example.	OK
UR-20	Display the simulation results	Only one result is visible.	-
UR-21	Display impact of deviations	Since only one result is displayed, the deviations are not visualized and only comparable in different models.	-
UR-22	Scenario comparison	Multiple models can be created to show different scenarios, however, there is no direct support for scenario comparison or modelling.	-
UR-23	Virtual data manipulation	A table can be specified as if it was virtual data, but the data is always regarded, not only in a simulation run.	-
UR-24	Virtual data manipulation does not affect production data	Virtual data is stored in the project, so it does not affect the operations of other applications.	OK
UR-25	Virtual data manipulation does not affect calculation of results	When virtual data is present it affects the whole calculation.	-
UR-26	Virtual data manipulation affects simulation results	Since the additional data is always regarded, it affects the results.	-
DR-01	Support for different data sources	The product supports different table-structured data sources such as Excel files and some SQL databases.	OK
DR-02	Extensibility for further data sources	A developer API is available.	OK
DR-03	User support through metadata	Metadata is not provided, the user must know the structure of the underlying data.	-
TR-01	Network connectivity	It can be assumed that database servers do not have to be locally available, thus they are connectible over a network.	OK
TR-02	Application of standards	Excel and SQL are supported de-facto standards in the industry.	OK
NFR-01	Availability	Access to the tool is available when a licence is present and the application is installed on the computer.	OK
NFR-02	Efficiency	During the evaluation of the tool, it responded quickly in the opinion of the author.	OK
NFR-03	Flexibility	The application is available for the Microsoft Windows operating only.	-
NFR-04	Integrity	Authorization and access restrictions are implemented.	OK
NFR-05	Interoperability	No interoperability and collaboration between systems is supported.	-
NFR-06	Reliability	When executing the same calculation, on the same model and data basis repeatedly, the results are always the same and thus predictable.	OK
NFR-07	Robustness	The tool shows red indicators and warnings when a formula or model element is not used as intended.	OK
NFR-08	Usability	With various training materials the basic use of the tool can be learned quickly. Most of the functionality is hidden behind toolbar buttons and tabs, in dialogues and sub-menues. The interface does not adhere to current design standards.	-
NFR-09	Maintainability	Powersim Software AS provides support for the products.	OK
NFR-10	Portability	The application cannot be used on other systems and except for sending project files to other machines.	-

NFR-11	Reusability	Due to the flexible modelling and data source connection possibilities, a reuse of the system is likely.	OK
NFR-12	Testability	Tangible test cases for a specific business case must be evaluated before employing it, therefore a demo version is available as a free download.	OK

Powersim Studio 10 fulfills 32 of the 46 requirements, thus 14 requirements are not fulfilled. In turn this results in 30.4% lack of fulfillment and 69.6% fulfillment.

11.2 Compatibility Matrix of CalcEngine Components

Compatibility Matrix

		Authenticator		ModelStorage			DataSource					Plugin		Statistics	
		SimpleAuth	T39DataSource	FilesystemStorage	MongoModelStorage	H2DataSource	HyperSQLDataSource	T39DataSource	CsvDataSource	VirtualCsvDataSource	T39GraphGenerator	VirtualCsvDataSource	LogStats	ConsoleStats
Authenticator	SimpleAuth			x	x	x	x	x	x	x	x	x	x	x
	T39DataSource	x		x	x	x	x	x	x	x	x	x	x	x
	FilesystemStorage	x	x			x	x	x	x	x	x	x	x	x
ModelStorage	MongoModelStorage	x	x	x		x	x	x	x	x	x	x	x	x
	H2DataSource	x		x	x			x	x	x	x	x	x	x
	HyperSQLDataSource	x		x	x								x	x
DataSource	T39DataSource		x	x	x						x		x	x
	CsvDataSource	x		x	x					x			x	x
	VirtualCsvDataSource	x		x	x								x	x
	T39GraphGenerator		x	x	x			x					x	x
Plugin	VirtualCsvDataSource	x		x	x				x	x			x	x
Stats	LogStats	x	x	x	x	x	x	x	x	x	x	x	x	
	ConsoleStats	x	x	x	x	x	x	x	x	x	x	x	x	x

Reading Example: Use the T39DataSource with a T39DataSource as Authenticator, a Filesystem or MongoModelStorage, and optionally the T39GraphGenerator plugin. You can use all the Statistics implementation independent from the rest.

