Object-Relational Mapping in 3D Simulation

Ann-Marie Stapelbroek,
Martin Hoppen
and Juergen Rossmann

Institute for Man-Machine Interaction
RWTH Aachen University
D-52074 Aachen, Germany
Email: ann-marie.stapelbroek@rwth-aachen.de {hoppen,rossmann}@mmi.rwth-aachen.de

***Abstract*—Usually, 3D simulation models are based on complex object-oriented structures. To master this complexity, databases should be used. However, existing approaches for 3D model data management are not sufficiently comprehensive and flexible. Thus, we develop an approach based on the relational data model as the most widespread database paradigm. To successfully combine an object-oriented 3D simulation system and a relational database management system, a mapping has to be defined bridging the differences in between. These are summarized as the object-relational impedance mismatch. Theoretical foundations of object-relational mapping are researched and existing, semi-automatic object-relational mappers are evaluated. As it turns out, existing mappers are not applicable in the presented case. Therefore, a new object-relational mapper is developed based on the utilized simulation system's meta information system. Key aspects of the developed approach are a necessary adaptation of the theoretical object-relational mapping strategies, database independence in conjunction with data type mapping, schema mapping by schema synchronization, and strategies for saving and loading model data as well as for change tracking. The developed prototype is evaluated using two exemplary simulation models from the fields of industrial automation and space robotics.**

Keywords*–Object-Oriented 3D Simulation System; Relational Data Model; Relational Database Management System; Object-Relational Mapper; Object-Relational Mapping; Object-Relational Impedance Mismatch.

I. Introduction

3D simulation systems are used in different areas like space robotics and industrial automation to derive properties – especially, spatial properties – of a planned or existing system's behavior. Usually, 3D simulations are based on complex and extensive models. Therefore, databases are appropriate for data management to master the complex object-oriented structures of 3D simulation models and to make simulation states persistent [1]. As a result, different simulation runs can be recorded and analyzed [2]. In comparison with flat file storage, the usage of databases has key advantages, in particular, if data independence, multi-user synchronization, data integrity, data security, reliability, efficient data access and scalability are required [3]. However, existing approaches for 3D model data management using databases are not sufficiently comprehensive and flexible, motivating the development of a new approach.

The most widespread database paradigm is the relational data model. If relational databases should be used as a persistence layer for object-oriented 3D simulation systems, a mapping has to be defined bridging the differences between both paradigms. These differences are summarized as the object-relational impedance mismatch. The term object-relational mapping (OR mapping) describes the process of mapping the objects of an application (here, a 3D simulation system) to table entries of a relational database and vice versa. A manual mapping between the object-oriented concept and the relational database model is complex and error-prone so that object-relational mappers (OR mappers) are used. An OR mapper is a tool that builds a translation layer between application logic and relational database to perform a semi-automatic object-relational mapping.

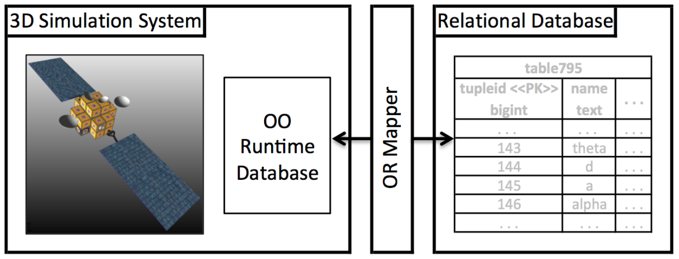


Figure 1. OR mapping for a 3D simulation system with an object-oriented runtime database (data: [4]).

In this paper, we present an OR mapper for 3D simulation systems with an object-oriented runtime database and a meta information system, see Figure 1. The work was conducted as a student project and is based on our previous work [1] [2]. A prototypical implementation is based on the 3D simulation system VEROSIM [5] and PostgreSQL [6] as a relational database management system (RDBMS). However, the underlying approach itself provides database independence allowing the usage of other RDBMSs. A key aspect of the presented OR mapping is the schema mapping that is build during a schema synchronization (based on [2]). The introduced concept considers both forward and reverse mapping. Furthermore, the OR mapper supports change tracking and resynchronization of changes. The OR mapper provides an eager and a lazy loading strategy. The prototype is evaluated using simulation models for industrial automation and space robotics (Figure 11).

The rest of this paper is organized as follows: In the next section, similar applications with a 3D context that integrate database technology are analyzed. Section III contains a short summary regarding the theoretical principals of OR mapping and Section IV introduces the utilized 3D simulation system VEROSIM and evaluates existing OR mappers. Subsequently, the newly developed concept for OR mapping is introduced in Section V and evaluated in Section VI. Finally, the paper is concluded in Section VII.

