Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, eds.

DATABASE-DRIVEN DISTRIBUTED 3D SIMULATION

Martin Hoppen
Michael Schluse
Jürgen Roßmann

Institute for Man-Machine Interaction
RWTH Aachen University
Ahornstrasse 55
D-52074 Aachen, GERMANY

Björn Weitzig

CPA-Systems GmbH
Auf dem Seidenberg 3a
D-53721 Siegburg, GERMANY

ABSTRACT

Distributed 3D simulations are used in various fields of application like geo information systems (GIS), space robotics or industrial automation. We present a new database-driven approach that combines 3D real-time simulation techniques with object-oriented data management. It consists of simulation clients that replicate from a central database object data as well as the data schema itself. The central database stores static and dynamic parts of a simulation model, distributes changes caused by the simulation, and logs the simulation run. Compared to standard decentralized methods this approach has several advantages like persistence for state and course of time, object identification, standardized interfaces for simulation, modeling and evaluation, as well as a consistent data schema and world model for the overall system, which at the same time serves as a means for communication.

1 INTRODUCTION

In this paper, we introduce a new approach for distributed 3D real-time simulation that uses a central object-oriented database for data management, communication and logging. The system can be used to flexibly build simulation applications in various fields of applications, like geo information systems (GIS) (Fig. 1), space robotics or industrial automation. It provides a central world model (CWM) for these applications as introduced in Freund, Müller, and Roßmann (1999), enhanced by a modern object-oriented approach using data and schema replication. First of all this means, the database is used for storing and managing all parts, static as well as dynamic, of the shared simulation model in a consistent data schema. Here the object-oriented modeling approach benefits the hierarchical 3D construction and environmental data typically used in 3D simulation applications. Secondly, the database also represents a central communication hub that drives the distributed simulation itself. Finally, a simulation run is implicitly logged by the database's versioning techniques allowing subsequent replay, analysis and archiving.

As described in Hoppen et al. (2011) we developed a flexible method allowing simulation clients to transparently access such an object-oriented database. These clients provide a local simulation database (think of a generalized scene graph) that acts as a real-time capable cache for the central database. Once during system startup, this likewise object-oriented database adopts the central database's schema, so both "speak the same language". During runtime, the simulation client can load (parts of) the model data by replicating it into its local simulation database, while local changes are tracked and synchronized back into the central database.

As mentioned above, not only are all static parts of a 3D simulation model (e.g., buildings and power poles in an urban model) stored in the database, but also all dynamic objects like simulated cars. A locally running simulation changes these dynamic objects (e.g. the car's position). These changes are likewise synchronized to the central database, hence communicating the new state of the simulation model

Figure 1: Example for an urban database-driven 3D simulation with a car and a helicopter (data: Düsseldorf).

to the CWM. The database's notification service actively notifies any subscribed simulation client about the changes in the shared world model. Each client adopts these changes by synchronizing their own local replicate copies accordingly.

As all intermediate states of the simulation are made persistent in the shared model, a simulation run can easily be captured by implementing a versioning mechanism on the database. The recorded time-stamped values not only represent a queryable 4D archive of the scenario. A simulation client can also be used in an off-line viewing mode to replay a simulation run step by step, allowing analysis and debriefing.

Many more advantages arise from using a central database for a distributed simulation system. Different applications (e.g. for authoring) can be employed using its standardized interfaces, an inherent rights management provides means for fine grained access control, and (spatial) queries allow very selective loading and changing of the simulation model.

Besides standardized interfaces, a standardized data schema like CityGML (2012) or SEDRIS Technologies (2012) fosters the accessibility of the shared simulation model. By adopting the database's schema, all participants of the distributed simulation system use this very same schema. To allow a simulation client to not only speak the same language but also to understand this language, there have to be certain mechanisms to translate parts essential to the functioning of the simulation (e.g. the dynamic modeling of a car) into the simulation system's own representation. This is a process we call functional data synchronization.

Thus, compared to the standard decentralized approach for distributed 3D simulation, our database-driven approach has several advantages, as it provides persistence for state and course of time, a simple solution to object identification ("id problem"), standardized interfaces, as well as a consistent data schema and world model, which at the same time serves as a means for communication. Figure 2 gives an overview of the overall architecture.

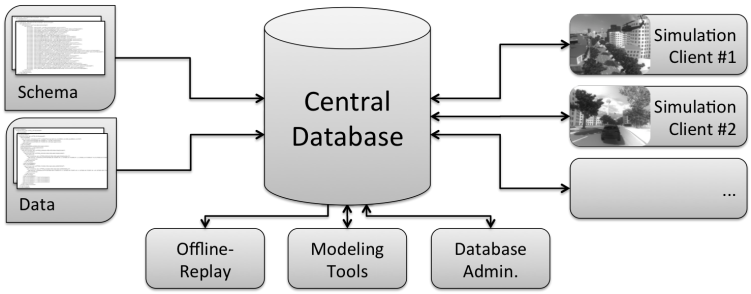

Figure 2: Architecture of the database-driven 3D simulation system.

