IB-CUE – Theoretical Concept and Implementation Aspects for a Software-based Information System for Innovative Budgeting in a Competitive University Environment

Markus Aleksy[1], Ingo Bayer[2], Tilo Dickopp[1], Axel Korthaus[1]
University of Mannheim, Schloss, L 5,5, 68131 Mannheim, Germany
[1]Department of Information Systems, [2]School of Business Administration, Dean's Office
{aleksy|dickopp|korthaus}@wifo3.uni-mannheim.de
bayer@bwl.uni-mannheim.de

Abstract

In our paper, we first describe a new conceptual approach to budget planning and allocation on the schools' level, which promotes autonomy and goal-orientation and provides incentives in order to optimize the performance and innovation outputs. Subsequently, we focus on the design and implementation of a software application system, which realizes computer support for this approach. An object-oriented UML model and the discussion of selected design decisions provide background information about a stand-alone prototype implementation. We provide an analysis of different aspects concerning a layered architecture for the final software product and the evolution of the prototype to become a distributed application.

Keywords: university budgeting information system, distributed object-oriented systems, object-oriented analysis and design.

Povzetek

IB-CUE: teoretična podlaga in različni vidiki realizacije informacijskega sistema za podporo inovativnega proračunskega načrtovanja v tekmovalnem univerzitetnem okolju

V članku najprej opišemo nov pristop k proračunskemu načrtovanju (načrtovanju denarnih pretokov) ter razdelitvi sredstev na ravni posameznih šol (fakultet). Pristop podpira avtonomijo in ciljno usmerjenost ter vsebuje spodbude za doseganje optimalnih rezultatov ter inovacij. Nato se osredotočimo na programsko opremo za računalniško podporo takega sistema. Prototip informacijskega sistema je realiziran kot izoliran (stand-alone) sistem, njegove lastnosti pa opišemo na podlagi objektno usmerjenega UML modela ter posameznih razvojnih odločitev. Sledi analiza različnih vidikov razslojene arhitekture končnega programskega produkta ter evolucija prototipa v smeri porazdeljene aplikacije.

Ključne besede: informacijski sistem za načrtovanje proračuna univerze, porazdeljeni objektno usmerjeni sistemi, objektno usmerjena analiza in načrtovanje.

1 Introduction

Today, state-owned German universities are facing major challenges due to dramatic changes in their environment. The devastating status of public finances as well as intensifying competition in all fields of education forces them to rethink their strategies and to adapt swiftly to these challenges, because only those who are able to "ride before the wave" of forced reforms imposed by the government have a chance to define their own future ([18], [19]). Therefore, they have to assure that their declining financial resources are not only allocated in an efficient way, but also in a way that advances the achievement of the university's goals as a whole. In our paper, we sketch the core ideas of an innovative budgeting model on the level of a school inside a university, because this is actually the place where the university's resources are transformed into teaching and research as the two major outputs of the university [2]. An in-depth discussion of design and implementation considerations for a software application system implementing the new approach follows in the second part of the paper.

2 Concepts for an innovative budgeting system

The budgeting system is based on an internal school organization, in which the chairs of the school belong to different departments, as, for example, Department of Accounting, Department of Management, and so on. The departments are responsible for teaching and research in their field of specialization, and the heads of the departments negotiate contracts with the dean's office about the output that has to be delivered to the school. There is a mutual under-standing that every department has to carry a part of the basic burden the school has to carry for the university as a whole. In addition to this basic output, every department is also expected to strive for innovations in research and teaching, which will help the school to excel as a competitor for public and private funds [7].

Thus, the budgeting model is based on the assumption that it should be useful in a threefold way. Firstly, it should enable the faculty members to carry out the basic workload the school is obliged to deliver in general. Secondly, it should reward those faculty members who perform in a way that enables the school to reach its more ambitious goals, and, last not least, it should boost innovation inside the school. Along these different goals, the budget model consists of three major pillars, which are:

- a Basic Budget,
- a Performance Budget,
- an Innovation Budget (cf. Figure 1).

The *Basic Budget* enables the school to meet basic operational requirements and provides resources for the general infrastructure, staff rooms etc. It can also reflect different performance levels of departments and be used to counteract the cementation of these differences due to different initial resources.