II. Related Work

In general, many applications with a 3D context have data management requirements similar to 3D simulation. Yet, the use of database technology is not widespread and files are still predominant. When databases are utilized at all, in many scenarios, they are used to store additional information (meta information, documents, films, positions, hierarchical structure . . .) on scene objects or parts [7] [8] [9] [10] [11] [12] [13]. Yet, we need to manage the 3D model itself to obtain all the benefits from using database technology. Another important aspect for 3D simulation is arbitrary application schema support to be able to work with native data and avoid friction loss due to conversions. Many systems use a generic (scene-graph-like) geometric model, in most cases with attributes [7] [14] [15] [16] [17]. In such scenarios, schema flexibility can be achieved to a certain extent by providing import (and export) to different file formats [15] [18] [19] [20]. Some approaches support different or flexible schemata. For example in [14], schema alteration is realized by adding attributes to generic base objects. Other systems support a selection of different static [15] or dynamic [16] [21] schemata. However, most approaches focus on a specific field of application, thus, requiring and supporting only a corresponding fixed schema [22] [23] [24]. While Product Data Management (PDM) systems [8] [9] or similar file vaulting approaches for 3D data [25] [26] [27] in principle support arbitrary schemata they are not explicitly reflected within the database schema due to their ”black box integration” approach. Most scenarios provide a distributed architecture in terms of multiuser support, a client-server model, or access control and rights management. However, only some build it on a Distributed Database (DDB)-like approach [21] [28] [17] with client-side databases. The latter is favorable for 3D simulation, e.g., to provide schema flexibility or a query interface on client-side, as well.

Altogether, while there are many existing approaches to use database technology in applications with a 3D context, none of them provides a comprehensive and flexible solution that fulfills all the requirements for 3D simulation.

III. Object-Relational Mapping

Some RDBMSs provide additional object-relational features. For example, PostgreSQL supports some object-oriented extensions like user defined types or inheritance. However, these features are not provided uniformly by all RDBMSs contradicting the desired database independence. Therefore, the OR mapping is realized with standard relational concepts only.

There are several references in literature dealing with the differences between object-oriented concepts and the relational data model. To solve the object-relational impedance mismatch and successfully generate an OR mapper, it is important to consider the properties of both paradigms and the consequent problems. For example, one main idea of object-orientation is inheritance [29]. However, the relational data model does not feature any comparable concept. Thus, rules have to be defined how inheritance can be mapped onto table structures. Further differences between both paradigms that contribute to the object-relational impedance mismatch are polymorphism, data types, identity, data encapsulation, and relationships.

The following subsections summarize the state-of-the-art of theoretical mapping strategies for inheritance, relationships and polymorphism.

A. Inheritance

The approaches to map objects onto tables differ in to how many tables one object is mapped. Most authors name three standard mapping strategies for inheritance. They are illustrated in Figure 3 regarding the exemplary inheritance hierarchy from Figure 2.

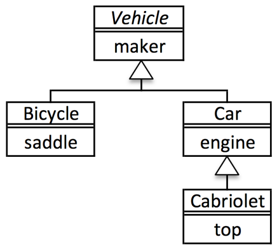
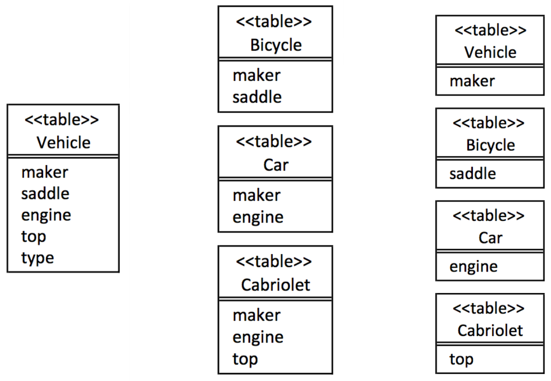


Figure 2. Exemplary inheritance hierarchy (adapted from [30, p. 62f]).

The first strategy is named *Single Table Inheritance* [30] and maps all classes of one inheritance hierarchy to one table, see Figure 3(a). A discriminator field is used to denote the type of each tuple [31]. An advantage is that all data is stored in one table preventing joins and allowing simple updates [30, p. 63]. Unfortunately, this strategy leads to a total denormalization, which is contrary to the concept of relational databases [31].



(a) Single Table Inheritance. (b) Concrete Table Inheritance. (c) Class Table Inheritance.

Figure 3. Standard mapping strategies for inheritance (adapted from [30, p. 62f]).

The *Concrete Table Inheritance* [30] strategy maps each concrete class to one table, see Figure 3(b). This mapping requires only few joins to retrieve all data for one object. A disadvantage is that schema changes in base classes are laborious and error-prone. [30, p. 62f]

The third standard mapping strategy for inheritance is named *Class Table Inheritance* [30] and uses one table for each class of the hierarchy, see Figure 3(c). It is the easiest approach to map objects onto tables [30, p. 62] and uses a normalized schema [31]. However, due to the use of foreign keys, this approach realizes an *is-a* relationship as a *has-a* relationship [31]. Thus, multiple joins are necessary if all data of one object is required. This aspect can have an effect on performance. [30, p. 62f] [32, p. 7]

Another possibility to map objects onto tables not mentioned in every reference on OR mapping is the generic approach [33]. It differs from the strategies mentioned above as it has no predefined structure. Figure 4 shows an exemplary set-up, which can be extended as required. The approach is particularly suitable for small amounts of data because it maps one object to multiple tables. It is advantageous if a highly flexible structure is required. Due to the generic table structure, elements can easily be added or rearranged. [33]

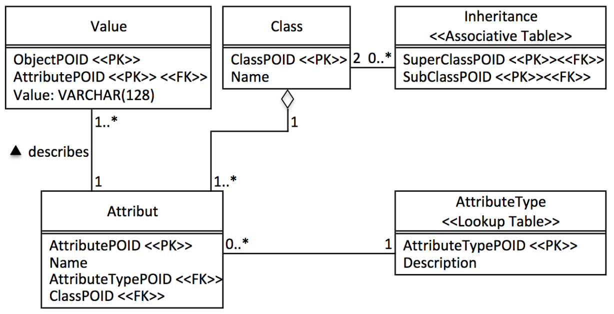


Figure 4. Map classes to generic table structure (adapted from [33, chapter 2.4]).

