# Managing Semantic World Models for eRobotics Applications
# Two Approaches Based on Object-Relational Mapping and on a Graph Database

Martin Hoppen, Juergen Rossmann, Ann-Marie Stapelbroek, and Sebastian Hiester

Institute for Man-Machine Interaction
RWTH Aachen University
Ahornstrasse 55
Aachen, Germany
Email: {hoppen,rossmann}@mmi.rwth-aachen.de,
{ann-marie.stapelbroek,sebastian.hiester}@rwth-aachen.de

***Abstract*—The main objective of eRobotics is to cope with the complexity in the development and lifecycle of current robotic and mechatronic systems. A key technology for eRobotics applications are Virtual Testbeds, a more holistic approach to 3D simulation that considers the entire system under study within its operational environment modeled in so-called Semantic World Models. Ideally, this complex data should be managed using database technology instead of flat file formats. However, existing related approaches fail to fulfill all the identified requirements. Thus, we present a new database synchronization concept whose basic idea is to use local in-memory simulation databases (SimDB) and synchronize them to a central, external database (ExtDB). In this paper, two new realizations of this concept are presented using two different choices for ExtDB: The first is based on an object-relational mapping approach, the second uses the graph database Neo4j. Both approaches are described in detail, evaluated, and finally compared to each other. In conclusion, both approaches are very well-suited but the actual choice depends the concrete application scenario.**

***Keywords–eRobotics; Semantic World Models; Data Management; Object Relational Mapping; Graph Database.***

## I. Introduction

The work at hand combines the results from our two previous publications [1] [2] and extends them by a more comprehensive motivation and a comparison of the two approaches.

Current robotic applications are characterized by complex system structures and expensive hardware prototypes. The eRobotics approach [3], a branch of eSystems Engineering like eHealth or eGovernment, is used to cope with these difficulties by combining the usage of electronic media, simulation technology and robotics concepts. It covers not only the development, but the whole lifecycle of robotic systems. The flexibility of eRobotics enables its usage in other fields of application: The modeling and simulation of environmental scenarios (forest, buildings or whole cities), industrial automation applications, and mechatronic systems like satellites in general (Figure 1).

An important method in eRobotics is 3D simulation. According to [4], the term simulation itself can be defined as the ”preparation, execution and evaluation of targeted experiments with a simulation model”, where a model is defined as a ”simplified reproduction of a planned or existing system”. Accordingly, a model used in 3D simulations focuses on a system’s spatial properties and behavior. The basis of any

Figure 1. Exemplary eRobotics applications: A wood harvester in its operational environment, a distributed city simulation scenario and Virtual Testbed of the International Space Station (ISS).

eRobotics application is a Semantic World Model, a comprehensive 3D model of the system under study itself and its operational environment. It focuses on the problem domain using a domain specific data schema and allows for an inherent interpretation of its meaning. In particular, such models go beyond purely geometric descriptions that are often optimized for rendering only.

Semantic World Models are the basis for so-called Virtual Testbeds (VTBs), a more holistic approach to 3D simulation that considers the entire system under development including its operational environment. In contrast, a standard simulation usually covers and examines only very specific aspects in detail. A VTB, in turn, can be used for various approaches such as simulation-based optimization, reasoning and control.

This can be summarized by the term Simulation-based *X*, with

- *X* = Optimization: The VTB is used during the development phase of the system. By running different simulations with varying system structures and properties the best alternative can be chosen.
- *X* = Reasoning: The VTB is used by the system in live operations, serving as a mental model of the system. This model can be used for optimized decision-making by simulating different action alternatives.
- *X* = Control: A control for a device or process is developed within the VTB (using above mentioned simulation-based optimization). By exchanging the virtual sensors and actuators with their physical counterpart, this control can then be used on the physical device or process, without further implementation or adaption.

An example is given in Figure 2.

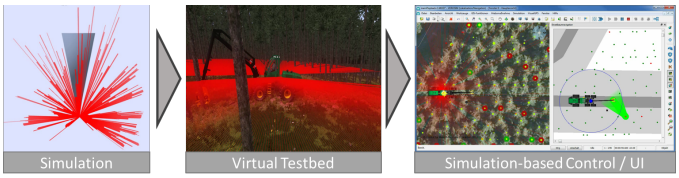



Figure 2. Difference between a simulation (of a single laser scanner), a Virtual Testbed (of a laser scanner equipped wood harvester and its operational environment) and Simulation-based *X* (user interface of a Simulation-based Control approach).

The inherent properties of Semantic World Models are very similar to those of 3D data like Computer-aided Design (CAD) data. Typically, they consist of a huge number of parts with many different types that feature a hierarchical structure with interdependencies. When developing eRobotics applications, this complexity needs to be managed appropriately. For that purpose, instead of widespread flat 3D file formats, the usage of database technology is recommended. Database technology provides many advantages for modern eRobotics applications. Multi-user support with access rights management, safe transactions and concurrency control can allow large development teams or distributed components to corporately work on complex systems. Semantic World Models like environmental models (e.g., forests or cities) or the ISS benefit from database management systems' (DBMS) support for managing huge amounts of data using techniques like (spatial) indexing or distributed databases. Query languages, data independence, and client-server architectures allow for the decoupling of concerns between simulation and data management. Schema support and metadata allow to centrally define a "common language" for (possibly heterogeneous) software components in the development and the whole lifecycle of eRobotics applications.

In particular, five main requirements for a data management approach for eRobotics applications can be identified:

1) Following [5], object-oriented data modeling is optimal to cope with the aforementioned complexity of CAD data and thus of Semantic World Models. Thus, it must be supported by the approach.
2) In eRobotics applications not only the data, but also the processes are complex. They are made manageable by using distributed approaches like distributed simulation, distributed data processing, distributed modeling (collaboration), distributed data acquisition, or distributed control architectures. Thus, a data management approach must also feature a distributed architecture, in particular, to replicate data to different clients, sites or projection screens.
3) For simulation and rendering, Semantic World Model data must be available in memory in every simulation cycle. Thus, the approach must use in-memory technology to keep necessary data available for (live) simulation.
4) Semantic World Models use domain specific data schemata that focus on the problem domain and allow for an inherent interpretation of their meaning. Thus, an appropriate data management approach for eRobotics applications must flexibly adapt to different schemata to make it universally applicable in many scenarios. Examples for such data schemata are CityGML for cities, ForestGML for forest and forestry models, Industrial Foundation Classes (IFC) [6] for building information modeling (BIM) scenarios, Automation Modeling Language (AutomationML) [7] in industrial automation applications, or Systems Modeling Language (SysML) [8] for system specifications.
5) Finally, Simulation-based *X* approaches need to be supported. For *X*=Optimization and *X*=Reasoning, simulation runs need to be recorded and analyzed. Ideally, this can be realized using an integrated versioning or temporal data management that is independent of the specific application schema to make it available to all eRobotics applications. For *X*=Reasoning and *X*=Control, the identical model and thus the identical data management approach must be reusable in live operation.

The rest of this contribution is structured as follows. In the next section, related work is identified representing the state of the art in managing 3D data using database technology. Section III introduces the basic idea of database synchronization that builds the basis for the two approaches presented in Section IV and Section V. In Section VI, both approaches are compared before Section VII gives a conclusion.

## II. Related Work

A variety of applications similarly work with 3D data like Augmented Reality applications, collaborative Virtual Environments, applications for city or building modeling and planning, or Product Data Management (PDM) systems, which are particularly used for managing CAD or other CA*x* (e.g., Computer-aided Engineering or Computer-aided Manufacturing) data. Mostly, they use object-oriented data modeling [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23]. Among such applications, different motivations for using database technology can be identified:

- Persistence [9] [13] [14] [15] [18] [19] [20] [21] [22] [23] [24]
- Query capabilities [13] [14] [15] [16] [17] [19] [20] [21] [22] [23]
- Multiuser support [10] [13] [14] [15] [16] [18] [20] [21] [22] [23] [25]
- Client-server model [10] [22] [23] [26]
- Large model data [10] [11] [12] [17] [18] [22] [23] [27]
- Access control and rights management [17] [22] [23] [24]
- Internet data provisioning [11] [12] [18] [19] [20] [21] [22] [23] [27]
- Temporal data management / versioning [19] [22] [23]
- Integration of existing databases [28]
- Integration of different data formats [11] [12] [22] [23]

Thus, related applications have similar reasons for the adoption of database technology. In all of them, different approaches are used to integrate databases into the application. Sometimes, databases are used to store additional information (meta information, documents, films, positions, hierarchical structure ...) on scene objects or parts [22] [23] [24] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37]. However, for eRobotics applications, the whole Semantic World Model should be managed using database technology to obtain the referred benefits for all its objects and their properties. This ensures that every aspect is queryable, distributable, and versionable and that the complete application schema is reproduced in the database to enforce it as a common language for all subsystems.

