Generic Business Simulation Using an In-Memory Column Store

Lars Butzmann[1], Stefan Klauck[1], Stephan Müller[1], Matthias Uflacker[1], Werner Sinzig[2], and Hasso Plattner[1]

[1]Hasso Plattner Institute, University of Potsdam, Germany, {firstname.lastname}@hpi.de
[2]SAP SE, Walldorf, Germany, werner.sinzig@sap.com

Abstract:
Value driver trees are a well-known methodology to model dependencies such as the definition of key performance indicators. While the models have well-known semantics, they lack the right tool support for business simulations, because a flexible implementation that supports multidimensional, hierarchical value driver trees and data bindings is very complex and computationally challenging. This paper tackles this problem by proposing an approach for generic enterprise simulations which are based on value driver trees. Our approach is two-fold: we present the definition of a simulation meta model at design time, and the run-time simulation tool. The simulation meta model describes the structure of the dependency graph, the data binding, and the parametrization of the model to simulate data changes. The simulation tool can then be used to create and edit simulation model instances and run simulations in real-time by leveraging an in-memory column store. Besides the formal description of the approach, this work presents a prototypical implementation of the simulation tool and an evaluation using data of a consumer packaged goods company.

1 Introduction

Companies invest a significant amount of time in their yearly budgeting process and the related quarterly or monthly forecasts. Especially in the higher management levels, this process is often questioned in terms of efficiency and effectiveness. Based on this deficiency, radical approaches such as *Beyond Budgeting* or alternative solutions such as *Balanced Scorecards* have been proposed. In this context, *Predictive Analytics* has been established with the goal to model cause and effect in an enterprise. This functionality can be used in the pre-budgeting process, be part of a forecast to evaluate scenarios in terms of their goal fulfillment, or try to support decisions in day to day operations.

Within the group of predictive analytics, value driver trees such as the DuPont model [CS00] are a well-known methodology. Examples for the usage of value driver trees can be found in several areas of an enterprise.

- **Procurement**: product costs depending on commodity prices, raw material purchase prices and currency exchange rates.

- **Production**: product costs depending on used energy, energy prices and production loss.
- **Sales**: sales depending on sales prices, behavior of competing companies, and available income of customers.
- **Finance**: profitability based on sales volume by market segmentation, sales prices, and currency exchange rates. Payment depending on delivery quality and adherence to delivery dates.

All these example use cases rely on linear relationships, which can be expressed by a single linear equation or by a set of linear equations with dependencies among another, as introduced by Zwicker [Zwi03]. The modeling of business processes using value driver trees provides a number of benefits: The decision making process can be supported by focusing on the key factors. This is accompanied by a reduction of the planning effort by focusing on the essentials. Also, it supports the collaboration among different business areas to create a common strategy. And if the key performance indicators are directly connected to the operational value drivers, planning for finance and accounting can become more realistic and efficient.

The challenge of using value driver trees is not the correct modeling of a company or the mathematical complexity of such a solution. It is more the acquisition of the required data, the limited capabilities of existing solutions and the expensive calculation of hierarchical dependencies on large data sets. The data is often not transparently available in ERP systems (e.g. bill of materials and routing coefficients). Also, it may be distributed among multiple systems (e.g. sales, production, and finance) and only be known implicitly (e.g. elasticity factors for the inverse demand function for products). Available solutions on the other hand are developed for specific use cases such as the calculation of product costs, the calculation of advertising effects, or the calculation of currency effects. With changing requirements however, the solution cannot be adapted but has to be reimplemented. One of the biggest drawbacks of current solutions is the run-time of calculations. While subsecond response times are required for a usage in the on-line decision making and planning process, a majority of existing solutions is running batch processes and therefore cannot meet this goal. Batch processes typically run several hours, most of the time over night.

A possible solution for this problem is the in-memory database technology. Previous work by Plattner [Pla09] has shown the benefits of columnar storage using main memory technology for analytical workloads. With the improved query performance, it is now possible to simplify complex data models [Pla14] that relied on materialized aggregates and other workarounds that were used to provide reasonable response times. The result is an increasing data transparency. Another benefit is the flexibility to query the data, even when the data sets are large and relations need to be included. This flexibility is important for retrieving the value drivers (sales volume, purchasing volume, expenses) as well as the coefficients between the drivers.