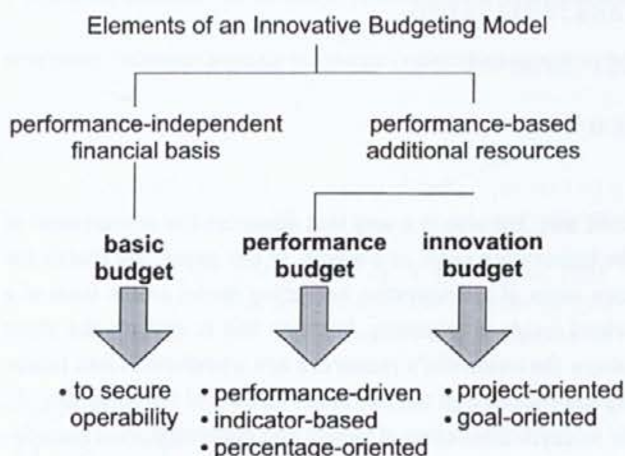

Figure 1: **Elements of an innovative budgeting model**

The *Performance Budget* rewards the faculty in accordance with the percentage of the overall workload of the school the individual faculty member or the individual department carries. This budget is distributed based on performance indicators that reflect the performance indicators on the university level, which are directly responsible for the budget volume of the school as a whole. In addition, performance budget distribution can also account for indicators that reflect the strategic goal settings of the school, so that goal-orientation is strengthened within the whole school.

The *Innovation Budget* has an "incubator function" for the school. It is used to stimulate innovative projects, which are in congruence with the school's strategy, like research projects, new teaching models etc.

To be as efficient as possible, administrative tasks such as budget planning activities have to be supported by adequate software-based information systems. In the remainder of the paper, we outline some basic aspects of the development of a software application system that can support the new budgeting approach explained above. IB-CUE ("Innovative Budgeting in a Competitive University Environment") is the acronym we chose as a name for this information system, which is currently being developed at the University of Mannheim. However, a working prototype of the application already exists.

Since the focus of this paper is to describe this prototype and to present further considerations for implementing a fully-fledged software solution to support our theoretical approach, we do not go into more detail about the budgeting methodology here. Readers who are interested to learn more about the business administrative concepts behind our approach are referred to [1].

3 Considerations for the development of the IB-CUE software application

When we planned the development process of IB-CUE, we identified two core milestones that could help to enforce an incremental approach that supports separation of concerns and step-wise refinement:

1. implementation of a stand-alone prototype providing the business logic, and
2. migration to a distributed software system.

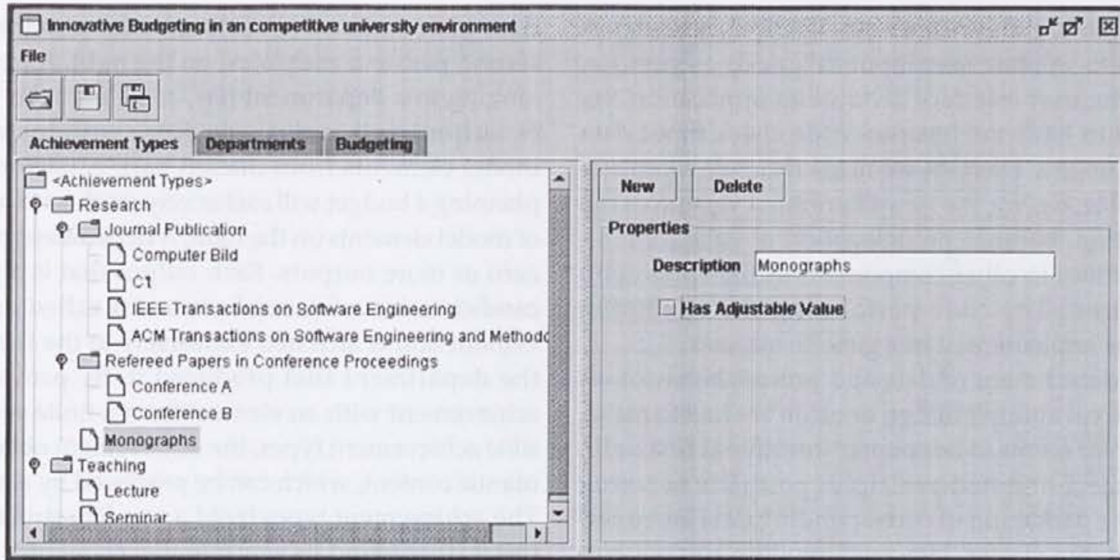

Figure 2: **Screenshot of the prototypical IB-CUE software application**

Starting with a prototypical implementation that only focused on the business logic of the budgeting application and deferred technical considerations until later had the big advantage, that many of the complex questions and design decisions of software development could be disregarded at first. By initially focusing on the functional aspects only, it became easier to provide a correct mapping of the requirements (defined by the concepts of the budgeting model) to the program code implementing the business logic. However, when designing the prototype, we kept in mind to encapsulate the business logic using clearly defined interfaces in order to allow for a layered approach during the evolution of the system. This software engineering method is called "Design by Contract" [10]. Thus, it will be possible to reuse the core business logic together with different middleware technologies for distributed communication or different clients for user interaction, for example. As mentioned before, this first step was already completed, and a Java-based prototypical implementation of the budgeting approach is available since December 2003. Figure 2 shows a screenshot of the prototype's GUI.