Illustration 41: Compatibility Matrix of Components

11.3 Performance Measurements

Each measurement was executed on the same computer running all services locally. This includes the auxiliary services required by the data source. For example, a database (MySQL) and search engine (elasticsearch) service for the T39 data source.

Table 14: Measurements of Model Performance on Different Data Sources (in ms)

Run No.	H2	Virtual CSV	T39
1	85	253	14272
2	54	173	10645
3	67	125	9695
4	45	146	9508
5	59	150	9009
6	64	114	8775
7	48	108	9164
8	49	156	8780
9	46	123	8597
10	50	145	8977
Average	**56.7**	**149.3**	**9742.2**

Table 15: Measurements of Server Loading Times for Various Model Sizes (in ms)

Nodes	Avg. Loading Time on Server in ms	Sum	Measurements				
			1	2	3	4	5
473	90.8	385.6	150	77	75	82	70
1014	738.4	1349.6	954	681	684	681	692
1711	1661.4	2953.6	1794	1780	1779	1825	1129
1851	334.8	1597.4	366	312	345	317	334
2865	3244.0	5296.8	3475	3118	3267	3244	3116
3704	613.2	3282.2	617	683	595	600	571
5621	1014.4	5008.2	1179	954	949	1030	960
10091	2092.8	9885.4	2138	2011	2068	2203	2044
14818	3252.4	14003.0	3333	3258	3162	3294	3215

Table 16: Measurements of Browser Loading Times for Various Model Sizes (in ms)

Nodes	Avg. Loading Time in Browser in ms	Sum	Measurements				
			1	2	3	4	5
473	294.8	385.6	307	328	279	290	270
1851	1262.6	1597.4	1224	1259	1303	1290	1237
3704	2669	3282.2	2550	3152	2577	2533	2533
5621	3993.8	5008.2	3899	4291	3908	3930	3941
10091	7792.6	9885.4	7189	8703	7359	8457	7255
1014	611.2	1349.6	601	585	648	622	600
1711	1292.2	2953.6	1124	1145	1222	1153	1817
2865	2052.8	5296.8	2040	2396	1923	1918	1987
14818	10750.6	14003.0	10663	10178	10024	12817	10071

12 List of Figures

13 List of Tables

14 Bibliography

Addicks, Jan Stefan; Steffens, Ulrike (2008): Supporting Landscape Dependent Evaluation of Enterprise Applications. In Martin Bichler, Thomas Hess, Helmut Krcmar, Ulrike Lechner, Florian Matthes, Arnold Picot et al. (Eds.): Multikonferenz Wirtschaftsinformatik, MKWI 2008, München, 26.2.2008 - 28.2.2008, Proceedings: GITO-Verlag, Berlin.

Altstaedt, Kai (2014): Origami. Wege zum kontrollierten Einsatz von Excel und Co. In *iX* (6), pp. 150–154. Available online at http://www.heise.de/artikel-archiv/ix/2014/06/150_Origami, checked on 7/2/2014.

Barjis, Joseph; Verbraeck, Alexander (2010): The Relevance of Modeling and Simulation in Enterprise and Organizational Study. In Will van der Aalst, John Mylopoulos, Norman M. Sadeh, Michael J. Shaw, Clemens Szyperski, Joseph Barjis (Eds.): Enterprise and Organizational Modeling and Simulation, vol. 63. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Business Information Processing), pp. 15–26.

Barzokas Vassilios (2014): Trends for Languages during 20/03/2014 - 27/03/2014. With assistance of Dr. Petalidis Nicholaos. Available online at http://trendyskills.com/, checked on 3/28/2014.

Bostock, Mike (2014): D3.js - Data-Driven Documents. Available online at http://d3js.org/, updated on 7/17/2014, checked on 7/24/2014.

Bott, Ed (2014): Microsoft Office for iPad Sets the Gold Standard for Tablet Productivity. Edited by ZDNet. CBS Interactive Inc. Available online at http://www.zdnet.com/microsoft-office-for-ipad-sets-the-gold-standard-for-tablet-productivity-7000027797/, updated on 3/27/2014, checked on 6/5/2014.

