Towards an Algebraic Foundation for Business Planning

Katrin Eisenreich
SAP Research CEC Dresden, SAP AG, Germany
katrin.eisenreich@sap.com
Advisor: Prof. Dr. Volker Markl

ABSTRACT

Modern applications for business planning and performance management offer a broad range of functions for planning, forecasting, and optimising business processes. Often, they must be integrated in a planning process involving different users and objectives. However, due to the heterogeneity of their implementations and BI platform requirements, this integration becomes a tedious task. In this paper, we propose an algebra as the foundation of a common implementation and execution platform for business planning applications. We outline the major topics involved in the development of this algebra and give pointers to interesting research questions we want to address in future.

1. Introduction

During the last decade, applications for business planning and various related functionalities in the field of Enterprise Performance Management (EPM) have become an indispensable part of the decision making process of any large company. Their functionalities are valuable to improve processes, exploit synergies, and drive revenues in a fast-moving business environment. Today’s planning applications offer proprietary and heterogeneous functionality supporting tasks such as strategic and concrete planning, forecasting, and simulation of sales, resources, revenues, etc. Often, within a single enterprise, multiple planning applications are used for different planning tasks and across business units. Furthermore, many applications rely on a specific BI infrastructure. This situation brings two major drawbacks. From the user perspective, reuse of existing plans and integrated planning is a tedious task. From the software vendors’ perspective, maintenance and development efforts must address different tools individually even when their functionalities are similar.

To address those problems, future planning applications should be based on a common planning engine offering a set of core functions. In our work, we propose an algebra comprising a set of logical planning operators as the formal foundation of such an engine. The design of this algebra must address two core requirements. First, it must be highly expressive, covering a broad range of planning and related EPM functionality. Second, its operator set should be minimal, with its characteristics well-defined so as to ensure sound plan execution. Another integral aspect in the context of this work is the physical counterparts to the logical operators. To speed up the often incremental planning process, their implementation should allow for a highly efficient and fast execution of individual planning steps.

We argue that the envisioned algebra provides an abstraction from application specific planning functions and underlying BI infrastructures. As a result, maintenance and optimisation efforts can be focused on the core operators.

1.1 Example Scenario

To give an idea of what we describe in the following as planning functionality, Figure 1 introduces a very basic use case featuring some aspects of planning. In the remainder of this paper, we will use this as a running example to illustrate our ideas.

The planning scenario illustrated here could be part of the sales planning of a car manufacturer. For example, a manager might want to evaluate unit sales for passenger cars in different regions of Germany and adjust quotas for the year 2009 in compliance with a specified goal, e.g., targeting a sales increase of 10% for underperforming regions, and 5% for regions that met this year’s sales target. Alternatively, he could apply a forecast function to derive the probable sales growth based on past performance in each region. Such a function could also discover seasonal trends or fluctuations in the sales amounts of previous years and incorporate them in the 2009 quotas.

Figure 1: A planning scenario
The desired result in this scenario is a plan containing sales quotas for the year 2009 for the North, East, South, and West of Germany. We assume that the planner has access to sales data of every relevant car dealer in Germany. He must process this data in several steps. First, relevant data must be selected, i.e., only data where year == 2008, product group == passenger cars, and country == Germany is used as basis for planning. Then, the values are summed up per region to achieve the desired granularity. After comparing the recorded and planned sales for 2008, the planner updates the quotas for 2009 according to the specified rule, resulting in a 5% increase for the North region, and a 10% increase for the quotas of all other regions. This data constitutes the result plan. In Section 3.3 we will describe those planning steps in a formalised fashion.

In the remainder of this paper, we first introduce the reader to the general concepts of business planning and the broader context of Enterprise Performance Management (EPM) in Section 2. We will outline and exemplify their application scope and core functionalities. In Section 3 we describe how the envisioned planning algebra fits into a planning application infrastructure and explain some of our initial ideas for the definition and composition of logical planning operators. Section 4 briefly addresses potential methods for increasing the efficiency of planning process execution. We give an overview of relevant previous research in Section 5 and finally conclude our ideas and describe directions for future research in Section 6.