A core task of the second phase of the development process will be the consideration of distribution aspects for the application. Since a school and its departments have a distributed organization structure, and faculty members should be able to enter their performance achievements into the system by themselves, it is mandatory that all people who are involved in the budgeting process must be able to work with the application simultaneously, without having to leave their offices. This requirement, which is crucial for user acceptance of the software, cannot be met by a monolithic stand-alone version of the application. For this reason, a suitable middleware technology has to be selected that provides the technical foundation for the collaboration of distributed application components.

4 Design and Implementation Aspects of the IB-CUE Prototype

In order to capture the functional requirements of the IB-CUE prototype, we performed object-oriented analysis and design activities using the Unified Modeling Language (UML) ([12], [13]). The resulting UML models served as a foundation for the implementation of the prototype in Java. During analysis and design, we had to make several decisions with respect to potential architectural options. For example, we attached great importance to the encapsulation of the functional business logic code by applying a set of design patterns and best practices [6], such as the "façade pattern" or "single point of access" paradigm. This modularization facilitates the reuse of the business logic and the distribution of the application in the next evolutionary step.

Another design decision referred to the question of where user input data is to be checked for correctness with respect to the business rules, before it is fed into

the budgeting algorithms etc. While it is common practice to validate user input as quickly as possible, i.e., in the user interface layer of an application, we decided to have the business code check input data for plausibility. Thereby, we made the code completely reusable, e.g. for the new distributed version of the application, because no delegation of validation responsibilities to other components of the application is required. The code performing the validation checks is implemented in a generic manner.

The detachment of data and process behavior represents yet another design decision we had to make. This choice seems to be counter-intuitive at first sight, since object-oriented principles postulate cohesion and close packaging of corresponding data and operations within classes. However, modern component-based approaches like Enterprise JavaBeans [5] etc. with their session-oriented and entity-oriented components recognize the benefits of separating data and behavior on a higher level of abstraction. Common design patterns such as the "value object pattern" or the "transfer object pattern" also promote the factoring out of data-centric classes. By separating these data classes, which are used by the business logic and also by the user interface (or the network layer in a future version of IB-CUE), we guarantee that new data format requirements resulting from new application components do not affect the code implementing the core business processes and rules of the system. E.g., since varying marshaling capabilities of different middleware technologies might result in different data type requirements, our approach can minimize the impact of switching the middleware technology.

In the remainder of this section, we would like to provide a detailed description of the design of our prototype, which was implemented in Java. Since the business logic was proven to be correct, other layers of the planned software architecture for the final software product will be implemented using the AspectJ technology [9]. For an overview of the architectural layers required for this system see section 5.

The first prototype mainly realizes the business logic layer, which will be reused in the implementation of the final software product. Therefore, we should have a detailed look at how this layer was designed. The business logic model consists of two parts, as can be seen in Figure 3: Classes that represent departments and their output make up the first part and are placed on the left side of the figure. Classes that deal with planning and allocating budgets make up the second part and are placed on the right. A person belonging to a department (i.e., someone who produces output) will exclusively work with instances of model elements from the left part. A person who is planning a budget will exclusively work with instances of model elements on the right. A department produces zero or more outputs. Each output that is a possible candidate for a financial reward is called an achievement and is attached exclusively to the instance of the department that produced it. By providing an achievement with an element from a finite set of possible achievement types, the achievement obtains a semantic content, which can be processed by a machine. The achievement types build a tree-like structure, but this feature found its way into the model only for convenience considerations and is not mandatory. The well-known "Composite" design pattern has been used to model this structure [6]. As the researcher is responsible for determining the type of an achievement, the instances of the AchievementType subclasses should be named in accordance with concepts that he can understand, e.g. "publication in journal XY", and not "publication in an A class journal".

The budgets are divided into different groups along three dimensions:

The "assignment dimension"
Budgets may or may not be assigned to a (not empty) set of departments. Unassigned budgets function as aggregate budgets and, therefore, they must have children, which either are assigned to departments or have children themselves. This is another application of the aforementioned "Composite" design pattern [6]. In a correct model, every budget has to be attributable to one or more departments either directly or indirectly through its child budgets. Furthermore, the planner can assign budgets to departments either manually, or they are assigned to a department automatically, because the department fulfils certain goals, i.e., it has achievements of certain types, which trigger the assignment of this budget. Typically, basic and innovation budgets will be assigned by hand, whereas performance budgets will be assigned automatically according to generally defined achievement goals.