Other systems use a database to store scene data itself, but not all of them with an object-oriented data modeling, which is ideally suited for Semantic World Models. Another important aspect for eRobotics applications is the support for arbitrary application schemata. Many systems use a generic (scene-graph-like) geometric model, in most cases with attributes [11] [17] [18] [19] [20] [26] [27]. In such scenarios, schema flexibility can be achieved to a certain extent by providing import (and export) to different file formats [11] [38] [39] [40]. Some approaches support different or flexible schemata. For example in [26], schema alteration is realized by adding attributes to generic base objects. Other systems support a selection of different static [11] or dynamic [10] [17] schemata. However, most approaches focus on a specific field of application, thus, requiring and supporting only a corresponding fixed schema. This is very common and described in many similar publications presenting applications with a 3D context, for example many from the field of BIM [41] [42] [43] [44] [45] [46] [47]. While PDM systems [22] [23] in principle support arbitrary schemata they are not explicitly reflected within the database schema due to their "black box integration" approach. Similar vaulting-based approaches with a 3D context can be found in other publications [48] [49] [50] [51] [52]. However, to flexibly support various eRobotics applications, arbitrary application schemata of Semantic World Models need to be explicitly supported. 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-like approach [10] [18] [25] with client-side databases. The latter is favorable for eRobotics applications, e.g., to provide schema flexibility or a query interface on client-side, as well. Finally, some approaches support temporal data management or versioning [19] [22] [23] [38] [39] [40] [41] [43] [44] [48] [52] [53] [54] [55]. In most cases, these approaches require special schema structures and many are based on file vaulting. To be flexibly used in eRobotics applications, however, it needs to be available for arbitrary application schemata.

Thus, especially due to deviant motivations, there is no single approach fulfilling all the requirements identified for managing Semantic World Models in eRobotics applications. This was the motivation for the development of our basic concept of database synchronization presented in the next section.

## III. Database Synchronization Concept

Based on the requirements described above and as previously shown in [56] [57], we developed a new database synchronization concept ideally suited for eRobotics applications. Its basic idea is to combine and synchronize two types of databases (Figure 3): Each 3D simulation system uses a local database (dubbed SimDB) as its core data management component. This database is an active, in-memory database whose main purpose is to cache the shared Semantic World Model and to build the basis for its runtime execution. All instances of SimDB are synchronized to an external database (dubbed ExtDB). It centrally manages the shared model, provides a persistence layer, can be used for versioning and serves as a communication hub for the distributed system.

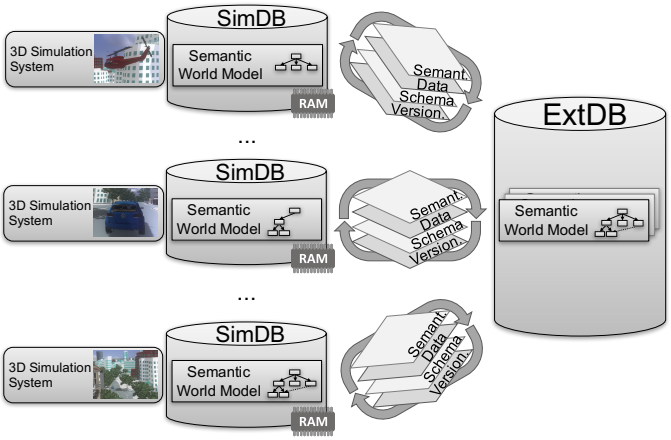



Figure 3. Basic idea of the database synchronization approach for eRobotics applications that synchronizes databases on four levels.

The synchronization component between each pair of ExtDB and SimDB performs a synchronization on four different levels:

1) On schema level, the schema is synchronized from ExtDB to every SimDB to make the whole distributed system ”speak the same language”.
2) On data level, instance data is replicated on demand from ExtDB to SimDB and kept in sync to make the shared Semantic World Model available to the 3D simulation systems.
3) On a semantic level, where necessary, native data in application specific schemata (e.g., SEDRIS [58]) is selectively translated to allow for an interpretation by the simulation system.
4) On a versioning level, a temporal database is used for ExtDB to synchronize snapshots to SimDB and to make the history of changes to the Semantic World Model persistent.

The question which arises now is, which database technology is best used when implementing this concept. For SimDB, we successfully utilize the simulation database VSD (Versatile Simulation Database) of the 3D Simulation System VEROSIM (Virtual Environments and Robotic Simulation [59]). VEROSIM was originally developed as a virtual reality and robot simulation system. Due to its flexible nature, it has become the basis for the development of various eRobotics applications in the fields of industrial automation, space robotics and environmental modeling. All its rendering and simulation techniques are based on VSD, an object-oriented, in-memory database. Similar to the common scene-graph, VSD builds a graph-like structure by means of objects and references in between. However, unlike scene-graphs, it does not only support a fixed set of generic node types. VSD rather provides means to freely configure a schema for its objects. By offering views on its contents, time-critical components like rendering and simulation can still quickly access their respective part of interest without repetitively traversing the whole graph. A VSD schema describes the structure as well as the behavior of its objects. That is, the simulation logic is not externally applied to the Semantic World Model but is in fact part of the objects it comprises. For that reason, VSD can be called an active database. Last but not least, VEROSIM and VSD are based on C++ and provide a proprietary meta system.

In VSD, objects are called instances and are characterized by properties. Such properties can either be value properties (Val-Properties) with basic or complex data types or reference properties. The latter model $1:1$ (Ref-Properties) or $1:n$ (RefList-Properties) directed relationships between instances. Furthermore, these relationships can be marked to *contain* target instances using an *autodelete* flag allowing to model UML (Unified Modeling Language) composite aggregations.

VSD comprises a meta information system providing access to its schema and also to specify its schema. So-called meta instances describe an instance’s class (name, inheritance, etc.) and so-called meta properties its properties (name, type, etc.). Figure 4 shows VSD’s data model.

For ExtDB, we currently use a generic, third-party object-relational mapper (OR mapper) called SupportGIS-Java (SGJ) [60] in combination with standard relational back-ends (mostly PostgreSQL [61]). SGJ’s main purpose and field of application

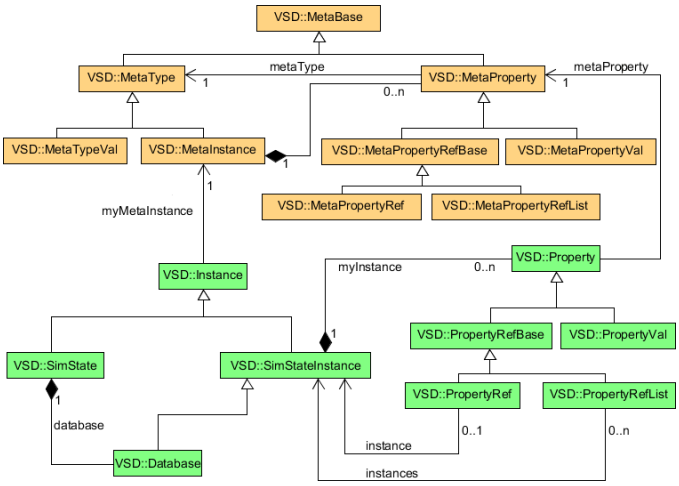



Figure 4. VSD data model (top: metadata, bottom: instance data).

is managing geodata for GIS (Geographic Information System). Thus, an import principle is its adherence to international standards like GML (Geography Markup Language [62]). SGJ flexibly supports arbitrary object-oriented application schemata by using a generic base schema within the back-end. This base schema is used to flexibly store a description of the respective application schema in terms of management table entries and generic data tables. This flexibility motivated our usage of SGJ in many different eRobotics applications.

However, a drawback of this current solution is the need for an additional translation layer between SimDB, the synchronization interface and ExtDB (Figure 5). To persist a new VSD instance, it firstly has to be translated into an object of the SGJ API (Application Programming Interface). This object representation is then translated into a relational representation. The same (in reverse order) applies to loading data into VSD.

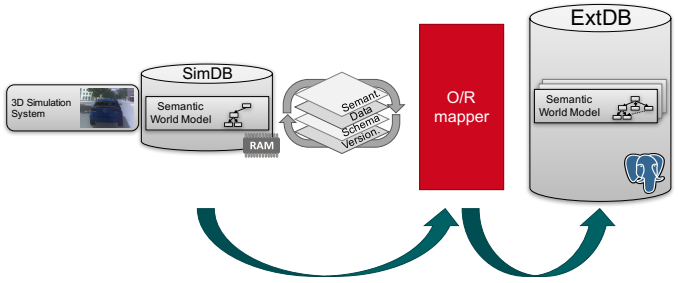



Figure 5. Drawback of the current SGJ-based solution: An additional translation layer between SimDB and ExtDB.

Thus, we evaluated alternative choices for ExtDB in two student projects. The first is an OR mapper solution directly built into the synchronization component and, thus, omitting the additional layer currently imposed by SGJ. In the prototype it is also combined with a PostgreSQL back-end. The main motivation for using an OR mapper is the same: The prevalence, matureness and well-defined theoretical basis of relational database management systems (RDBMS) lead to an extensive software availability but also to wide user acceptance especially in business and industry. The OR mapper-based approach is presented in the next section. The second approach

uses graph database technology, namely the graph database Neo4j [63] [64]. The study was mainly motivated by VSD's graph database like structure and the drawbacks of OR mapper solutions subsumed by the well-known object-relational impedance mismatch. This second approach is presented in Section V.