2. BUSINESS PLANNING APPLICATIONS IN THE EPM PROCESS

There is no clear definition of what constitutes business planning applications as such. Initially, we want to keep a broad scope and embrace in our considerations a whole range of functionalities from the field of EPM. This includes, e.g., standard planning of measures and key figures, but also technologies for elaborate forecasting, what-if analysis, and specialised planning and optimisation approaches such as Activity Based Costing (ABC).

We want to reemphasise our motivation for a common planning algebra by giving a brief outline of the steps involved in an EPM process, as illustrated in Figure 2. Consider as the first step in this cycle the creation and prioritisation of (high-level) business objectives and strategies for their achievement. Such an objective can, for example, be the decrease of expenditures or an increase of product quality. Based on such strategic decisions, the next step in the cycle consists of planning and executing business processes against the qualitative goals associated with the set objectives. For example, to achieve the goal of decreasing departmental costs by some percentage, headcount, travel costs, etc. must be planned accordingly. The actual outcome of the planned processes is then monitored, consolidated, and analysed. Based on the resulting data, the fourth step involves modelling and simulation of different business scenarios with the goal to further optimise process outcomes. Finally, in response to the given business situation and challenges, the cycle starts anew with the setting of strategic goals.

Note that the described steps are partially featured in our very basic example: First, we analyse sales data recorded for the year 2008. Then, having set the goal of increasing overall sales, we plan increased sales quotas for 2009 for each of the considered regions.

![Figure 2: Steps within an EPM cycle](image)

Clearly, the concrete characteristics of such a complex process as is the EPM cycle can vary considerably, involving different tools for strategy management, planning, consolidation, and forecasting - as well as responsible employees from different units and levels in the company hierarchy who use those tools. Consequently, one can differentiate involved applications along several dimensions. They can differ, for example, with respect to the business discipline where they are employed (e.g., financial, production, or sales planning), the applied techniques (e.g., ABC, strategy planning, what-if analysis), or the targeted user group and roles (e.g., power users or casual users). Furthermore, applications can be applicable to different stages within the planning process, covering short-, medium-, and long-term planning horizons or different directions of planning (i.e., top-down or bottom-up).

The data and knowledge underlying the described process comes in the form of multidimensional and relational data, as well as quantitative and qualitative metrics, business rules, and dependencies between business processes and their parameters. To enable the EPM process, applied tools need to access and “understand” this data and different tools must be integrated such that the results of one step can be reused in the next step. Though in many cases the applied model of multidimensional data is similar, this integration can become a big challenge due to sometimes subtle inconsistencies. For example, the underlying data might be organised with respect to slightly varying criteria or certain properties might be irrelevant in one tool but essential in another. Also, requirements for unit and currency translation can vary, and definitions of business rules and key performance indicators might have to be mapped between tools.

Planning applications nowadays enable users to implement customised planning processes in varying degrees of flexibility, usability, and integratability with other tools. Plans can be assembled by combining simpler functions in sequences, or complementing them with custom formulas or function calls. The core of most such tools, however, is implicitly built by a relatively stable set of operations building
the “common denominator” of planning functions, such as:

- **Copying** and moving plan figures within the plan
- **Revaluating** figures by applying a function, e.g., increasing figures by a given factor
- **Distributing** a value to a number of target items, e.g., over a given time interval, based on some reference to existing data
- **Forecasting** figures from one period to another using a specified forecasting model

In spite of the considerable functionality overlap, the implementation of those functions differs widely between tools and their integration can become a tedious task, as described above. In the next section, we lay out our initial concepts for the proposed planning algebra and describe how its application can increase efficiency of planning application development, use, and integration.

3. A PLANNING ALGEBRA

The central idea of the proposed algebra is to provide an abstract description formalism for developers to define arbitrary planning functionality. Figure 3 illustrates how this approach fits into an overall planning application infrastructure. The architecture outline shows how different planning applications are implemented on top of a common planning engine. The planning algebra forms the central layer of this engine. It comprises sets of logical and physical operators, and specifies how the former are mapped to, or are implemented by, the latter. The layer below realises the implementation of the physical operators, preferably applying parallelisation techniques. The layer above the algebra layer provides an interface to applications or users. This could be a language for specifying planning expressions based on the underlying algebra. Alternatively, a graphical notation could provide an intuitive means of supporting users in the planning process.