The "amount type dimension"
Each budget has an associated rule that is used to calculate the amount for this budget. It should be noticed

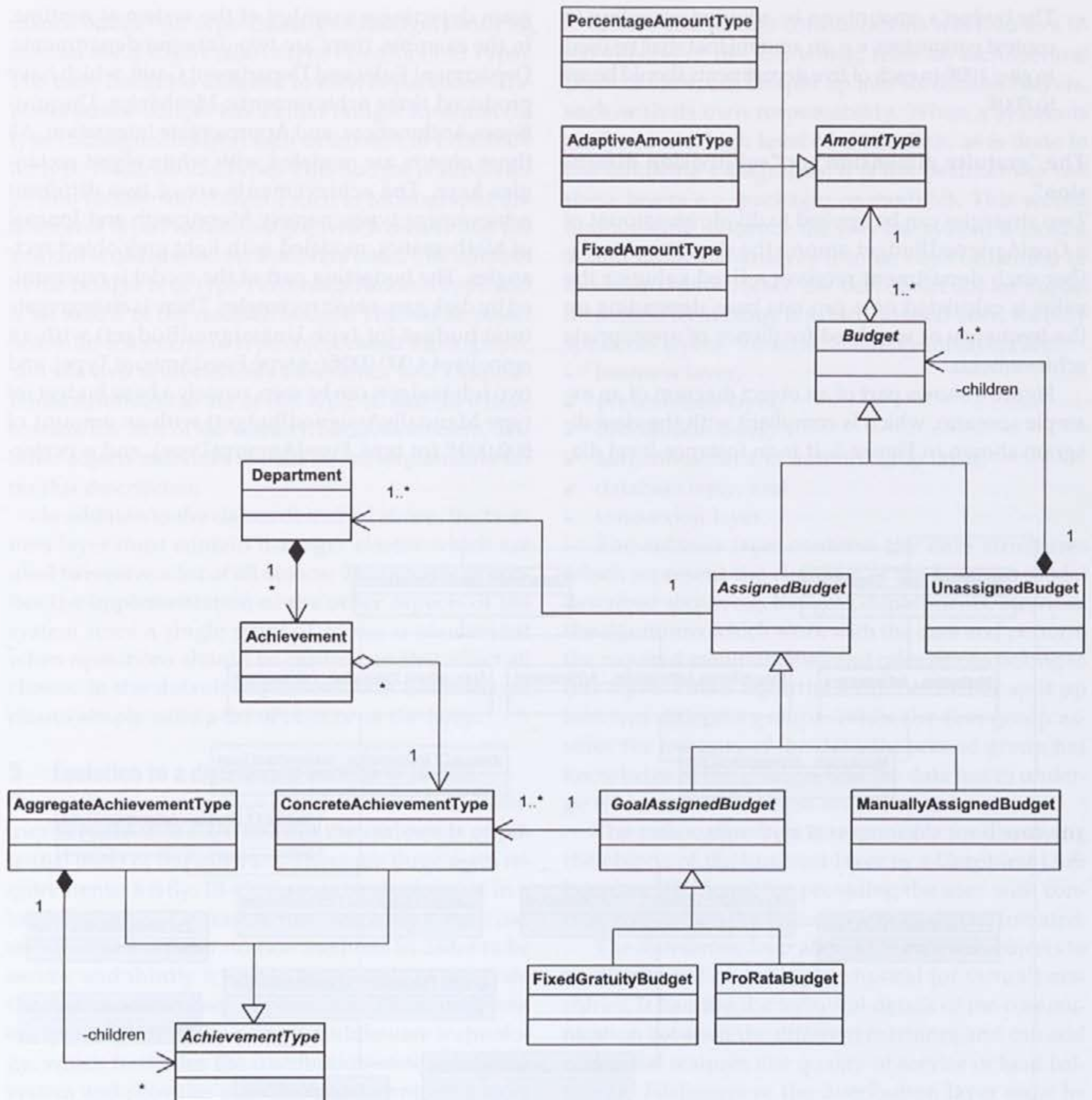

Figure 3: **UML class diagram of the budgeting system problem domain**

that more than one department might share the amount of a budget, so that an adequate partitioning scheme must determine the amount granted to each department. There are three kinds of rules which can be used to achieve this:

- The budget's amount can be a fixed value. The root budget, which is the aggregate of all budgets, may only be set based on this kind of rule.
- The budget's amount can be a fraction of the amount of its parent budget, specified in percent.

- The budget's amount can be adjusted according to context parameters, e.g. an amount that shall be used to give 100€ to each of five departments should be set to 500€.

The "gratuity dimension" or "subdivision dimension"

Two strategies can be applied to divide an amount of a GoalAssignedBudget among the departments. Either each department receives a fixed value or the value is calculated on a pro rata basis depending on the frequency or weighted frequency of appropriate achievements.