## IV. Direct Object-Relational Mapping 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.

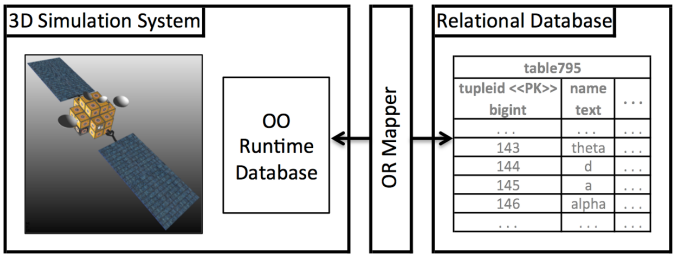



Figure 6. OR mapping for a 3D simulation system with an object-oriented runtime database (data: [65]).

In this section, we present an OR mapper for 3D simulation systems with an object-oriented runtime database and a meta information system, see Figure 6. The work was conducted as a student project and is based on our previous work as introduced above. A prototypical implementation is based on the 3D simulation system VEROSIM and PostgreSQL as an 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 built during a schema synchronization. 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 16).

### A. State of the Art

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 [66]. 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.

*1) 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 8 regarding the exemplary inheritance hierarchy from Figure 7.

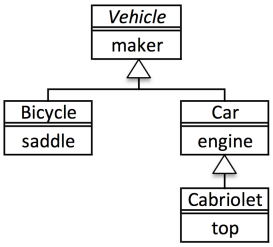
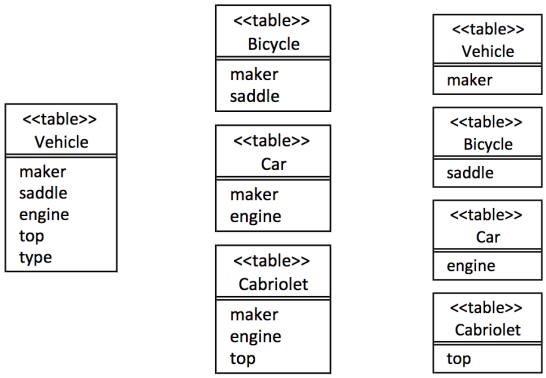



Figure 7. Exemplary inheritance hierarchy (adapted from [67, p. 62f]).

The first strategy is named *Single Table Inheritance* [67] and maps all classes of one inheritance hierarchy to one table, see Figure 8(a). A discriminator field is used to denote the type of each tuple [68]. An advantage is that all data is stored in one table preventing joins and allowing simple updates [67, p. 63]. Unfortunately, this strategy leads to a total denormalization, which is contrary to the concept of relational databases [68].




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

Figure 8. Standard mapping strategies for inheritance (adapted from [67, p. 62f]).

The *Concrete Table Inheritance* [67] strategy maps each concrete class to one table, see Figure 8(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 [67, p. 62f].

The third standard mapping strategy for inheritance is named *Class Table Inheritance* [67] and uses one table for each class of the hierarchy, see Figure 8(c). It is the easiest approach to map objects onto tables [67, p. 62] and uses a normalized schema [68]. However, due to the use of foreign keys, this approach realizes an *is-a* relationship as a *has-a* relationship [68]. Thus, multiple joins are necessary if all data of one object is required. This aspect can have an effect on performance [67, p. 62f] [69, p. 7].

Another possibility to map objects onto tables not mentioned in every reference on OR mapping is the generic approach [70]. It differs from the strategies mentioned above as it has no predefined structure. Figure 9 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 [70].

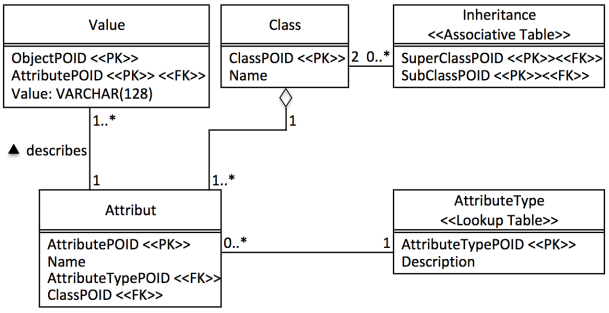



Figure 9. Map classes to generic table structure (adapted from [70, Chapter 2.4]).

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 [67, p. 63].

*2) 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 [69]. 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 [70] [67, p. 58f]. In case of $n$:$m$ relationships, additional tables are mandatory: A so-called association table is used to link the participating tables [70] [67, 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 [70].

Several references describe the aforementioned mapping strategies for relationships. Besides, [71] describes an approach using an additional table regardless of the cardinality. Thus, objects can be mapped onto tables regardless of their relationships. Following [71], 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 [71, p. 1446f].

*3) 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 [72, p. 89]. Thus, a mapping is required to map polymorphic associations onto a relational database. Following [72], [73], 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 10. 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.

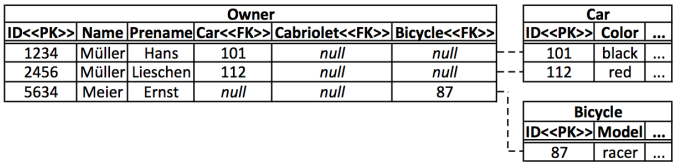



Figure 10. Mapping of polymorphic associations using *Exclusive Arcs*.

Another approach is named *Reverse the Relationship* and is shown in Figure 11. 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 [72] [73]. The application has to ensure that only one entry of all subordinate tables is assigned to the entry of the superordinate table [72, p. 96ff].

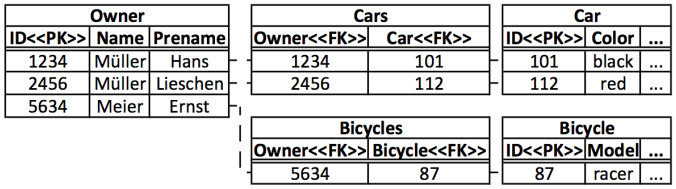



Figure 11. Mapping of polymorphic associations using *Reverse the Relationship*.

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 12.

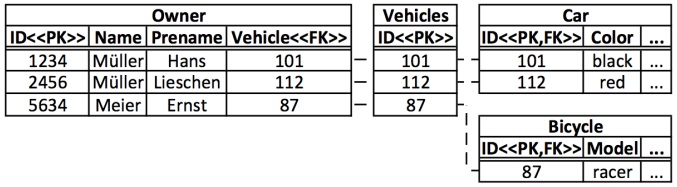



Figure 12. Mapping of polymorphic associations using *Base Parent Table*.

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 [72, p. 100ff].

*4) Existing OR Mappers:* 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 [68].

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 [68].

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 [74] and NHibernate [75] recommend an XML-based mapping while mappers like ODB [76] and QxORM [77] 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. 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 SGJ-based approach used in our previous work 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 (see Figure 5).

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.

### B. Approach

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 [68]. 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 13).

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 13.

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 [57]. 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.

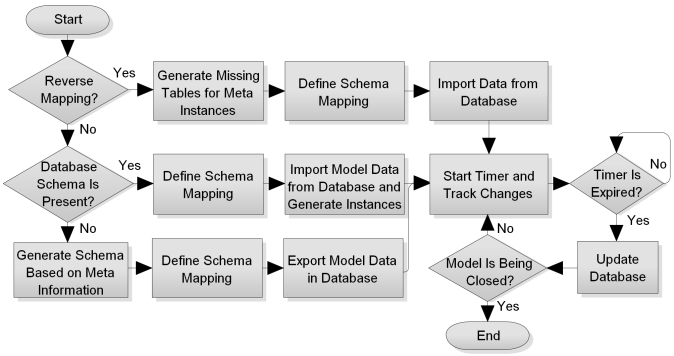



Figure 13. Sequence diagram of the presented OR mapping approach.

In most cases, structural aspects are associated with OR mapping. Behavioral and architectural aspects are often con-

sidered secondarily although they are not less important [67, 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, [67] 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 [67, 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 [67], the pattern *Unit of Work* is indispensable to solve this behavioral and concurrency problem, see Figure 14. 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 [67, p. 54f].

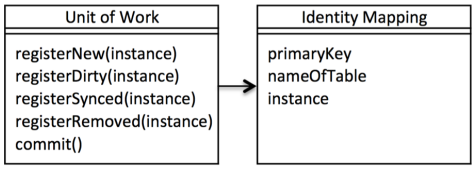



Figure 14. Combination of the patterns *Unit of Work* and *Identity Mapping* (adapted from [67]).

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 14. 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 [67, p. 55f]. Compared to literature [67] 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 IV-A. 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 IV-A3 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 12) 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 15. 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.