In this work, we introduce a solution to address the mentioned challenges and problems. First, we introduce a meta model how dependency graphs can be created and configured. Second, we explain the simulation tool and how it creates and runs simulations. Third, we

present a prototypical implementation of the HPI Business Simulator tool and demonstrate it using a real-world example. Finally, we analyze the query performance for both storage types, a row store and a column store.

2 Approach

Our approach is divided into three parts to build a tool for business simulations: the definition of a generic calculation model, the data binding and the simulation.

2.1 Calculation Model

Simulation models are hypergraphs. Each node has a name, available dimensions and can be connected with other nodes by *operations*, which are hyperedges. The simulation model specifies the available dimensions with their hierarchy levels. Time can for example be hierarchically structured into years, months, and days. The dimensions of a specific node are determined by the data binding or the operation and dimension specification of connected nodes in case the node has no data source.

Operations define the dependencies between nodes. Each operation has to define the way it calculates the values for the result nodes. Thereby, the approach can define the calculations of common operations as addition, subtraction, multiplication and division, which work in a similar way as for one-dimensional data. Combining the values of nodes with different levels of hierarchies may require a preaggregation phase. In other cases, the operation cannot be executed at all. The signature and algorithm of each operation has to be defined. The addition operation connects two nodes with the same dimensions and creates a node with values for these specific dimensions. The hierarchy level of the resulting node is the minimum of both input nodes.

2.2 Data Binding

Nodes of the calculation model can obtain their values in two ways. On the one hand, they can query their data directly from database tables. On the other hand, they can calculate their values by solving the equation defined by the operation between connected nodes and itself. For the first case, data bindings are required. Our work focuses on relational databases as a data source, although it is also possible to use other sources. Data bindings define the database connection and SQL queries for all supported dimension, filter criteria and hierarchy levels. A generic way to implement it without specifying each single possible query is to define the SQL query to request the data for all dimensions at the lowest hierarchy levels. Based on this data, filter criteria and aggregations can be applied to support selections and views of the data at higher hierarchy levels. Figure 1 shows a dependency graph with marked nodes for data sources.

