

A Novel Distributed Database Synchronization Approach with an Application to 3D Simulation

Martin Hoppen and Juergen Rossmann

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

Email: {hoppen, rossmann}@mmi.rwth-aachen.de

Abstract—3D (three-dimensional) simulation applications from various fields benefit from the usage of database technology. In contrast to the prevailing naive file-based approach, simulation models can be managed more efficiently, temporal databases can be used to log simulation runs, and active databases provide a means for communication. Thus, we use a central database to manage shared simulation models. To enable real-time access, each simulation client caches the model to its local runtime (in-memory) simulation database. For that purpose, a pairwise synchronization is needed between each runtime database and the central database. After a synchronization on schema level, each client replicates data on-demand. In this publication, we give a detailed description of our notification-based approach to keep master copies in sync with their replicate copies. The state of synchronization in between a pair of copies as well as allowed state transitions are comprehensively modeled using state machines. Moreover, we present three representative applications already using the approach, proving its practicability: City simulations, a Virtual Testbed for space robotics, and a forest inventory, management and simulation system.

Keywords—Database Synchronization; 3D Simulation; Distributed Database; Applications.

I. INTRODUCTION

In this publication, we extend our previous work from [1]. In particular, we describe more aspects of our novel distributed database synchronization technique and give a more detailed insight into three application scenarios using the presented approach.

Simulation applications in general and 3D simulation applications in particular all follow the basic principle of applying simulation techniques to a corresponding model. Hence, the discipline is called modeling and simulation. A simulation model however needs some kind of data management. Up to now, files are still common for this task. In [2], we present a database-driven approach to overcome the associated disadvantages. Here, a central database is used to manage the shared simulation model, while simulation clients perform an on-demand replication of the model to their respective runtime database. The latter is an in-memory database providing the necessary real-time access. A revised version of this system was shown in [3], where the central database is even used as a communication hub to drive and log distributed 3D simulations.

In this paper, we add a detailed description of the

notification-based synchronization approach used in this scenario. However, its specification should be preferably universal to allow for its adoption with different database systems. For that purpose, general requirements towards the two involved database systems – generically referred to as ExtDB (the central database) and SimDB (the runtime simulation database) – were compiled [4]. They incorporate methods adopted from Model-Driven Engineering (MDE) [5] and allow to use the concepts of the Unified Modeling Language (UML) [6] to give generalized method specifications for the different components of the overall approach [7]. Thus, in this publication, the synchronization approach will also be presented using UML metaclasses.

The synchronization approach relies on change notifications. Hence, ExtDB and SimDB need an according service. Using the notifications, the state of synchronization between both databases is monitored and modeled in a state machine for each pair of master and replicate copy. For resynchronization, transactions are scheduled and either executed or canceled out. Furthermore, notifications are used to confirm transactions and to detect change conflicts. A particular challenge in this scenario is to keep the state machine models "stable", i.e., not to miss or misinterpret notifications.

The approach is already used in different fields of applications, three of which are presented in detail in this paper: In various city and urban (distributed) 3D simulation scenarios, huge city models are stored in databases. Simulation clients use the presented approach to access the data and distribute changes like the movement of a car or a helicopter. In a Virtual Testbed for space robotics, planetary surveying, landing and exploration missions are developed and simulated using a shared world model. Different clients use the approach to access the model stored in a central database to deposit sensor data, extract maps, and utilize them for navigation and localization. Finally, in a large area forest inventory, management and simulation system, remote sensing data is used to extract semantic forest models managed in databases. Different stakeholders in the forestry sector can access these shared models to update, refine, simulate with, and analyze the data.

The rest of this paper is organized as follows: Section II presents work related to our own. In Section III, the foundations of the database-driven approach for 3D simulation are recapitulated. Section IV summarizes the system requirements

and the applied approach for method specifications using the UML metamodel. Both sections pave the way for the main Section V, where we present the notification-based synchronization approach. In Section VI, exemplary applications are shown. Finally, in Section VII, we conclude our work and present some future work.

II. RELATED WORK

Regarding database synchronization for 3D simulation systems and similar software only few approaches can be found. In [8], a combination of scene-graph-based 3D clients with a federation of databases connected by the Common Object Request Broker Architecture (CORBA) is proposed. On client-side, a local object-oriented DBMS (OODBMS) provides an in-memory scene object cache connected to the federation using an Object Request Broker (ORB). Cached objects are bidirectionally replicated to the scene graph. Concurrency control among the federated databases and the local object caches allows multi user interaction between the clients.

A mobile Augmented Reality (AR) system combining distributed object management with object instantiation from databases is described in [9]. Objects are distributed shallowly by creating "ghost" copies retaining a master copy only at one site. Such a ghost is a non-fully replicated copy of its master allowing simplified object versions to be transmitted (e.g., with sufficient parameters for rendering). Changes to the master copy are pushed to all its ghosts. Remote systems can change a master copy by sending it a change request.

In [10], [11], a Virtual Reality (VR) system is combined with an OODBMS to provide VR as a multi-modal database interface. In [12], a revised version adds collaborative work support. For update propagation, VR clients issue changes to the shared virtual environment as transactions to the back-end they are connected to. After an interference check they are committed to the database and distributed by a separate notification service. The system uses transactions with regular ACID properties (e.g., for "Create box B") committed as a whole as well as special continuous transactions for object movements. For the latter, atomicity does not apply as movements are committed incrementally to frequently propagate updates.

The "Collaborative Urban Planner" described in [13] is based on the multi-user Virtual Environment system DeepMatrix [14], extended by a relational DBMS back-end providing persistency. Clients allow for so-called shared operations like "rotate object" that are sent to the server for distribution and persistency. A server application provides concurrency control, message distribution and data management. It represents the single point of access to the database ensuring consistency among the clients' shared operations. The database primarily contains meta information on shared objects (position, texture).

In [15], a "Virtual Office Environment" contains 3D data and semantics managed by a DBMS to allow semantic-based queries and collaboration. Clients' actions are issued as queries to the shared database. Changes are distributed to all other clients, which adopt them locally.

A "shared mode" for database-driven collaboration is presented in [16]. In a chess application example with two players a shared database with the game's setting is alternately updated by the one client while being polled for changes by the other, which subsequently reflects the changes in his own virtual scene instance.

Compared to our approach, [8] comes close but lacks details and is only a proposal without known implementations. The ghosts in [9] may suffice for rendering but are restricted for sophisticated simulation applications. Furthermore, not all objects are managed by the database. In [10], [11], [12], [15], only VR-specific data and operations are supported. [13] does not manage the model data itself using the database. Finally, the approach in [16] is similar to our own but only demonstrates a very limited type of change distribution. Altogether, no other approach offers a comparably tight integration of database technology into 3D software or simulation systems.

Similarities to our MDE-based approach for the general assessment of database compatibility can be found in generic model management. [17] introduces different generic schema operations like match, merge, translate, diff, and mapping composition. The work gives an overview but concentrates on tool support for semi-automatic mappings. Our own approach can be seen as an implementation of the "ModelGen" operator that automatically translates a schema from one metamodel into another, including mapping creation. However, in contrast, we provide an automatic mapping of schemata and a runtime approach instead of a static mapping.

Another implementation is provided in [18]. A pivotal supermodel is used to transform schema as well as data. In [19], the same system is extended to provide runtime transformations with read-only access. A similar approach is taken in [20] using a proprietary pivotal graph-based representation. [21] presents an approach for transforming schema and data between the Extensible Markup Language (XML) and the Structured Query Language (SQL). However, none of these approaches use standardized metamodeling and model transformation languages as used in our approach.

Besides these database-centric approaches, related work can also be found in the field of parallel and distributed simulation. Overviews can be found in [22] or [23]. In this field, approaches focus on the synchronization of events and time in simulation – somehow similar to our synchronization using change notifications. However, they cannot be directly applied to the presented problem of distributed data management. Here, events (change notifications) can only be monitored – they cannot be affected as in discrete event simulation.

III. DATABASE-DRIVEN 3D SIMULATION

Using a central database (ExtDB) to manage a shared simulation model has several advantages. In contrast to a classical file based approach, databases provide a very efficient data management, well-defined access points, e.g., using a query language or an Application Programming Interface (API), a consistent data schema for structured data, and concurrent access for multiple users. This allows to persist the current state of a 3D simulation model comprising its static (e.g., building, tree, work cell) as well as dynamic (e.g., vehicle, robot) parts. During a simulation run, the state of its model's dynamic parts changes. This is an inherent property of simulation. To capture this process over time, a temporal database [24] can be used. Here, any change to the simulation model causes the previous state's conservation as a version. Altogether, this also allows to persist the course of the simulation itself. Besides these more or less passive activities, a database can also be used as an active part of the simulation. One approach is to use it as an active communication hub. An active database [24] is needed

that can provide the necessary change notifications to inform clients of changes to the shared simulation model.

