A UML-based Metamodeling Architecture for Database Design

Marie-Noëlle Terrasse, Marinette Savonnet and George Becker*
Laboratoire LE2I - Université de Bourgogne - France
E-mail: {terrasse, savonnet, becker}@khali.u-bourgogne.fr

Abstract

We use metamodeling for Object Oriented database design and management. First, in the context of the UML-based metamodeling, we provide an overview of what is achievable, e.g., more exhaustive descriptions of classes (their behaviors and their dependencies), validation of external schemas of a database, as well as determination of requirements for software tools to enhance system security. Second, we propose an extension –using our specific metamodeling architecture– for management of federated databases.

1. Introduction

Our objective is to study to what extent the Object Oriented database design and management –including distributed or federated databases– can be improved by using a UML-based metamodeling architecture.

First, we give an overview of data distribution in the context of distributed and federated databases. Then, we present metamodeling as a two-fold abstraction mechanism: multi-view models integrated into a metamodel, and metamodeling architectures. Section 2 gives guidelines for using each view of the UML multi-view model in the database design. Section 3 extends this work to management of federated databases through our specific metamodeling architecture. Section 4 concludes by a review of our on-going work.

1.1 Distributed Databases: an Overview

The world-wide development of networks –particularly the Internet– allows a new type of organization for enterprises. In this context, data and their operations can be distributed over the net. Data distribution encompasses various issues such as reduction of the cost of data transfers (by allocating the data where it is often used), improving computation performance (by using the parallelism due to distribution), and improving reliability (by duplication of data).

Both distributed and federated databases use data distribution as well as distribution of operations. Their main difference lies in constraints that the global system applies to local databases, with federation being less restricting to local databases. The design of distributed databases [8, 29] encompasses two main steps: fragmentation (which splits data into fragments), and allocation (which chooses a site –at least one site– for each fragment). When autonomous databases have to cooperate, the process is different in the sense that fragmentation as well as allocation are given. Designing a federated database [30] consists in integrating databases, i.e., proposing to users a global view of their contents.

1.2 Metamodeling: An Overview

Metamodeling proposes a two-fold abstraction mechanism in order to model increasingly complex systems (many different users being offered different functionalities, complex structures of data, complex behaviors, etc.) under more and more demanding requirements (mainly requirements for validation and refinement of models [10, 26], but also requirements for semi-automatic translation of models into executable codes [18, 25]).

Abstraction by conceptualization is no more than an extension of the database standard dichotomy between data and a model. In the OMG’s metamodeling architecture four different levels of abstraction describe respectively:

- Instances through a snapshot of the system.
- Models through a multi-view representation whose main structural component is similar to the OO-conceptual schema of databases. Each model is an abstract description of all possible snapshots for a given database. Furthermore, the model limits –through be-
behavioral description– the way in which the system can evolve from one snapshot to another one.

- Metamodels correspond to different application domains. Metamodels are abstract descriptions of models: they define which components of the modeling language are necessary for describing models of that application domain. In most cases, ad-hoc components of the language may be defined by specializing plain components of the modeling language; see, for example, the extension mechanisms of the UML.

- Meta-metamodels indicate the filter through which the real world is seen: for example, is boolean logic satisfactory or is it necessary to use a temporal logic?

As implemented by description-driven systems, see for example Chevenier & al. [9], abstraction by conceptualization allows modelers to split the system description into several well-defined\(^1\) levels, making their modeling strategy clearer: what is relevant to the application itself and what is supposed to be more general.

**Abstraction by projection** implements through a metamodel the principles of separation and combination of concerns [3, 5, 23, 34]. In the context of complex problems, the information to be taken into account becomes overwhelming for modelers. Separation of concerns proposes to model, by syntactically separate units, different and partial points of view of the system (such as structural and behavioral descriptions, sets of main functionalities offered to different types of users, organisation of physical components, etc). Since those views are semantically related –through the overall meaning of the system being modeled– their relationships must be expressed as inter-view constraints and the global view of the system must be defined as well. As illustrated in Figure 1, the metamodel is in charge of defining –for a given application domain– which views are necessary, what are their mandatory inter-view constraints, and how they must be combined to describe the system.

Abstraction by projection allows modelers to choose the best descriptive language for each point of view, and enables the use of formal tools for validation and refinement of models [16, 18, 25]. Furthermore, when the metamodel is well-defined\(^2\), the global description is guaranteed to be consistent and complete.

---

1. This may be argued when using a loose metamodeling architecture; see [1, 27] for more details about loose versus strict metamodeling. Furthermore, many extensions of the UML combine those levels.

2. Such a metamodel with a well-defined semantics is very difficult to obtain. For example, the UML metamodel’s semantics is often said not to be precise enough [6, 7, 13, 14, 26]. This is a major issue of metamodeling.