In conclusion, there is no single perfect approach to map objects onto tables yielding an optimal result in all situations. Instead, a decision has to be made from case to case depending on the most important properties. For this purpose, the three standard mapping strategies for inheritance can also be combined, however, to the disadvantage of more complexity. [30, p. 63]

B. Relationships

In contrast to relationships between two objects, which can be unidirectional, relationships between tables in a relational database are always bidirectional. In unidirectional relationships, associated objects do not know if and when they are referenced by another object [32]. Due to the mandatory mapping of unidirectional onto bidirectional relationships, information hiding cannot be preserved regardless of the relationship's cardinality, i.e., 1:1 (one-to-one), 1:n (one-to-many) or n:m (many-to-many) relationships.

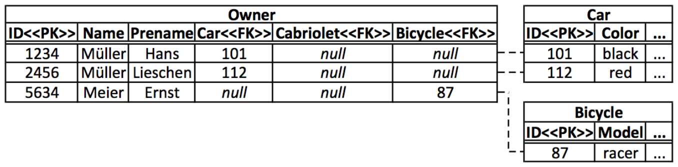
1:1 relationships can simply be mapped onto tables using a foreign key. To map 1:n relationships, structures have to be reversed. [33] [30, p. 58f] In case of n:m relationships, additional tables are mandatory: A so-called association table is used to link the participating tables. [33] [30, p. 60] It is also possible to map an n:m relationship using multiple foreign keys in both tables if constant values for n and m are known. [33]

Several references describe the aforementioned mapping strategies for relationships. Besides, [34] describes an approach using an additional table regardless of the cardinality. Thus, objects can be mapped onto tables regardless of their relationships. Following [34], one disadvantage of the aforementioned approaches is the violation of the object-oriented principle of information hiding and abstraction. Furthermore, tables are cluttered by foreign key columns which reduce maintainability and performance. The authors prove (by a performance test) that their own approach shows no performance degradation. [34, p. 1446f]

C. Polymorphism

Polymorphism is an essential concept in object-orientation. However, relational databases do not have any feature to reference entries of different tables by one foreign key column. The target table and column have to be explicitly defined for each foreign key constraint. It is not possible to define a foreign key that references more than one table [35, p. 89]. Thus, a mapping is required to map polymorphic associations onto a relational database. Following [35], [36], there are three mapping approaches for polymorphic associations.

The first approach is named *Exclusive Arcs* and uses a separate foreign key column for each table that can be referenced by the polymorphic association, see Figure 5. This approach requires NULL values for foreign key columns. For each tuple, at most one of the foreign key columns may be unequal to NULL. Due to foreign key constraints, referential integrity can be ensured. However, the administrative effort for the aforementioned NULL rule is high. An advantage of this approach is that queries can easily be formulated.



Owner

ID<<PK>>	Name	Prename	Car<<FK>>	Cabriolet<<FK>>	Bicycle<<FK>>
1234	Müller	Hans	101	*null*	*null*
2456	Müller	Lieschen	112	*null*	*null*
5634	Meier	Ernst	*null*	*null*	87

Car

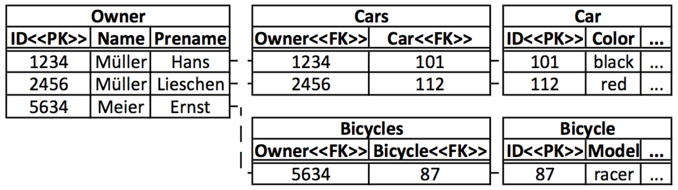
ID<<PK>>	Color	...
101	black	...
112	red	...

Bicycle

ID<<PK>>	Model	...
87	racer	...

Figure 5. Mapping of polymorphic associations using *Exclusive Arcs*.

Another approach is named *Reverse the Relationship* and is shown in Figure 6. It uses an intermediate table with two foreign key columns like the aforementioned approach for n:m relationships. Such an intermediate table has to be defined for each possible type (table) that can be referenced by the polymorphic association. [35], [36] The application has to ensure that only one entry of all subordinate tables is assigned to the entry of the superordinate table. [35, p. 96ff]



Owner

ID<<PK>>	Name	Prename
1234	Müller	Hans
2456	Müller	Lieschen
5634	Meier	Ernst

Cars

Owner<<FK>>	Car<<FK>>
1234	101
2456	112

Car

ID<<PK>>	Color	...
101	black	...
112	red	...

Bicycles

Owner<<FK>>	Bicycle<<FK>>
5634	87

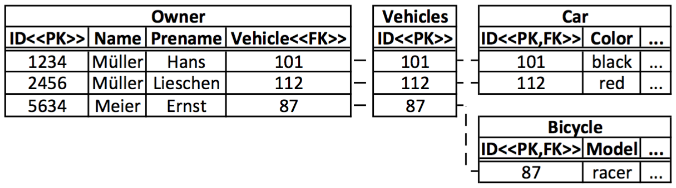
Bicycle

ID<<PK>>	Model	...
87	racer	...