Figure 4 shows part of an object diagram of an example scenario, which is compliant with the class diagram shown in Figure 3. It is an instance-level diagram depicting a snapshot of the system at runtime. In the example, there are two different departments, Department Euler and Department Gauß, which have produced three achievements: Mechanica, Disquisitiones Arithmeticae, and Approximate Integration. All those objects are modeled with white object rectangles here. The achievements are of two different achievement types, namely Monograph and Journal of Mathematics, modeled with light grey object rectangles. The budgeting part of the model is represented by dark grey object rectangles. There is an aggregate total budget (of type UnassignedBudget) with an amount of 4.000.000€ (of type FixedAmount Type), and two sub-budgets can be seen, namely a basic budget (of type ManuallyAssignedBudget) with an amount of 500.000€ (of type FixedAmountType), and a perfor-

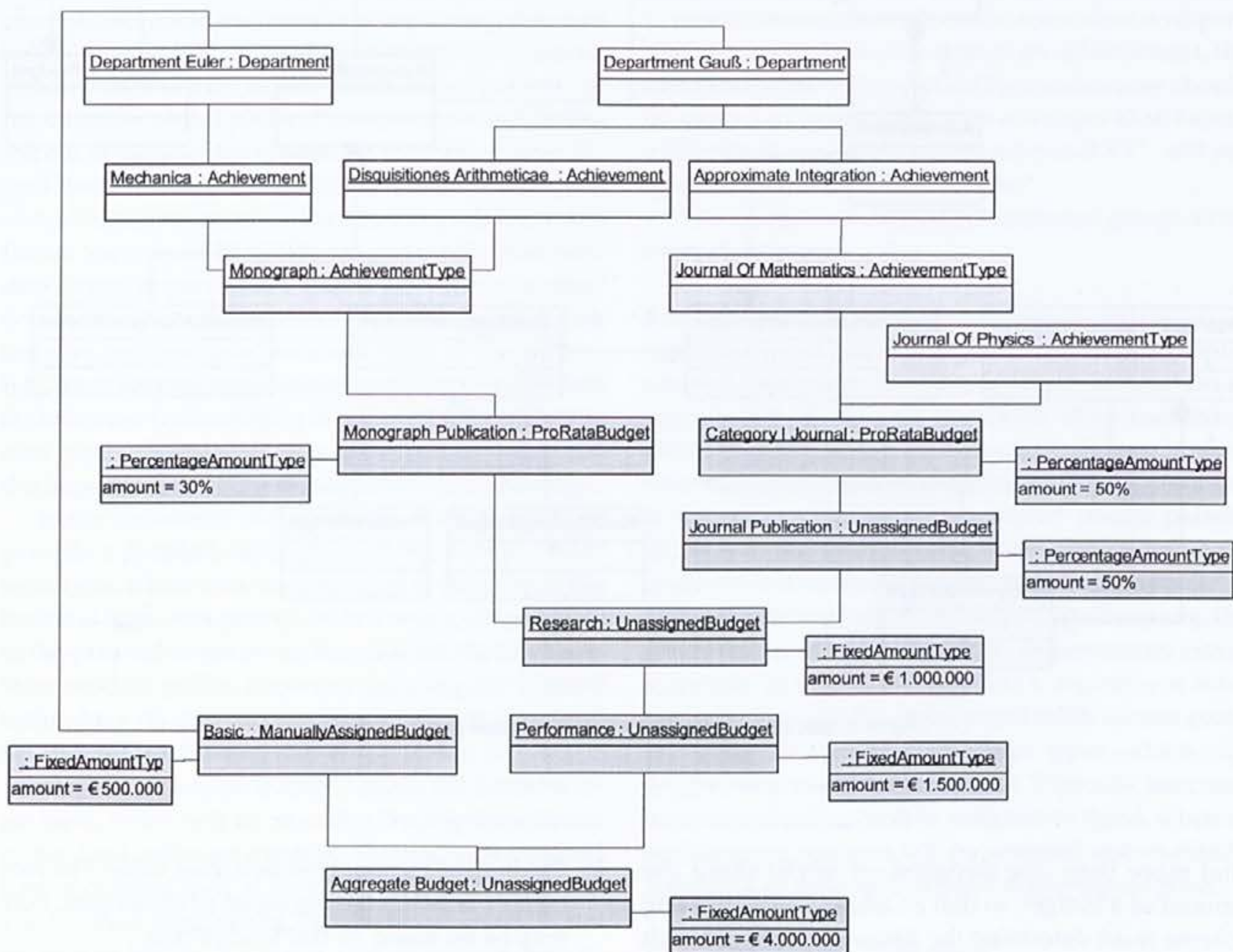

Figure 4: **UML object diagram of an example scenario**

mance budget (of type UnassignedBudget) with an amount of 1.500.000€ (also of type FixedAmount Type). The basic budget is assigned to each department. The performance budget has a child budget Research (of type UnassignedBudget) with an amount of 1.000.000€ (of type FixedAmountType). This budget is subdivided into further sub-budgets, such as MonographPublication of type ProRataBudget, which means that the amount is distributed on a pro rata basis. The amount of the budget is of type PercentageAmountType and is set to 30% of the research budget. There is an object link to an achievement type object called Monograph, and the two achievements Mechanica and Disquisitiones Arithmeticae are of this type, so that they have to share the 30% of the research budget's amount. The other objects and links should be self-explanatory after this description.