The rest of this paper is organized as follows. In Section 2 work related to our own is evaluated. In Section 3 we introduce the 3D simulation system VEROSIM and its simulation database, which is

the basis of the simulation part of the developments presented in this paper. As the database part of our work, in Section 4 the utilized object-oriented database technology SupportGIS Java is presented. In Section 5 we explain the synchronization process in general and schema as well as data synchronization in detail. In Section 6 we give details on functional data synchronization while Section 7 describes how the simulation system accesses versioned data within the central database. The paper ends with some examples of applications in Section 8 as well as a perspective and conclusion in Section 9. Compared to our previous publication the main contributions of this paper are the database-driven distributed simulation, the concept of functional synchronization and the usage of database versioning for simulation replay.

2 RELATED WORK

Databases have been integrated into 3D systems in many different ways. But no approach provides our tight integration of real-time 3D simulation and object-oriented database technology including persistence, collaboration, arbitrary data schema, and versioning. In the simplest scenarios, the database is only used to store object positions etc. (Manoharan, Taylor, and Gardiner 2002). More sophisticated systems use the database to store the scene data itself, where some do support collaboration (Schweber, Erick Von 1998; Julier et al. 2000; Watanabe and Masunaga 2002; Manoharan et al. 2002; Kaku et al. 2005; Walczak 2012) while others do not (Vakaloudis, A. and Theodoulidis, B. 1998; Schmalstieg et al. 2007). A flexible support for different data schemata is not widespread among these systems (Watanabe and Masunaga 2002; Schmalstieg et al. 2007; Walczak 2012; Dassault Systèmes 2008; Autodesk 2005). Finally, versioning is only supported in CVS-like systems (Dassault Systèmes 2008; Autodesk 2005) on file level or using special schemata as in Vakaloudis and Theodoulidis (1998).

3 SIMULATION SYSTEM

One of the requisites of our database-driven architecture is a simulation client with a highly flexible data management. On the one hand it must provide the modeling capabilities of an object-oriented database. This is needed to adopt the central database's schema, our basis for a consistent data model throughout the entire distributed system. Furthermore, similar to the central database, the internal data management also needs to provide a notification mechanism, so that local changes can be detected and written back into the CWM. On the other hand, it must be efficient to support real-time 3D simulation with high-performance simulation and rendering techniques based on database data.

The 3D simulation system VEROSIM provides such a flexible yet efficient data management. The system's internal data management is an event driven and object-oriented graph database called VSD (VEROSIM Active Simulation Database), which describes the components as well as their behavior and provides methods for parallel and distributed computing and visualization. Nodes of this database not only provide data encapsulation but also mechanisms for interaction with the simulation environment implementing their behavior – this is what distinguishes this database from standard scene graphs.

All active components of the database are derived from an "Instance" class. Simulation data as well as the topology of the database is stored in so called properties. In this case "active" means that whenever a property is changed a message is sent to all registered listeners. Adding or removing instances from the database triggers a message as well. This implements the internal notification mechanism mentioned above.

To be able to adopt another database's schema, VSD's meta information has to be accessible. This is realized by a reflection mechanism providing so called meta-instances describing classes of instances (name, inheritance, etc.) and meta-properties describing their properties (name, type, multiplicity, etc.). This enables the simulation system to query meta-instances and meta-properties by name. Furthermore, dynamic schema generation allows for the creation of new meta-instances and meta-properties at run-time. This mechanism is needed for the on-demand adoption of previously unknown schemata. More details on schema adoption are given in the Section 5.

4 DATABASE

For the central database implementing the CWM of the distributed simulation system we chose SupportGIS Java, an object-oriented database technology with focus on compliance to international standards (OGC 2012), spatial data, functional range, large data sets, and performance. It can be accessed as an application or by use of its public API. Different storage containers can be used as a back-end. Furthermore, it provides adapters for several databases (e.g., PostgreSQL, Oracle, MySQL, . . .), for a transient cache, a WFS-server, and some more (Fig. 3). The different storage containers are encapsulated by the kernel-based API so users and application developers can use one unified interface.

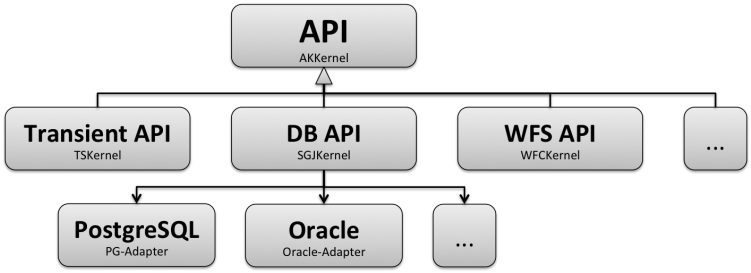

Figure 3: Hierarchy of the SupportGIS Java API.

