GignoMDA - Generation of Complex Database Applications

Sebastian Richly, Dirk Habich, and Wolfgang Lehner
Dresden University of Technology
Database Technology Group
dbgroup@mail.inf.tu-dresden.de

Abstract

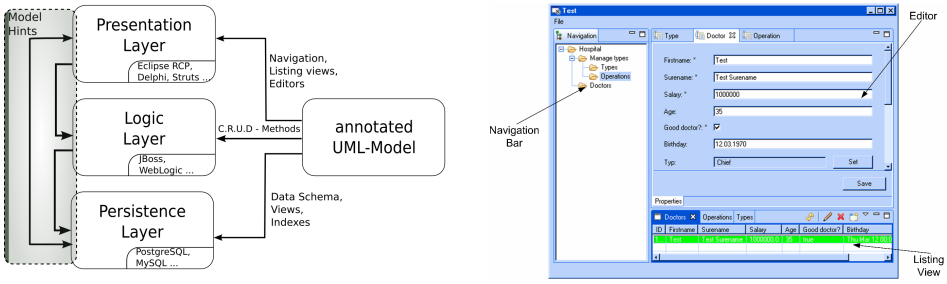
Complex database applications feature a large amount of structural and content-related aspects, and with each project, these aspects either have to be implemented completely again or must be realized by adopting and adjusting available program code. Based on the MDA concept (Model-Driven Architecture), the GignoMDA Project aims at the enrichment of the automatic generation of complex 3-layer applications through the consideration of non-functional properties. Aside from the automation aspect, the optimal mapping of annotated UML models to multi-layer architectures plays a central role here. That means, our approach provides a single point of truth describing all aspects of database applications (e.g. database schema, project documentation, etc.) with a great potential of cross-layer optimization. These new cross-layer optimization hints as non-functional properties are a novel way for the challenging global optimization issue of multi-tier database applications.

1 Introduction

Relational database systems are often used as persistent layer for a vast range of applications, especially for multi-tier applications. A huge number of such small to mid-size database applications require similar data maintenance and retrieval activities. On the one hand, writing such kind of code is neither challenging nor free of errors. On the other hand, the global optimization of such applications is a very difficult task because required knowledge is normally hidden within the individual components of multi-tier applications.

The utilization of models has a long tradition in the software technology and is the standard proceeding since the definition of UML. The *Model Driven Architecture* (MDA) approach [3,6], coined by the Object Management Group, places the UML models in the mid-point of the development process of applications. One of the main goals of MDA is to separate software design from architecture and realization technologies facilitating that design and architecture can alter independently. The design addresses only the functional requirements while architecture provides the infrastructure through which non-functional requirements like scalability, reliability and performance are realized. From the architecture independent UML model, various architecture specific models and program codes can be derived.

GignoMDA is based on the Model-Driven Architecture approach and extends classic methods for code generation through targeted control based on a central application specification. Not only does our GignoMDA approach enables a central and initially implementation- and architecture-invariant description of database applications, we also attempt to allow for an optimized implementation. In order to deal with the challenging global optimization issue of multi-tier database applications, we extend the MDA concept for the description of functional dependencies and specification techniques for the implicit and explicit modeling of optimization hints as non-functional properties. These non-functional properties provide a novel way for cross-layer optimization steps, which is not possible within regular software development process. With the optimization hints in the model we able to derive different optimization strategies for different architectures and therefore we are confirm with the general MDA concept.



(a) GignoMDA Approach (b) Generated Eclipse Frontend

Figure 1: GignoMDA Project

To put it in a nutshell, we present our GignoMDA approach which is illustrated in Figure 1(a) by providing contributions in the following areas: (i) New annotations for UML models, so that these models are the mid-point of database application development addressing software design and optimization issues of multiple layers, (ii) explicit and implicit hints as the foundation for cross-layer-optimization techniques, especially for deriving physical database design decisions, and (iii) prototypical realization as proof-of-concept of all proposed UML annotations and optimization hints based on the AndroMDA framework [1].

The rest of the paper is organized as follows: In Sections 2 and 3, we give an brief overview of our UML profile extensions and application interaction patterns. In Section 4, we present our optimization hints in a more detail. This paper closes with a description of the GignoMDA prototype and a conclusion in 5.

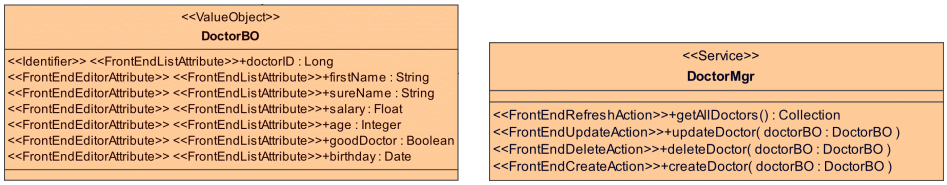
2 UML Profile Extensions

Since our GignoMDA project is based on the idea of the MDA approach [3, 6], we start with a very short overview of MDA concepts. The MDA approach introduces a *Platform-Independent Model* (PIM), which is an abstract model of the software system that does not incorporate any implementation choice, mostly used to describe the business logic. Furthermore, the PIM can be extended by application designers to a "*marked PIM*" where the model elements are marked with (1) stereotypes to define their functionality within the application, and (2) tagged values to add additional information for the code generation process. The *Platform-Specific Model* (PSM), consisting of the target application platform, is derived from this PIM. That means, the UML model as PIM is the mid-point within the MDA world and the vision is to generate full-operating application for different platforms from this single UML based specification.