However, a steady, direct data exchange with ExtDB is not advisable for 3D simulation. This would lack real-time capabilities and impose a strong coupling on each and every component of the simulation system with the utilized database system. Instead, we use an approach that combines ExtDB with a local runtime database (SimDB) for each simulation client. The lower part of Figure 1 shows the principle structure of this approach for a single pair of ExtDB and SimDB instance. By replicating required contents from ExtDB to SimDB, the simulation system can use the cached copies and the nature of ExtDB can be hidden away.

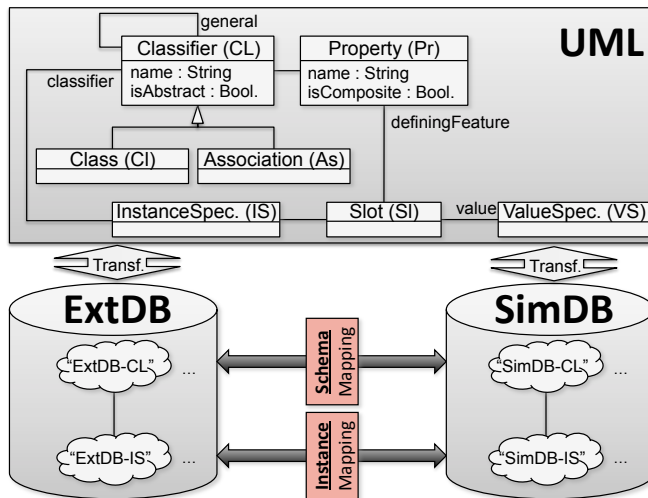


Figure 1. Principle structure of the approach for database-driven 3D simulation.

The two databases are synchronized on schema and data level. During the former, the schema description is transferred from ExtDB to SimDB so both systems "speak the same language". This builds up a schema mapping between the databases and is done once during system startup. Note however that this does not imply a semantic mapping like mapping an address represented by a single string to a fielded address representation (name, street, etc.). Instead, only the different modeling concepts (i.e., the utilized metaclasses) are mapped.

During runtime, data is loaded, i.e., replicated, from ExtDB to SimDB. Here, based on the schema mapping, the appropriate schema components are instantiated, values are copied, and an instance mapping (compare Figure 8) is stored to keep the relationship between master and replicate copy. Copies no longer required can also be unloaded, i.e., removed from SimDB provided they have not been changed. Changes are tracked and resynchronized to keep both master and replicate in sync. This is realized using notification services of ExtDB and SimDB. The approach is presented in detail in Section V. Besides for schema and instance data, synchronization can also be required on a semantic level. In functional data synchronization, the meaning of a modeled item is made available to the simulation system by translating it to a representation it can interpret. An example could be an engine modeled in the SEDRIS schema [25]. To allow a simulation of such a component it must be translated to the appropriate primitives

of the simulation system.

IV. SYSTEM REQUIREMENTS

To generalize the approach system requirements were identified [4]. The aim is to make it universally available for different implementations of ExtDB and SimDB. First of all, a general compatibility of the two databases' modeling concepts is stipulated. For that purpose, both their metamodels are taken into account. A database's metamodel represents its abstract syntax, thus its modeling concepts. Metamodels shall not only comprise metaclasses for describing schema components like tables, classes, or attributes. They must also contain the corresponding instantiation concepts (e.g., metaclasses for rows, objects, or values). This is needed to also enable data synchronization. The two metamodels' compatibility can then be expressed with a model transformation, e.g., using the ATL Transformation Language (ATL) [26].

To provide a common basis for arbitrary database metamodels, a pivotal metamodel with transformations from and to both databases' metamodels is stipulated as well. The pivot's metaclasses can be used to indirectly refer to SimDB's or ExtDB's metaclasses using the demanded mapping. In the context of 3D simulation, Geographic Information Systems (GIS), Computer-Aided Design (CAD), or other 3D software, an object-oriented modeling is advisable, as such data usually consists of a huge number of hierarchically structured parts with interdependencies [24]. Thus, the UML (language unit classes) is a reasonable choice for a pivot. Figure 1 gives an overview. It also comprises the mainly utilized UML metaclasses. Altogether, this allows to generically refer to the structure of SimDB and ExtDB using UML concepts. Therefore, the method specification in the next section uses concepts like object, link, class, or property although including any database metamodel that can be mapped to the UML metamodel.

Note, however, that this mapping to UML structures is conceptually needed to show the databases' compatibility and to obtain a means for generalized method specifications. The actual implementation of the synchronization approach is done on API or query language level – in particular to ensure real-time capabilities.

The two databases are also required to provide a notification service. In terms of the UML metamodel, notifications shall provide information on object insertions and removals, on property updates, and on link insertions as well as removals. Furthermore, they must provide a reflection interface to access schema components and instantiate corresponding data. Finally, objects must be uniquely identifiable.

V. NOTIFICATION-BASED DATABASE SYNCHRONIZATION

This section represents the main contribution of this article: A detailed description of the notification-based synchronization approach.

A. Comparison with Distributed Databases

Following the definition in [24], the presented scenario, i.e., the combination of SimDB and ExtDB, would be a distributed database (DDB). Figure 2 depicts a classical DDB structure. Several databases build a virtual database that is transparently accessed via the DDBMS. In the example, a set of Door objects is horizontally fragmented, allocated to the different

databases and thus partially replicated. Similarly, our approach aims at transparency of the distribution. In Figure 3, it is depicted correspondingly. However, it is a special case, in which SimDB is a cache for ExtDB. Simulation clients access the shared simulation model only via SimDB. The nature and (for the most part) the existence of ExtDB are hidden away. The master copy of the simulation model is stored in ExtDB. An exception are local changes in a SimDB instance that are not yet synchronized to ExtDB, thus residing only at that client. In contrast, a classical DDB is accessed as a whole from the outside and the DDBMS hides away its distributive nature.

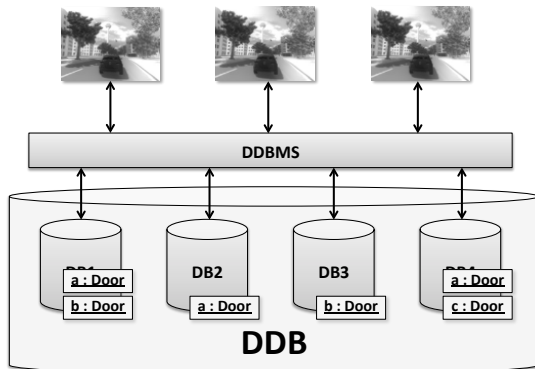


Figure 2. Classical distributed database with a centralized DDBMS.

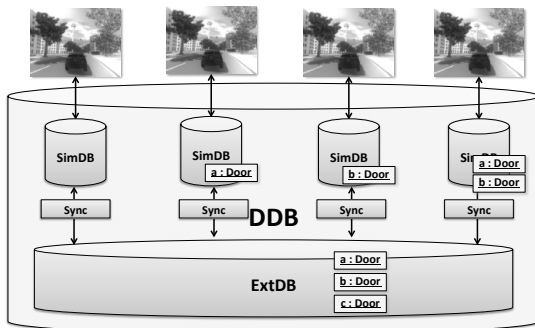


Figure 3. The presented approach can also be interpreted as a distributed database consisting of the central ExtDB and several SimDB instances.

Important DDB concepts are fragmentation, allocation and replication, as well as autonomy and heterogeneity. We use horizontal fragmentation splitting up object sets (but not objects themselves) between the central ExtDB and the connected SimDBs. All fragments are allocated to ExtDB. Further allocation, i.e., replication, to the different SimDBs is realized on-demand as shown in [7]. Thus, in the example in Figure 3, all Door objects are allocated to ExtDB and some of them are also allocated (i.e., replicated) to the connected SimDBs. While ExtDB is fully autonomous, SimDB is limited to the schema adopted from ExtDB. As both databases usually are different systems – e.g., SimDB is a runtime database – the assumed DDB is heterogeneous.

One or more instances of SimDB have a star-shaped connection to one instance of ExtDB. Changes are synchronized independently between each pair of SimDB and ExtDB. In the example in Figure 3, there are four SimDBs connected

with their respective synchronization component to the central ExtDB. Differences in between such a pair are resynchronized periodically but not synchronously. Thus, we have a similar scenario as described in [27] for replication servers with asynchronous replication. However, in contrast to mobile databases, the connection is always kept alive and resynchronization is typically short-term. Furthermore, there is no global transaction or recovery manager. Changes to ExtDB by any client or to SimDB by any client component are committed without control of the synchronization component, which can merely monitor such changes. Thus, following durability (as in Atomicity, Consistency, Isolation, Durability (ACID)) they cannot be undone. Durability is important as an online (i.e., live) 3D simulation cannot be reset in the middle of a run.

B. Lock-free Approach for Simulation

One way to treat concurrent changes is an active concurrency control using locks. For distributed concurrency control, one approach is to choose a so called distinguished copy, which holds a representative lock for all its replicate copies [24]. In our case, the master copies in ExtDB could be adopted for this purpose as they are shared among all clients. However, locking is not recommendable here as acquiring locks would be time-consuming (as an ExtDB access would be necessary each time) and possible deadlocks may interrupt a running simulation.