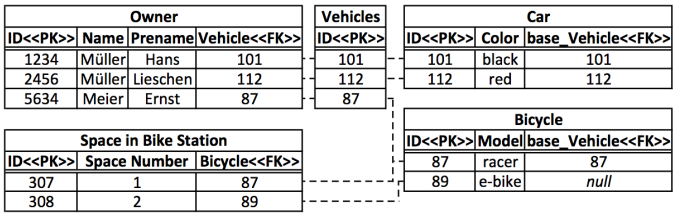



Figure 15. Adapted *Base Parent Table* mapping of polymorphic associations.

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 [78] (QSqlDatabase) allows for a vendor-independent database communication. Altogether, the developed OR mapper can easily support different RDBMSs.

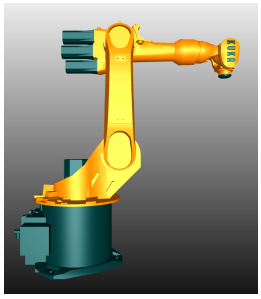
After schema synchronization, model data can be loaded from the relational database populating the simulation database with corresponding instances. A so-called *eager loading strategy* is used to immediately load and generate all model instances. The Unit of Work generates an identity mapping for each 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 [67, 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 [67, p. 227ff]. The presented OR mapper uses a *Ghost Identity Mapping* (Figure 14). The advantage of this modified approach is that only "complete" instances are present in the 3D simulation system's runtime database.

### C. Evaluation

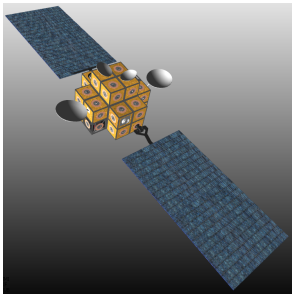
As mentioned before, schema generation and synchronization work independently of the selected simulation model. All required structures are defined during schema generation. In the evaluated configuration of the 3D simulation system VEROSIM, 910 tables, 1,222 foreign key columns, and 2,456 association 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 16(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: [65]).

Figure 16. Evaluated simulation models.

The second model (Figure 16(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.

An overview of the performance values compared to the proprietary VEROSIM MOD file format is given in Table I. The results are about factor two slower for loading data, which is acceptable for a first prototype. However, the overhead for writing data is especially larger for the more complex satellite model. This can mainly be explained by the structures described in Subsection IV-B that have to be build up in the relational database. This process is far more complex than linearly writing out model data to the (highly optimized) MOD file format. Yet again, the optimization of the prototype might improve these results.

As mentioned in Subsection IV-A4, 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.

TABLE I. LOADING AND SAVING TIMES OF THE ORM-BASED PROTOTYPE COMPARED TO PROPRIETARY VEROSIM MOD FILE FORMAT.

| | ORM | | File | |
|---|---|---|---|---|
| | Robot | Satellite | Robot | Satellite |
| Loading | 4.7$s$ | 7.1$s$ | 2.5$s$ | 3.5$s$ |
| Saving | 0.4$s$ | 22$s$ | 0.5$s$ | 1.0$s$ |

## V. GRAPH DATABASE APPROACH

Currently, relational databases are dominating the market. Due to different problems with scalability and effective processing of big data with relational databases the field of NoSQL ("Not only SQL") databases has emerged [79]. In this context, the approach of graph databases (GDBs) has become popular. GDBs save their data in the nodes and edges of a mathematical graph, in particular, to manage highly linked information. As such, they are ideally suited for 3D simulation models. As mentioned above, like in CAD, such 3D data usually comprises a huge number of parts of many different types (mostly, each with only few instances), structured hierarchically with interdependencies. This recommends a graph-like data structure. For the same reason, the scene graph is a common approach to manage 3D data at runtime.

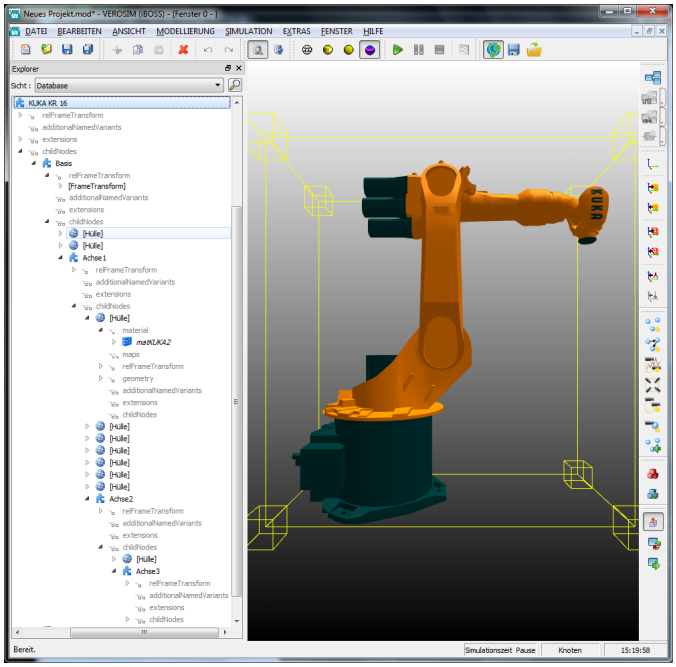


Figure 17. Robot model (from Figure 24) loaded from Neo4j into the in-memory simulation database VSD.

In this section, we present an approach for a synchronization interface between a GDB and a 3D simulation database, i.e., the runtime database of a 3D simulation system. The applied data mapping strategy is bidirectional and in part incremental. The approach was developed in the context of a student project and is based on our previous work. Its feasibility is shown with a prototypical implementation using

the GDB Neo4j and the 3D simulation system VEROSIM and its VSD database (Figure 17).

### A. State of the Art

In this section, the necessary basics for our work are presented.

*1) Graph Database Basics:* The idea of a GDB relies on the mathematical graph theory. Information is saved in the nodes (or vertices) and edges (or relationships) of a graph as shown in Figure 18. A graph is a tuple $G = (V, E)$, where $V$ describes the set of nodes and $E$ the set of edges, i.e., $v_i \in V$ and $e_{i,j} = (v_i, v_j) \in E$ [80]. To specify records, properties of nodes and (depending on the GDB) even relationships can be described by key-value pairs [63]. An important aspect of GDBs is the fact that all relationships are directly stored with the nodes so that there is no need to infer them as in relational databases using foreign keys and joins. Hence, read operations on highly connected data can be performed very fast. During a read access, the graph is traversed along paths so that the individual data records (nodes and edges) can be read in situ and do not have to be searched globally. Therefore, the execution time depends only on the traversal's depth [79].

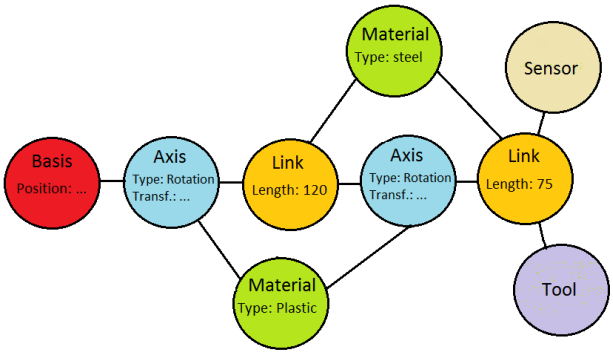



Figure 18. Graph database of a robot model.

GDBs also provide standard database features like security, recovery from hard- or software failures, concurrency control for parallel access, or methods for data integrity and reliability.

In contrast to flat files, using a (graph) database, data can be modified with a query language. Such languages are a powerful tool to manipulate the database content so that the data is not only stored persistently and securely but can also be handled simply.

*2) Neo4j:* Neo4j is a GDB implemented in Java. It can be run in server or embedded mode. Figure 19 shows its data model. Central elements are nodes and relationships containing the stored records. These records are described by an arbitrary number of properties (key-value pairs). Neo4j offers the concept of *labels* and *types* to divide the graph in logical substructures. A node is extendible with several labels characterizing the node's classification. Similarly, a relationship is identified by a type (exactly one). Besides the classification of the data, this also improves reading performance as just a part of the graph must be traversed to find the desired record [63] [64]. Apart from that, Neo4j is schemaless, i.e., it does not require any metadata definition before inserting actual user data.

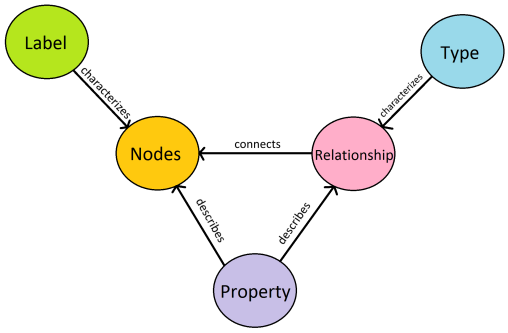



Figure 19. Neo4j data model.

All Neo4j accesses are processed in ACID (Atomicity, Consistency, Isolation, Durability) compliant transactions guaranteeing the reliability, consistency and durability of the database content [79]. Accesses are either performed with Neo4j's own query language called Cypher or using its Java API.