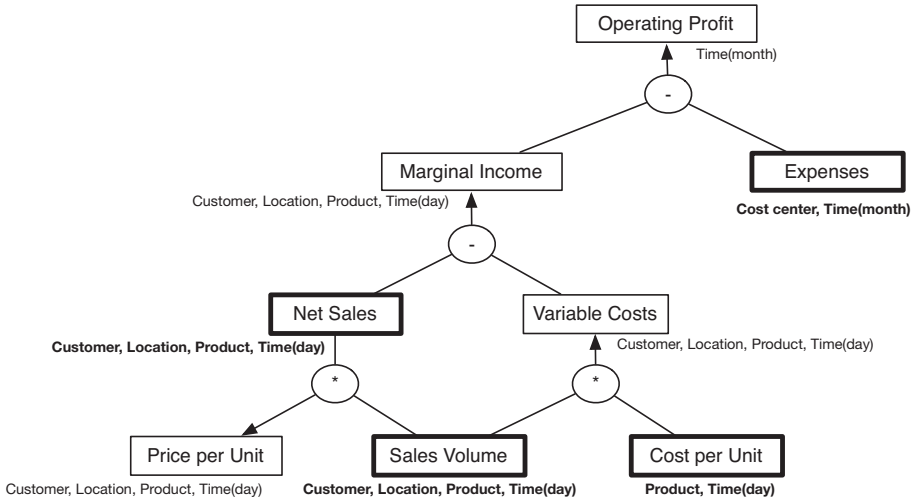


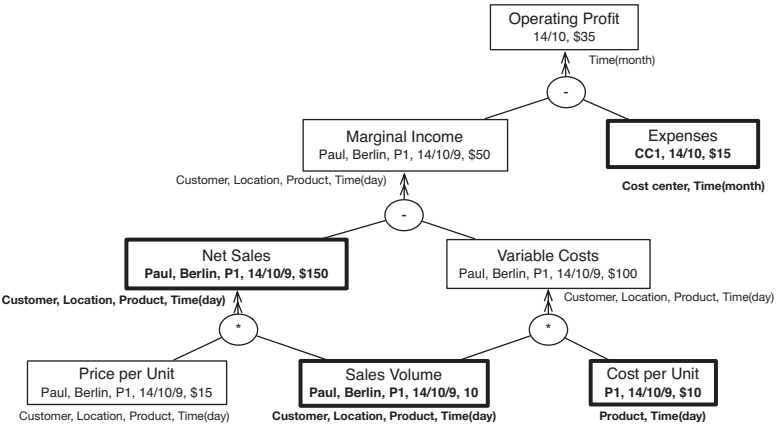
Figure 1: Dependency graph for the operation profit: Bold nodes represent data sources with their dimensions. The values of nodes which are marked by arrows can be calculated recursively by following the operations from which the arrows come from.

2.3 Simulation

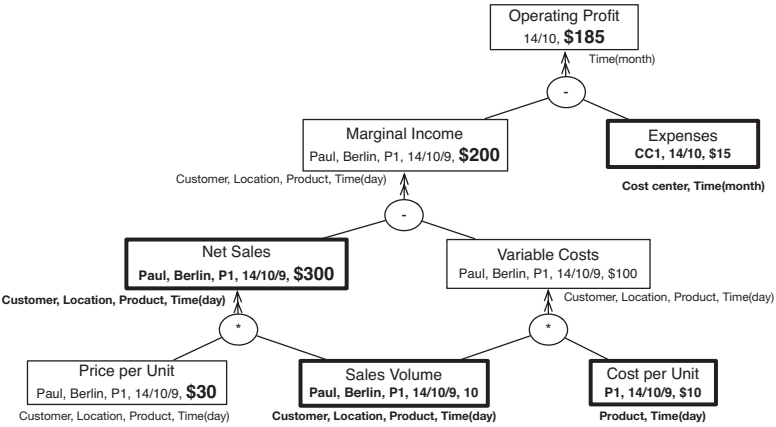
The simulation tool supports creating, changing and running simulation models. In general, different user groups are responsible for model editing and specifying simulation scenarios. The choice of KPIs and its adaption to better fit the company's structure is executed at management level. As the management level does not have a detailed knowledge of the data schema, it will delegate the task of defining data bindings to more technical staff. The individual simulations can then be done in collaboration with controllers. Hence, the tool has two modes to separate these activities: one for editing simulation model instances, the other for executing simulations. The editor mode comprises an interface to create dependency models as described in Section 2.1 and allows to specify the data binding. A graphical modeling tool can be used as user-friendly interface, as well as a text editor to describe the model with structured text, e.g. JSON or XML.

Simulation model instances could then be validated before switching to the simulation mode. The simulation mode allows drilldowns, i.e. filtering the node values by dimensions and hierarchy levels. The number of dimensions specifies the number of drilldown possibilities. The depth of the hierarchy specifies how deep the drilldown goes. If a node does not support the current drilldown level, it is excluded from the simulation. For each drilldown level, simulation parameters can be specified. Thereby, the model has to define directions for edges to specify which nodes are affected by the parameterization of connected nodes.

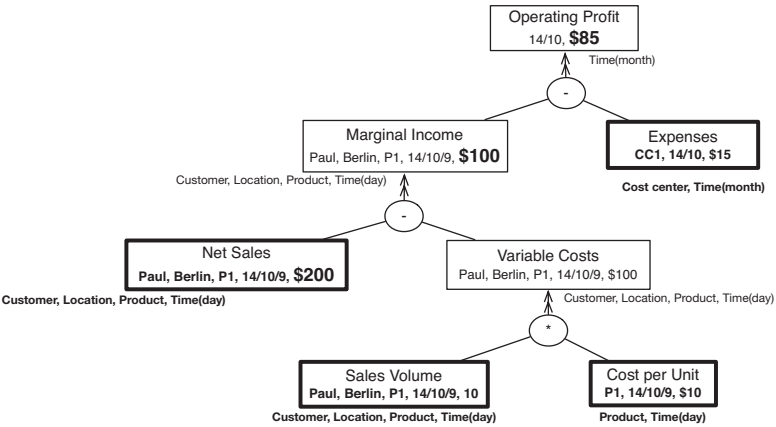
Figure 2 shows an example of a simulation. Based on the dependency graph as given in



(a) Initial state with data from the database table.