Nowadays, the most database applications typically consist of three layers: (i) presentation layer based either on web technology or on a rich client platform (RCP), (ii) business logic layer implementing the structure and the behavior of business objects, and (iii) the persistence layer implemented by a standard (mostly relational) database system. The current powerfulness of the UML model is not sufficient in the context of database applications and therefore we extend the PIM with the following concepts:

1. New stereotypes for all three layers, especially we integrate the modeling of the presentation layer in the whole process, which is totally missing in many approaches. With our extensions we are able to design complete database applications and automatically generate full-operating application from the model.

2. Moreover, we introduce novel stereotypes enabling cross-layer optimization for database applications. These stereotypes are considered in generation process and the resulting application is optimized regarding all layers.



(a) Presentation Object (b) Service

Figure 2: Sample Presentation and Service

3. New tagged values to annotate the UML model which is used in the UML2Code transformation. These new tagged values specify the utilization of the new stereotypes.

We illustrate the new GignoMDA designing process and the subsequently generation process with the following database application, where we record doctors and their assigned surgeries within a hospital environment. This example application requires typical data maintenance and retrieval activities as other database application and we do not consider further extended business logic. In this example, the presentation layer is an Eclipse-based [2] user-frontend (see Figure 1(b), which communicates via RMI with JBoss as business logic layer). The persistence layer is modeled via EJB supporting any relational database system. On this example database application we evaluate several cross-layer optimization hints, which are described in Section 4.

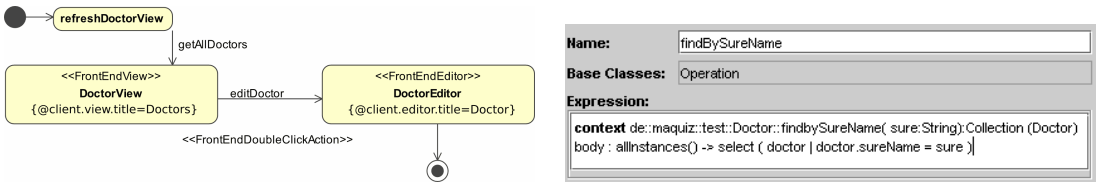
We start with the frontend modeling whereby the application frontend as depicted in Figure 1(b) is divided into three major parts: *Navigation bar*, *Listing View* and *Editor*. The navigation bar illustrates the overall structure of the application areas and provides the entry points to the listing views and individual editors for data maintenance. The navigation tree is defined by an UML use-case diagram. The hierarchy of the navigation node can be modeled by dependency connections between the use cases, annotated with a new stereotype.

The listing view comprises the result set of a database query, specified by an OCL constraint (see Section 3). The tabular view displays only attributes of the underlying object which are annotated with the new stereotype `<<FrontEndListAttribute>>` (see Figure 2(a)) or included in the tagged value `@client.view.table.columns`.

The editor view provides a mechanism to view and manipulate individual records shown in the listing view. The appearance and the rules for updates of the attributes and all participating associations are controlled by the stereotype `<<FrontEndEditorAttribute>>` (see Figure 2(a)). Additional information—e.g. the label caption—are specified by tagged values. Moreover, the user's input can be validated by given OCL constraints [4].

3 Application Interaction Patterns

Besides the design of the frontend application we have also to define the set of interactions between the different parts of the application by a UML activity graph. Figure 3(a) shows a sample interaction pattern. With the activity state *RefreshDoctorView*, the activation of a navigation node opens the listing view *DoctorView* by calling the method *getAllDoctors*. As can be seen in Figure 2(b), this `<<FrontEndRefreshAction>>`-method is part of the set of C.R.U.D. (acronym for the life cycle operations Create, Retrieve, Update, and Delete) methods associated with each business object manager element (i.e. a class with the stereotype `<<Service>>`). These methods are either provided with default semantics (e.g. get all instances of the underlying business object for *getAllXXX*-methods) or replaced by methods with corresponding application-specific semantics. The presentation type of an activity state is set using the stereotypes `<<FrontEndView>>` and `<<FrontEndEditor>>`. Actions, like the opening of an editor or the deletion of an individual object, are modeled as transitions with the `<<FrontEndAction>>` stereotype.



(a) Activity Diagram (b) OCL query

Figure 3: Sample Activity Diagram and OCL query

Additionally, Figure 3(a) shows the user interaction path from a listing view to the editor view by issuing a double click on a selected data record in the list. The *DoctorEditor* appears by calling the frontend-action *editDoctor*.