There are several importers and exporters to handle schema (e.g. XML Schema) and instance data (e.g., XML or GML documents). A generic base schema allows for the arbitrary definition of complex object-oriented data schemata with relations or embedded elements. SupportGIS Java also provides built-in geometry data types (e.g., Polygon, Line, Point) that are used just like other data types (e.g., Long, String) in schemata like SEDRIS or CityGML. Furthermore, it can represent XML and GML data, which allows for the execution of complex queries compliant with OGC filter encoding.

For a database-driven 3D simulation, the central database must provide a notification system that is able to exchange data between its clients. Thus, a first attempt was to use the notification mechanism included in the PostgreSQL back-end of SupportGIS Java. But this turned out to be only usable for slow communication. Furthermore, a notification could only be received after sending the next SQL statement and no push service was provided. We therefore implemented our own service. In a first step, this client-side approach uses its own database connection and table to avoid interference with regular traffic. The sending client actively writes notification data to the database (push), which is read and evaluated by the listening clients via polling the dedicated notification table.

Technically, each notification is represented by a row in the table that contains a unique notification ID, an ID of the sending client, a time-stamp and the content. The content describes the type of action (e.g., insert, update, delete) and the affected objects. To avoid messages getting lost caused by database latency, the commit-time of the database connection is used as a time-stamp. Since this given commit-time can differ a little from the actual time the data gets visible to the polling clients, they use a small time slot looking for new messages. A cache containing the last messages is used to prevent firing a message twice by checking its unique ID. SupportGIS Java uses this base communication layer to enable adding of schema- and object-specific change listeners that allow for easy and efficient data monitoring. Such listeners are registered by the simulation clients to receive external notifications as described in Section 5.

Practical tests have shown that this procedure works fine (see Section 8), but it reaches its limits under heavy load. In future, the current procedure will be extended to a “notification system 2.0”. This would change the base communication layer from polling into a dedicated client-server layer with a separate server software that allows to push messages.

SupportGIS Java furthermore provides a versioning mechanism to track objects over time and to analyze historic states. To achieve good performance even when changing lightweight values (e.g. a single integer value) in otherwise heavyweight objects (e.g. containing binary (BLOB) values), internally, the versioning

mechanism is split into two complementary parts: The object-history keeps track of the creation and deletion of whole objects, while the attribute-history monitors the changes in an existing object. Both of them are again split into two parts: One for the current object state and one for all other (usually old) states. This keeps the 'current' state fast, which is the common way of access.

5 DATABASE INTERFACE

The database interface for simulation clients provides a dynamic two-level synchronization. By replicating objects from the central database to the internal simulation database and keeping them "in sync" not only object data is synchronized. Also, the simulation system adopts the data schema of the central database by synchronizing it with its internal simulation database.

This approach has several advantages over the direct usage of objects from the central database. By using the (replicated) objects in the simulation database, details of the central database are hidden from all internal components of the simulation system (particles, physics, sensor data processing, rendering, etc.) allowing for transparent data access, and real-time visualization and simulation. The internal simulation database works as an "intelligent" data cache to speed up repeating access patterns, which would otherwise lead to repetitive queries to the central database. Additionally, using a central database provides the simulation system with persistence as changes made to replicated objects can be resynchronized back into the database. Other clients or tools that are not realized using VEROSIM can still directly access the database, without invalidating the dynamic two-level synchronization as a whole (see Fig. 2).

5.1 Schema Synchronization

The first level of this generic approach is the schema synchronization between the central database and the internal database of a simulation client. The result of this synchronization is a schema mapping: classes are mapped to meta-instances and attributes (or relations) to meta-properties (Fig. 4 left). The schema synchronization can be realized in two ways: Either by schema matching or by schema transfer.

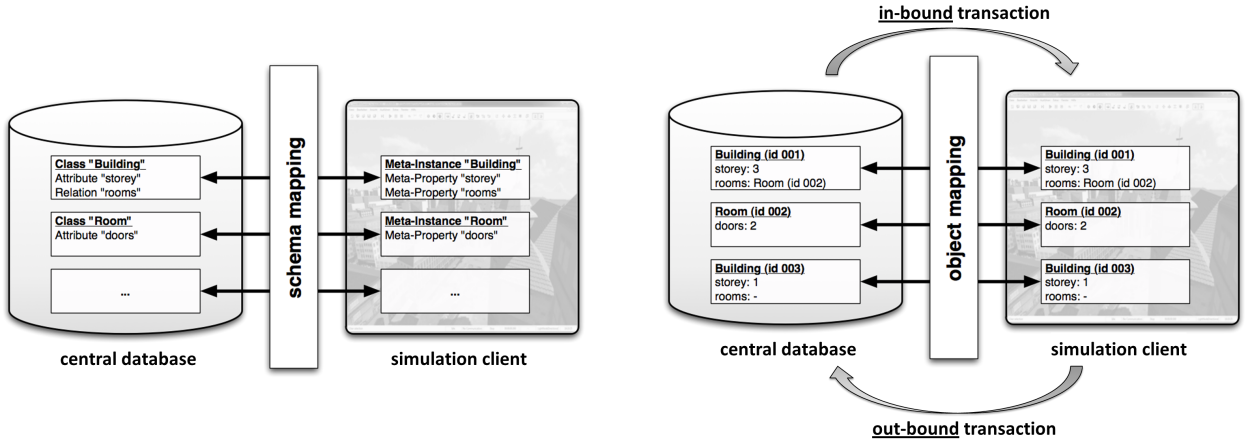

Figure 4: Schema mapping (left) and object mapping (right).