(b) Simulation with an absolute value for price per unit.



(c) Simulation with an absolute value for net sales.

Figure 2: Examples of Simulations.

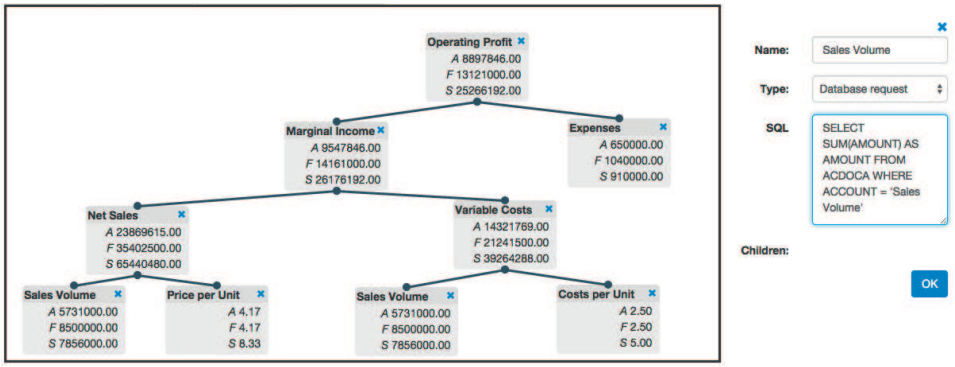


Figure 3: The editor mode of the HPI Business Simulator.

Figure 1 with data sources for net sales, sales volume, cost per unit and expenses, the other nodes can be calculated forming the graph as presented in Figure 2a. The values are the basis of the simulation. The double arrows specify in which directions the values are propagated. Each value, independent whether it was directly queried from a database table or was calculated, can be overwritten by simulation parameters. Figure 2b shows the simulation of an increased price per unit ($30 instead of $15) and how it propagates via the endpoints with double arrows. When parameterizing nodes with incoming double arrows, e.g. setting net sales to $200, the connected nodes are ignored (cf. Figure 2c).

3 Proof of Concept

In this section, we explain the capabilities of the HPI Business Simulator. For a proof of concept (POC), we have engaged with a Fortune 500 company in the consumer packaged goods (CPG) industry and discussed their deficiencies and needs in the area of business simulation. Based on the input and data set they provided to us, we created a prototype of our simulation model.

3.1 HPI Business Simulator

In the first step, we created an instance of a simulation model using the editor mode of our tool. The created graph consists of 8 nodes (the node for sales volume is represented twice) and 4 multidimensional attributes called dimensions. The complete tree is shown in Figure 3.

```
{
    "dimensions": {
        "time": ["year", "month", "day"],
        "product": ["product"],
        "customer: ["customer"],
        "region": ["country", "city"]
```

```
    },
    "nodes": {
        "node1": {
            "name": "Sales Volume",
            "dimensions": {"customer": "customer", "region": "country", "time": "day
                ", "product": "product"},
            "unit": "kg",
            "SQL": "SELECT SUM(AMOUNT) AS AMOUNT FROM ACDOCA WHERE ACCOUNT = 'Sales
                Volume'"
        },
        "node2": {
            "name": "Price per Unit",
            "dimensions": {"customer": "customer", "region": "country", "time": "day
                ", "product": "product"},
            "unit": "USD/kg",
        },
        "node3": {
            "name": "Net Sales",
            "dimensions": {"customer": "customer", "region": "country", "time": "day
                ", "product": "product"},
            "unit": "USD",
            "SQL": "SELECT SUM(AMOUNT) AS AMOUNT FROM ACDOCA WHERE ACCOUNT = 'Net
                Sales'"
        }
    },
    "operations": {
        "node3": "node1 * node2"
    }
}
```

Listing 1: An extract of the model definition using the JSON format.