*3) Other Graph Databases:* Besides Neo4j, there are many other GDBs in the market. They differ in their conceptual structure and application area.

DEX is a GDB based on the labeled and directed attributed multigraph model. All nodes and edges are classified (labeled), edges are directed, nodes can be extended with properties (attributes), and edges can be connected with more than two nodes (multigraph) [81]. The graph is represented by bitmaps and other secondary structures. DEX has been designed for high performance and scalable graph scenarios. The good performance is achieved by the bitmap-based structure and the indexing of all attributes, which are efficiently processed by the C++ kernel [82].

Trinity [83] [84] is a memory-based graph store with many database features like concurrency or ACID-conform transactions. The graph storage is distributed among multiple well connected machines in a globally addressable memory address space yielding big data support. A unified declarative language provides data manipulation and message passing between the different machines. The great advantage of Trinity is the fast access to large data records. It is based on a multigraph model, which can exceed one billion nodes. Since there is no strict database schema, Trinity can flexibly be adapted to many data sets.

HypergraphDB stores its data in a directed multigraph, whose implementation is based on BerkeleyDB. All graph elements are called atoms. Every atom is characterized by its atom arity indicating the number of linked atoms. The arity determines an atom's type: An arity larger than zero yields an edge atom, or else, a node atom. Each atom has a typed value containing the user data [85].

InfoGrid is a framework specialized in the development of REpresentational State Transfer (REST)-full web applications. One part of this framework is a proprietary GDB used for data management. The graph's nodes are called MeshObjects, which are classified by one or more so-called EntityTypes, properties, and their linked relationships. MeshObjects not only contain the user data but also manage events relevant to the node [86].

Infinite Graph is a GDB based on an object-oriented concept. All nodes and edges are derived from two basic Java classes. Thus, the database schema is represented by user-defined classes. Besides data management, Infinite Graph provides a visualization tool [87]. Since the database can be distributed on multiple machines working in parallel, Infinite Graph can achieve a high data throughput. To manage concurrency, a lock server handles the different lock requests [82].

AllegroGraph [88] provides a REST protocol architecture. With this interface, the user has full control of the database including indexing, query and session management. All transactions satisfy ACID conditions.

Despite this wide range of GDBs, for the following reasons, we decide to use Neo4j in our approach:

- In many tests it proves to process data fast and efficiently,
- it can handle more than one billion nodes – even enough for extremely large 3D simulation models – which could be useful in coming stages of extension,
- Neo4j is a full native GDB so that traversal and other graph operations can be performed efficiently,
- Neo4j provides a comprehensive and powerful query language (e.g., for efficient partial loading strategies in future versions of the presented prototype),
- directed edges allow to model object interdependencies more accurately, however, without disadvantages in traversal performance,
- properties on relationships allow for a more flexible modeling (e.g., to distinguish between shared and composite aggregation relationships),
- finally, Neo4j is currently the most prevalent GDB in the market indicating it to be especially well explored and developed. Hence, it provides the best prospects of success.

Note that while we choose Neo4j for the reasons given above, the presented concepts are mostly independent of the choice of the particular GDB.

### B. Concept

In this section, we describe the fundamental concept and the required features of our synchronization component's prototype. Its implementation using Neo4j and VSD is described in Subsection V-C.

*1) Structure Mapping:* An essential question when synchronizing two databases is: How do we map the different data structures? Depending on the database paradigm, entities with attributes and relationships (connecting two or more entities) are represented differently. For example, a relational database uses relations, attributes and foreign keys while a GDB uses nodes, relationships and properties.

1) **Schema Mapping:** Before synchronizing user data, a generic schema mapping is performed mapping the metadata of one database to the other as described in [57]. This is performed once on system startup. For example, when performed between a relational and an object-oriented database, each table of the former might be mapped to a corresponding class of the latter (columns and class attributes accordingly).
2) **Schemaless Approach:** When a schemaless database is involved, a different approach has to be applied. Here, metadata from a non-schemaless database must be mapped onto the user data of the schemaless one. For example, class names from an object-oriented database are mapped onto node labels of a schemaless GDB.

For the schemaless Neo4j, in our prototype, we chose the second approach.

*2) Object Mapper:* Another key aspect of the concept is the object mapper. It maps objects from one database to an equivalent counterpart in the other database. For example, an object from an object-oriented database is mapped to a corresponding node in the GDB. The mapping is based on the counterparts' identities and includes a transfer of all property (or attribute) values in between. Based on these mappings, individual object or property changes can be tracked and resynchronized. Summarized the mapping is bidirectional and in part incremental.

*3) Transactions:* Any changes (insert, update, delete) to the data are tracked and stored in transactions, which can be processed independently. By executing these transactions, data is (re)synchronized on object level. During the accumulation (and before the execution) of such transactions, the operations stored within can be filtered for redundancies. For example, a transaction for creating a new object followed by a transaction for deleting the very same object can both be discarded.

### C. Prototype

This section gives an insight into the prototypical implementation of the interface between VSD and Neo4j. The prototype should have the ability to save simulation data from VSD in Neo4j and to load it back into VSD. Initially, when storing a simulation model in Neo4j, VSD's contents are archived once. Subsequently, changes in VSD are tracked and updated to Neo4j individually. That is, when a VSD instance has been changed just the changes are transferred as mentioned above. In the current version of the interface, only changes within VSD are tracked for resynchronization. Thus, Neo4j serves as database back end, which can store simulation models persistently.

The prototype is realized in a C++ based VEROSIM plugin, which uses Neo4j's Java API in embedded mode in order to communicate with Neo4j.

*1) Data Mapping:* In the context of this work, synchronization represents data transfer from one database to another. However, the structure of one database's data elements often differs from those of another. Thus, it becomes necessary to map these different structures on each other. Figure 20 shows our intuitive approach.

Single VSD instances are mapped to single Neo4j nodes and references (Ref/List-Properties) from one instance to another are represented by relationships between the corresponding nodes. The relationship is orientated to the referenced node's direction. Furthermore, we transfer the Val-Properties of a VSD instance to Neo4j node properties.

As mentioned above, a basic difference between VSD and Neo4j is that the former comprises metadata describing (and prescribing) a schema while the latter is schemaless. VSD metadata contains important information for the simulation and

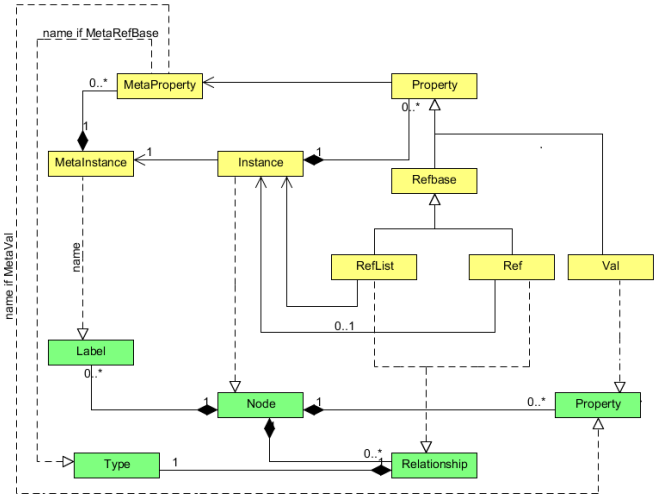



Figure 20. Data mapping of the synchronization component (top: VSD data model, bottom: Neo4j data model).

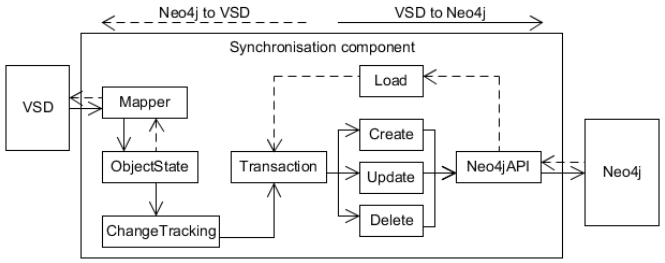



Figure 21. Synchronization component.

| VSD-Id | Neo4j-Id | State | Pending-Transactions |
|---|---|---|---|
| ... | | | |
| 6049 | 0 | SYNCED | - |
| 6050 | 1 | PENDING_UPDATE | Update-Transaction1, Update-Transaction2 |
| 6051 | 2 | PENDING_DELETE | Delete-Transaction |
| ... | | | |

Figure 22. Exemplary list of object states of the mapper.

is indispensable for a correct data mapping. Thus, it is essential to transfer this informations as well. We store VSD metadata on Neo4j's object level:

1) A VSD instance's class name is mapped to its Neo4j node's label,
2) a VSD Ref/List-Property's name is mapped to its Neo4j relationship's type, and
3) a VSD Val-Property's name is mapped to its Neo4j property's key.

Val-Property values are handled depending on their data type. If the type corresponds to one of Neo4j's supported basic types (e.g., integer, float, string, boolean, etc.) the value will be transfered directly. More complex data structures (e.g., mathematical vectors, etc.) are serialized to a binary representation and transfered as such.