Therefore, we developed a lock-free approach using notifications. For each pair of SimDB and ExtDB, the mechanism monitors changes by listening to the notifications. For resynchronization, it schedules transactions of the respective database. Due to the monitoring approach, they can only comprise a single data operation. The approach is similar to optimistic concurrency control (OCC) [27]. However, transactions cannot be rolled back when changes are conflicting. Instead, conflicts are only implicitly resolved: The last client changing a value is given precedence. Altogether, it is crucial that the synchronization component always knows about the state of synchronization for each copy. However, besides resynchronization and passive monitoring, the mechanism cannot and must not intervene, e.g., by rejecting changes as mentioned above.

C. Notifications and Transactions

For each pair of SimDB and ExtDB, a change tracking component connects to the notification services of SimDB for so-called *internal* notifications and of ExtDB for so-called *external* notifications. Notifications include insertions and removals of objects and links, as well as updates of object properties. A link between objects can only be removed or inserted but not updated, as its identity is only derived from the connected objects (and the corresponding association on schema level).

For the sake of simplicity, external notifications from ExtDB are abbreviated as extInsert, extUpdate, and extRemove, internal notifications from SimDB as simInsert, simUpdate, and simRemove, accordingly (Table I).

During runtime, these notifications are evaluated. Depending on the current state of the corresponding pair of master and replicate copy represented by an instance mapping entry, a transaction may be scheduled that can later be used to resynchronize the detected change from the one to the

TABLE I. TYPES AND ABBREVIATIONS OF NOTIFICATIONS FROM SIMDB AND EXTDB.

	internal (SimDB)	external (ExtDB)
property update	simUpdate	extUpdate
instance insertion	simInsert	extInsert
instance removal	simRemove	extRemove

other database. A scheduled transaction comprises one data operation with its kind (insert, remove, or update), the affected instance (object or link) or its id, and for updates the affected property. Table II gives an overview over the utilized transactions and their abbreviations. A transaction for transferring a change from SimDB to ExtDB is called an *out-bound* transaction and will be abbreviated with the prefix *sim2ext*.

For example, when detecting an object insertion within SimDB by a *simInsert* notification, a new *sim2extInsert* out-bound transaction may be scheduled. Its (future) execution will insert an equivalent object of the corresponding ExtDB-Classifier (using the schema mapping) into ExtDB. Here, the current property values are retrieved from the SimDB object's slots and are replicated for the new ExtDB object. Finally, the new object complements the corresponding instance mapping entry with its identifier. This can be seen as the complementing operation to the loading of objects. Links are treated accordingly but without the need for property value replication. An instance's removal (object or link) from SimDB, notified by a *simRemove* notification, may lead to a *sim2extRemove* transaction whose (future) execution will remove the associated ExtDB instance. A *simUpdate* notification signals the change of a SimDB object's property and may be scheduled as a *sim2extUpdate* transaction to transmit the value change from SimDB to ExtDB. Similar to *sim2extInsert* transactions, a *sim2extUpdate* transaction's execution retrieves the current value of its corresponding property from SimDB and replicates it to ExtDB.

TABLE II. TYPES AND ABBREVIATIONS OF TRANSACTIONS BETWEEN SIMDB AND EXTDB.

	out-bound: SimDB→ExtDB	in-bound: ExtDB→SimDB
property update	sim2extUpdate	ext2simUpdate
instance insertion	sim2extInsert	ext2simInsert
instance removal	sim2extRemove	ext2simRemove

Accordingly, external notifications may lead to the scheduling of *in-bound* transactions for resynchronizing global changes from ExtDB to SimDB. They are prefixed by *ext2sim*: *ext2simInsert*, *ext2simRemove*, and *ext2simUpdate*. Responses to external notifications are mostly identical to their internal counterparts. However, due to the nature of SimDB being a cache for ExtDB, a variation applies when treating external insertions. New objects or links within ExtDB may be handled by different strategies. They may be ignored or subsequently taken into account by a loading transaction (*ext2simInsert*). In this paper, the latter approach is chosen. Alternatively, one could consider to reevaluate previously executed queries to determine the "interest" in the new instance.

D. Change Propagation – The Basic Idea

Figures 4–7 show the basic idea of the notification-based distributed synchronization using an example with a central

ExtDB and two SimDB clients. A door object is replicated to two simulation databases. One client changes the door's state indicated by a notification. The change is synchronized to the central database where another notification is issued. The latter causes another synchronization of the change to the second client.

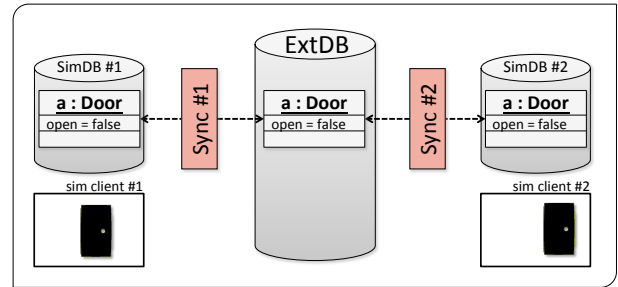


Figure 4. 1st step: Initial situation of a distributed synchronization example: all databases in sync.

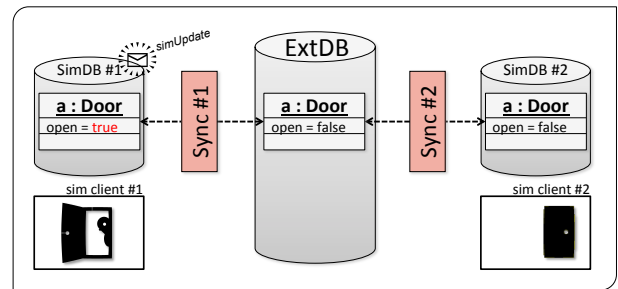


Figure 5. 2nd step: Client #1 changes the door's state; its SimDB issues a *simUpdate* notification.

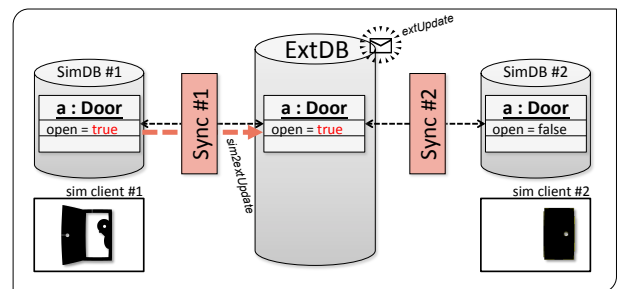


Figure 6. 3rd step: Sync component #1 responds using an out-bound *sim2extUpdate* transaction to synchronize the change to ExtDB, which in turn issues an *extUpdate* notification.

E. Modeling Synchronization With State Machines

Altogether, instance mapping entries (i.e., pairs of master and replicate copy) can be seen as to reside in a certain state of synchronization. Some examples are given in Figure 8: An object *a : Door* may exist in the central ExtDB without being loaded (i.e., replicated) to SimDB (i.e., no mapping exists), a replicated object *b : Door* may be unchanged (in sync), an object *c : Door* may only exist in SimDB (a transaction for its insertion in ExtDB is pending), a previously replicated object *d : Door* may be deleted in SimDB (a transaction

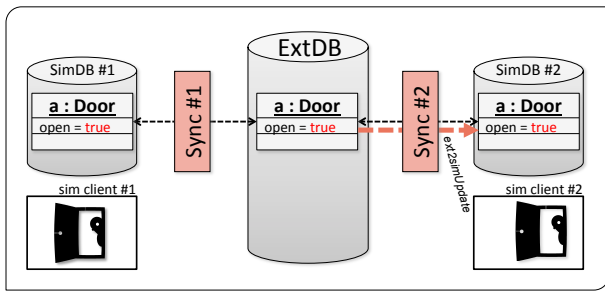


Figure 7. 4th step: Sync component #2 uses an in-bound ext2simUpdate transaction to adopt the change from ExtDB to SimDB #2.

for its deletion within ExtDB is pending), or a replicated object e:Door’s property may be changed within SimDB (a transaction for synchronization to ExtDB is pending).

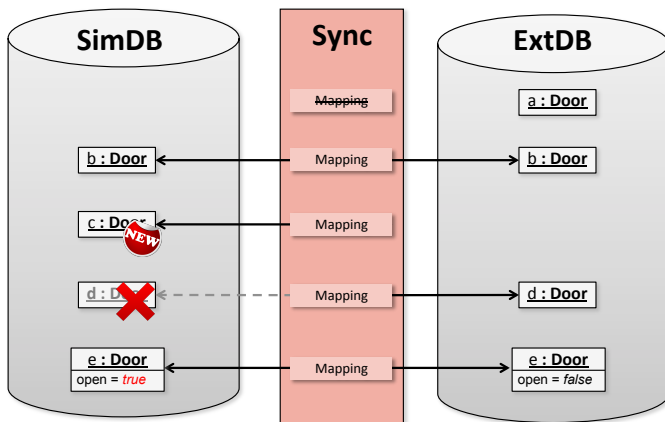


Figure 8. Exemplary mapping states between pairs of master and replicate copy.