We use the JavaScript Object Notation (JSON) as a data format to store the model information. Each node is represented as an object, specifying its name and corresponding properties. The ID is used as key for the corresponding node object. Listing 1 is an example for a subgraph that appears in Figure 3. The first level of the JSON structure contains three values, the existing dimensions, the nodes itself, and the operations which represent the connections between the nodes. As explained in Section 2.1, a node consists of a set of properties. The node with the name *Sales Volume* has a data source and therefore has an SQL query that defines how its data can be retrieved. In contrast, the connected node *Price per Unit* has no data binding and calculates its values using the formula defined in the operations list. In this subgraph, the units are not equal for all nodes. In cases where the units are not the same, the editor ensures that the operations return the correct units. While creating the model, the editor constantly checks the model for consistency and correctness. In this example, the unit of *Sales Volume* is kg and the unit of *Price per Unit* is USD/kg. Since the unit *kg* is in the denominator and numerator, they cancel each other out, leaving USD as the unit for *Net Sales*.

3.1.1 Consumer Packaged Goods Data

The underlying data set we got had a normalized data schema with one transactional table (ACDOCA) and multiple corresponding dimension tables. The transactional table consists of approximately 300 million line items which represent a period of 5 years. Dimensions that were included are the account type, cost center, product information, and location information. For our scenario, we used a denormalized data schema to remove the join

Table 1: Example data for the underlying ACDOCA table with four dimensions.

DocID	G/L Account	Cost Center	Product	Customer	Location	Date	Qty	Amount
01	COGS		Product 1	Paul	Berlin	10/9/2014	10	100.00
02	COGS		Product 2	Paul	Berlin	10/9/2014	5	75.00
03	Inventory					1/1/2014		-175.00
04	Revenue		Product 1	Paul	Berlin	10/9/2014	10	150.00
05	Revenue		Product 2	Paul	Berlin	10/9/2014	5	125.00
06	Receivables			Paul	Berlin	10/9/2014		275.00
07	Expenses	Cost Center 1				10/2014		15.00

complexity for the model creation process. Table 1 shows example data for the relevant columns used by the business simulator. The two main columns are *G/L Account* and *Quantity* respectively *Amount* which are included in every query. *G/L Account* describes the type of account a line item corresponds to. The other five dimension columns are only included into the SQL query when they are required.

Based on the customer feedback, the HPI Business Simulator (cf. Figure 3) shows three values per node. First, the actual value (*A*) for the current period which includes all values that exist until today. Second, the forecasted value (*F*) for the complete period that is based on A and plan data (*P*). Third, the simulated value (*S*) which is based on *F* and changes calculated by the simulation model. By default, the forecast and the simulation value are equal because no parameter has been changed. By adjusting a node with an absolute or relative value, the valued driver tree is recalculated and updated with the new results. As a result, the simulation value changes and consequently reflects the changes compared to the forecast value. The user is able to select certain filters to drill down into a dimension, e.g. a certain product. In this case, the adjustments are only applied for the selected dimensions.

3.2 In-Memory Column Store

A key enabler for our tool is in-memory column stores, such as SAP HANA [FCP+11]. By keeping all data in main memory, the run-time of analytical queries, as required for the business simulator, can be reduced significantly compared to traditional disk-based systems. The second benefit is the columnar data layout. Since the mapping of the simulation model only accesses columns that are required for a calculation, the amount of processed data is kept to a minimum. This functionality is especially beneficial in the context of enterprise systems where examples show that tables can have more than 300 columns. The columnar layout is also well suited for analytical queries, e.g. aggregate queries, where larger data sets of a column have to be scanned and aggregated.

A third benefit is an aggregate cache which is a transparent caching engine inside the database [MP13]. Other than traditional materialized views, the aggregate cache does not create a hard copy of the data and as a result does not return stale data. The aggregate cache leverages the internal table representation in certain column stores like SAP HANA or Hyrise [GKP+10] and always works on the latest data. During a typical simulation session, different scenarios are analyzed and compared. A scenario includes a set of changes in the value driver tree. The differences between the scenarios vary, but are usually small.