To store additional meta information about VSD Ref/List-Properties, we take advantage of Neo4j's feature to add properties to relationships. Currently, every relationship gets a boolean property with the key *autodelete* as introduced above. Additionally, a RefList-Property entry's order is stored as an index in a relationship property.

*2) Synchronization Component:* Figure 21 depicts the structure of the synchronization component (based on [89]). Its core is the (object) mapper managing mappings between pairs of VSD instances and Neo4j nodes. Each mapping is stored in form of a so-called ObjectState (OS) holding all relevant information. The OS contains both objects' ids, all collected (but not executed) transactions and the state of the relation between the two. This state indicates whether the pair is synchronous, i.e., equal, or whether one of them has been changed and differs from its counterpart. An exemplary list of object states of the mapper is given in Figure 22.

Each change to a VSD instance is encapsulated in a transaction stored in the appropriate OS. Subsequently, they can be executed. Depending on the change's type, a create, update, or delete transaction is generated. Furthermore, a separate load transaction is used to load Neo4j contents into VSD. Each transaction comprises all type specific information necessary for its execution. For example, a create transaction contains the names and values of all Val-Properties, the class name, and information on Ref/List-Properties like target ids, reference names and *autodelete* values.

The last part of the synchronization component is the Neo4jAPI, which interacts with Neo4j's Java API.

*VSD to Neo4j:* VSD is an active database. One aspect of this activity is that changes to its instances are notified to registered components like the synchronization component presented in this work. Notifications include all relevant information about the modification like the instance's id or the changed property. The synchronization component encapsulates this information in an appropriate transaction. Using the instance id, the mapper is able to identify the corresponding OS and retrieve the mapping's state. A change tracking mechanism is used to filter redundant transactions as mentioned above (more details are given in Section V-C3).

When the user or some automatic mechanism (e.g., a timer) triggers a resynchronization, all collected transactions are executed modifying Neo4j's contents accordingly.

*Neo4j to VSD:* When loading a Neo4j database's contents to VSD, the Neo4jAPI traverses the graph and generates a load transaction for each visited node. All data is read from Neo4j before entries are stored in the mapper. Load transactions contain the respective node's id, all its property keys and values and the ids of adjacent nodes of outgoing relationships and their respective properties (*autodelete* and index for RefList-Properties). Subsequently, the synchronization component executes all load transactions. For each, a new VSD instance with appropriate properties is created and its id is stored in an OS with the corresponding node's id.

*3) Change Tracking:* As mentioned above, when collecting transactions, newly created ones may cancel out older ones. A change tracking mechanism performs the necessary filtering of such redundant transactions.

Change tracking is based on the current state of the considered OS. Depending on the incoming transaction's type, the state changes and the list of collected transactions is updated.

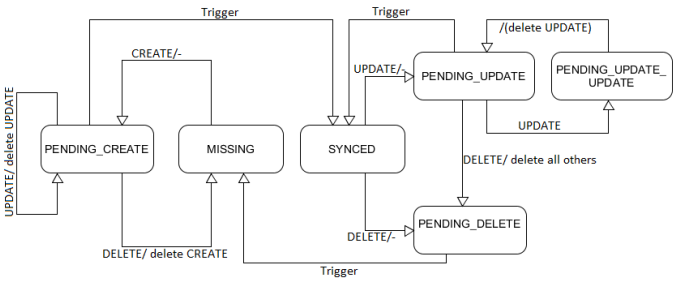



Figure 23. State machine of the change tracking mechanism.

Change tracking is modeled as a state machine as depicted in Figure 23. Here, the input (triggering state transitions) is represented by the incoming transaction type and the output (emitted during state transitions) describes the transaction list's modification. The initial state of any OS for a newly created VSD instance is the *MISSING* state as there is no corresponding Neo4j node. This intermediate state is left as soon as the corresponding create transaction is generated and the state changes to *PENDING_CREATE*. If this VSD instance is deleted before the transaction of type create has been executed, both transactions (create and delete) are removed and the whole OS is deleted. Else, upon a resynchronization trigger, a corresponding Neo4j node is generated, the state changes to *SYNCED*, and all executed transactions are removed from the list. The *SYNCED* state means that a Neo4j node and its VSD instance counterpart are in sync. It is reached every time a resynchronization was performed and is left when the VSD instance is modified (*PENDING_UPDATE*) or deleted (*PENDING_DELETE*).

In *PENDING_UPDATE* state, the changed property of an additional update transaction is compared to existing update transactions to avoid multiple updates of the same property. If two transactions modify the same property only one of them needs to be stored. This is represented by the intermediate *PENDING_UPDATE_UPDATE* state.

### D. Evaluation

Finally, the interface's effectiveness and performance have been evaluated using the same two simulation models of an industrial robot and a satellite as used in the OR mapping approach. As above, given the current functional range of the presented prototype, further tests do not appear to provide more insights. Initially, both models are stored in a Neo4j database and, subsequently, loaded back into an (empty) VSD. The robot model yields 170 Neo4j nodes and 209 relationships. The more complex satellite about 20,000 nodes and 25,000 relationships. The highly connected nature of the 3D simulation data is apparent making a GDB ideally suited for its storage.

Figures 24 and 17 give an impression of the interface's effectiveness. Figure 24 shows an excerpt of the robot model data within Neo4j. Figure 17 shows the same data loaded into the VSD in-memory simulation database. The data mapping operates generically, i.e., independent from the actual data, making the whole synchronization component very flexible. The interface can synchronize arbitrary VSD contents to a Neo4j database.

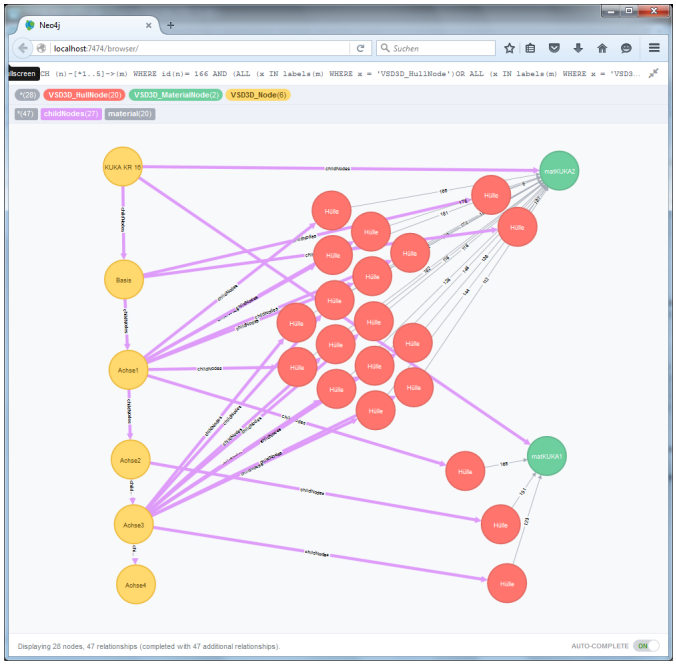



Figure 24. 3D simulation model data (excerpt) of an industrial robot stored within Neo4j.

TABLE II. LOADING AND SAVING TIMES OF THE NEO4J-BASED PROTOTYPE.

| | Neo4j | | File | |
|---|---|---|---|---|
| | Robot | Satellite | Robot | Satellite |
| Loading | 0.14 | 3.99 | 0.1 | 2.8 |
| Saving | 2.63 | 10.53 | 1.8 | 9.9 |

In Subsection V-C, we present the interface's functionality to selectively resynchronize changes to VSD instances. This feature has been tested by changing some VSD properties (e.g., name or position of a component). In the Neo4j browser, we verified that these modifications were transferred correctly. Inversely, changes to node properties from the Neo4j browser show up in VEROSIM when the model is reloaded. This also shows the advantage of selectively modifying data within a database in contrast to a file-based approach.

Another important aspect of the evaluation is the interface's performance. Here, the initial storage of a simulation model into Neo4j and the loading of a whole simulation model from Neo4j were examined and compared to saving and loading models to and from the native VEROSIM file format. Results are given in Table II. The access operations to the GDB are only somewhat slower than the native file operations. For a prototypical implementation from a student project, these results are very promising. First of all, compared to the highly optimized code for reading and writing the native file format, the current prototype is only optimized to a certain degree. Furthermore, the more high-level database access operations will always remain a little more complex than simple, sequential file reading or writing. Yet, the additional benefit from a full-fledged database (providing security, multi-user support, etc.) more than compensates for this small drawback.

Altogether, this shows that a GDB like Neo4j is well suited

for highly connected 3D simulation model data and can be handled fast.

## VI. Comparison

First of all, compared to our previous solution using the generic OR mapper SGJ, the presented two prototypes avoid the usage of an additional layer between the synchronization component and ExtDB, i.e., PostgreSQL and Neo4j. Instead, they realize a direct database access. Besides fewer transformations (presumably, leading to better performance) this also allows for a more flexible adaptation to the needs of eRobotics applications and Semantic World Models.