Figure 6. Mapping of polymorphic associations using *Reverse the Relationship*.

The third approach uses a super table (or "base table") and is named *Base Parent Table*. It is based on the basic idea of polymorphism where subtypes can be referenced using a common, often abstract supertype. In most cases, these supertypes themselves are not mapped to the relational database.

The strategy uses a table to represent a supertype for all its subtypes' tables as shown in Figure 7.



Owner			
ID<<PK>>	Name	Prename	Vehicle<<FK>>
1234	Müller	Hans	101
2456	Müller	Lieschen	112
5634	Meier	Ernst	87

Vehicles
ID<<PK>>
101
112
87

Car		
ID<<PK,FK>>	Color	...
101	black	...
112	red	...

Bicycle		
ID<<PK,FK>>	Model	...
87	racer	...

Figure 7. Mapping of polymorphic associations using *Base Parent Table*.

Such a base table only consists of one column containing a primary key value. The assigned subordinate entry has the same primary key value as the entry of the base table. Thus, an unambiguous assignment is possible. This approach has the big advantage that base tables do not have to be considered in queries. They are only used to ensure referential integrity. [35, p. 100ff]

IV. Existing Solutions

For a long time, differences between both the object-oriented and relational paradigm were bridged by simple protocols like Java Database Connectivity (JDBC) and Open Database Connectivity (ODBC), which provide a general interface to different relational databases. These interfaces have the disadvantage that the programmer itself is responsible for data exchange between objects and tables. Due to the mixing of SQL statements and object-oriented commands, this usually leads to complex program code that is not easily maintained. [31]

OR mappers are used to realize a simpler and smarter mapping between objects and table entries on the one side and a clear separation between the object-oriented and relational layer on the other side. Thus, the application can be developed independently of the mapping and the database. As a consequence, different development teams can be deployed. [31]

There are several tools for OR mapping with different features and documentation. Examples are Hibernate (Java), NHibernate (.NET), ADO.NET Entity Framework (.NET), LINQ to SQL (.NET), Doctrine (PHP), ODB (C++), LiteSQL (C++), and QxOrm (C++). Not every existing mapper features all three standard mapping strategies for inheritance. Another main difference is how the mapping approach can be specified. In particular, OR mappers like Hibernate [37] and NHibernate [38] recommend an XML-based mapping while mappers like ODB [39] and QxORM [40] recommend the opposite.

The applicability of an OR mapper depends on the utilized application. In the presented scenario, this is the 3D simulation system VEROSIM, which is subsequently introduced before presenting the evaluation of existing OR mappers.

VEROSIM uses an in-memory runtime database called Versatile Simulation Database (VSD) for its internal data management. It is an object-oriented graph database providing the means to describe structure as well as behavior of a simulation model. Besides interfaces for data manipulation, it provides a change notification mechanism for updates, inserts and deletes. In VSD, objects are called instances and their classes are described by so-called meta instances representing the meta information system of VSD. Instances can have properties for simple values (e.g., integers or strings) or for referencing other instances – either a single target (1:1) or a list of targets (1:n). During runtime, instances can be identified by a unique id. VEROSIM and VSD are implemented in C++. [5]

Thus, an OR mapping is required that maps data of a runtime database like VSD onto a relational database. None of the existing OR mappers support a direct mapping of a runtime database's meta information system. They only map object-oriented classes and objects of a specific programming language. Similarly, the approach used in our previous work [2] maps a relational database to a generic object interface that is subsequently mapped to VSD. Thus, if one of these mappers is used, a second mapping is required to map between the meta information system and the object-oriented layer of the OR mapper.

Based on meta instances, any VSD instance can be classified during runtime. This is a key advantage for the OR mapping with regard to the generation and maintenance of all mappings. Thus, the decision was made to develop a new OR mapper. This allows the OR mapping to be tailored to the requirements of runtime simulation databases like VSD.

V. Object-Relational Mapper for 3D Simulation Systems

A basic decision criterion for OR mapping is the definition of the database schema. Given an existing object-oriented schema, *forward mapping* is used to derive a relational database schema. In contrast, if the initial situation is a given relational database schema, *reverse mapping* is used to derive an object-oriented schema. As already mentioned, database independence is a key aspect of OR mapping. In reverse mapping, this aspect is omitted as a specific database schema of a particular RDBMS is used as the basis for the mapping. [31] The focus of the presented OR mapper is forward mapping to map existing model data of the 3D simulation system onto an arbitrary relational database. Nevertheless, reverse mapping is supported in the concept as well to use the 3D simulation system for other existing databases (see the upper path in Figure 8).

The designed forward mapping of the presented OR mapper is briefly described in the following paragraph and the overall structure of the OR mapper is shown in Figure 8.

First of all, the database schema has to be generated to be able to store object-oriented simulation data in the relational database. Subsequently, a schema synchronization defines a schema mapping between the object-oriented and the relational schema. More details on this are given in [2]. The schema mapping defines which meta instance is mapped to which table. Based on this mapping, initial simulation model data can be stored. *Generate Schema Based on Meta Information* and *Export Model Data in Database* are performed only once and can be seen as the initialization of the OR mapping. Subsequently, model data can be loaded from the relational database and updated within the simulation database. A change tracking mechanism keeps track of changes within the simulation database and allows for their resynchronization to the relational database.