---

**Figure 1. The Metamodel as an Implementation of the Principles of Separation and Combination of Concerns**

2. Guidelines for modelers using a UML-based metamodeling architecture

Let us consider a UML-based metamodeling architecture to show what can be useful to modelers of databases. In such an architecture, e.g., Figure 2, the metamodel provides modelers with an a-priori multi-view model that encompasses 9 diagrams to describe a system: structural view (using Class and Collaboration Diagrams), behavioral view (using StateCharts, Activity and Sequence Diagrams), physical view (using Components and Deployment Diagrams), as well as views of external functionalities and instances (using Use Case and Object Diagrams).

2.1 Using abstraction by projection

**Structural view** Our belief is that working within a UML-based metamodeling architecture improves the description of the structural view of the database. Such a structural view—which is classically expressed through a conceptual schema (classes and their relationships)—takes into account dependencies between classes. In the context of OO modeling, due to the requirement of encapsulation, dependencies are considered to be visible at three different levels [12]: Conceptual dependencies are defined between instances (e.g., compositions or aggregations). Contextual dependencies are due to signatures of methods (a method of a class has a parameter that belongs to another class). Operational dependencies are due to bodies of methods (a method uses a local object belonging to another class).
Conceptual dependencies, as well as most of the contextual dependencies, are expressed with ease by the conceptual schema. Operational dependencies are more difficult to express (due to the requirement of stability of relations). In [33], we extended an OO conceptual schema into an extensive model of dependencies between classes of a database. Such information may be also expressed by the Collaboration Diagram of the UML.

Analogously, role modeling (see, for example [22]) captures dependencies that are limited to a particular domain: they cannot be integrated into the Class Diagram but they can be captured within the Collaboration Diagram.

Thus, Class and Collaboration Diagrams enable modelers to express a dual view of all dependencies between classes. For more detail, see [24] where we proposed an algorithm for improvement of the adequacy between Class and Collaboration Diagrams for expressing the structural view of a database.

User view Within the Unified Development Process [19], the Use Case Diagram is fundamental for modeling of information systems: use cases are first described and then used as a basis for determining the Class and Sequence diagrams of the UML model. By using the traceability information (between use cases and classes), it is easy to determine which classes are necessary for each type of users. Such information may be useful for validating (or defining) external schema (views) of the database.

Behavioral view Behavioral description of the system is becoming increasingly important for those new databases that are mostly used through the Internet. Because users of the databases are no longer limited to local and well-informed users, it is necessary to provide the users with a precise and readable description of the operations that can be activated. Many authors study the problem of the readability for a casual user of those behavioral descriptions, e.g., the “light-weight tools” of Heitmeyer & al. [15].

Physical view The physical view of the system (Component and Deployment Diagrams) has two major applications in distributed databases. First, description of the physical components of the systems, combined with the localization of those components on the different sites (host machines) allows modelers to describe the distribution itself. And second, the Deployment Diagram which describes the type of links between the sites can be combined with models of interactions between components in order to provide security specialists with basic and necessary information for designing security tools. See, for example, Jurjens’ extension of the UML [20] for development of secure systems.

2.2 Using abstraction by conceptualization

A current trend in metamodeling – which can be applied to databases as well – is to define, for each application domain, a specific metamodel. For example, the OMG defines specific profiles –as extensions of the UML metamodel – for various domains such as real-time or CORBA [28]. Many authors define extensions of the UML metamodel for specific domains, e.g., Architecture Description Language [32], hypermedia systems [2], synchronization [17], etc.

A model of each application is derived from the specific metamodel of the domain. Such a strategy provides two main advantages: improvement of reuse [21] since common features are described only once in the metamodel, and an explicit basis of agreement for distributed databases since the common metamodel is supposed to describe –at an abstract level – the basis of agreement.

3. Extension to federated databases

In this section we present our metamodeling architecture and its application for federated databases.

3.1 Our Metamodeling Architecture

UML-based metamodeling faces a dilemma. On one hand, since UML is a wide-purpose language, the semantics of UML metamodel is far from being precisely defined. On the other hand, formal metamodels are necessary in order to propose modeling environments with formal tools for validation and refinement of specifications.

In order to manage both formal and ambiguous descriptions of metamodels, we proposed [36] an UML-based metamodeling architecture that reorganizes the two uppermost layers of the architecture into a mirroring structure: abstract descriptions (called modeling paradigms) combining several different languages at the meta-metamodel level.
Meta-metamodel layer Our meta-metamodel layer comprises modeling paradigms that describe—in terms of concepts that are interrelated by constraints—the semantics of the real world. As defined in [35], a modeling paradigm mp is described—using English language, logic and set theory—through two sets, $\mathcal{E}^3(mp)$ and $\mathcal{C}^3(mp)$. The set $\mathcal{E}^3(mp)$ contains descriptions of elementary concepts. The set $\mathcal{C}^3(mp)$ contains constraints among the concepts of $\mathcal{E}^3(mp)$. For example, the general OO modeling paradigm (as it appears in the UML approach [39], and which we denote by gnp) encompasses concepts such as object (with identity, state, and behavior), class, generalization, as well as constraints like each object belongs to a class.