In the remainder of this section, we describe the data structure over which our algebra is defined, and introduce an initial set of core planning operators.

### 3.1 Data structure

In the scenario described in Section 1.1, we introduced some important aspects of the data used for planning. This data exhibits a multidimensional structure in that it contains a set of figures (e.g., sales units) that are recorded, analysed and planned along various dimensions (e.g., the time at which sales were made). This structure resembles the standard data cube definition from Online Analytical Processing (OLAP) first introduced in [5]. We therefore consider the data cube an appropriate construct in the context of planning. By nature, planning always involves analytical steps to some extent, be it the basic computation of aggregates or the application of forecast techniques. However, planning furthermore exhibits special requirements not sufficiently addressed by the standard OLAP cube model:

- Creation and modification of data is a crucial aspect of planning — indeed, ultimately it is the goal of every planning process as such. Writing new data back to store is therefore a decisive requirement for planning as opposed to mere analytical applications.
- Planning is an inherently iterative and cooperative process in which newly created data often serves as input to further planning steps. To ensure traceability and consistency, the applied data structure should contain information about the source and “reliability” of data at hand. For example, if data comes from an integrated source such as an ERP system, we would be rather confident in its quality. On the other hand, data derived using, e.g., a forecast strategy is not perfectly reliable. Such information about uncertainty and lineage of data should be traceable throughout the planning process.
- The dimension of Time plays a central role in the planning process and therefore deserves special consideration when representing planning data. Writing forward data in time based on past, present, and future achievements and goals is the focus of planning. Many functions, such as forecasting based on time series, are defined specifically as a function over time.
- The concept of changing dimensions bears special relevance in planning. Restructuring of company units, product portfolios, etc., can be both trigger and intent of a planning process — or even a means of analysis applied within such a process. Therefore, we must enable such changes and adequately consider them in querying and writing planning data. Note that this also raises the need to handle time on a meta level (When have planning structures changed?), as opposed to the level of plan data itself (When was a sale made?).

In the following, we differentiate the (abstract) specification of a cube’s structure from its (concrete) instances, that is, the actual materialisation of a cube. We adopt concepts from [13], illustrating each of them using our running example. We denote the cube specification \(CS = (D, L, M, h)\), comprising:

A set of dimensions \(D = \{D_1 = D_T, D_2, \ldots, D_n\}\) where each \(D_i, 1 < i < n\) denotes a dimension along which the data can be analysed. We use \(D_T\) to explicitly accommodate the prominence of the (mandatory) time

![Figure 3: A layered planning engine architecture](Image 60x137 to 286x316)
A set of dimension levels $L = \{L_1, L_2, ..., L_m\}$ where each $L_i$, $1 < i < m$ denotes an attribute describing a dimension at some level of granularity. We use $\text{dom}(L_i)$ to refer to the domain, i.e., all possible values, of $L_i$.

A set of measures, $M = \{M_1, M_2, ..., M_p\}$ where each $M_i$, $1 < i < p$ represents a figure to be analysed and planned. We use $\text{dom}(M_i)$ to refer to the domain of $M_i$. For our example we define $M_S = \{\text{sales}\}$.

A mapping function $h : D \rightarrow L$ mapping each dimension $D_i$ to a corresponding set of levels, which are mutually disjoint, i.e., $v_i$, $j$, $i \neq j : h(D_i) \cap h(D_j) = \emptyset$. For the scenario, we define $h_S(\text{Time}) = \{\text{day}, \text{month}, \text{year}\}$, $h_S(\text{Location}) = \{\text{store}, \text{city}, \text{region}, \text{country}\}$, $h_S(\text{Product}) = \{\text{product}, \text{p_group}\}$ and $h_S(\text{Version}) = \{\text{version}\}$. Furthermore, an ordering $<_{v_i}$ is defined on $h(D_i)$. $<_{v_i}$ reflects a dimension hierarchy path, with attributes higher in the path describing a coarser granularity of the dimension $D_i$. For example, for dimension $D_i = \text{Time}$, the ordering $\text{day} < \text{month} < \text{year}$ applies. For the sake of simplicity in the context of this paper, we assume there exists only one hierarchy path for each dimension. We will need to consider the handling of several hierarchy paths in future work. Below, we use a function $\text{levels}_h(D_i)$ to retrieve the order number of a level in its respective hierarchy. For example, $\text{levels}_h(\text{day}) = 1$.