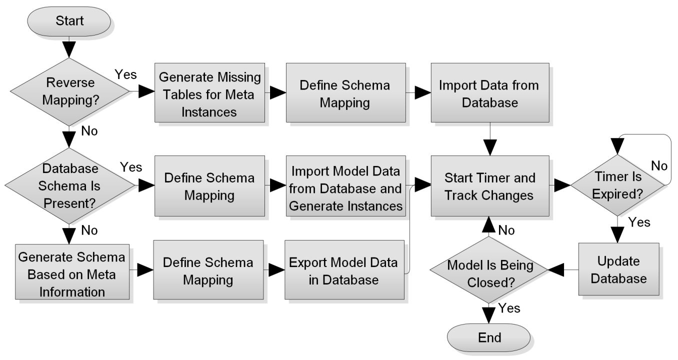

Start
Reverse Mapping?
Yes
Generate Missing Tables for Meta Instances
Define Schema Mapping
Import Data from Database
No
Database Schema Is Present?
Yes
Define Schema Mapping
Import Model Data from Database and Generate Instances
Start Timer and Track Changes
Timer Is Expired?
No
Yes
No
Generate Schema Based on Meta Information
Define Schema Mapping
Export Model Data in Database
Model Is Being Closed?
Update Database
No
Yes
End

Figure 8. Sequence diagram of the presented OR mapping approach.

In most cases, structural aspects are associated with OR mapping. Behavioral and architectural aspects are often considered secondarily although they are not less important [30, p. 58]. All three aspects should be regarded when developing an OR mapper.

Architectural aspects define the communication between business logic and database. The basic principle is not to mix up the business logic with SQL statements, but rather to use separate classes for database access. These can be classified in four strategies: *Row Data Gateway*, *Table Data Gateway*, *Active Record* and *Data Mapper*. To completely isolate business logic, [30] recommends a Data Mapper. Although this is the most complex strategy, it is used for the developed OR mapper to realize an independent layer between the 3D simulation system and the selected relational database. As a result, both systems can independently be extended. Furthermore, Data Mapper is especially well suited for complex structures. [30, p. 49f]

Behavioral aspects define how data can be loaded from or saved to the relational database. With only a few instances to manage, it is easy to keep track of loaded, modified or removed instances and to synchronize these changes with the database. The more instances must be managed, the more complex this process gets. In addition, if various users or processes can access the database, it is even more complex. Here, it has to be ensured that a loaded instance contains valid and consistent data. Following [30], the pattern *Unit of Work* is indispensable to solve this behavioral and concurrency problem, see Figure 9. A Unit of Work can be seen as a control for OR mapping. It registers all loaded, removed or newly created instances as well as changes. A central concept of the Unit of Work is that it aggregates all changes and synchronizes them in their entirety rather than letting the application call separate stored procedures. Alternatives to a central Unit of Work are to immediately synchronize changes or to set dirty flags for each changed object. [30, p. 54f]

Given its many advantages, a Unit of Work is used in the presented OR mapper. To avoid repetitive loading of the same instances, the Unit of Work is combined with the pattern *Identity Mapping* as shown in Figure 9. An Identity Mapping records each instance loaded into the simulation database and maps it to the related tuple in the relational database. Before loading an instance from its tuple, the Unit of Work checks if there already is an Identity Mapping for this instance, which is especially important for lazy loading. [30, p. 55f] Compared

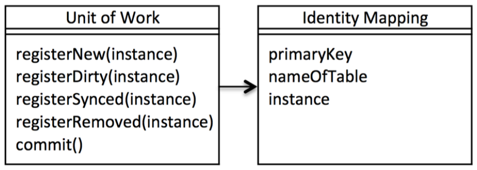
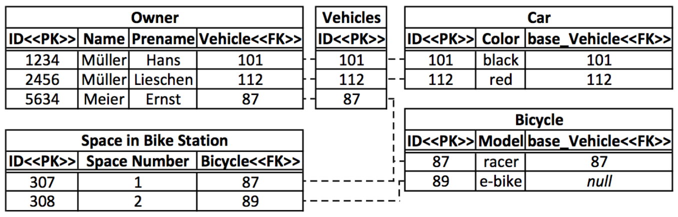


Figure 9. Combination of the patterns *Unit of Work* and *Identity Mapping* (adapted from [30]).

to literature [30] we extended the dirty mechanism. Instead of only registering whole instances as dirty, modified properties are registered as well. This allows to synchronize changes more efficiently.

The fundamentals of structural aspects are described in Section III. To minimize the overall number of joins, the Concrete Table Inheritance strategy was chosen for mapping inheritance. Furthermore, two strategies are selected to map relationships. 1:1 relationships are mapped to simple foreign key columns whereas 1:n relationships are mapped to association tables. However, this is only possible for monomorphic associations. For the polymorphic case, the strategies described in Subsection III-C have to be evaluated. Due to the high administrative effort, Exclusive Arcs is inapplicable. The other two strategies are compared regarding the formulation of queries. Base Parent Table allows for simpler queries. However, the theoretical mapping of this strategy (Figure 7) does not fit in combination with the aforementioned selected mappings for inheritance and monomorphic associations. In practice, a subordinated instance can be referenced by both a monomorphic and a polymorphic association of superordinated instances. As a consequence, the foreign key constraint could be violated. So the theoretical mapping of Base Parent Table is adapted to fit in combination with the aforementioned selected mappings for inheritance and monomorphic associations as shown in Figure 10. As an advantage, both the base table and the additional foreign key column do not need to be considered in queries. They are only used to ensure referential integrity.