This can be modeled as a state machine in statechart notation [28] for each object’s or link’s instance mapping entry. For objects, this state machine is given in Figure 11, for links in Figure 13. It may be in a synchronous state (*Synced*), a *Loading* or *Unloading* state, a state representing its absence or non-management (*NonManaged*), or a state of pending transaction (*ext2simInsertPending*, *ext2simRemovePending*, etc.). For update transactions, the synchronization states of an object’s properties are concurrently modeled in the sub states of state *UpdatesPending* shown in Figure 12.

In these state machines, events comprise internal and external notifications, as well as some management events for object loading and unloading, failure thereof, and update completion. To simplify the state machine diagrams, any event undefined for a state shall trigger a transition to an omitted error state. Notification events are also implicitly filtered based on the related instance to match the considered instance mapping. I.e., the state machine for a certain instance mapping will only receive notifications for the corresponding instances from ExtDB or SimDB. Note that transitions depicted with italic text are special conditions dealt with in the Subsection V-G.

F. Event Handling Within the State Machines

An exemplary chain of events – depicted in Figure 9 – would be the insertion of a new door object into SimDB

leading to a transition guarded by simInsert from the initial state NonManaged to state sim2extInsertPending shown in Figure 10.

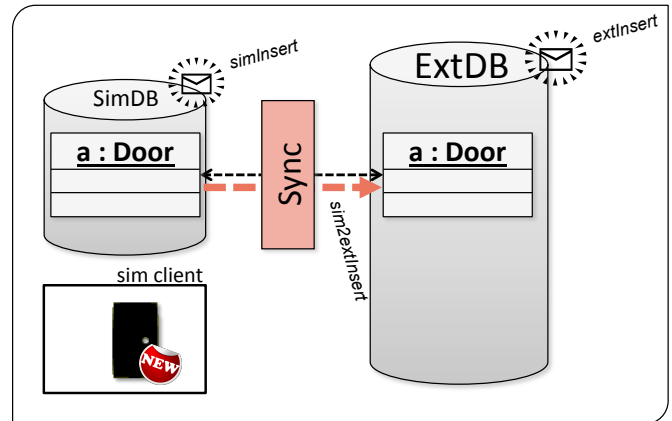


Figure 9. Exemplary insertion of a door object into SimDB and subsequent synchronization to ExtDB using a sim2extInsert transaction.

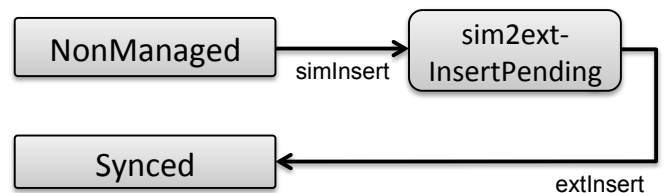


Figure 10. Excerpt from Figure 11 for the state transitions accompanying the exemplary insertion depicted in Figure 9.

In this example, a sim2extInsert transaction is scheduled for the new object. When the transaction is executed (see Subsection V-J), an equivalent object is inserted into ExtDB eventually causing the database to issue an extInsert notification. In turn, this event triggers a transition from the sim2extInsertPending to the Synced state. Thus, the extInsert event confirms the insertion into ExtDB and is used as a receipt to acknowledge a transaction’s successful execution. This is especially useful for handling concurrent changes within SimDB and ExtDB occurring during other transaction’s execution.

The receipt handling mechanism is also used to handle mutual changes that cancel each other out. An example are mutual removals: An instance is, e.g., first removed from ExtDB and subsequently from SimDB by independent processes. Thus, a previously scheduled ext2simRemove transaction with pending execution (in state ext2simRemovePending) is canceled out by the incoming simRemove notification for the same instance. The event causes a transition to the NonManaged state.

The same effect can be observed for the insertion of links. As before, links identify only by their member objects. In contrast to objects, they can be inserted identically but independently into both databases. In the state machine for links, this is represented by a transition from, e.g., sim2extInsertPending to Synced triggered by an extInsert for the identical link without having executed the scheduled transaction.

The state machines for objects and links differ only in some aspects. The latter allows an additional transition from the pending remove states back to the Synced state.

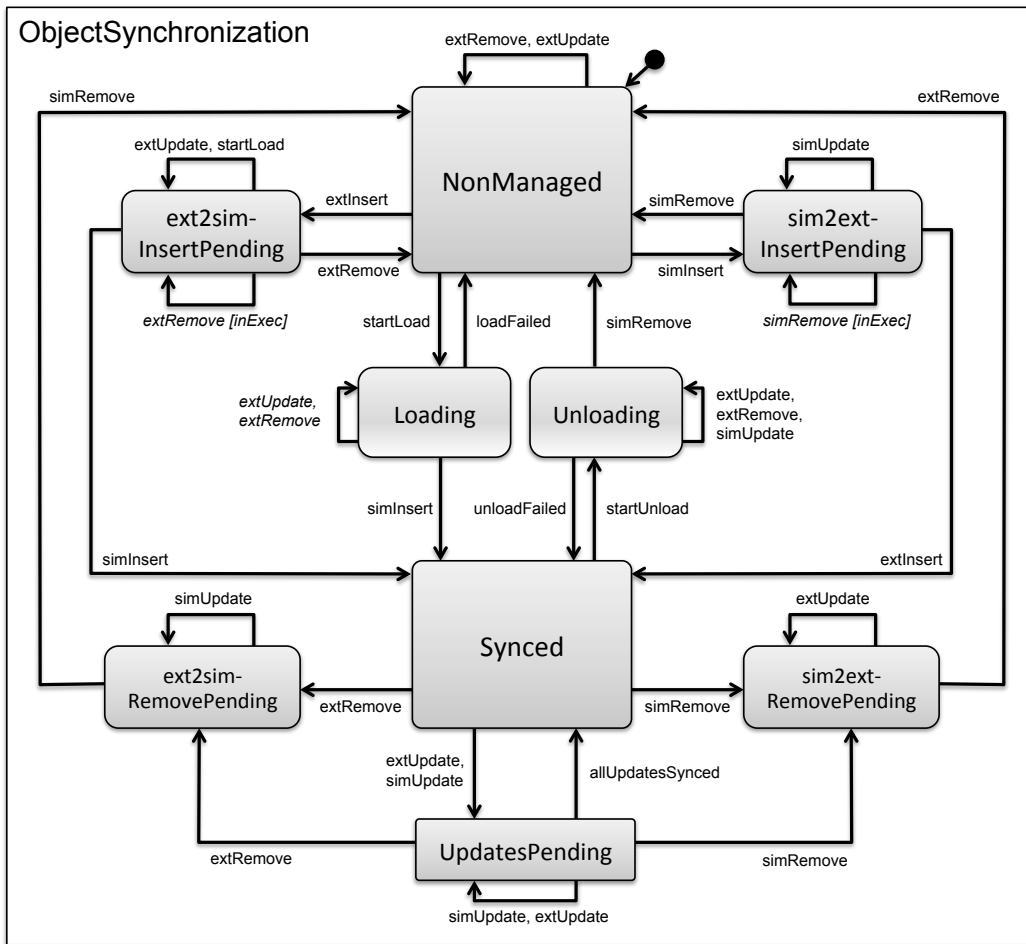


Figure 11. Synchronization states of an object's instance mapping.

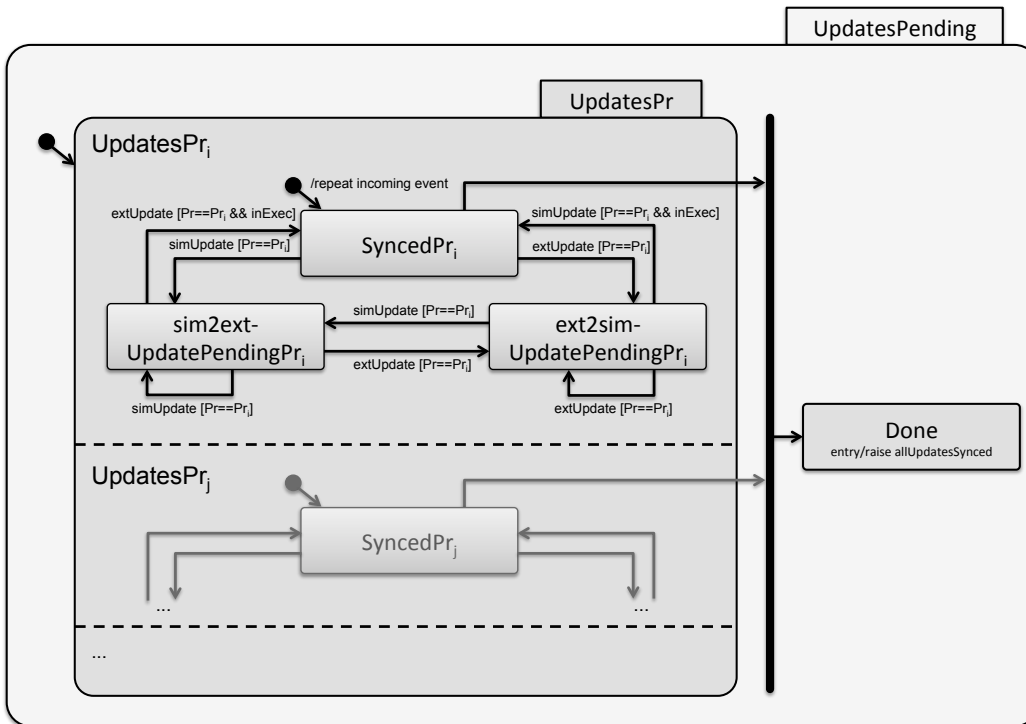


Figure 12. Sub structure of state UpdatesPending from Figure 11 for property updates.