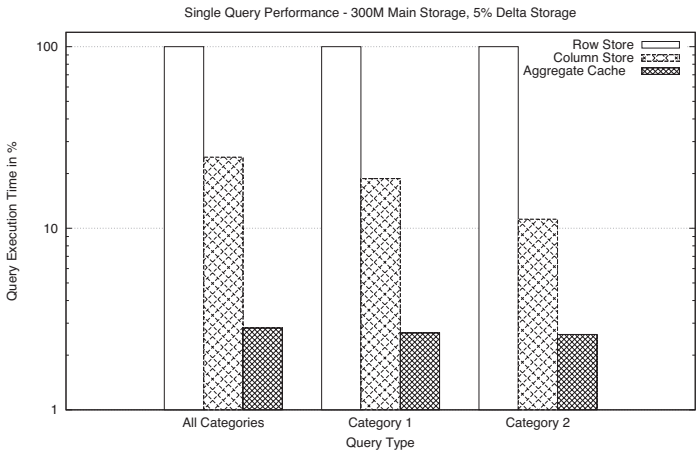


Figure 4: A performance comparison between a row store, a column store and a column store with an aggregate cache for three query types.

Consequently, the executed queries are similar and can therefore benefit from the aggregate cache. Further, the created scenarios are constantly reviewed and the updated forecast is compared to the simulated value. In that process, the simulator can rely on the aggregate cache to speed up the query execution and reduce the overall system load.

3.2.1 Performance Analysis

In our evaluation, we measured the performance of the queries using a row store, a column store and a column store with an aggregate cache inside SAP HANA [FCP+11]. The benchmarks have been conducted on a server featuring 128 CPUs (Intel Xeon X7560) with 2.27GHz and 24MB cache each. The entire machine was comprised of 512GB of main memory. The data set size on disk was approximately 130GB. Every benchmark was run at least five times and the displayed results are the median of all runs.

In Figure 4, the query performance for three types of queries is evaluated. The three types differ in the amount of data they have to aggregate. To demonstrate the impact of a column store and the aggregate cache, we compared the performance with an in-memory row store inside SAP HANA. The results are shown relatively to the row store, which serves as the baseline for the other two approaches. With the first query, the value of a single value driver, in this case *Net Sales*, for all categories was retrieved. Using the column store, the query is accelerated by a factor 4. For the aggregate cache, it is accelerated by a factor of 35. For the second and third query, the same query was used with a filter on the category. Category 1 is the biggest category including approximately 40% of all products. Category 2 is a smaller category including only 10% of the products. The execution time for the row store resulted in the same execution time as for the first query, even though the number of records that had to be aggregated was smaller. In contrast, the performance of the column store increased with an increasing selectivity resulting in an acceleration

of a factor 10 for the third query. The performance using the aggregate cache was only slightly faster for the second and third query (factor 38 compared to the row store). This is a result of its characteristic since it only caches the results of the main storage and still has to aggregate the delta storage on the fly. The records in the delta storage just slightly decreased for query 2 and 3. A deeper evaluation of the aggregate cache was done by Müller and Plattner [MP13].

The benchmark shows that a column store is superior to a row store for typical business simulator queries. However, more investigation and benchmarking is necessary, especially in the context of executing multiple similar queries at the same time, as required by the business simulator.

4 Related Work

Besides sophisticated controlling software, spreadsheets, MS Excel in particular, are the main simulation tools in enterprises. The RS-Controlling-System [1] offers Excel templates for target-performance comparisons of balance sheets, profit and loss statements and capital flows. Spreadsheets have an easy to understand interface, but do not build on consolidated enterprise data, which is stored in a RDBMS. On the other side, SQL lacks the support for array-like calculations as Witkowski et al. claim in [WBB+03]. Their idea is to combine both and offer a spreadsheet-like computation in RDBMS through SQL extensions. In [WBB+05], they continue that research and introduce a way to translate MS Excel computations in SQL. The simulation model is thereby expressed in MS Excel. This approach comes with two drawbacks. First, it does not encapsulate the definition of the simulation model so that it is not tangible, but only expressed by multiple formulas spread over many table cells. Second, MS Excel is bound to the two-dimensional representation and cannot visualize graphical dependencies very well.