Owner			
ID<<PK>>	Name	Prename	Vehicle<<FK>>
1234	Müller	Hans	101
2456	Müller	Lieschen	112
5634	Meier	Ernst	87

Vehicles
ID<<PK>>
101
112
87

Car		
ID<<PK>>	Color	base_Vehicle<<FK>>
101	black	101
112	red	112

Space in Bike Station		
ID<<PK>>	Space Number	Bicycle<<FK>>
307	1	87
308	2	89

Bicycle		
ID<<PK>>	Model	base_Vehicle<<FK>>
87	racer	87
89	e-bike	*null*

Figure 10. Adapted *Base Parent Table* mapping of polymorphic associations.

Another important part of an OR mapper is data type mapping. Data types of the object-oriented data model can differ from those of the relational data model. Thus, a data type mapping has to be defined. The developed OR mapper comprises an interface to use a dynamic data type mapping, which can be adapted for each database and its related data types. This is one main aspect of the supported database independence. Furthermore, the utilized Qt framework [41] (QSqlDatabase) allows for a vendor-independent database communication. Altogether, the developed OR mapper can easily support different RDBMSs.

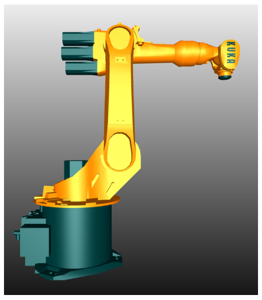
After schema synchronization, model data can be loaded from the relational database populating the simulation database with corresponding instances. A so-called *eager loading strategy* is used to immediately load and generate all model instances. The Unit of Work generates an identity mapping for each loaded instance. This provides an unambiguous mapping between each loaded instance and the corresponding tuple in the relational database. Furthermore, a so-called *lazy loading strategy* is specified for selectively loading model data from the database. It is based on the ghost strategy presented in [30, p. 227ff]. Here, typically necessary information, like primary key and table name, is determined for all tuples from all tables regardless whether the instance is loaded or not. Ghost instances are generated containing only this partially loaded data. [30, p. 227ff] The presented OR mapper uses a *Ghost Identity Mapping* (Figure 9). The advantage of this modified approach is that only "complete" instances are present in the 3D simulation system's runtime database.

VI. Application

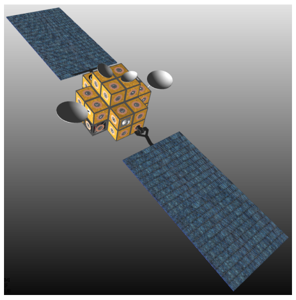
As mentioned before, schema generation and synchronization work independently of the selected simulation model. All required structures are defined during schema generation. In the evaluated configuration of the 3D simulation system VEROSIM, 910 tables, $1,222$ foreign key columns, and $2,456$ association tables are generated to map all meta instances and $1{:}1$ as well as $1{:}n$ relationships. The schema generation takes about 200 seconds on a local PostgreSQL 9.4 installation. The required schema mapping is built up during schema synchronization and takes about 2.7 seconds.

Due to its flexibility, the OR mapper can be used for any simulation model. The prototype is evaluated using two exemplary models from two different fields of application: industrial automation and space robotics. Given the current functional range of the presented prototype, further tests do not appear to provide any additional insights.

Figure 11(a) shows the first model from the field of industrial robotics. The robot model contains only a few objects so that only 173 primary keys have to be generated to map all objects to table entries. It takes about 0.43 seconds to store the whole robot into the relational database and about 4.7 seconds to load it.



(a) Industrial robot simulation model.



(b) Modular satellite simulation model (data: [4]).

Figure 11. Evaluated simulation models.

The second model (Figure 11(b)) is a modular satellite. In comparison, it contains much more objects so that $19,463$ primary keys are generated to map all objects to table entries. In this case, it takes about 22 seconds to store all objects of the satellite and about 7.1 seconds to load all of them from the relational database.

As mentioned in Section IV, a comparable interface to existing ORM solutions would be less efficient as well as more complex and time-consuming to realize due to the necessary second mapping. Thus, we refrain from performing such comparisons.

VII. Summary, Conclusion and Future Work

In contrast to flat files, database technology provides many advantages for managing 3D simulation models. However, existing approaches for database integration into applications with a 3D context do not provide a sufficiently comprehensive and flexible solution. Given the prevalence of relational DBMS and the preferred object-oriented modeling of 3D simulation models, an OR mapping approach is recommended. Using existing ORM solutions (including our previous work [2]), an intermediate layer cannot be avoided. Thus, in this work, we develop a direct OR mapping approach.