Based on the definition of $CS$, we can create an actual cube instance\footnote{For brevity we will use “cube” to refer to a cube instance from now on.} complying with the structural characteristics defined by $CS$. A cube $C$ is defined as a 5-tuple $(D_C, L_C, M_C, C_h, R_C)$, where $D_C, L_C$, and $M_C$ define the structure of the cube, i.e., the sets of included dimensions $D_C \subseteq CS.D$, dimension levels $L_C \subseteq CS.L$, and measures $M_C \subseteq CS.M$. Each $L_i \in L_C$ describes the hierarchy level of dimension $D_i \in D_C$ at which the data is currently reflected, $\forall L_i \in L_C, L_i \in h(D_i)$, $C_h$ refers to the base cube. That is, $C_h = (CS.D, L_h, CS.M, C_h, R_h)$ reflects all dimensions and measures defined by $CS$, $\forall L_i \in L_h, \text{levels}_h(L_i) = 1$, and $R_h$ contains the underlying data in its finest granularity. We must keep a reference to $C_h$ to calculate aggregated data for $C$. Finally, $R_C$ is the set of cell data of $C$, containing for each cell $c$ in the cube a tuple $c = (d_1, ..., d_n, v)$, where $\forall i \in [1, ..., n], d_i \in \text{dom}(D_i)$ and $\forall j \in [1, ..., p], m_j \in \text{dom}(M_j)$. For ease of notation we denote $\text{dimensions}(c) = [d_1, ..., d_n]$ and $\text{measures}(c) = [m_1, ..., m_p]$. Note that below we use the term cube also to refer to an “empty” cube, where only the structural characteristics, but none of $C_h$ and $R_C$, are assigned.

At the present point, we rely on this model to describe our first ideas. Incorporating constructs for handling of changing dimensions (e.g., [2]), as well as lineage and uncertainty information (e.g., [12]), is an important point for future investigations.

3.2 Planning Operators

As the basis for our planning algebra, we need to evaluate planning functionality at its lowest granularity and factor out the core functionalities to be encapsulated in atomic operators. Each such operator takes as input cube instance(s) $C_i$, and possibly additional parameters, and outputs a cube instance $C_{out}$. The resulting operator set should be such that it is both complete, i.e., all desired functionality can be described as expressions over the algebra, and minimal, i.e., there should be no redundancy in the operator set.

The standard operators defined on the OLAP data cube, such as Slice, Dice, Drill-Down and Roll-Up, can serve as a starting point for our algebra. Once again, we need to extend this set to duly address the specific characteristics of planning. This concerns the modification of existing data, the incorporation of forecast functionality, and the storage of resulting plan data.

The following shows an initial draft of a possible set of planning operators. After operators used to LOAD and STORE cubes, we list SLICE, DICE, ROLLUP, and DRILLDOWN as standard OLAP operators. Following those, the operators ASSIGN, MAP, PAIR, and FUNCTION focus on modification of plan data.

We have derived this first set of operators from evaluations of existing planning applications and possible application scenarios. We hold that the captured functionalities can be covered applying the listed operators. Please note, however, that this reflects our preliminary ideas only, with operator descriptions still at a conceptual level and not properly formalised. The granularity at which those operators should be defined in order to achieve both a complete and minimal operator set is not yet clear. Also, up to now, we have not considered how the handling of changing dimensions or uncertainty could be reflected in our algebra.

LOAD :: $C_0 \times F \rightarrow C$ loads a cube $C$ applying selections on dimensions and dimension values, based on a given cube definition $C_0 = (D_0, L_0, M_0, \emptyset, \emptyset)$ and a list of conditions $F$. Each $f_k \in F$ is a tuple $(D_i, O_{f_k}, v_i)$, $D_i \in D_0$, $v_i$ being a value range from $\text{dom}(D_i)$, and $O_{f_k} \in \{<, >, =\}$ being an operator applicable to $\text{dom}(D_i)$. The result cube is $C = (D_0, L_0, M_0, C_h, R_C)$, where all cells $c_j \in R_C$ comply with the conditions $f_k \in F$, i.e., $\forall f_k \in F, c_j \in R_c, c_j[i] O_{f_k} v_i = true$.