BrightPoint Consulting Inc. (2014): Federal Budget Data Visualization D3.js. Available online at http://www.brightpointinc.com/interactive/budget/index.html?source=d3js, updated on 7/24/2014, checked on 7/24/2014.

British Psychological Society (BPS) (2012): Office Workers Spend Too Much Time At Their Desks, Experts Say (ScienceDaily). Available online at www.sciencedaily.com/releases/2012/01/120113210203.htm, updated on 1/15/2012, checked on 5/7/2014.

Brock, D. C.; Moore, G. E. (2006): Understanding Moore's Law: Four Decades of Innovation: Chemical Heritage Foundation. Available online at http://books.google.de/books?id=woBkE-_SOCUC.

Chemuturi, Murali (2013): Requirements Engineering and Management for Software Development Projects. New York, NY: Springer.

Cormen, Thomas H. (2009): Introduction to Algorithms. 3rd ed. Cambridge, Mass: MIT Press.

Crothers, Brooke (2014): Microsoft Office for iPad Bags 27M Downloads in Short Order. Microsoft's Julia White says Office for iPad apps have been downloaded 27 million times to date. That's more than double the number announced last month. Edited by CNET. CBS Interactive Inc. Available online at http://www.cnet.com/news/microsoft-office-for-ipad-bags-27-million-downloads-in-short-order/, updated on 5/12/2014, checked on 6/5/2014.

Currie, Wendy; Hlupic, Vlatka (2000): Strategic Management Support in Changing Environments: Simulation Modelling: The Link Between Change Management Panaceas. In : Proceedings of the 32Nd Conference on Winter Simulation. San Diego, CA, USA: Society for Computer Simulation

International (WSC '00), pp. 2022–2028. Available online at http://dl.acm.org/citation.cfm?id=510378.510674.

DATEV eG (2014): Kontenrahmen SKR 70. Hotel/Gaststätten. Nürnberg (10138). Available online at http://www.datev.de/portal/ShowPage.do?pid=dpi&nid=61938, checked on 5/5/2014.

Deutschland123.de (2014): Deutschland: Hotels + Tourismus-Statistik. Available online at http://www.deutschland123.de/deutschland-hotels-tourismus-statistik, checked on 5/5/2014.

Earthy, Jonathan; Sherwood, Brian; Bevan, Nigel (2001): The Improvement of Human-centred Processes - Facing the Challenge and Reaping the Benefit of ISO 13407. In *International Journal of Human-Computer Studies* 55 (4), pp. 553–585. DOI: 10.1006/ijhc.2001.0493.

Ernst, Alexander; Lankes, Josef; Schweda, Christian; Wittenburg, Andre: Tool Support for Enterprise Architecture Management - Strengths and Weaknesses. In : 2006 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06). Hong Kong, China, pp. 13–22.

Fernandez, Adrian; Insfran, Emilio; Abrahão, Silvia (2011): Usability Evaluation Methods for the Web: A Systematic Mapping Study. In *Information and Software Technology* 53 (8), pp. 789–817. DOI: 10.1016/j.infsof.2011.02.007.

Fitz-Gibbon, Carol Taylor (1990): Performance Indicators. Clevedon, Avon, England, Philadelphia: Multilingual Matters (BERA dialogues, 2).

Gamma, Erich; Johnson, Ralph; Vlissides, John; Helm, Richard (1995): Design Patterns. Elements of Reusable Object-oriented Software. Reading, Mass: Addison-Wesley (Addison-Wesley professional computing series).

Gartner, Inc. (Ed.) (2013): Magic Quadrant for Operational Database Management Systems. With assistance of Donald Feinberg, Merv Adrian, Nick Heudecker. Available online at https://www.gartner.com/doc/2610218, updated on 10/21/2013, checked on 3/28/2014.

Gautier de Montmollin (2013): The Transparent Language Popularity Index. Results: July 2013 update. Available online at http://lang-index.sourceforge.net/, updated on July 2013, checked on 3/28/2014.

Goldsmith, R. F. (2004): Discovering Real Business Requirements for Software Project Success: Artech House, Incorporated. Available online at http://books.google.de/books?id=3XSL7kS8GEgC.