The presented OR mapper allows a flexible and generic mapping between an object-oriented runtime simulation database and a relational database. It is based on the meta information system so that an OR mapping can be performed for arbitrary simulation models. The mapper detects schema changes, i.e., new or modified meta instances, and automatically adapts the used mapping without the need for a manual definition of persistent elements. Hence, compared to existing OR mappers, a complex and error-prone manual maintenance of the defined mapping can be omitted. The presented OR mapper separates the 3D simulation system and the used relational database so that business logic is not mixed with SQL statements. As a result, the 3D simulation system can be developed independently from data storage. Future projects can profit by time saving as they do not have to realize persistent data storage separately. Following [42, p. 525], the programming effort for storing objects in relational databases accounts for 20-30% of the total project effort. Finally, the presented OR mapping is successfully evaluated using two simulation models from two different fields of application (industrial automation and space robotics).

In future, data type mapping can be extended by more specialized data types and further RDBMSs can be combined with the prototype. Furthermore, the currently generated structures within the relational database do not contain explicit information on the inheritance relationships as they are not needed by the simulation system itself (they can be retrieved from its meta information system). However, to allow third party applications to interpret the data, inheritance structures would be of interest. Another aspect to investigate is the mapping of queries and operations. For the former, an object-based query language meeting VSD's demands, e.g., XQuery or (a variation of) Java Persistence Query Language (JPQL) or Hibernate Query Language (HQL), needs to be mapped to proper SQL queries. Further performance optimizations and, with an extended functional range of the mapper, evaluations beyond the results from the student project could be performed, as well. Finally, we could examine further applications, e.g., from other fields like forestry.

REFERENCES

[1] M. Hoppen and J. Rossmann, “A Database Synchronization Approach for 3D Simulation Systems,” in DBKDA 2014,The 6th International Conference on Advances in Databases, Knowledge, and Data Applications, A. Schmidt, K. Nitta, and J. S. Iztok Savnik, Eds., Chamonix, France, 2014, pp. 84–91.

[2] M. Hoppen, M. Schluse, J. Rossmann, and B. Weitzig, “Database-Driven Distributed 3D Simulation,” in Proceedings of the 2012 Winter Simulation Conference, 2012, pp. 1–12.

[3] A. Kemper and A. Eickler, Database Systems – An Introduction (orig.: Datenbanksysteme–Eine Einführung), 9th ed. München: Oldenbourg Verlag, 2013.

[4] J. Weise et al., “An Intelligent Building Blocks Concept for On-Orbit-Satellite Servcing,” in Proceedings of International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS), 2012, pp. 1–8.

[5] J. Roßmann, M. Schluse, C. Schlette, and R. Waspe, “A New Approach to 3D Simulation Technology as Enabling Technology for eROBOTICS,” in 1st International Simulation Tools Conference & EXPO 2013, SIMEX’2013, 2013, pp. 39–46.

[6] The PostgreSQL Global Development Group, “PostgreSQL: About,” 2015, URL: http://www.postgresql.org/about/ [retrieved: May, 2016].

[7] B. Damer et al., “Data-Driven Virtual Environment Assembly and Operation,” in Virtual Ironbird Workshop, 2004, p. 1.

[8] U. Sendler, The PLM Compendium: Reference book of Product Lifecycle Management (orig.: Das PLM-Kompendium: Referenzbuch des Produkt-Lebenszyklus-Managements). Berlin: Springer, 2009.

[9] Verein Deutscher Ingenieure (VDI), “VDI 2219 - Information technology in product development Introduction and economics of EDM/PDM Systems (Issue German/English),” Düsseldorf, 2002.

[10] Y. Zhao et al., “The research and development of 3D urban geographic information system with Unity3D,” in Geoinformatics (GEOINFORMATICS), 2013 21st International Conference on, 2013, pp. 1–4.

[11] D. Pacheco and S. Wierenga, “Spatializing experience: a framework for the geolocalization, visualization and exploration of historical data using VR/AR technologies,” in Proceedings of the 2014 Virtual Reality International Conference, 2014.

[12] A. Martina and A. Bottino, “Using Virtual Environments as a Visual Interface for Accessing Cultural Database Contents,” in International Conference of Information Science and Computer Applications (ICISCA 2012), Bali, Indonesia, 2012, pp. 1–6.

[13] T. Guan, B. Ren, and D. Zhong, “The Method of Unity3D-Based 3D Dynamic Interactive Query of High Arch Dam Construction Information,” Applied Mechanics and Materials, vol. 256-259, 2012, pp. 2918–2922.

[14] G. Van Maren, R. Germs, and F. Jansen, “Integrating 3D-GIS and Virtual Reality Design and implementation of the Karma VI system,” in Proceedings of the Spatial Information Research Centre’s 10th Colloquium. University of Otago, New Zealand, 1998, pp. 16–19.

[15] J. Haist and V. Coors, “The W3DS-Interface of Cityserver3D,” in European Spatial Data Research (EuroSDR) u.a.: Next Generation 3D City Models. Workshop Papers : Participant’s Edition, Kolbe and Gröger, Eds., Bonn, 2005, pp. 63–67.

[16] M. Kamiura, H. Oisol, K. Tajima, and K. Tanaka, “Spatial views and LOD-based access control in VRML-object databases,” in Worldwide Computing and Its Applications, ser. Lecture Notes in Computer Science, T. Masuda, Y. Masunaga, and M. Tsukamoto, Eds. Springer Berlin / Heidelberg, 1997, vol. 1274, pp. 210–225.