In contrast to objects, an identical link can be reinserted after its removal. The former includes an additional update management for objects. The UpdatesPending state encapsulates a sub state structure for managing property updates (Figure 12). Primarily, it contains a super state UpdatesPr with concurrent regions for each of the object's properties, e.g., region UpdatesPr_{*i*} for the object's *i*th property. A region for Property Pr_{*i*} has three states representing an unchanged property value (SyncedPr_{*i*}), a property value changed within SimDB (sim2extUpdatePendingPr_{*i*}), and a property value changed within ExtDB (ext2simUpdatePendingPr_{*i*}). Internal transitions are triggered by update notifications (simUpdate and extUpdate) filtered for the corresponding property. To simplify modeling, the update event that caused the transition to UpdatesPending shall be repeated to initially activate the appropriate sub state, e.g., activate sim2extUpdatePendingPr_{*i*} by repeating a simUpdate event. Further updates to the object's value for Pr_{*i*} can be ignored when they stem from the same database (i.e., both SimDB or both ExtDB). For example, in state ext2simUpdatePendingPr_{*i*}, further extUpdate notifications for Pr_{*i*} can be ignored as the new value has to be transferred to SimDB, anyway. However, a subsequent update to the same property from within SimDB causes a change conflict (see Subsection V-G). The modeled strategy is to give precedence to the more recently notified change. Thus, a transition to sim2extUpdatePendingPr_{*i*} is triggered. When the transaction implicitly scheduled on entering one of the update states is executed, a notification is needed as a receipt. However, in contrast to insert or remove transactions, there is no "natural" counterpart for update transactions. An executed ext2simUpdate transaction causes a simUpdate notification that is indistinguishable from any other third party change. Thus, before execution, an "inExec" flag is set. For ext2simUpdate, the next simUpdate notification for Pr_{*i*} will trigger a transition back to the synced state of this property (the inExec flag will be reset). When all concurrent regions are in their respective synced state, a synchronized (in terms of concurrency) transition to the Done state is triggered (modeled by the vertical bar). On entering this state, the allUpdatesSynced event is raised triggering a transition from the super state UpdatesPending to the Synced state (see Figure 11).

In some situations, events may also be ignored. Within the state machines, this may be modeled as self-transitions. For example, in sim2extRemovePending, further extUpdate events from ExtDB can be ignored as the corresponding object will be removed from ExtDB, anyway. The same applies to pending insertions as property values will be replicated on execution time. For a pending ext2simInsert, the startLoad event has to be ignored, which will be emitted by the loading process used by the transaction. For non-managed instances within ExtDB, remove and update notifications can be ignored as they are of no interest to SimDB. Note that the sub regions in Figure 12 do not explicitly model ignored update events for properties $Pr \neq Pr_i$. Nevertheless, they shall be ignored and not cause a default transition to the omitted error state.

Besides change tracking, both state machines also contain states for the loading and unloading of objects or links. To announce a currently non-managed instance's loading, the methods presented in [7] shall raise an additional startLoading event. The instance remains in the Loading state either until its insertion into SimDB is acknowledged by an appropriate

simUpdate receipt event or until the load process fails. Accordingly, before unloading a synchronized instance, a startUnload event shall be raised and the Unloading state is activated. Reaching this state, subsequent changes to the instance are ignored and consequently get lost. A simRemove notification serves as a receipt to acknowledge the unloading process and triggers a transition to the NonManaged state. Failing this, an unloadFailed event is raised to return to the Synced state.

G. Change Conflict Handling Using State Machines

As mentioned above, changes (insertions, removals, and updates) from ExtDB and SimDB may conflict when they occur to the same instance (and property) before executing the corresponding transaction. For example, in a city scenario, a building's street number is locally changed within SimDB causing a sim2extUpdate transaction. Before this change is made persistent and globally available within ExtDB by executing the transaction in a resynchronization run, the very same number is changed within ExtDB (e.g., by another simulation client). Following the strategy modeled above, the previous change is omitted and instead a new ext2simUpdate transaction is stored.

In general, different strategies to handle such situations could be thought of. First of all, conflicts can be avoided beforehand by giving only mutual exclusive write access to instances. This approach could be used in distributed simulation scenarios where separate objects are simulated by different clients without interaction. This can be managed by a superordinate simulation control. Avoiding the occurrence of conflicts could also be realized by explicitly locking changed instances or their property value in the respective other database. However, this may stall or even reset a simulation run as mentioned above.

Thus, a monitoring, i.e., reactive handling of change conflicts as mentioned above is inevitable. The presented methods' strategy is embedded in the given state machines. For change conflicts, two scenarios can be distinguished: A conflict may either occur *before* or *during* a transaction's execution. Before executing a transaction, conflict handling can be realized straightforward. It is modeled with simple transitions within the state machines. One example is the precedence for more recently notified updates as shown above. Another strategy is that object removals are final and thus "always win". Id est, a pending remove transaction for an object precedes all update events for the object. For pending object insertions, conflicts cannot occur as the corresponding object does not exist in the respective other database. In the context of links, the behavior slightly differs. Again, as a link identifies only by the objects it connects, one and the same link can be independently inserted and removed from both databases. Here, the same strategy as for the update of an object's property value applies: the more recently notified change precedes previous changes.

As long as a transaction is still pending, incoming events can always be processed by state transitions to reflect the relation between SimDB and ExtDB. In a resynchronization run, the current state of each state machine is evaluated (compare Subsection V-J). If a state with pending transaction *T1* is determined *T1* is executed. However, this decision is made independently at each client. A notification from a previously committed, conflicting transaction *T2* may arrive just after *T1*'s execution is started. In some cases, *T1* may

still be abortable. But the notification may just as well arrive when T_1 commits. So, while native transactions of the utilized DBMSs themselves are usually isolated the decision to start a pending transaction is not. This limits transaction isolation (i.e., ACID properties) in the distributed system.

The same applies to the reading of property values. Objects can be removed, and links can be removed and inserted based only on the information from the corresponding notification. For object insertions and property updates however, the current state of the respective source database has to be retrieved as notifications themselves do not contain the corresponding values. Thus, when such a transaction is executed the source values may have already been changed by subsequent transactions whose notifications may either have not yet arrived or transaction execution may already have started as described above. This also limits transaction isolation.

Thus, a strategy had to be found for dealing with such situations. Otherwise, scenarios where a change in one database is neither reflected within the other database nor within the instance mapping's state machine may occur. For example, a property value is changed in ExtDB, but its instance mapping's state machine is in state Synced although SimDB still holds the previous value.

The primary instrument to handle such interfering changes is the aforementioned usage of notifications as receipts. For that purpose they must have the following features:

- 1) A notification's arrival guarantees the corresponding operation to be executed.
- 2) The order of arrival of a single database's notifications is identical to the execution order of the corresponding operations.
- 3) Between one running instance of SimDB and ExtDB there is at most one transaction being executed at a time (see Subsection V-J).

Based only on these assumptions, a conflict management can be stable. However, one should keep in mind:

- 1) A notification not yet received does not imply that the corresponding operation is not yet executed (notifications may be delayed).
- 2) On arrival of a notification, the *current* state within the database must not be consulted for further state transitions. By time of arrival it may already have been changed several times.
- 3) The order of arrival between notifications from ExtDB and notifications from SimDB is arbitrary.

Based on these considerations, a special event handling can be implemented to process the queued events after a transaction's execution. As stated above, the main problem are notifications arriving between the start of a transaction's execution and the arrival of the corresponding receipt notification. For a proper event handling, these events must sometimes be reordered. To be precise, they are captured and reinserted into the event queue just after the receipt event. This ensures their correct processing in terms of state transitions. The procedure is necessary for object or link insertions, link removals, object updates, and object or link loading. In the state machines, transitions with italic text particularly model this case. In the sub states of UpdatesPending, this highlighting is omitted as the same transitions are needed for standard and for this special event handling.