Gruman, Galen (2013): Why iPad Apps Can't Replace Your Desktop Software - yet. It's as facile to say iPads can replace PCs as it is to say they can't - the story of mobile apps isn't black-and-white. Edited by Inc. Infoworld. International Data Group (IDG). Available online at http://www.infoworld.com/d/mobile-technology/why-ipad-apps-cant-replace-your-desktop-software-yet-216006, updated on 4/9/2013, checked on 6/5/2014.

H2 Open-Source Project (2014): Change Log. Available online at http://www.h2database.com/html/changelog.html, updated on 7/13/2014, checked on 7/16/2014.

Haedrich, Günther; Kaspar, Claude; Kreilkamp, Edgar; Klemm, Kristiane (1998): Tourismus-Management. Tourismus-Marketing und Fremdenverkehrsplanung. 3rd ed. Berlin: De Gruyter.

Hao, Ming C.; Keim, Daniel A.; Dayal, Umeshwar; Schneidewind, Jörn (2006): Business Process Impact Visualization and Anomaly Detection. In *Inf Vis* 5 (1), pp. 15–27. DOI: 10.1057/palgrave.ivs.9500115.

Harris, Peter J.; Brander Brown, Jackie (1998): Research and Development in Hospitality Accounting and Financial Management. In *International Journal of Hospitality Management* 17 (2), pp. 161–182. DOI: 10.1016/S0278-4319(98)00013-9.

Hevner, Alan R.; March, Salvatore T.; Park, Jinsoo; Ram, Sudha (2004): Design Science in Information Systems Research. In *MIS Q* 28 (1), pp. 75–105. Available online at http://dl.acm.org/citation.cfm?id=2017212.2017217.

Höß, Oliver; Falkner, Jürgen; Weisbecker, Anette (2014): Mit Anschluss. Cloud-Plattformen zum Bereitstellen und Verwalten von Web-APIs. In *iX* 2014 (6), pp. 64–71. Available online at http://www.heise.de/artikel-archiv/ix/2014/06/064_Mit-Anschluss, checked on 7/7/2014.

Java Community Process (2013): JSR-000353 Java API for JSON Processing - Final Release. With assistance of Jitendra Kotamraju. Available online at https://jcp.org/aboutJava/communityprocess/final/jsr353/index.html, updated on 5/23/2013, checked on 7/24/2014.

Jokela, Timo; Iivari, Netta; Matero, Juha; Karukka, Minna (2003): The Standard of User-centered Design and the Standard Definition of Usability: Analyzing ISO 13407 Against ISO 9241-11. In : Proceedings of the Latin American Conference on Human-computer Interaction. New York, NY, USA: ACM (CLIHC '03), pp. 53–60. Available online at http://dl.acm.org/citation.cfm?id=944519.944525.

Kerzner, H. R. (2013): Project Management Metrics, KPIs, and Dashboards: A Guide to Measuring and Monitoring Project Performance: Wiley. Available online at http://books.google.de/books?id=BRNwAAAAQBAJ.

Kim, Soyoung; Stoel, Leslie (2004): Apparel Retailers: Website Quality Dimensions and Satisfaction. In *Journal of Retailing and Consumer Services* 11 (2), pp. 109–117. DOI: 10.1016/S0969-6989(03)00010-9.

Kitchenham, Barbara; Charters, Stuart (2007): Guidelines for Performing Systematic Literature Reviews in Software Engineering. Keele University and Durham University Joint Report (EBSE 2007-001 ;).

Lagerström, Robert; Baldwin, Carliss; MacCormack, Alan; Dreyfus, David (2013): Visualizing and Measuring Enterprise Architecture: An Exploratory BioPharma Case. (13-105). Available online at http://nrs.harvard.edu/urn-3:HUL.InstRepos:11591704.

Letondal, Catherine; Chatty, Stéphane; Phillips, W. Greg; André, Fabien; Conversy, Stéphane: Usability Requirements for Interaction-oriented Development Tools. Available online at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.223.180.

Marcus, Aaron (2013): Design, User Experience, and Usability. Second International Conference, DUXU 2013, Held as Part of HCI International 2013, Las Vegas, NV, USA, July 21-26, 2013, Proceedings, Part IV. Berlin, New York: Springer (LNCS sublibrary. SL 3, Information systems and application, incl. Internet/Web and HCI, 8012-8015).