[17] E. V. Schweber, “SQL3D - Escape from VRML Island,” 1998, URL: http://www.infomaniacs.com/SQL3D/SQL3D-Escape-From-VRML-Island.htm [retrieved: May, 2016].

[18] T. Scully, J. Doboš, T. Sturm, and Y. Jung, “3drepo. io: building the next generation Web3D repository with AngularJS and X3DOM,” in Proceedings of the 20th International Conference on 3D Web Technology, 2015.

[19] Z. Wang, H. Cai, and F. Bu, “Nonlinear Revision Control for Web-Based 3D Scene Editor,” in Virtual Reality and Visualization (ICVRV), 2014 International Conference on, 2014, pp. 73–80.

[20] J. Doboš and A. Steed, “Revision Control Framework for 3D Assets,” in Eurographics 2012 - Posters, Cagliari, Sardinia, Italy, 2012, p. 3.

[21] D. Schmalstieg et al., “Managing complex augmented reality models,” IEEE Computer Graphics and Applications, vol. 27, no. 4, 2007, pp. 48–57.

[22] M. Nour, “Using Bounding Volumes for BIM based electronic code checking for Buildings in Egypt,” American Journal of Engineering Research (AJER), vol. 5, no. 4, 2016, pp. 91–98.

[23] B. Domínguez-Martín, “Methods to process low-level CAD plans and creative Building Information Models (BIM),” Doctoral Thesis, University of Jaén, 2014.

[24] S. Hoerster and K. Menzel, “BIM based classification of building performance data for advanced analysis,” in Proceedings of International Conference CISBAT 2015 Future Buildings and Districts Sustainability from Nano to Urban Scale, 2015, pp. 993–998.

[25] H. Eisenmann, J. Fuchs, D. De Wilde, and V. Basso, “ESA Virtual Spacecraft Design,” in 5th International Workshop on Systems and Concurrent Engineering for Space Applications, 2012.

[26] M. Fang, X. Yan, Y. Wenhui, and C. Sen, “The Storage and Management of Distributed Massive 3D Models based on G/S Mode,” in Lecture Notes in Information Technology, vol. 10, 2012.

[27] D. Iliescu, I. Ciocan, and I. Mateias, “Assisted management of product data: A PDM application proposal,” in Proceedings of the 18th International Conference on System Theory, Control and Computing, Sinaia, Romania, 2014.

[28] H. Takemura, Y. Kitamura, J. Ohya, and F. Kishino, “Distributed Processing Architecture for Virtual Space Teleconferencing,” in Proc. of ICAT, vol. 93, 1993, pp. 27–32.

[29] D. J. Armstrong, “The Quarks of Object-Oriented Development,” Communications of the ACM, vol. 49, no. 2, 2006, pp. 123–128.

[30] M. Fowler, Patterns of Enterprise Application Architecture, 1st ed. Addison Wesley, 2002.

[31] A. Schatten, “O/R Mapper und Alternativen,” 2008, URL: http://www.heise.de/developer/artikel/O-R-Mapper-und-Alternativen-227060.html [retrieved: May, 2016].

[32] T. Neward, “The Vietnam of Computer Science,” 2006, URL: http://www.odbms.org/2006/01/the-vietnam-of-computer-science/ [retrieved: May, 2016].

[33] S. W. Ambler, “Mapping Objects to Relational Databases: O/R Mapping In Detail,” 2013, URL: http://www.agiledata.org/essays/mappingObjects.html [retrieved: May, 2016].

[34] F. Lodhi and M. A. Ghazali, “Design of a Simple and Effective Object-to-Relational Mapping Technique,” in Proceedings of the 2007 ACM symposium on Applied computing. ACM, 2007, pp. 1445–1449.

[35] B. Karwin, SQL Antipatterns: Avoiding the Pitfalls of Database Programming, 1st ed. Railegh, N.C.: Pragmatic Bookshelf, 2010.

[36] ——, “Practical Object Oriented Models in Sql,” 2009, URL: http://de.slideshare.net/billkarwin/practical-object-oriented-models-in-sql [retrieved: May, 2016].

[37] Hibernate, HIBERNATE–Relational Persistence for Idiomatic Java, 2015, URL: http://docs.jboss.org/hibernate/orm/5.0/manual/en-US/html/index.html [retrieved: May, 2016].

[38] NHibernate Community, NHibernate–Relational Persistence for Idiomatic .NET, 2015, URL: http://nhibernate.info/doc/nhibernate-reference/index.html [retrieved: May, 2016].

[39] Code Synthesis Tools CC, ODB: C++ Object-Relational Mapping (ORM), 2015, URL: http://www.codesynthesis.com/products/odb/ [retrieved: May, 2016].

[40] L. Marty, QxOrm (the engine) + QxEntityEditor (the graphic editor) = the best solution to manage your data in C++/Qt !, 2015, URL: http://www.qxorm.com/qxorm_en/home.html [retrieved: May, 2016].

[41] The Qt Company, “Qt Documentation,” 2016, URL: http://doc.qt.io/qt-5/index.html [retrieved: May, 2016].

[42] A. M. Keller, R. Jensen, and S. Agarwal, “Persistence Software: Bridging Object-Oriented Programming and Relational Databases,” in ACM SIGMOD Record, vol. 22, no. 2. ACM, 1993, pp. 523–528.