H. Exemplary Change Conflict Handling

One example for change conflict handling are updates (Figure 12). A property's update transaction can be examined separately as updates of different properties are independent from each other. Table III lists an exemplary sequence of events for some integer property and the associated actions, state machine states, values in SimDB and ExtDB, and emitted notifications. In the example, the local property's slot value in SimDB is updated several times even while changes are replicated to ExtDB. Notifications are used to ensure that all updates are reflected within the state machine's current state.

Initially (step #1), SimDB and ExtDB are in sync at value 10. The value in SimDB is changed to 20 (#2) and the corresponding simUpdate notification (a) triggers a state machine transition (#3). At some point in time, the client starts the resynchronization process (#4). Then, a first interfering update (#5) changes the value to 30. As property update notifications do not contain a value it must be retrieved from the respective database at transaction execution time (#6). Afterwards, a second interfering update (#7) changes the value to 40. In #8, the read value 30 is replicated to ExtDB. As mentioned above, the order, in which notifications from SimDB and ExtDB are received, is arbitrary. Thus, notifications simUpdate (a) and (b) may be processed first (#9, #10). As the "inExec" flag is set, all notifications are stored (instead of ignored without the "inExec" flag being set) until the corresponding receipt notification extUpdate is processed in #11. Subsequently, the flag is reset and both stored notifications are reinserted into the event queue. While the receipt notification eventually yields a transition back to the Synced state (#12), notification reinsertion causes the necessary transition back to the state of pending updates (#13) to replicate the value of 40 from SimDB to ExtDB. The additional simUpdate notification (c) only yields a self-transition (#14) as an update is already pending. Another resynchronization run would replicate the value to ExtDB starting at #15.

This approach to capture and reinsert notifications is needed as it is unknown whether an interfering update was done before (#5) or after (#7) reading the current value from SimDB in #6 to execute the sim2extUpdate transaction in #8. Note that when only interfering updates of the first type occur, the additional simUpdate notifications are in fact redundant. However, this is acceptable to guarantee that no updates are lost between SimDB and ExtDB. In case of interfering updates from other clients to ExtDB, additional extUpdate (instead of simUpdate) notifications are emitted. Here, notifications need not be stored as the first extUpdate notification is simply interpreted as the expected receipt and subsequent extUpdates yield normal state transitions. Finally, the same store-and-reinsert strategy is used similarly in the other use cases mentioned above (object insertions, link removals, and object or link loading).

Another example is a pending link insertion from SimDB to ExtDB (sim2extInsertPending) in Figure 13. After starting the transaction's execution, the link may concurrently be removed and reinserted into SimDB several times by different components of the simulation system. It may also be inserted into ExtDB from a third party client. In this case, sim2extInsert's execution will have no further effect. Subsequently, the link may even be removed and reinserted again within ExtDB. However, after the execution process, all corresponding (pos-

TABLE III. EXAMPLE OF A LOCAL INTERFERING UPDATE OF SOME INTEGER PROPERTY WITHIN A SINGLE SIMDB.

#	action	state machine	SimDB val.	ExtDB val.	notification
1	(initial state)	Synced	10	10	
2	update 10 → 20 in SimDB		20		simUpdate (a)
3	process event simUpdate (a)	→ UpdatesPending / sim2extUpdatePendingPr _i			
4	start resync	inExec := true			
5	update 20 → 30 in SimDB (1st interference)		30		simUpdate (b)
6	read current value from SimDB				
7	update 30 → 40 in SimDB (2nd interference)		40		simUpdate (c)
8	execute transaction sim2extUpdate			30	extUpdate
9	process event simUpdate (b)	[inExec=true] ⇒ store simUpdate (b)			
10	process event simUpdate (c)	[inExec=true] ⇒ store simUpdate (c)			
11	process event extUpdate	→ UpdatesPending / SyncedPr _i → Done inExec := false reinsert simUpdate (b) in event queue reinsert simUpdate (c) in event queue			allUpdatesSynced
12	process event allUpdatesSynced	→ Synced			
13	process event simUpdate (b)	→ UpdatesPending / ext2simUpdatePendingPr _i			
14	process event simUpdate (c)	(self-transition)			
15	start resync			

sibly queued) events are processed until the receipt arrives. As the first extInsert notification is the receipt, no other extInsert and thus no extRemove notification can precede it. Hence, only simRemove and simInsert events have to be captured. They represent the current state of the link within SimDB. When the extInsert event arrives, these captured SimDB notifications are reinserted into the state machine's event queue just after the extInsert receipt. After the extInsert triggers a transition to the Synced state, the reinserted notification events are processed regularly. Note that it is irrelevant whether the link insertion in ExtDB actually originates from the successful execution of the sim2extInsert transaction or from a third party's operation. In both cases, the link is established and an extInsert notification is produced.

I. An Interim Conclusion

Regarding change tracking and conflicts, dependencies between different operations could also be a problem. Inserting an object usually also causes a parent link's insertion and removing an object cause a removal of all corresponding links and all (hierarchically) descendant objects. However, all these operations – dependent or not – cause the same kind of notifications and all interdependencies are resolved by the respective database itself. Thus, such dependent changes can be treated by the standard mechanisms and need no special handling.

Altogether, as mentioned above, this approach cannot avoid or fix conflicts but only detect them and react on them. However, the utilized SimDB and ExtDB themselves are not corrupted as they provide safe standard database access methods. Thus, only the distributed synchronization state must be kept free of corruptions. This is ensured by the presented approach.

J. Resynchronization

In resynchronization, all scheduled transactions are executed to bring the two databases back in sync. This process can be triggered in several ways. When the approach is applied in a collaborative scenario, it can be initiated manually. For

immediate response from and to other users, it can also be automatically triggered after each transition to a state with pending transaction. In distributed simulation, typical access patterns include constantly repeated changes of the same few property values, e.g., a moving car and a moving helicopter. In such scenarios, transactions can be aggregated within short but arbitrary periods to lower the impact on traffic. However, this includes a trade-off between traffic and update rate.

Transactions are processed in groups of the same type to gain advantage from dependencies and optimize execution performance. An overview is given in the list below. At first, all ext2simRemove and sim2extRemove transactions for objects are executed to remove deprecated objects. As an optimization measure, this is done in reversed order of occurrence of the corresponding notifications. System requirements for the notification services specify the removal of object subtrees to be notified in bottom-up order. Additionally, subtrees may be removed manually in terms of further subtrees piece by piece. Thus, the transaction for a removal of the ExtDB root object of a completely removed subtree will always be preceded by all its descendant objects. Processing transactions in reversed order, the corresponding root object within SimDB will be deleted first and its instance mapping entry is removed. Due to the semantics of composite aggregation, this deletion will recursively delete all SimDB descendant objects in one operation (typically optimized by SimDB). Pending ext2simRemove transactions for descendant objects receive a notification for the removal of their target object, which serves as a receipt triggering a transition to the NonManaged state.

- 1) ext2simRemove and sim2extRemove for objects (in reverse order of occurrence)
- 2) ext2simInsert and sim2extInsert for objects
- 3) ext2simChange and sim2extChange (for objects)
- 4) ext2simRemove and sim2extRemove for links
- 5) ext2simInsert and sim2extInsert for links

Next, all ext2simInsert and sim2extInsert transactions for object insertions are processed. For each ext2simInsert the standard loading mechanism is used, including replicating