In addition to the classes described above, the business layer must contain manager classes which are used to receive a list of all objects. This greatly simplifies the implementation of the other aspects of the system since a single point of access is convenient when operations should be carried out that affect all classes. In the default implementation the manager classes simply store a list of objects on the heap.

5 Evolution to a distributed version of IB-CUE

Currently, we are working on a distributed, multi-user version of IB-CUE to meet the real needs of potential users of the software. There are three main requirements: firstly, IB-CUE must be deployable in a heterogeneous IT infrastructure, secondly it must use standardized communication methods in order to be secure, and thirdly it should be possible to integrate the system with legacy applications. These problems are typically solved by using a middleware technology, which facilitates the distribution of the software system and provides standardized technical services for common infrastructure problems. It is obvious that the correct choice of this technology is crucial to the overall robustness and maintainability.

In order to be able to use a middleware technology at all, we encapsulated the business code into well-defined components. So, whatever method is chosen for distribution, it will result in network components functioning as proxies [6] for the business components, which do the actual work. This way, the code which is proven to be correct (cf. above) does not need to be changed.

Those and further considerations will lead to a final version of IB-CUE which, from an architectural point of view, can be split up into six different layers, each with its own responsibility. When a system is analyzed at a high level of abstraction, as is done in the following paragraphs, it is not pertinent to call these layers e.g. packages or modules. This would undoubtedly influence the way the system is looked at and induce to analyze it in an object-oriented or modular fashion. Since the best choice for an implementation technology is to be discussed later, we only speak of "layers" for now. The relevant layers are:

- business layer,
- presentation layer,
- distribution layer,
- authentication and authorization layer,
- database layer, and
- transaction layer.

The *business layer* contains the data structures which represent the elements of the business model described above, e.g. budgets, departments, etc. Also, the algorithms which work with the data and perform the required manipulations and calculations belong to this layer. These algorithms can be further split up into two different groups: While the first group assures the integrity of the data, the second group has knowledge of the changes that the data has to undergo when certain business actions are taken.

The *presentation layer* is responsible for displaying the objects of the business layer in a Graphical User Interface (GUI) and for providing the user with controls with which the business actions can be initiated.

The *distribution layer* allows the business objects to be distributed on multiple physical (or virtual) machines. It handles the technical details of the communication between the different machines and can add additional features like quality of service or load balancing. Furthermore the distribution layer must be able to communicate with legacy systems in order to integrate data which is hold there into the application.

The *authentication and authorization layer* governs a list of users who have access to the system. The permission to perform certain actions can be granted to or removed from a user at this layer.

The *database layer* is responsible for long-term persistence of the business objects in a database.

The *transaction layer* protects the system from damage which could arise from inconsistent data due to concurrent use of business data.

Although the described functionality can be implemented in several ways, in this case an approach must be chosen which has minimum impact on the business layer. Otherwise the testing and verification of the business algorithms which has already been done would have been futile. This means the interdependencies between the layers have to be designed in a way that allows every layer to interact with the business layer while the business layer itself exists in a clean room environment without any outgoing dependencies. Figure 5 shows which layer must have knowledge of which other layers. A layer must have knowledge of another layer only if it needs to influence the mode of operation of that other layer.

In the following subsections the dependencies between the layers are described briefly by giving examples of how a layer influences the operation mode of another layer. This will be the basis for the discussion on how the problem is to be addressed. The order of description of these dependencies matches the numbering of the arrows in Figure 5.

Dependency between transaction and database layers

The transaction layer needs to coordinate transactions with the database layer. Since a database management system (DBMS) usually comes with its own transaction processing monitor (TPM), in some cases the TPM of the DBMS can also be used to manage transactions outside of the DBMS, but in most cases it is the responsibility of the transaction layer to nest the database transactions in the application transactions [20].