To describe simulation models, Golfarelli et al. introduce a methodology to formally express and build what-if analyses in [GRP06]. They divide the process to design simulations into seven phases: goal analysis, business modeling, data source analysis, multidimensional modeling, simulation modeling, data design and implementation, and validation. To describe the actual simulation model they propose an extension of UML 2 activity diagrams [GR08]. However, they do not focus on how to develop generic simulation applications that are based on the formal descriptions.

5 Conclusions and Future Work

This paper proposes a new approach to build business simulations. It consists of a multidimensional generic model to describe the dependencies of value drivers as basis of simulations and the concept of a simulation tool to create, edit and parameterize model instances

[1] http://www.controllingportal.de/Shop/Excel-Tools/RS-Controlling-System.html

to simulate scenarios.

The main benefits of our solution can be summarized as follows: The generic business simulation approach enhances the focus on key factors that drive the business. Further, it reduces the planning effort and drives cross-functional collaboration and alignment. By directly using transactional data as a basis for the simulation, we exclude consistency issues. Most importantly, the simulation solution provides leadership with visibility into different alternatives to achieve desired outcomes.

The calculation model and database binding are described in a semi-formal way in prose text and JSON. Future work can formalize the model and the multidimensional operations. Of special interest are the possibilities and specifications of simulation changes. Beside, an investigation of optimization strategies for the required database queries, which base on groupings with different granularities, can be done.

References

[CS00] A.D. Chandler and S. Salsbury. *Pierre S. Du Pont and the Making of the Modern Corporation*. BeardBooks, 2000.

[FCP+11] Franz Färber, Sang Kyun Cha, Jürgen Primsch, Christof Bornhövd, Stefan Sigg, and Wolfgang Lehner. SAP HANA database: data management for modern business applications. In *ACM SIGMOD*, pages 45–51, 2011.

[GKP+10] Martin Grund, Jens Krüger, Hasso Plattner, Alexander Zeier, Philippe Cudre-Mauroux, and Samuel Madden. HYRISE: A Main Memory Hybrid Storage Engine. *PVLDB*, 4(2):105–116, 2010.

[GR08] Matteo Golfarelli and Stefano Rizzi. UML-Based Modeling for What-If Analysis. In *DaWak*, pages 1–12, 2008.

[GRP06] Matteo Golfarelli, Stefano Rizzi, and Andrea Proli. Designing What-if Analysis: Towards a Methodology. In *DOLAP*, pages 51–58, 2006.

[MP13] Stephan Müller and Hasso Plattner. Aggregates Caching in Columnar In-Memory Databases. In *IMDM @ VLDB*, pages 62–73, 2013.

[Pla09] Hasso Plattner. A common database approach for OLTP and OLAP using an in-memory column database. In *ACM SIGMOD*, pages 1–2, 2009.

[Pla14] Hasso Plattner. The Impact of Columnar In-Memory Databases on Enterprise Systems. *PVLDB*, 7(13):1722–1729, 2014.

[WBB+03] Andrew Witkowski, Srikanth Bellamkonda, Tolga Bozkaya, Gregory Dorman, Nathan Folkert, Abhinav Gupta, Lei Shen, and Sankar Subramanian. Spreadsheets in RDBMS for OLAP. In *ACM SIGMOD*, pages 52–63, 2003.

[WBB+05] Andrew Witkowski, Srikanth Bellamkonda, Tolga Bozkaya, Aman Naimat, Lei Sheng, Sankar Subramanian, and Allison Waingold. Query by Excel. In *VLDB*, pages 1204–1215, 2005.

[Zwi03] E. Zwicker. *Prozeßkostenrechnung und ihr Einsatz im System der integrierten Zielverpflichtungsplanung*. Techn. Univ. Berlin, 2003.