In **schema matching**, a previously defined schema for the simulation database is matched against the same schema in the central database (e.g., class *Building* matches meta-instance *Building*, attribute *storeys* matches meta-property *storeys* ...). Technically the internal schema is hard-coded and defined at compile-time. The matching process itself is name- and type-based using VSD's reflection mechanism mentioned above.

In contrast, **schema transfer** is entirely performed at run-time using dynamic schema generation, also described above. The central database's schema is evaluated and meta-instances are defined (e.g. a

meta-instance *Building* is created) without previously being hard-coded. Based on the class's attributes and relations, the appropriate meta-properties are created (e.g. meta-instance *Building* gets a meta-property *rooms*) defining data and structure of the schema within the simulation system.

These two principles of schema synchronization can of course be mixed, so that classes or attributes missing during the schema matching process can be added dynamically by schema transfer.

5.2 Data Synchronization

After synchronizing the schemata, which is done only once during run-time initialization, data is synchronized on the object-level (including attributes and relations). This synchronization provides the basis for our fundamental principle of data access: transparency. No component of the simulation system has direct access to the central database – only the interface itself. Instead, these components access database objects by means of the local copies in the internal simulation database, i.e. replicated versions of database objects. This built-in transparency encapsulates the central database. Furthermore, the replicated objects within the simulation database serve as cached versions of the external database's objects, necessary for real-time data access in the simulation. Based on this principle, data synchronization now includes object loading, management and unloading, as well as change tracking and bidirectional resynchronization. It spans the complete run-time of the system.

Object loading implies that an object from the database is replicated into the simulation database. This is realized by querying the object and instantiating an instance of the corresponding meta-instance (known from the schema mapping). Then all attributes and relations are copied to the corresponding properties and a mapping between the database's object and the replicated instance is inserted into the object mapping (Fig. 4 right). Hence after loading an object, the data it represents is transparently available as a local instance to the rest of the simulation system.

A crucial part of data synchronization is change tracking. For this, the database interface registers itself to the notification systems of the simulation database as well as of the central database. Change notifications include messages about the insertion of new objects, the deletion of existing objects, the modification of an object attribute's value, and the setting or unsetting of a reference between two objects. Notifications of the simulation database are called *internal* while the central database's notifications are called *external*. Change tracking is then done as follows:

1. An incoming notification is first checked for its relevance. For example, external messages about a non-replicated object being changed or deleted can be ignored.
2. In the next step, a transaction is created for the notification. An internal notification leads to an *out-bound* transaction for transferring the change to the external central database. Accordingly, an external notification causes an *in-bound* transaction (Fig. 4 right).
3. Before queuing the new transaction for later execution, previously queued transactions for the same object are examined for possible neutralizations. For example, an existing in-bound transaction for object deletion and a new out-bound transaction for the deletion of the very same object can both be discarded ("case closed").

When a command for bidirectional resynchronization between local simulation and external central database is issued, the database interface switches from the aforementioned change tracking mode into sync mode. Here, the list of queued transactions is processed. Afterwards the interface is switched back to change tracking mode. New notifications, which were queued during sync mode, are ignored for changes caused by own transactions or processed as usual for other one's changes.

The command for resynchronization can be carried out immediately or by user request, depending on the application. For database-driven distributed simulation, it can for example be bound to a resynchronization timer synchronizing every n ms (if anything was queued).

In addition to providing the basis for the distributed simulation, resynchronization can also be perceived as the implementation of a persistence layer for the simulation system as changes to the virtual environment can be made permanent. Thus, simulation results can be saved and virtual environments can be manipulated permanently, right within the simulation system giving it more flexibility in the choice of possible applications (see Section 8).

Besides loading and resynchronization, instances can also be unloaded, provided that no out-bound transactions have been stored for them and that they are currently not used in the simulation. Unloading is simply realized by deleting the instance from the internal simulation database and removing the object mapping. Usually, unloading is used for dynamically streaming parts of large data sets, e.g. city models.

6 FUNCTIONAL DATA SYNCHRONIZATION

When using third party data schemata like SEDRIS or CityGML, the simulation client might be confronted with objects of previously unknown classes. Based on reflection and inheritance from built-in base classes, even for these new classes the system provides generic methods for data access to internal simulation components and standard user interfaces so they are nevertheless able to read and write the objects and their attribute values.