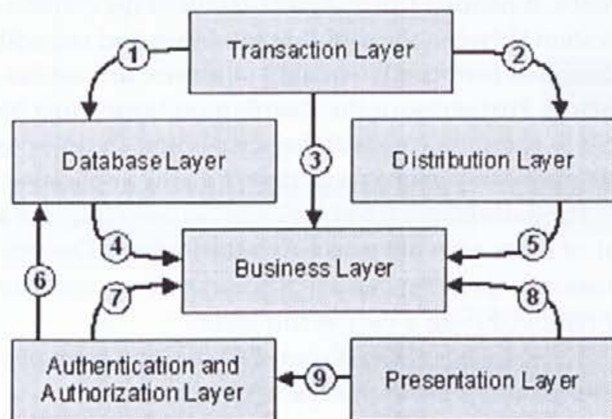

Figure 5: **Dependencies between the layers**

Dependency between transaction and distribution layers

In cases when several clients access data concurrently over a network or when data is held at many different locations, the transaction layer must be designed with respect to the special situations. One possible scenario is that a transaction needs to be rolled-back when a participant in the network exceeds response time. The example points up that the transaction layer must have knowledge of the details of the distributed system. Further examples can be found in [16].

Dependency between transaction and business layers

One of the most common activities of transaction processing is that resources are locked by the transaction manager – in this case the transaction layer. This can only be done when the transaction layer has access to the resources either directly or through a resource manager, but in both cases profound knowledge of the implementation details of the business layer is essential.

Dependency between database and business layers

The database layer not only needs to read all the values encapsulated in the business layer in order to store them in a DBMS, but it is also responsible for deciding which data needs to be kept on the heap at all. This requires keeping a record of the frequency in which the data in the business layer is accessed.

Dependency between distribution and business layers

In order to marshal data that is to be transmitted over the network, its structure must be known. This means that the distribution layer needs read and write access to the business layer.

Dependency between authentication/authorization and database layers

Similar to the example given above a DBMS usually has its own security system, which in most cases requires a form of authentication [14]. One responsibility of the authentication and authorization layer is to provide the DBMS with data necessary to authenticate the user. While the database login might differ from the login used to gain access to IB-CUE, the authentication and authorization layer need to be able to map one to the other.

Dependency between authentication/authorization and database layers

Not every user is authorized to work with all business data and to initialize every business action. The authentication and authorization layer therefore must have the possibility to disallow certain actions depending on the *identity* of the data and the user – it is not sufficient to control data access by only taking the *type* of the data into account, e.g. when a university member has the permission to view the budget of his school he does not at the same time have the permission to view the budget of every possible school.

Dependency between presentation and business layers

The presentation layer takes the role of the view and the controller in the commonly used model view controller pattern Š6Ć. The relationship between the view and the model can be designed to be a one-way relationship as between the presentation layer and the business layer in IB-CUE.

Dependency between presentation and authentication/ authorization layers

The presentation layer must display a login dialog that provides the authentication and authorization layer with the data necessary for the authentication of the user.

The usual way to implement the described layers is to choose appropriate technologies and design the classes in a way that respects the requirements of the technologies to be used. For example, if the distribution layer should be implemented using Enterprise JavaBeans (EJBs) [5], the classes have to implement certain interfaces and use certain implementation patterns.

The most appropriate approach in this case is, however, to use a technique known as "aspect-oriented programming" (AOP) [8]. The name aspect-oriented programming is a summary for generic techniques which allow the software designer to separate concerns like distribution, transaction processing or authorization and authentication in a different module. The main business classes can still be written in a traditional object-oriented language. This is accomplished by describing the transformation that each class must undergo when a certain layer – or aspect like it is called in AOP – is added. For example the transaction aspect, when implemented in a naïve way, might make it necessary to add a flag to every class which tells the TPM if the class is currently in use and therefore access operations should be locked. Since transaction processing is not part of the core business logic the code should also be kept in a different module. This is exactly how AOP works: the code of the various aspects is woven into the core business code previous to the compilation of the system.

6 Conclusion

As a means to support state-owned universities in facing deteriorating financial conditions and an intensified performance competition, we have introduced an innovative budgeting system on the schools' level, which is especially suited as a controlling instrument to guarantee basic operation of the schools, reward outstanding performance and to bring forward innovation. We have discussed design and implementation aspects of our Java-based monolithic software prototype IB-CUE, which implements this approach. Furthermore, we provided background information about the current evolution of this prototype into a distributed application with separate client and server components. We described basic considerations concerning a layered approach to the design of the final software product's architecture. The next steps in our project will be to select suitable technologies for the implementation of those layers. For example, we will have to analyze the pros and cons of some of the currently most popular middleware technologies, among which are heavy-weight technologies such as the Common Request Broker Architecture (CORBA) [11], Java 2 Enterprise Edition (J2EE) [17], and the Common Object Model Plus (COM+) [15], as well as more lightweight technologies like Web Services [3], which have become very popular recently.

References

[1] M. Aleksy, I. Bayer, T. Dickopp and A. Korthaus. Innovative Budgeting in a Competitive University Environment – Theoretical Concept and Design Considerations for an Information System. Proceedings of the International Workshop on Business and Information (BAI 2004), Taipei, Taiwan, March 26-27, 2004, chapter II-4, (2004).