STORE :: $C \rightarrow -$ writes a given cube $C$ back to store.

DICE, SLICE, ROLLUP, and DRILLDOWN are defined similar to their homonymous OLAP counterparts. For space restrictions, we only briefly describe their functionalities here. More details can be found in related work about OLAP (see, e.g., [9]). The DICE operator is used to restrict cells of a cube by applying a selection condition $f$ similar to the one defined for the LOAD operator. SLICE is used to cut out a dimension of the cube, aggregating all measures over all values of that dimension. ROLLUP decreases the granularity of a cube by aggregating cell values along some dimension’s hierarchy path (i.e., increasing the corresponding dimension level), while its inverse operator DRILLDOWN increases the granularity of a cube along a dimension.

ASSIGN, in contrast to the previously listed operators, serves to actually modify cube data by assigning a new
value to a given dimension attribute for all cells in a cube. ASSIGN :: $C_1 \times a \rightarrow C_2$, where $a$ is a tuple $(D_a, v)$, assigns a value $v$ to the current attribute level of dimension $D_a$ of a given cube $C_1$ such that the result is cube $C_2 = (D_1, L_1, M_1, C_0, R_2)$, where $R_2 = \{ c | (d_1, \ldots, d_{a-1}, v, d_{a+1}, \ldots, d_b) \}$. For example, if $D_a = D_T = Time$, and $v = 2009$, the value for the year attribute would be set to the year 2009 in $C_2$.

Conceptually, this means that the data in $C_1$ is moved to another “section” of the result cube $C_2$ – in our example case, all cell data is moved to the year 2009 in $C_2$.

MAP :: $C_1 \times f_{MAP} \rightarrow C_2$ applies a function $f_{MAP}$ to the measure value(s) in each cell $c_{1,i} \in R_1$, resulting in a new value for the corresponding cell $c_{2,i} \in R_2$. That is, $C_2 = (D_1, L_1, M_1, C_0, R_2)$, such that $\forall c_{1,i} \in R_1, c_{2,i} \in R_2 \rightarrow measures(c_{2,i}) = f_{MAP}(measures(c_{1,i}))$. A possible application of the MAP operator is, e.g., the revaluation of some measure by a given factor.

PAIR :: $C_1 \times C_2 \times f_{PAIR} \rightarrow C_3$ applies a function $f_{PAIR}$ to the values of corresponding cells $c_{1,i} \in R_1$ and $c_{2,i} \in R_2$ from two equal sized cubes $C_1$, $C_2$. The result cube is $C_3 = (D_1, L_1, M_1, C_0, R_3)$ where $\forall c_{1,i} \in R_1, c_{2,i} \in R_2 \rightarrow c_{3,i} \in R_3, measures(c_{3,i}) = f_{PAIR}(measures(c_{1,i}), measures(c_{2,i}))$. The PAIR operator can be applied to compute an output cube based on an evaluation (e.g., a comparison of measure values) of the cell values of two input cubes, as illustrated below in Figure 5.

FUNCTION :: $C_1 \times C_2 \times f_{GEN} \rightarrow C_3$ applies a generic function $f_{GEN}$ to the data provided in the cells of two cubes to create values for the cells of a result cube $C_3$. Unlike the MAP and PAIR operators, we do not require that structure and size of the input and result cubes are equal and, consequently, the values referenced and produced by $f_{GEN}$ are not assumed to reside in corresponding cells. Therefore, we must introduce some mechanism to ensure that $f_{GEN}$ is actually applicable to the input cubes, i.e., that the input cubes are compliant with the input specification of $f_{GEN}$.

The motive for including this yet rather conceptual operator is the intended integration of functionalities requiring more complex “parameter settings” than the previous operators. This concerns, e.g., forecast functions applying some time series model or the distribution of values based on given reference data.