For data like geometries, transformations or dynamics parameters there is demand for a mapping to specific representations in the simulation database so that their functional meaning can be identified and used by the simulation algorithms. For example, the internal renderer needs an appropriate geometry representation to be able to process geometry data. We call this the "functional data synchronization", which produces a "functional data mapping". Functional data synchronization (and thus mappings) can be *schema-aware* or *schema-unaware*, *bidirectional* or *unidirectional*, and *substituting* or *supplementing*.

Schema-unaware synchronization does not use any prior knowledge on the employed schema and can thus be treated in a generic fashion. For example it can be used to map the built-in geometry type (see Section 4) of a geo database to the simulation system's geometry representation. Another example of use would be for standard variations of simple attribute mappings like mapping an integer attribute to a color property (think of $10526880 = \#A0A0A0$ being translated to $(160,160,160)$) instead of an integer property (the standard behavior), see Fig. 5 left. Most schema-unaware mappings are also **bidirectional**. Regarding the aforementioned geometry example this means, when changing an object's geometry in the central database, the functionally mapped geometry in the simulation database gets updated accordingly and vice versa. As schema-unaware synchronization is independent from a special data schema it is usually implemented directly within the database interface itself and can be chosen for selected attributes by configuring the interface.

We call a functional mapping **substituting**, when the representation in the simulation database replaces the standard representation. This is the usual case for schema-unaware synchronization. In the example (Fig. 5 left), the integer attribute in the central database would directly be synchronized to a color property in the simulation database. So the color property replaces the default integer property that would otherwise be used. This can only be realized by directly accessing the central database (using its API). Thus, a substituting synchronization should be realized within the database interface's loading and resynchronization mechanisms itself to adhere the principle of transparency (note the combined object and functional mapping in Fig. 5 left).

In contrast, a functional mapping is called **supplementing** when the database interface's standard mechanisms are combined with a downstream functional synchronization mechanism within the local simulation database. One can apply this principle to the simple integer-to-color example above (Fig. 5 right). In a first step, the integer attribute in the central database would be mapped to an integer property in the simulation database, thus being managed using a standard object mapping of the database interface. In a downstream functional mapping, this integer property is synchronized to an additional color property (both within the simulation database). For the schema-unaware case, a supplementing synchronization is

usually chosen when both the original representation and the functional translation are needed within the simulation system or very special translations are needed in between.

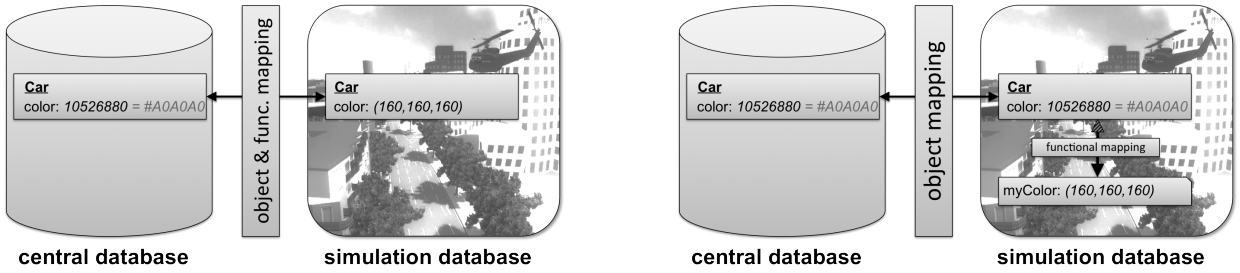

Figure 5: Exemplary *schema-unaware* functional data mappings. Left: *Substituting* and *bidirectional*. Right: *Supplementing*, and *bidirectional* or *unidirectional* (depending on striped arrowhead).

Schema-aware functional synchronization is needed for simulation data that is somehow encoded within its schema. The synchronization mechanism has to be aware of the specific classes and attributes of the schema that has to be interpreted and mapped to the appropriate representation in the simulation database. For example, it has to "know" if dynamics parameters were encoded in SEDRIS's Property_Value structures (Fig. 6). Schema-aware synchronization should therefore be supplementing. This way, the database interface can be used for loading and resynchronizing the encoded data without knowing anything about the schema's meaning. A downstream mechanism for locally synchronizing the encoded data to the supplementing representation within the simulation database can then exploit its knowledge about the used schema. This can be done unidirectionally for functional translations that won't change within the simulation or whose changes are only of local interest. Using bidirectional downstream synchronization, changes to the supplementing representations are resynchronized to their encoded counterparts that for their part are resynchronized to the central database by the generic database interface. All in all, when changes to a functionally translated representation shall be distributed via the central database, the downstream synchronization to the encoded data needs to be bidirectional. This also holds for the schema-unaware case.