[2] I. Bayer. "Strategische und operative Führung von Fakultäten - Herausforderungen durch Autonomie und Wettbewerb". Dissertation, Universität Mannheim, *Hemmer Scientific, Frankenthal*, pp. 4-7, (2002).

[3] D. Booth et al. "Web Services Architecture", W3C Working Group Note, (2004).

[4] T. Cormen, C. Leiserson, R. Rivest. "Introduction To Algorithms", *MIT Press and McGraw Hill*, (1994).

[5] L.G. DeMichiel. "Enterprise JavaBeans™ Specification, Version 2.1", Final Release, Sun Microsystems, (2003).

[6] E. Gamma, R. Helm, R. Johnson, J. Vlissides. "Design Patterns: Elements of Reusable Object-Oriented Software," *Addison-Wesley*, (1995).

[7] E. Kappler. "Die Universität kann Autonomie lernen". In: Tischer, S., Winckler, G., Biedermann, H. et al.: Universitäten im Wettbewerb – Zur Neustrukturierung österreichischer Universitäten. *Rainer Hampp Verlag, München*, pp. 297-330, (2000).

[8] G. Kiczales, J. Irwin, J. Lamping, J.-M. Loingtier, C. V. Lopes, C. Maeda, and A. Mendhekar, "Aspect-oriented programming". *European Conference on Object-Oriented Programming 1997 (ECOOP 1997)*, LNCS 1241, pp. 220-242. Springer, (1997).

[9] R. Ladded. "AspectJ in Action", *Manning Publications*, (2003)

[10] B. Meyer. "Object-Oriented Software Construction", Sams, (1997).

[11] Object Management Group. "Common Object Request Broker Architecture: Core Specification", OMG Adopted Specification, Tech. Doc. No. formal/04-03-12, (2004).

[12] Object Management Group. "UML 2.0 Infrastructure Specification", OMG Adopted Specification, Tech. Doc. No. ptc/03-09-15, http://www.omg.org/docs/ptc/03-09-15.pdf, (2003a).

[13] Object Management Group. "UML 2.0 Superstructure Specification", OMG Adopted Specification, Tech. Doc. No. ptc/03-08-02, http://www.omg.org/docs/ptc/03-08-02.pdf, (2003b).

[14] G. Pernul. "Database Security", *Advances in Computers*, Vol. 38, *Academic Press*, pp. 1-74, (1994).

[15] D.S. Platt. "Understanding COM+", *Microsoft Press, Redmond, WA*, (1999).

[16] M. Rangarao, A. Vogel. "Programming with Enterprise JavaBeans, JTS and OTS, Building Distributed Transactions with Java and C++", *John Wiley & Sons*, (1999).

[17] B. Shannon. "Java™ 2 Platform Enterprise Edition Specification, v1.4", Final Release, Sun Microsystems, (2003).

[18] B. Sporn. "Adaptive University Structures - An Analysis of Adaptation to Socio-economic Environments of US and European Universities". *Jessica Kingsley, London*, (1999).

[19] W.-D. Webler. "Qualität der Lehre als Gegenstand staatlicher Steuerung". In: Neusel, A., Teichler, U., Winckler, H. (Hrsg.): Hochschule-Staat-Politik, Christoph Oehler zum 65. Geburtstag, Frankfurt, New York, pp. 235-256, (1993).

[20] G. Weikum, G. Vossen. "Fundamentals of Transaction Information Systems: Theory, Algorithms, and Practice of Concurrency Control and Recovery", *Morgan Kaufmann Publishers*, (2001).

Dr. Markus Aleksy studied in Management Information Systems at the University of Mannheim, Germany. He holds a doctorate degree from the University of Mannheim. His research interests include analysis, design, implementation, and evaluation of distributed systems, especially based on CORBA.

Tilo Dickopp studied Computer Science and Applied Mathematics at the University of Mannheim. He is currently working on his doctorate degree. His research interests include object-oriented analysis, design, implementation, and evaluation of distributed systems, especially based on J2ME, as well as mobile electronic commerce.

Dr. Ingo Bayer studied Business Administration at the University of Mannheim. He holds a doctorate degree from the University of Mannheim and is currently Managing Director of the Business School at the University of Mannheim. His research interest lies in the transition Process from a state dominated system of higher education to a more market oriented model, especially in the areas of strategic planning, management, finance and budgeting.

Dr. Axel Korthaus studied Management Information Systems at the University of Mannheim. He holds a doctorate degree from the University of Mannheim and is currently working on his professorial dissertation. He is the project leader of a project on collaborative, component-based business application software development and gives lectures on OOA/OOD, OODB, and CBSE. His research interests include object-oriented analysis, design, implementation, and evaluation of distributed systems, especially based on J2EE, as well as knowledge management in software engineering.