Matthes, Florian; Neubert, Christian; Steinhoff, Alexander (2011): Hybrid Wikis: Empowering Users to Collaboratively Structure Information. In : ICSOFT (1)'11, pp. 250–259.

Microsoft Corporation (2005): INFO: Genauigkeit und Genauigkeit in Gleitkommatypen Berechnungen. Available online at http://support.microsoft.com/kb/125056, updated on 2/24/2005, checked on 8/18/2014.

Microsoft Corporation (2014a): File Formats That Are Supported in Excel. Available online at http://office.microsoft.com/en-us/excel-help/file-formats-that-are-supported-in-excel-HP010014103.aspx, updated on 7/24/2014, checked on 7/24/2014.

Microsoft Corporation (2014b): Filter Data in a Range or Table. Available online at http://office.microsoft.com/en-us/excel-help/filter-data-in-a-range-or-table-HP010073941.aspx, updated on 7/24/2014, checked on 7/24/2014.

Microsoft Corporation (2014c): Import Data From External Data Sources. Available online at http://office.microsoft.com/en-us/excel-help/import-data-from-external-data-sources-HA104003952.aspx, updated on 7/24/2014, checked on 7/24/2014.

Microsoft Corporation (2014d): Secure a Workbook With a Password. Available online at http://office.microsoft.com/en-us/excel-help/secure-a-workbook-with-a-password-HP001112410.aspx, updated on 7/24/2014, checked on 7/24/2014.

MongoDB, Inc. (2014): Introduction to MongoDB. Available online at http://www.mongodb.org/about/introduction/, updated on 7/2/2014, checked on 7/15/2014.

Morato, E. A. (2013): Business Decision Making: eBookit.com.

Nazemi, Kawa; Breyer, Matthias; Kuijper, Arjan (2011): User-Oriented Graph Visualization Taxonomy: A Data-Oriented Examination of Visual Features. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell et al. (Eds.): Human Centered Design. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture notes in computer science), pp. 576–585.

Nielsen, Jakob (2010): Website Response Times. Edited by Nielsen Norman Group. Available online at http://www.nngroup.com/articles/website-response-times/, updated on 6/21/2010, checked on 6/5/2014.

Olafsson, Stefan (2013): Tempo. The Intuitive JavaScript Template Engine. Available online at http://tempojs.com/, updated on 3/7/2013, checked on 7/21/2014.

Oracle (Ed.) (2014): LogManager (Java Platform SE 7). Available online at http://docs.oracle.com/javase/7/docs/api/java/util/logging/LogManager.html, updated on 5/8/2014, checked on 7/18/2014.

Oracle Corporation (2014a): Javadoc Tool Home Page. Available online at http://www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html, checked on 7/24/2014.

Oracle Corporation (2014b): Java SE Technologies - Database. Available online at http://www.oracle.com/technetwork/java/javase/jdbc/index.html, updated on 7/24/2014, checked on 7/24/2014.

Parmenter, David (2010): Key performance indicators. Developing, implementing, and using winning KPIs. 2nd ed. Hoboken, N.J: John Wiley & Sons.

Pelkmann, Thomas (2010): Vor Excel kapituliert. Business Intelligence. Edited by CIO.com. IDG Communications Media AG. Available online at http://www.cio.de/knowledgecenter/bi/2240667/index3.html, updated on 7/26/2010, checked on 7/2/2014.

Powersim Software AS (2012): Get Started. Start a New Model Using Wizard. Available online at http://www.powersim.com/main/business-simulation/get-started/get-started2/, updated on 3/1/2012, checked on 7/24/2014.

PrincewaterhouseCoopers LLP (2014): PwC's 2014 Global Economic Crime Survey. Economic Crime: A Threat to Business Globally. With assistance of Steven Skalak, Darshan Patel, Alex Tan, Claudia Nestler, Ian Elliott, Maniu Thoithi et al. Edited by PrincewaterhouseCoopers LLP.

PrincewaterhouseCoopers LLP (Global Economic Crime Survey, 7). Available online at http://www.pwc.com/gx/en/economic-crime-survey/index.jhtml, checked on 7/7/2014.