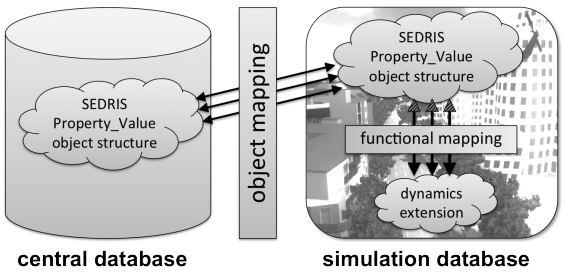

Figure 6: Exemplary *schema-aware* functional data mapping: *supplementing*, and *bidirectional* or *unidirectional* (depending on the striped arrowheads).

7 VERSIONING INTERFACE

As described above, the central database provides a versioning mechanism that archives every previous state of the object data as a time-stamped version. It also keeps a global log of the changes that led to these versions. The versioning interface is a component of the simulation system's database interface providing transparent access to these historic states and the change log. A user interface element or any other component of the simulation system can then set a reference time and the versioning interface takes care of reloading the appropriate versions of the object data into the simulation database. The point in time

can either be arbitrary or it can be chosen from the change log, which is also available from the versioning interface. The change of reference time leads to an update of the replicated data to match the chosen historic state, meaning attributes and references may change, and replicates may be removed or inserted.

Using a naive approach each new request would lead to a reload of the total of replicated objects. This can significantly be sped up by incorporating the change log into the reload. For that, the versioning interface has to keep track of the requested points in time. When receiving a new request t_i, it retrieves all log entries between this and the previous point in time t_{i-1}. This list of changes contains all objects that actually have to be updated. When going forward in time ($t_i > t_{i-1}$), the changes are processed in ascending order to preserve their chronological sequence. When going back in time ($t_i < t_{i-1}$) instead, not only the order of changes must be inverted but also their meaning. The deletion of an object is inverted to its insertion and vice versa. The same applies to the setting or unsetting of references. Only attribute changes need not be adjusted. Finally, reloading the (possibly inverted) changes is realized using the very same transaction mechanism as described in Section 5 by interpreting the list of changes as subsequent external notifications from the central database.

8 APPLICATIONS

Using the presented approach different categories of applications have already been realized. Besides the focus in this paper on distributed simulations, this also includes distributed algorithms and distributed real world processes.

8.1 Distributed Simulation

In a distributed urban 3D simulation scenario the architecture has been incorporated to its full extent including the database-driven distribution of simulation data and a downstream off-line viewer to replay the archived run of a simulation (Fig. 1). The scenario consists of a CityGML-based model (City of Düsseldorf) as static model data and SEDRIS-based car and helicopter models as dynamic model data. The model data is loaded from the central SupportGIS Java database into two attached VEROSIM simulation clients where simulation specific data is reconstructed using functional data synchronization. With an id-based approach for the configuration of active and passive components one client controls the car while the other controls the helicopter. Movements are resynchronized to the central database and distributed to the respective other client. Afterwards, the versioning mechanism allows an off-line replay of the simulation run on one of the clients.

To measure the performance of the distributed simulation we set up two personal computers (PC1, PC2) connected via a Gigabit Ethernet switch. PC1 runs the central database and one VEROSIM client, PC2 the second VEROSIM client. On one machine the car is configured to automatically drive in a circle while at the same time on the other machine the helicopter is configured to turn itself around an axis. On each machine both movements are visualized - one locally simulated and one transmitted. Thus both clients need to permanently read from and write to the central database to resynchronize their data. Two passes with an alternating assignment of configuration (helicopter, car) to machine (PC1, PC2) ensure comparability. Table 1 shows the associated measurements for reading and writing (example: when the helicopter is simulated locally on PC1 the system needs 0.65ms/op to read a new value (for the car) from the central database).

Table 1: Performance in milliseconds per read (left) and (write) operation.

read	helicopter	car
PC1	0.65ms/op	0.72ms/op
PC2	1.28ms/op	1.32ms/op

write	helicopter	car
PC1	2.30ms/op	2.26ms/op
PC2	4.70ms/op	4.60ms/op

As writing values triggers versioning and notifications it is slower than reading values (factor $3-3.5$). The results for PC2 are slower because PC1 locally runs the central database (factor 2), which could be

improved using other network hardware. To obtain another reference we also measured the pure writing performance to the database using its API without VEROSIM. When constantly writing new values to a local database we reached about 2.00ms/op (including versioning and notifications). Compared to 2.30ms/op and 2.26ms/op the simulation system's overhead seems to be within tolerance.

8.2 Distributed Algorithms

In the research project FastMap (Roßmann et al. 2011) the same architecture is used in the development of future space missions. The major goal of FastMap is the automatic generation of navigation maps as a basis for self-localization and navigation of mobile robots during the exploration of planetary surfaces. The architecture is used to connect a Virtual Testbed, a key concept in the field of eRobotics (Roßmann and Schluse 2011), and other data generators to a central database, providing the common data store and communication framework of the mission. In the context of the complementing research project SELOK (Roßmann, Sondermann, and Emde 2011) a new approach for a self-localization and navigation unit for mobile robots in extraterrestrial environments is developed (Fig. 7 left) using the same central database.