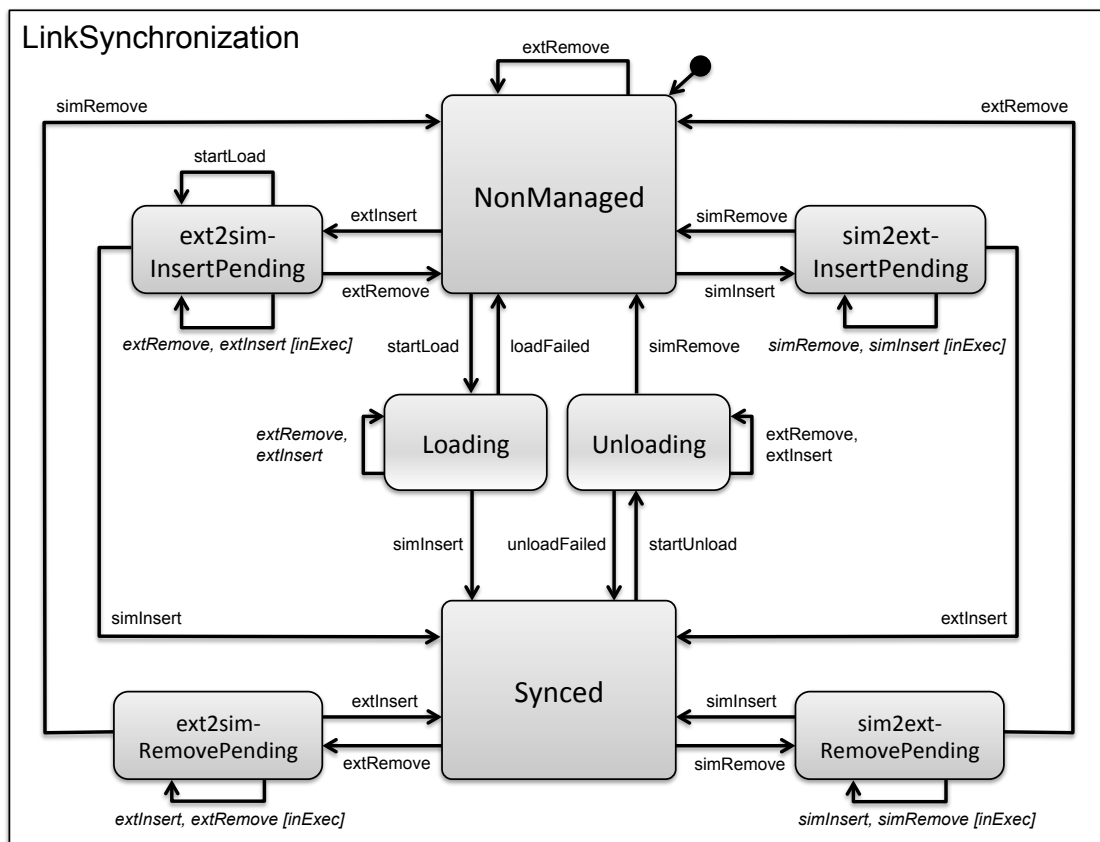


Figure 13. Synchronization states of a link's instance mapping.

current property values from the corresponding slots. Loading an object also includes loading all the object's ancestors to provide parent-child-relations within SimDB. Upon their loading, a **startLoad** event will be emitted to provoke a transition into the Loading state (for objects and parent links). Only for the object to be inserted itself and its parent link this event will be ignored. **sim2extInsert** transactions for objects make new objects globally available within ExtDB. The necessary operations can be seen as the counterpart to the loading operations.

Performing all object removals and insertions first assures further link operations are executed optimally. Beforehand, all **ext2simUpdate** and **sim2extUpdate** transactions are executed applying the current property values from ExtDB to SimDB and vice versa. In the next step, links are removed from SimDB and ExtDB by executing all corresponding **ext2simRemove** and **sim2extRemove** transactions. As with dependent object removals, each link already removed receives a receipt notification causing a transition to the NonManaged state. All remaining pending link removals are executed normally.

Finally, all **ext2simInsert** and **sim2extInsert** transactions are executed. The former are treated like their counterparts for objects by loading the corresponding link into SimDB using standard mechanisms. For parent links previously inserted in conjunction with an object insertion, receipt notifications may already have triggered a transition to the Synced state. As with link removals, all other link insertions are executed normally. When a designated link lacks a member object within the target

database, the transaction will (detectably) fail to execute, reset the **inExec** flag, and stay in pending state. This may occur although object insertions are executed before link insertions. For example, an object and a link may be inserted while the resynchronization process has already executed all other object insertions. Likewise, a member object may be removed in the target database. However, the next resynchronization cycle will take care of this.

VI. APPLICATIONS

Using the presented approach, different kinds of applications have already been realized.

A. City and Urban (Distributed) 3D Simulation

An application from the field of geo information systems are city simulations. Nowadays, a 3D city model exists for many cities in Germany like Dusseldorf or Stuttgart. Typically, these are often maintained by urban authorities, e.g., by land-registry. A widespread data model is the CityGML [29] standard, which is based on the Geography Markup Language (GML). A problem is to access, display, and use these huge data sets efficiently. Most tools can either access data in the original, highly semantic CityGML format but cannot efficiently display its 3D content or even do 3D simulation. Others can only efficiently use derived and optimized geometric representations, e.g., in VRML (Virtual Reality Modeling Language), which lack the semantic richness and need to be created by offline conversion. Using the presented approach

however, city data can be accessed in its original highly semantic schema making it available to a real-time capable 3D simulation system at the same time.

Prototypical 3D simulation scenarios include driving cars (with realistic physical correct behavior) or flying a helicopter through different city models (Figure 14). At the same time, the associated descriptive data can be accessed as the connection between geometry and object is never broken. As the proposed approach has read and write support (compare Section V), city data can even be updated from within the simulation system. In the context of CityGML, functional data synchronization has been used to supplement so-called implicit geometry representations with the corresponding referenced detail model. This was used for street furniture like street lights or benches or vegetation like trees. The implicit representation therefore contains references to external model files (e.g., in VRML format), which are subsequently loaded into the simulation system. For vegetation it may also just denote a tree type. Additionally, pose and size information are given, which are applied to the supplement.



Figure 14. Bringing large city models to life by combining database and simulation technology (data: Dusseldorf).

In another urban scenario, the presented methods have been used as the basis for a new approach to distributed 3D simulation using a central database. An important precondition is that the shared simulation model within ExtDB not only contains static model data like buildings, street furniture or vegetation. It also has to contain dynamic objects like a car or helicopter. A locally running simulation changes these dynamic objects (e.g., the cars position) within SimDB. Using the mechanisms described in Section V, these changes are synchronized to ExtDB, hence communicating the new state of the simulation model to the shared model. The database's notification service actively notifies any subscribed simulation client. Each client adopts the changes by synchronizing their own local replicate copies accordingly. All in all, this realizes a database-driven distributed 3D simulation.

In training scenarios using 3D simulation techniques – like driving or flying a virtual vessel or operating a virtual model of a machine – there is an interest in recording a simulation run. Recorded data can be used for replay, analysis, debriefing,

and archival. This may be realized using external tools that log all interactions between the participants of the distributed simulation. Here, additional tools are needed to replay these logs. In the presented database-driven approach however, all changes are centrally routed through the shared model within ExtDB. Thus, as all intermediate states of the simulation are made persistent in the shared model, a simulation run can easily be captured by using a temporal database for ExtDB. 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.

Using these two core concepts, a database-driven distributed 3D simulation application has been prototypically realized (Figures 15–16). The scenario consists of a heterogeneous shared model with a generic village in CityGML format as static model data and SEDRIS-based [25] car and helicopter models as dynamic model data. Data is loaded from a central database into two attached VEROSIM simulation clients. Here, simulation specific data is reconstructed using functional data synchronization. In particular, this includes relative transformations, which are synchronized bidirectionally to distribute movements of the dynamic objects. Dynamic parameters of the simulation model like physical characteristics of a car's drive were also encoded within the SEDRIS schema and reconstructed as supplementing structures in SimDB. However, these active components of the simulation are specially treated to configure responsibilities. I.e., the car shall only be actively simulated by one simulation client and only passively retraced within the other. The same applies to the helicopter vice versa. An id-based approach is used to configure these active and passive components per client. After replicating the data, the simulation is started in each client. Movements in one client are resynchronized to the central database and distributed to the respective other client. At the end of the simulation, the aforementioned usage of a temporal database allows an off-line replay of the simulation run.

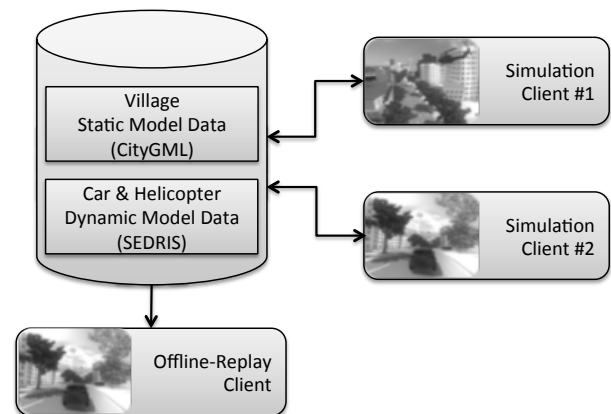


Figure 15. Architecture of the database-driven distributed 3D simulation application.

A performance evaluation yielded 0.7 – 1.3ms for single reading and 2.3 – 4.6ms for single writing operations (note that writing includes the versioning mechanism), which is acceptable for the given task. The central database uses a polling-based notification approach. It worked well but reached



Figure 16. The presented approach is used to actively drive a distributed simulation scenario.

its limits under heavy load making a push-based approach more advisable. For more details see [3].

Usually, such applications use amounts of files for data management combined with a decentralized communication infrastructure, e.g., based on the High Level Architecture (HLA) [30], and separate logging components are needed to archive a simulation. In contrast, we provide a more integrated approach. This avoids divergence between data management and the corresponding change distribution mechanism, no separate mechanism is needed to access logged data, and a consistent data schema provided by the central database is used throughout the distributed system.

B. Virtual Space Robotics Testbed

So-called Virtual Testbeds follow a holistic approach to 3D simulation. In contrast, conventional 3D simulation applications typically use very specialized techniques and focus on certain details. Virtual Testbeds however incorporate a diversity of problem aspects and their interactions into a single application, which integrates all relevant objects and parameters, their environment, and documentation. Similar to concepts for Product Data Management (PDM) systems [31], [32], they are also used in a much broader scope from design to testing, production and training. The Virtual Space Robotics Testbed (Figure 17) integrates the combined efforts of the research projects FastMap [33], [34], SELOK [35], [36], and Virtual Crater [37]. It provides means for the development, testing, and evaluation of robotic systems for space missions.