From a performance point of view – as far as this is comparable for two prototypes both developed in the context of student projects – the results for the Neo4j-based prototype are better than those of the ORM-based approach. In fact, this was our expected result as the graph paradigm obviously fits better to the structure of Semantic World Models. The complex mapping rules needed to properly represent inheritance, relationships and polymorphism to overcome the object-relational impedance mismatch take their toll, especially when writing data. However, further optimization of these first prototypes might change these differences.

Regarding the implementation of schemata, in both cases, special structures within the relational or graph database are necessary that have to be interpreted by the respective synchronization component. Other applications accessing these databases must have knowledge about this "encoding" – especially when writing data. The schemaless Neo4j obviously has drawbacks as opposed to the ORM-based approach. In the latter, many object-oriented schema structures from VSD can be mapped to their relational counterparts (e.g., attributes). This allows to secure more structural requirements on the side of ExtDB rather than the synchronization interface.

Another important aspect is the dissemination of the two systems. While the graph database paradigm might be more suited for storing Semantic World Model data, RDBMSs are far more pervasive. Thus, it is easier to find stable DBMSs, tool or query language (SQL vs. Cypher) support, or even specialized developers and administrators. Finally, customer acceptance is also a key decision criterion. Many users of eRobotics applications in business and industry already use a DBMS in their enterprise. Mostly, these are well-established relational solutions like Oracle, Microsoft SQL Server, MySQL or PostgreSQL – and mostly they want to keep using these solutions for different reasons (existing infrastructure, existing maintenance contracts, existing personnel etc.).

Finally, by being based on the general database synchronization concept presented in Section III, both approaches fulfill most of the five requirement for data management for eRobotics applications:

1) both map VSD's object-oriented data to either PostgreSQL or Neo4j,
2) both realize a distributed architecture by synchronizing the local VSD to a central PostgreSQL or Neo4j,
3) both integrate the in-memory database VSD as a cache for the shared Semantic World Model in PostgreSQL or Neo4j,
4) both flexibly adapt to different schemata by mapping them appropriately to PostgreSQL or Neo4j structures,
5) and both DBMS allow for a usage in simulation as well as live operation scenarios. However, regarding temporal data management, in both cases, further research needs to be conducted for a realization using PostgreSQL or Neo4j.

TABLE III. Comparison chart of the two presented approaches.

| | **ORM** | **Neo4j** |
|---|---|---|
| Additional transformations | no | no |
| Impedance mismatch | yes | no |
| Native schema support | yes | no |
| Encoding of OO schema | complex | straightforward |
| Dissemination of DBMS | high | low |
| Fulfillment of 5 main requirements | all | all |
| Prototype performance | limited | good |

Table III gives an overview.

Thus, while both approaches are applicable for eRobotics applications, each has its advantages and drawbacks.

## VII. Conclusion and Future Work

In this paper, we give an introduction to eRobotics applications and define their requirements for data management. We present the various benefits from using database technology to manage the underlying Semantic World Models instead of using flat file formats. Furthermore, we show how existing approaches for database integration into applications with a 3D context do not provide a sufficiently comprehensive and flexible solution to fulfill all these requirements. This motivated the development of our new database synchronization concept for eRobotics applications. Its basic idea is to use local in-memory simulation databases (SimDB) and to synchronize them to a central, external back-end database (ExtDB). This concept has previously been realized using a generic OR mapper with the drawback of an additional translation layer. Thus, in this paper and based on this concept, two new direct approaches are presented omitting this additional layer. The first is an integrated OR mapping approach, directly mapping the schema and data from SimDB to a central relational ExtDB without an intermediate representation. The second uses a graph database for ExtDB, benefiting from the graph-like structure of Semantic World Models. For both cases, the state of the art (theory and existing solutions) is analyzed, based on which the approaches are developed. The respective evaluations and the final comparison show the basic suitability and practical feasibility of both prototypes for the management of eRobotics applications' Semantic World Models. While the graph database approach is advantageous in terms of performance the OR mapping approach is more suited to integrate into existing corporate infrastructures. Thus, the decision for a database paradigm for ExtDB depends on the concrete application scenario.

In future, both prototypes might receive more development and optimization. For the direct OR mapper, 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. For the Neo4j-based approach, also further performance optimizations and evaluations beyond the results from the student project could be performed. For instance, better traversal algorithms might improve loading speed. Another idea is to use Neo4j's batch inserter in contrast to the transactional structure to reduce resynchronization time. Furthermore, Neo4j might be used as a central database in a distributed simulation scenario with several VEROSIMs and VSDs. Here, an equivalent notification mechanism is needed for Neo4j to be able to track modifications in the central database. Finally, apart from the two presented approaches, further database paradigms like in-memory databases (e.g., SAP HANA, H2 or Redis) could be examined as candidates for ExtDB.

## REFERENCES

[1] A.-M. Stapelbroek, M. Hoppen, and J. Rossmann, "Object-Relational Mapping in 3D Simulation," in DBKDA 2016, The 8th International Conference on Advances in Databases, Knowledge, and Data Applications, Lisbon, Portugal, 2016.

[2] M. Hoppen, J. Rossmann, and S. Hiester, "Managing 3D Simulation Models with the Graph Database Neo4j," in GraphSM 2016, The 3rd International Workshop on Large-scale Graph Storage and Management, Lisbon, Portugal, 2016.

[3] J. Rossmann, M. Schluse, M. Rast, and L. Atorf, "eRobotics Combining Electronic Media and Simulation Technology to Develop (Not Only) Robotics Applications," in E-Systems for the 21st Century: Concept, Developments, and Applications, S. Kadry and A. El Hami, Eds. Apple Academic Press, 2015.

[4] Verein Deutscher Ingenieure (VDI), "VDI 3633 - Simulation of systems in materials handling, logistics and production - Fundamentals," Düsseldorf, 2013, URL: https://www.vdi.de/richtlinie/entwurf_vdi_3633-simulation_von_logistik_materialfluss_und_produktionssystemen_begriffe/ [retrieved: 2017.05.17].

[5] R. Elmasri and S. B. Navathe, Database Systems: Models, Languages, Design, And Application Programming, 6th ed. Prentice Hall International, 2010.

[6] buildingSMART Internatioinal, "Industry Foundation Classes (IFC)," URL: http://www.buildingsmart-tech.org/specifications/ifc-releases/summary [retrieved: 2017.05.17].

[7] A. e.V., "AutomationML," URL: https://www.automationml.org [retrieved: 2017.05.17].

[8] SysML Open Source Specification Project, "SysML Specification," URL: http://www.sysml.org/sysml-specifications/ [retrieved: 2017.05.17].

[9] S. Julier, Y. Baillot, M. Lanzagorta, D. Brown, and L. Rosenblum, "Bars: Battlefield augmented reality system," in NATO Symposium on Information Processing Techniques for Military Systems, 2000, pp. 9–11.

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

[11] 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.

[12] J. Haist, R. Schnuck, and T. Reitz, "Usage of persistence framework technologies for 3D geodata servers," in AGILE 2006, 9th AGILE Conference on Geographical Information Science. Shaping the future of Geographic Information Science in Europe, 2006, pp. 43 – 50.

[13] Y. Masunaga and C. Watanabe, "Design and implementation of a multimodal user interface of the Virtual World Database system (VWDB)," in Proceedings Seventh International Conference on Database Systems for Advanced Applications. DASFAA 2001. IEEE Comput. Soc, 2001, pp. 294–301.

[14] Y. Masunaga, C. Watanabe, A. Osugi, and K. Satoh, "A New Database Technology for Cyberspace Applications," in Nontraditional Database Systems, Y. Kambayashi, M. Kitsuregawa, A. Makinouchi, S. Uemura, K. Tanaka, and Y. Masunaga, Eds. London: Taylor & Francis, 2002, ch. 1, pp. 1–14.

[15] C. Watanabe and Y. Masunaga, "VWDB2: A Network Virtual Reality System with a Database Function for a Shared Work Environment," in Information Systems and Databases, K. Tanaka, Ed., Tokyo, Japan, 2002, pp. 190–196.

[16] K. Kaku, H. Minami, T. Tomii, and H. Nasu, "Proposal of Virtual Space Browser Enables Retrieval and Action with Semantics which is Shared by Multi Users," in 21st International Conference on Data Engineering Workshops (ICDEW'05). IEEE, apr 2005, pp. 1259–1259.

[17] 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.

[18] E. V. Schweber, "SQL3D - Escape from VRML Island," 1998, URL: http://www.infomaniacs.com/SQL3D/SQL3D-Escape-From-VRML-Island.htm [retrieved: 2017.05.17].

[19] A. Vakaloudis and B. Theodoulidis, "Spatiotemporal Database Connection to VRML," in Proceedings of the 9th UK Electronic Imaging & the Visual Arts Conference, EVA '98, Cambridge, 1998.

[20] K. Walczak, "Dynamic Database Modeling of 3D Multimedia Content," in Interactive 3D Multimedia Content, W. Cellary and K. Walczak, Eds. London: Springer London, 2012, ch. 4, pp. 55–102.