We want to exemplify the intention behind this generic operator by introducing two examples. Consider the operator FORECAST :: $C_1 \times C_2 \times f_{FORECAST}(dom(D_T))$ as a special instance of FUNCTION explicitly operating over the dimension $D_T = Time$. Cube $C_2$ could contain some reference data, either given by previously recorded data (e.g., recorded sales for a ten-year time interval) or retrieved from external sources (e.g., available data about the past or current economic situation). Then, based on some forecasting model defined by $f_{FORECAST}$, a forecast could be run on the data in $C_1$ 2

2 Without loss of generality, we assume that due to the equal structure and size of input and output cubes their cells can be ordered with respect to a common key. Therefore the $i$th cells in the cubes correspond to each other.

3.3 Function composition

By composing the basic operators above, we can build up more complex planning functionality such as provided by current planning applications. Such composition can either be done by the user or by an application which in turn provides composite functions as larger building blocks to the user. Latter option can help making planning more intuitive to certain user groups, while at the same time it implies losing some flexibility in plan modelling. As already stated, we want to provide the algebra as an intermediate layer so that applications on top can use and compose functionalities as required.

Figure 4 shows how we can form a planning process made up of several steps using intermediate plan cubes. A planning language PL on top of our algebra could be used to apply several planning operators in one planning expression, constituting one step in the process. At the beginning of the process, data must be loaded from storage into a cube residing in memory. As illustrated, each intermediate cube in the process is the result of the previous planning expression and serves as input to the next planning expression in the process. Planning often is an iterative process during which the planner successively analyses alternatives and builds up a more consolidated plan. Intermediate cubes need not necessarily be stored, but are kept in memory where they serve for analysis and as starting point for further planning steps.

When the planning process has come to an end, the result cube is written to storage.

For now we assume that each planning step applies only a single operator. Figure 5 shows our example scenario as an instance of a planning process, illustrating the application of several operators (depicted as labelled arrows). Cube $C_1$ is loaded from store as defined by the shown parameterisation. $DICE1$ and $DICE2$ each restrict the value for the version attribute, resulting in cubes $C_2$ and $C_3$ containing actual and plan data, respectively. PAIR sets the new quota values based on a comparison of values from $C_2$ and $C_3$. Finally, ASSIGN sets the year value to 2009, resulting in the result cube $C_5$, which is then stored.

4. IMPLEMENTATION OF PHYSICAL OPERATORS
Once we have defined a first set of logical operators we must decide how those should be mapped to and implemented by physical operators. As stated above, the operator implementations shall be highly efficient in order to speed up the processing of planning steps. One means to achieve this goal could be the parallelisation of operator processing. We plan to investigate this aspect using existing implementations of the MapReduce programming model (e.g., [3]). Alternatively, one could exploit efficient in-memory processing exploiting dedicated server hardware, such as described in [6].

Irrespective of the concrete implementation, we must analyse how different operators are actually suited for parallelisation. That is, we need to investigate how the processing steps required to realise the functionality of each respective operator can be aligned with a parallel processing approach. Applying a MapReduce-like approach to implement a certain operator can only lead to the desired performance increase if some of its tasks and the underlying data partitions can be processed concurrently. Furthermore, the performance gain made by running operators in parallel must not be cancelled out by the costs incurred by partitioning, communication, and synchronisation.

We exemplify this aspect for two operators introduced in Section 3. First, we consider the PAIR operator, which applies a function \( f_{PAIR} \) to input cubes \( C_1, C_2 \) to produce an equally structured and sized output cube \( C_3 \), where the cell values \( c_{3,i} \) in \( C_3 \) are calculated as \( c_{3,i} = f_{PAIR}(c_{1,i}, c_{2,i}) \).

We can see that there exist no interdependencies between calculations for different cells in \( C_3 \) and that the new cell value depends only on the corresponding cells of the input cubes. Therefore, parallel execution could be very beneficial. To realise a MapReduce-like approach, we would implement 1) a map function mapping cells of \( C_1 \) and \( C_2 \) to their index \( i \), thus producing a set \( Map = \{(i, c_{1,i}, c_{2,i})\} \) and 2) a reduce function, each instance of which gets as input an equally sized partition of \( Map \) and applies \( f_{PAIR} \) to produce result values for each \( c_{3,i} \).