8.3 Distributed Real World Processes

Another major application is the Virtual Forest, a science project (Roßmann et al. 2008) whose goal is to provide new means for forest inventory, serving as the basis for the development of new decision support systems and the application of state of the art industrial automation techniques to the forest industry. In the context of the Virtual Forest, the proposed architecture has been used for simulations (Roßmann et al. 2011) and an integrated tool for forest inventory (Fig. 7 right).

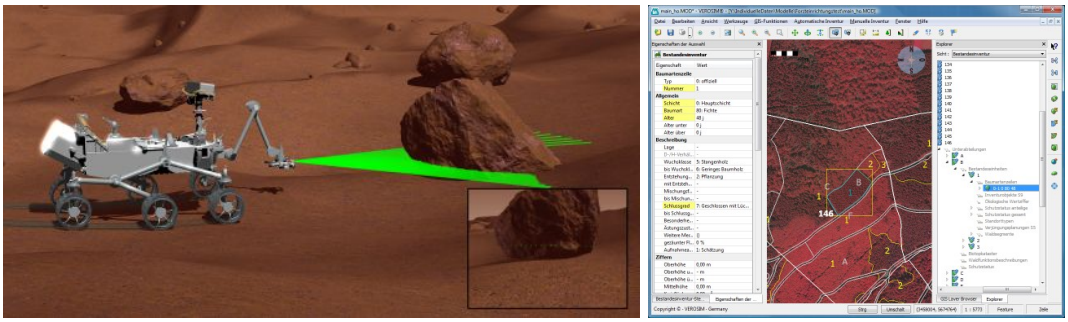

Figure 7: SELOK Virtual Testbed with Mars rover (left), Virtual Forest inventory tool (right).

9 CONCLUSION AND FUTURE WORK

We presented a new architecture for database-driven distributed 3D simulation that combines 3D real-time simulation techniques with object-oriented data management. Simulation clients replicate schema and object data from a central database that stores static and dynamic parts of a simulation model, distributes changes caused by the simulation, and logs the simulation run. The presented architecture has already proven its feasibility, performance and flexibility from urban scenarios to space robotics and back on earth to forestry. It will be the basis for many more simulation applications and Virtual Testbeds in these and other fields of applications like industrial automation. Its fundamental advantages are persistence, a consistent data model throughout the distributed system, a solution to the "id problem", a versioning mechanism for built-in logging, and standard interfaces to access shared model data by simulation clients as well as clients for data generation or evaluation.

Currently, we are evaluating the internal simulation database's data modeling infrastructure using the Meta Object Facility (MOF). As a next step we plan to analyze concurrency control in the distributed system including the handling or avoidance of conflicting in- and out-bound transactions, as well as locking

strategies for the central database. Furthermore, the versioning mechanism of SupportGIS Java is currently extended to support multiple time types (i.e. notions of time). Supplementary to the system time of the server, this allows to manage time-stamps for e.g., a simulation time, representing a simulation's actual time response, or a validation time, representing the time a value is delivered to the public. The logical handling of combinations of different time types requires clear rules that will be investigated in future work.

ACKNOWLEDGMENTS

We like to thank our colleagues Ralf Waspe, Malte Rast, and Thomas Steil at the Institute for Man-Machine Interaction and Christoph Averdung, Martin Krückhans, Stephan Struppler and Ralf Stüber at CPA Systems for supporting the presented work.

Virtual Forest: This project is co-financed by the European Union and the federal state of North Rhine-Westphalia, European Regional Development Fund (ERDF). Europe - Investing in our future.

FastMap and SELOK: Supported by German Aerospace Center (DLR) with funds of the German Federal Ministry of Economics and Technology (BMWi), support codes 50 RA 1034 and 50 RA 0911.

REFERENCES

Autodesk 2005. "Best Practices for Implementing Autodesk Vault".

CityGML 2012. "Virtual 3D City Models". Accessed Apr. 2, 2012. http://www.citygml.org.

Dassault Systèmes 2008. "ENOVIA V6 Architecture Performance Capability Scalability a Product Lifecycle Management Whitepaper Prepared by ENOVIA , a Dassault Systèmes Brand Executive Summary".

Freund, E., M. Müller, and J. Roßmann. 1999. "Data storage and flow control in automation systems by means of an active database.". In *Computational intelligence for modelling, control & automation '99.*, edited by M. Mohammadian. IOS Press Amsterdam.

Hoppen, M., J. Roßmann, M. Schluse, R. Waspe, and M. Rast. 2011. "Combining 3D Simulation Technology with Object-Oriented Databases: A Database Oriented Approach to Virtual Reality Systems". In *Proceedings of the ASME 2011 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE)*, edited by B. Epureanu and H. Dankowicz, 1545–1554.