[21] K. Walczak and W. Cellary, "Building database applications of virtual reality with X-VRML," in Proceeding of the seventh international conference on 3D Web technology - Web3D '02. New York, New York, USA: ACM Press, feb 2002, pp. 111–120.

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

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

[24] T. Manoharan, H. Taylor, and P. Gardiner, "A collaborative analysis tool for visualisation and interaction with spatial data," in Proceedings of the seventh international conference on 3D Web technology. ACM, 2002, pp. 75–83.

[25] 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.

[26] 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.

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

[28] C. Borgatti, M. Felicori, M. Mauri, L. Calori, A. Guidazzoli, S. Pescarin, T. Diamanti, M. Liguori, and L. Valentini, "Databases and virtual environments: a good match for communicating complex cultural sites," in ACM SIGGRAPH 2004 Educators program. ACM, 2004, p. 30.

[29] H. Richards-Rissetto, F. Remondino, G. Agugiaro, J. Von Schwerin, J. Robertsson, and G. Girardi, "Kinect and 3D GIS in archaeology," in Proceedings of the 2012 18th International Conference on Virtual Systems and Multimedia, VSMM 2012: Virtual Systems in the Information Society, 2012, pp. 331–337.

[30] M. Forte and G. Kurillo, "Cyberarchaeology: Experimenting with

teleimmersive archaeology," in 2010 16th International Conference on Virtual Systems and Multimedia, VSMM 2010, 2010, pp. 155–162.

[31] A. Guarnieri, F. Pirotti, and A. Vettore, "Cultural heritage interactive 3D models on the web: An approach using open source and free software," Journal of Cultural Heritage, vol. 11, no. 3, 2010, pp. 350–353.

[32] M. F. Shiratuddin and W. Thabet, "Utilizing a 3D game engine to develop a virtual design review system," Journal of Information Technology in Construction (ITcon), vol. 16, no. Special Issue Use of Gaming Technology in Architecture, Engineering and Construction, 2011, pp. 39–68.

[33] S. Wang, Z. Mao, C. Zeng, H. Gong, S. Li, and B. Chen, "A new method of virtual reality based on Unity3D," in 2010 18th International Conference on Geoinformatics, 2010, pp. 1–5.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

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

[41] J. Beetz, L. van Berlo, R. de Laat, and P. van den Helm, "bimserver.org - An Open Source IFC Model Server," in Proceedings of the CIB W78 2010: 27th International Conference, no. Weise 2006, Cairo, Egypt, 2010, pp. 16–18.

[42] H.-s. Kang and G. Lee, "Development of an object-relational IFC server," in ICCEM/ICCPM, 2009.

[43] S. Malaikrisanachalee and H. Vathananukij, "Integration of java-based BIM with spatial database," International Journal of Civil Engineering, vol. 9, no. 1, 2011, pp. 17–22.

[44] V. Tarandi, "The BIM Collaboration Hub: A model server based on IFC and PLCS for Virtual Enterprise Collaboration," in Proceedings of the CIB W78-W102 2011: International Conference, Sophia Antipolis, France, 2011, pp. 26–28.

[45] 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.

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

[47] 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.

[48] 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.

[49] M. Mahdjoub, D. Monticolo, S. Gomes, and J. Sagot, "A collaborative design for usability approach supported by virtual reality and a multi-agent system embedded in a PLM environment," Computer-Aided Design, vol. 42, no. 5, 2010, pp. 402–413.

[50] M. Roberts, N. Ducheneaut, and T. F. Smith, "The "3d Wiki": Blending virtual worlds and Web architecture for remote collaboration," in 2010 IEEE International Conference on Multimedia and Expo, ICME 2010, 2010, pp. 1166–1171.

[51] 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.

[52] 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.

[53] C. Shahabi, F. Banaei-Kashani, A. Khoshgozaran, L. Nocera, and S. X. S. Xing, "GeoDec: A Framework to Visualize and Query Geospatial Data for Decision-Making," IEEE Multimedia, vol. 17, no. 3, 2010, pp. 14–23.

[54] M. Lobur, O. Matviykiv, A. Kernytskyy, and R. Dobosz, "Presentation of the heat simulation model with use of relational data model," 2011 11th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics CADSM, 2011, pp. 228–229.

[55] A. Stadler, C. Nagel, G. König, and T. H. Kolbe, "Making interoperability persistent : A 3D geo database based on CityGML," 3D Geo-Information Sciences, 2009, pp. 175–192.

[56] M. Hoppen and J. Rossmann, "A novel distributed database synchronization approach with an application to 3d simulation," IARIA International Journal on Advances in Software, vol. 7, no. 3 and 4, December 2014, pp. 601–616.

[57] 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.

[58] SEDRIS Associates & Partners, "SEDRIS," 2013, URL: http://www.sedris.org [retrieved: 2017.05.17].

[59] 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, J. F. M. Van Impe and F. Logist, Eds., Brussels, Belgium, 2013, pp. 39–46.

[60] CPA Geo-Information, "CPA SupportGIS Java (SGJ)," URL: http://www.supportgis.de [retrieved: 2017.05.17].

[61] The PostgreSQL Global Development Group, "PostgreSQL: About," 2015, URL: http://www.postgresql.org/about/ [retrieved: 2017.05.17].

[62] Open Geospatial Consortium (OGC), "Geography Markup Language (GML)," URL: http://www.opengeospatial.org/standards/gml [retrieved: 2017.05.17].

[63] M. Hunger, Neo4j 2.0 A graph database for everyone (orig.: Neo4j 2.0 Eine Graphdatenbank für alle), 1st ed. entwickler.press, 2014.

[64] Neo4j Team, "The Neo4j Manual v2.2.5," 2015, URL: http://neo4j.com/docs/stable/ [retrieved: 2017.05.17].

[65] 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.

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

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

[68] A. Schatten, "O/R Mappers and Alternatives (orig.: O/R Mapper und Alternativen)," 2008, URL: http://www.heise.de/developer/artikel/O-R-Mapper-und-Alternativen-227060.html [retrieved: 2017.05.17].

[69] T. Neward, "The Vietnam of Computer Science," 2006, URL: http://www.odbms.org/2006/01/the-vietnam-of-computer-science/ [retrieved: 2017.05.17].

[70] S. W. Ambler, "Mapping Objects to Relational Databases: O/R Mapping In Detail," 2013, URL: http://www.agiledata.org/essays/mappingObjects.html [retrieved: 2017.05.17].

[71] 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.

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

[73] ——, “Practical Object Oriented Models in Sql,” 2009, URL: http://de.slideshare.net/billkarwin/practical-object-oriented-models-in-sql [retrieved: 2017.05.17].

[74] Hibernate, HIBERNATE–Relational Persistence for Idiomatic Java, 2015, URL: http://docs.jboss.org/hibernate/orm/5.0/manual/en-US/html/index.html [retrieved: 2017.05.17].

[75] NHibernate Community, NHibernate–Relational Persistence for Idiomatic .NET, 2015, URL: http://nhibernate.info/doc/nhibernate-reference/index.html [retrieved: 2017.05.17].

[76] Code Synthesis Tools CC, ODB: C++ Object-Relational Mapping (ORM), 2015, URL: http://www.codesynthesis.com/products/odb/ [retrieved: 2017.05.17].

[77] 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: 2017.05.17].

[78] The Qt Company, “Qt Documentation,” 2016, URL: http://doc.qt.io/qt-5/index.html [retrieved: 2017.05.17].

[79] I. Robinson, J. Webber, and E. Eifrem, Graph Databases-New Opportunities For Connected Data, 2nd ed. O’Reilly, 2015.

[80] R. Diestel, Graph Theory, 2nd ed. Springer, 2000.

[81] N. Martinez-Bazan, S. Gomez-Villamor, and F. Escale-Claveras, “Dex: A high-performance graph database management system,” in Data Engineering Workshops (ICDEW), 2011 IEEE 27th International Conference on, April 2011, pp. 124–127.

[82] R. Kumar Kaliyar, “Graph databases: A survey,” in Computing, Communication Automation (ICCCA), 2015 International Conference on, May 2015, pp. 785–790.

[83] Microsoft, “Graph engine 1.0 preview released,” 2016, URL: http://research.microsoft.com/en-us/projects/trinity/ [retrieved: 2017.05.17].

[84] B. Shao, H. Wang, and Y. Li, “Trinity: A distributed graph engine on a memory cloud,” in Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM, 2013, pp. 505–516.

[85] B. Iordanov, “HyperGraphDB: A Generalized Graph Database,” in Web-Age information management. Springer, 2010, pp. 25–36.

[86] InfoGrid Team, “Infogrid: The web graph database,” 2016, URL: http://infogrid.org/trac/ [retrieved: 2017.05.17].

[87] J. Peilee, “A survey on graph databases,” 2011, URL: https://jasperpeilee.wordpress.com/2011/11/25/a-survey-on-graph-databases/ [retrieved: 2017.05.17].

[88] “Allegrograph,” 2016, URL: http://franz.com/agraph/allegrograph/ [retrieved: 2017.05.17].

[89] 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.