Reschenhofer, Thomas (2013): Design and Prototypical Implementation of a Model-based Structure for the Definition and Calculation of Enterprise Architecture Key Performance Indicators. Master Thesis. Technische Universität München, München. sebis, checked on 7/6/2014.

Reverb Technologies, Inc. (2014): Swagger: A Simple, Open Standard for Describing REST APIs with JSON. Available online at https://helloreverb.com/developers/swagger, updated on 7/21/2014, checked on 7/21/2014.

Robertson, Suzanne; Robertson, James (2006): Mastering the Requirements Process. 2nd ed. Upper Saddle River, NJ: Addison-Wesley (ACM Press books).

Russell, Karl; Carter, Shan (2009): How the Giants of Finance Shrank, Then Grew, Under the Financial Crisis. Edited by The New York Times. Available online at http://www.nytimes.com/interactive/2009/09/12/business/financial-markets-graphic.html?_r=0, updated on 9/20/2009, checked on 5/13/2014.

SAP SE: Calculating with High Precision - Business Intelligence - SAP Library. Available online at http://help.sap.com/saphelp_nw70ehp3/helpdata/en/4a/fa9225912432c7e10000000a421937/content.htm, checked on 8/18/2014.

Schadow, Dominik (2014): Java-Web-Security. Sichere Webanwendungen mit Java entwickeln. Heidelberg: Dpunkt.verlag.

Schütz, Alexander; Widjaja, Thomas; Kaiser, Jasmin (2013): Complexity in Enterprise Architectures - Conceptualization and Introduction of a Measure from a System Theoretic Perspective. In : Proceedings of the 21th European Conference on Information Systems.

SoftWatch (2014): SoftWatch Benchmark: Real Usage of MS Office Apps. Available online at http://media.wix.com/ugd/f2b43d_e07a8d63359b49a58e8d4a7544841a80.pdf, updated on April 2014, checked on 6/5/2014.

solid IT (2014): DB-Engines Ranking. Available online at http://db-engines.com/en/ranking, updated on March, 2014, checked on 3/28/2014.

Spence, Robert (2007): Information Visualization. Design for interaction. 2nd ed. Harlow, England, New York: Addison Wesley.

Statistisches Bundesamt (2014): Beherbergungskapazitäten. und Auslastung. Edited by Statistisches Bundesamt. Wiesbaden. Available online at https://www.destatis.de/DE/ZahlenFakten/Wirtschaftsbereiche/BinnenhandelGastgewerbeTourismus/Tourismus/Tabellen/BeherbergungAuslastung.html, checked on 5/5/2014.

Stephen O'Grady (2013): The RedMonk Programming Language Rankings: January 2013. Available online at http://redmonk.com/sogrady/2013/02/28/language-rankings-1-13/, updated on 2/28/2013, checked on 3/28/2014.

TIOBE Software BV (Ed.) (2014): TIOBE Index for March 2014. F# on its way to the top 10. Available online at http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html, checked on 3/28/2014.

van der Lans, Rick F. (2012): Data Virtualization for Business Intelligence Systems. Revolutionizing Data Integration for Data Warehouses. [S.l.]: Morgan Kaufmann.

W3C (2012): Standards - W3C. Available online at http://www.w3.org/standards/, updated on 4/24/2014, checked on 7/24/2014.

Widjaja, Thomas; Kaiser, Jasmin; Tepel, Dennis; Buxmann, Peter (2012): Heterogeneity in IT Landscapes and Monopoly Power of Firms: A Model to Quantify Heterogeneity. In *ICIS 2012 Proceedings*. Available online at http://aisel.aisnet.org/icis2012/proceedings/BreakthroughIdeas/3.

Wiegers, Karl Eugene (2003): Software Requirements. Practical Techniques for Gathering and Managing Requirements Throughout the Product Development Cycle. 2nd ed. Redmond, Wash: Microsoft Press.

wordnik (2014): Swagger RESTful API Documentation Specification. Reverb Technologies, Inc. Available online at https://github.com/wordnik/swagger-spec/blob/master/versions/1.2.md, updated on 6/6/2014, checked on 7/21/2014.