Now, consider the operator \( MAP \). If this operator is applied to a relatively “small” cube (i.e., a cube with a relatively small number of cells) using a “simple” (in terms of required computation time) \( f_{MAP} \) function, we might not want to parallelise its processing since the overhead of parallelisation itself could outweigh any potential processing speedup. On the other hand, if we apply a more complex function \( f_{MAP} \) or the input cube is very large, parallelisation could increase the overall processing performance considerably.

We can see that, in order to determine the suitability of different operators for parallelisation, we need to take into account several factors. Those depend both on the characteristics (and parameters) of the applied operator and on the statistics of the cubes on which it is applied. Furthermore, statistics of the underlying data and the computational properties of operators can also be valuable for optimising logical operator composition plans and their mapping to physical operators. The idea here is to apply heuristics and techniques similar to those used in database query optimisation. We have not yet investigated those issues in depth, but believe that any such optimisation could be one of the major technical benefits of the proposed algebraic approach.

5. RELATED WORK

Multidimensional modelling and algebras for OLAP have been addressed by many researchers over the last decade (e.g., [8]). The proposed algebras differ in the granularity and scope of their operators. There have been efforts to classify multidimensional models [1] and to find a common framework for a corresponding algebra [9]. To our knowledge, there have been no efforts extending the analysis-focused algebras towards an application context heavy on modification, such as planning. The same applies to the integration of forecasting technologies in such an algebra. Temporal OLAP [12] addresses the problem of changing dimensions and the consequent need for time-dependent querying of data cubes. Probabilistic database systems such as Trio [2] consider the problem of representing and handling uncertainty and lineage information in relational data. Probabilistic OLAP [7] addresses a similar problem in the multidimensional domain. This work could serve us as valuable input for the integration of uncertainty or lineage information in our algebra. Multidimensional query languages [8] such as MDX [10] have been developed to ease the analysis of multidimensional data. As related in Section 3, a planning language on top of our algebra could be built by extending such an existing language.

Today, many systems use parallel, in-memory data processing (e.g., [6, 4]) to speed up OLAP queries, but do not offer efficient functionalities for modification and write access.
to multidimensional data. The MapReduce programming model (e.g., [3]) can be applied for efficient parallel processing and generation of large datasets. Existing implementations such as Hadoop [11] can build the basis for an efficient implementation of our physical planning operators.

6. CONCLUSION AND NEXT STEPS

This paper marks the starting point for our research on an algebraic planning foundation. We substantiated the need for an integrating planning algebra due to the functional overlap and implementational heterogeneity of business planning applications. We then described our first ideas for its design and illustrated how its operators can be applied to compose functionalities in a planning process. Finally, we briefly addressed some aspects of efficient plan execution before giving pointers to relating work. To conclude the paper we now describe important open issues and potentially related topics for our future work, some of which we have already addressed throughout this paper.

- **Consolidate scope and applicability of the planning algebra** As stated before, the range of planning applications considered in this work covers different, often overlapping, aspects of EPM, including standard planning functionality, forecasting, and specialised planning techniques. We have indicated some of the integration challenges that occur in an EPM cycle. To fully understand which functionalities our algebra shall cover and how it can be applied in concrete business scenarios, we still need to thoroughly investigate such scenarios and the tools currently applied therein. Also, we want to achieve a deep insight into the technical challenges through discussions with application experts and developers.

- **Defining a sound set of core operators** Based on the identified requirements, we must define a minimal set of well-formalised core operators that provide the required expressivity. We need to further evaluate and formalise characteristics of the identified operator set, such as operator parameterisation and the computational properties of the algebra.

- **Integration of forecast functionality** Enabling the generic integration of forecasting techniques with the proposed algebra constitutes an interesting topic for our future research. So far, we have addressed this issue merely by providing a general description of the FORECAST operator. As a next step, we need to evaluate how such an operator can be formalised, and which restrictions the underlying data must comply with.

- **Parallel execution and optimisation** We have briefly described our ideas to speed up planning task execution by applying techniques such as parallel processing of operators. As a next step, we want to evaluate the realisation of operators based, e.g., on an implementation of the MapReduce approach, such as Hadoop [11]. To pursue this idea further, we need to analyse how well different operators can be parallelised, and how the parallelisation can be automated based on the characteristics of operators and underlying data. A derived cost model and execution statistics could then be applied to heuristically optimise plan execution.

7. REFERENCES