As already pointed out, the interaction patterns rely on methods which are performing the application logic, i.e. retrieving the underlying objects in the simplest case. For the ongoing example, Figure 2(b) shows all necessary model elements which are part of the logic layer for a specific entity, i.e. a Doctor object.

All C.R.U.D.-methods of an entity used by the editor or listing views are supposed to be defined within an object marked as `<<Service>>`-object.

For example, operations with the stereotype `<<FrontEndCreateAction>>` will be implemented as a create method with standardized behavior according to the EJB specification. Furthermore, every retrieve operation must match a `<<FinderMethod>>` method of the—via `<<EntityRef>>`—referenced entity. The specific semantic of this method, e.g. the corresponding query, can be defined by an OCL constraint (see Figure 3(b)), or using the new tagged value `@persistence operation.query`.

4 Model Hints

In addition to frontend and application patterns, the underlying database objects have to be modeled. The major aspects of data modeling are already provided by UML and the classes with the stereotype `<<Entity>>` correspond to database objects. Additional aspects are (1) check constraints or triggers, are modeled through OCL constraints and (2) database indexes for several attributes through a tagged values.

Within our GignoMDA project, we exploit the fact that the design and the potential content of the database have an impact on the presentation layer and vice versa by introducing the concept of *model hints*. For example, the application designer—usually supported by the domain expert—may specify the number of expected instances of objects or attributes already during the design phase. Such hints result in changes of the default behavior of the application (e.g. prompting for a search dialog to avoid mass loading when activating the listing view) and in additional DDL operations wrt. underlying database system (e.g. enabling partitioning). Hints, in general, are therefore a central mechanism for cross-layer optimizations in large applications. Within the GignoMDA project, we distinguish two kinds of model hints:

- **Explicit Hints:** Explicit hints are added by the application designer via stereotypes or tagged values. An explicit hint annotates an model element a specific role or trait. For example, the stereotype `<<LookupEntity>>` tells the underlying system that the data are mostly read-only. The other extreme of the expected behavior can be annotated by adding the stereotype `<<UpdateEntity>>` telling the code generator to optionally add database parameter adjustments for extensive logging and locking.

- **Implicit Hints:** Implicit hints are derived by the generator from the specified structure and behavior and cannot be added by the application designer. For example, every association between two objects holds a tagged value `@client.association.displaytype`

telling the code generator whether the association should be displayed as a set of check boxes or using a tabular list. By indicating the "check box"-style, the designer implicitly denotes that the associated table will contain only a few objects. This information, can be used for example to exploit object-relational functionality of the underlying database system and create a schema holding the associated objects with a nested table.

Hints therefore play a general role in enabling cross-layer optimization out of the central specification. The examples given above are supposed to illustrate the power of model hints of which GignoMDA supports a large variety. Furthermore, we are able to derive different optimization strategies for different architectures from the hints in the architecture-independent UML-model.

5 GignoMDA Prototype and Conclusion

Our prototypical implementation is based on the AndroMDA Framework [1]. On the modeling side, as mentioned above, the approach proposes an extension of the UML design methodology via a UML profile to specify persistence aspects, security, business logic and potential user interactions. On the code generation side, the prototype extends the AndroMDA framework by adding additional MDA cartridges and extending other already existing modules to consider the additional semantics specified in the UML model.

As target platform for the presentation layer, GignoMDA currently supports the Eclipse-RCP platform [5] using the dynamic component model for plug-ins, update management, menu and preferences management. On the middle tier, annotated business objects are represented as session- and entity beans. The prototype extends the existing EJB AndroMDA cartridge to cover the additional semantics and hints of the UML model. Moreover, the final application is deployed in the open source application server JBoss. In order to capture the special adjustments required to reflect the semantics of implicit and explicit hints, we are currently restricting the use of database systems to MySQL, PostgreSQL, Microsoft SQL Server and Oracle 10 Xe.

In this paper, we have presented our GignoMDA Project that aims at the enrichment of the automatic generation of complex mulit-layer database applications through the consideration of non-functional properties. In the near future, building large applications will definitely based on an extensive portion of generated code. Furthermore, efficient implementations require a global view on the general problem ranging from the presentation layer down to the persistence layer and database optimization layer. In summary, we see GignoMDA as a first step towards a new application development paradigm: Just model and click the "Generate"-button, whereas several implicit and explicit optimization tasks are automatically considered.

References

[1] AndroMDA. http://www.andromda.org.

[2] Eclipse RCP framework. http://www.eclipse.org/rcp.

[3] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained. The Model Driven Architecture: Practice and Promise*. Addison-Wesley, 2003.

[4] OCL 2.0 Spezifikation. http://www.omg.org/docs/ptc/03-10-14.pdf.

[5] Eclipse technology overview. http://www.eclipse.org/whitepapers/eclipse-overview.pdf.

[6] Dave Thomas and Brian M. Barry. Model driven development: the case for domain oriented programming. In *OOPSLA '03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 2–7, New York, NY, USA, 2003. ACM Press.