Julier, S., Y. Baillot, M. Lanzagorta, D. Brown, and L. Rosenblum. 2000. "Bars: Battlefield augmented reality system". In *NATO Symposium on Information Processing Techniques for Military Systems*, edited by J. Grosche, 9–11.

Kaku, K., H. Minami, T. Tomii, and H. Nasu. 2005, April. "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)*, edited by A. Makinouchi and M. Zaki, 1259–1259: IEEE.

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

OGC 2012. "Open Geospatial Consortium". Accessed Apr. 2, 2012. http://www.opengeospatial.org.

Roßmann, J., and M. Schluse. 2011. "Virtual Robotic Testbeds: A Foundation for e-Robotics in Space, in Industry - And in the Woods". *Developments in eSystems Engineering, International Conference on* 0:496–501.

Roßmann, J., M. Schluse, and A. Bücken. 2008. "The virtual forest - Space- and Robotics technology for the efficient and environmentally compatible growth-planing and mobilization of wood resources". *FORMEC 08 - 41. Internationales Symposium*:3 – 12.

Roßmann, J., M. Schluse, R. Waspe, and R. Moshammer. 2011, December. "Simulation in the Woods: From Remote Sensing based Data Acquisition and Processing to Various Simulation Applications". In

Proceedings of the 2011 Winter Simulation Conference, edited by S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, 984 – 996. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Roßmann, J., B. Sondermann, and M. Emde. 2011. "Virtual Testbeds for Planetary Exploration: The Self-Localization Aspect". In *Advanced Space Technologies in Robotics and Automation*.

Roßmann, J., N. Wantia, M. Springer, O. Stern, H. Müller, and M. Ellsiepen. 2011. "Rapid Generation of 3d Navigation Maps for Extraterrestrial Landing and Exploration Missions : the Virtual Testbed Approach". In *Advanced Space Technologies for Robotics and Automation*.

Schmalstieg, D., G. Schall, D. Wagner, I. Barakonyi, G. Reitmayr, J. Newman, and F. Ledermann. 2007. "Managing complex augmented reality models". *IEEE Computer Graphics and Applications* 27 (4): 48–57.

Schweber, Erick Von 1998. "SQL3D - Escape from VRML Island". Accessed Apr. 2, 2012. http://www.infomaniacs.com/SQL3D/SQL3D-Escape-From-VRML-Island.htm.

SEDRIS Technologies 2012. "Environmental Data Representation & Interchange". Accessed Apr. 2, 2012. http://www.sedris.org.

Vakaloudis, A. and Theodoulidis, B. 1998. "Spatiotemporal Database Connection to VRML".

Walczak, K. 2012. "Dynamic Database Modeling of 3D Multimedia Content". In *Interactive 3D Multimedia Content*, edited by W. Cellary and K. Walczak, Chapter 4, 55–102. London: Springer London.

Watanabe, C., and Y. Masunaga. 2002. "VWDB2: A Network Virtual Reality System with a Database Function for a Shared Work Environment". In *Information Systems and Databases*, edited by K. Tanaka, 190–196. Tokyo, Japan.

AUTHOR BIOGRAPHIES

MARTIN HOPPEN studied computer science at the University of Bonn, Germany. Since 2007 he is a Ph.D. student and research associate at the Institute for Man-Machine Interaction of RWTH Aachen University (Germany) where he works on combining 3D simulation systems with databases. His email address is hoppen@mmi.rwth-aachen.de.

MICHAEL SCHLUSE studied electrical engineering at the University of Dortmund, Germany. Between March 1996 and February 2005 he was researcher and team leader "System Technology" at the Institute of Robotics Research (IRF) in Dortmund. In May 2002, he received his doctorate at the University of Dortmund. Between March 2005 and March 2006 he was department head at the EFR-Systems GmbH and responsible for the development of modern 3D simulation technologies and versatile Virtual Reality systems. Since April 2006 he is senior engineer at the Institute for Man-Machine Interaction at the RWTH Aachen University. His email address is schluse@mmi.rwth-aachen.de.

JÜRGEN ROSSMANN studied electrical engineering at the Universities of Dortmund and Bochum, Germany. After his studies he worked as a researcher and team leader at the Institute of Robotics Research (IRF) in Dortmund. He received his doctorate in 1993 and was named junior professor for robotics and computer graphics at the University of Southern California in 1998. He habilitated in 2002 at the University of Dortmund and was managing director of EFR-Systems GmbH in Dortmund from 2005 to 2006. Since 2006 he is director of the Institute for Man-Machine Interaction and full professor at the RWTH Aachen University in Aachen, Germany. His research interests are projective virtual reality, multi-agent control and supervision, multi-sensor integration, system simulation and optimization techniques, computer vision, real-time visualization and man-machine interaction. His email address is rossmann@mmi.rwth-aachen.de.

BJÖRN WEITZIG received a diploma in computer science from the University of Dortmund and a Ph.D. for his work on 3D city modeling at the University of Bonn. He is a senior consultant at the GIS company CPA-Systems. His email address is weitzig@supportgis.de.