Modeling paradigms are partially ordered (by inclusion of concepts and subsumption of constraints, see [35] for more details). Thus, our meta-metamodel layer may be structured as a poset of modeling paradigms.

Metamodel layer Our objective is to build our metamodel layer as a mirror of the poset of modeling paradigms: the generic modeling paradigm gnp is instantiated into the UML metamodel itself, and other modeling paradigms are instantiated into specializations of the UML metamodel. We note that such an objective differentiates modeling paradigms from the MOF of the OMG [38] which is used for alignment of existing metamodels rather than for their creation. Analogously to modelers' practice [11, 17, 31, 32] of extending the UML metamodel for various application domains, we use the tailoring mechanisms of UML (constraints, tag values and stereotypes) in order to instantiate modeling paradigms as metamodels. Let us consider a modeler to provide a high-level framework for interoperability. We propose to adapt such a framework for management of federated databases. Let us assume that an unambiguous metamodel is provided for any database belonging to the federation. Interoperation of two databases DB$_1$ and DB$_2$ is carried out by an abstract basis of agreement—at the metamodel level—built from their metamodels mm$_1$ and mm$_2$, respectively. In order to build the basis of agreement, we need formal operations defined as follows: Given two metamodels mm$_1$ and mm$_2$, we build the unions of their sets of concepts and constraints, respectively. If the union of constraints contains no contradiction: 1) the metamodels mm$_1$ and mm$_2$ are said to be consistent with each other, and 2) the metamodel defined by those unions is said to be integrated from mm$_1$ and mm$_2$.

The strategy we propose consists of constructing an unambiguous metamodel that can be used as a basis of agreement. As long as possible, the inheritance hierarchy of metamodels is used for such a construction: we propose a metamodel-based agreement. In case of failure, we use the meta-metamodel level to build a modeling paradigm for agreement. More precisely, the following cases are possible when looking for a basis of agreement relying on mm$_1$ and mm$_2$: The corresponding schema is given in Figure 4: large dark circles indicate, in each case, the actual basis of agreement.

- Agreement on a relaxed metamodel

See part (a) of Figure 4. Let us denote by mm the first common ancestor of mm$_1$ and mm$_2$ in the inheritance hierarchy. The metamodel mm corresponds to weaker constraints than those of both mm$_1$ and mm$_2$ metamodels. If mm is an unambiguous metamodel, mm is then the basis of agreement for mm$_1$ and mm$_2$.

### Figure 3. A Mirroring Structure of Modeling Paradigms and Metamodels

**Meta-metamodel layer**

![Meta-metamodel layer](image)

**Metamodel layer**

![Metamodel layer](image)
• Agreement on an integrated metamodel
  See part (b) of Figure 4. If the common ancestor \( m m \) is an ambiguous metamodel, let us consider \( mm_1 \) and \( mm_2 \) that are the first two unambiguous metamodels on the path from \( mm \) to \( mm_1 \) and to \( mm \) to \( mm_2 \), respectively. If \( mm_1 \) and \( mm_2 \) are consistent, then it is possible to use their integrated metamodel as a basis of agreement.

• Agreement on an instantiated metamodel
  See part (c) of Figure 4. If \( mm_1 \) and \( mm_2 \) are not consistent, it is impossible to construct an integrated metamodel from \( mm_1 \) and \( mm_2 \). The solution is to move up to the meta-metamodel layer. Let us denote by \( mp \) the meta-metamodel corresponding to the informal metamodel \( mm \). We choose as a basis of agreement a modeling paradigm \( mp' \), i.e., one of the unambiguous modeling paradigms that are close to \( mp \) into the poset (see [35] for more detail).

4. Conclusion

We are convinced that metamodeling can substantially improve the design of distributed databases by using multi-view models. The structural view –consisting of Class and Collaboration Diagrams– can be used for exhaustive and consistent description of dependencies between classes. The user view –Use Case Diagram– provides a support for validating the design of external schemas of the databases. The behavioral view –Statecharts, Sequence and Activity Diagrams– provides external users with an accurate description of the semantics of available classes and operations. The physical view –Component and Deployment Diagrams– can be used for designing security of distributed databases.

Furthermore, by using our specific UML-based architecture (which provides a two-fold mirroring structure of the meta-metamodel and metamodel layers), we can provide federated databases with abstract bases of agreement. Our on-going work first consists in the implementation of our framework for high-level interoperability. Then, we plan to test our framework within different application domains.

References