This Virtual Testbed's purpose is to support the development of future space missions. The three research projects pursue different aspects of this very same goal. The major goal of the research project FastMap is the automatic generation of navigation maps from images taken during the descent phase of a planetary landing mission. To calibrate the Virtual Testbed – in particular for camera and lighting – a physical planetary landing mockup is used. It consists of scaled realistic surface models and two robot arms carrying a light source and a camera. The flexibility of the Virtual Testbed even allows for its usage as a control system of this facility. The Virtual Testbed contains a model of a planetary descent scenario and additionally a virtual model of the physical mockup. All three scenarios (virtual mission, virtual mockup and physical mockup) are driven by the very same software components.

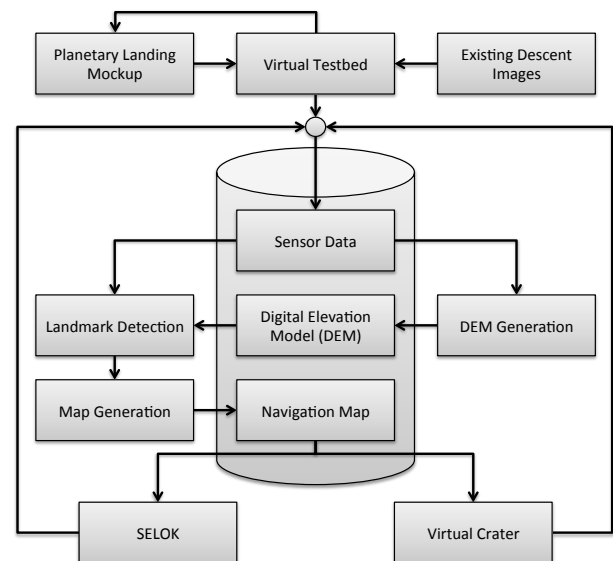


Figure 17. Structure of the projects FastMap, SELOK, and Virtual Crater in the Virtual Space Robotics Testbed.

A central database is used to manage a shared data model with a GML-based application schema. The central active database is used to make the shared model persistent and to communicate changes between the clients while deriving the navigation map. Hence, in this scenario, the approach is used for distributed data processing. The state of this distributed process is reflected by a central *mission* object holding a phase indicator. In each phase, a different client is responsible for finishing the phase (by incrementing it). Phase changes and all other data modifications including the insertion of new objects are actively communicated to all clients to drive the distributed approach.

In the first phase, a client (using the approach presented in this paper) processes sensor data, which can either stem from the physical mockup's camera or from one of the virtual scenarios (Figure 18). For each of these captured images, an object with the image's data is stored in the central database. When all images are captured, the phase is incremented. However, following a pipelining approach, as soon as the first image objects are stored in the database, the second client is notified and fetches these objects. A digital elevation model (DEM), i.e., a 3D model of the planet's ground, is constructed. Subsequently, DEM fragment objects holding the rasterized elevation data are also stored in the central database. When all captured images are processed, the phase is once again incremented. This phase change triggers the third client. Based on the images and the DEM, it applies different detection algorithms to create a map of landmarks like craters, rocks, or mountain tops. These landmarks are also stored in objects of the corresponding classes of the FastMap application schema. To end the distributed data processing, the detector client increments the phase of the mission for a last time.

Subsequently, the completed map is used as a basis for self-localization and navigation of mobile robots during the exploration of the planetary surface (Figure 19). This is the subject of the research project SELOK. Here, mobile robots use sensors like stereo cameras or laser scanners to capture

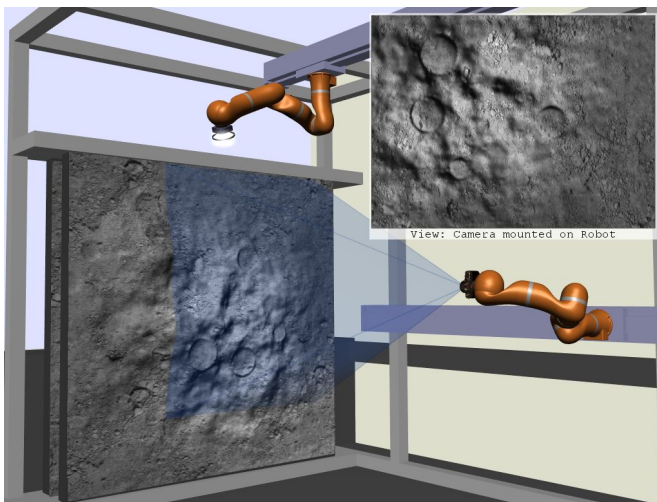


Figure 18. A simulation of a physical mockup for a planetary landing mission (project FastMap). The presented approach provides access to the shared world model.

their surroundings. The Virtual Testbed allows to flexibly combine and interchangeably use simulated and physical sensors. Within the data, landmarks like the aforementioned rocks or craters are detected. These so-called local landmarks are then matched against the global landmarks from the navigation map yielding a self-localization of the robot.

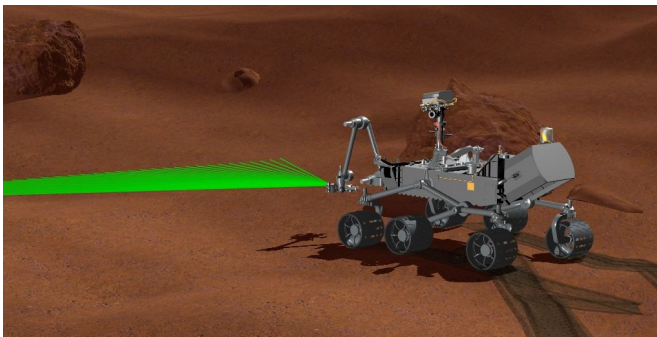


Figure 19. Planetary localization and navigation using the maps extracted after the landing phase. The presented approach is used to access the shared world model.

Finally, the third research project Virtual Crater is concerned with the development of a Virtual Testbed for mobile robots with planetary exploration missions. The testbed allows for a cost-efficient and realistic simulation of mobile robots in a virtual lunar environment. Here, they can be developed, programmed, tested, and optimized. Furthermore, the project comprises a physical testbed to compare and verify simulation results with reality. The results of Virtual Crater can be complemented by the navigation map and DEM from FastMap the self-localization methods from SELOK.

As this scenario provides similar conditions for the integration of the presented approach, evaluation results from Subsection VI-A are accordingly applicable.

C. Forest Inventory, Management and Simulation

In the field of forestry, the approach has successfully been employed for forest inventory, management and simulation. Most of the works were realized in the context of the research project Virtual Forest [38], [33], [39]. One of the core ideas of this project is a consistent, shared data model and data management in the Virtual Forest database (Figure 20). Provided to all stakeholders in this field, it facilitates the exploitation of know-how and synergies. Furthermore, it supports the transfer of industrial automation techniques to the forest industry.

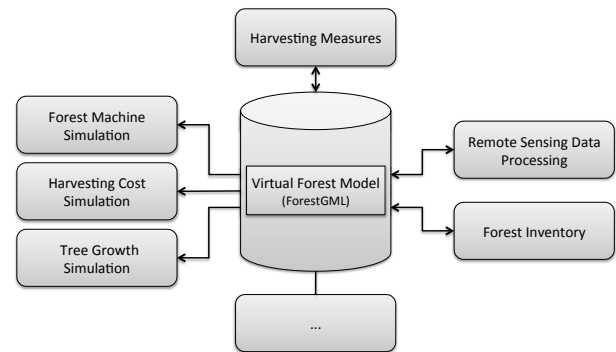


Figure 20. Architecture of the Virtual Forest scenario.

Integrating the database and all applications, the Virtual Forest constitutes a 4D geo-information system (GIS). It combines an object-oriented, semantic world modeling with 3D geo data with the fourth dimension of time in terms of a temporal database and simulation techniques. While the temporal database is used to preserve and access previous states, different simulation techniques are provided for prediction of future scenarios.

Forest inventory is the acquisition and management of environmental data in forestry. In the Virtual Forest, this is realized using a semi-automatic approach. Based on remote sensing and other data, algorithms for tree species classification, single tree delineation and attribution, and forest stand delineation and attribution are used to automatically process huge amounts of data. The results of these semantic world modeling processes are stored in the common Virtual Forest model. Expert users can then use them for quality inspection and data refinement. To improve their work, another important aspect are convenient user interfaces to algorithms and supportive tools (Figure 21).

Based on inventory data, other applications in the forestry context can be driven. There are simulation applications for decision support to predict harvesting costs or forest growth. Driver assistance or 3D simulations can be used to support and to train the usage of harvesters (see Figure 22) and forwarders in real world environments. Techniques for the automated feedback from harvesting allows for a permanent inventory. Here, results from felling measures are transferred back into the central model to keep it up-to-date. User-friendly query interfaces and reporting tools can help to evaluate and interpret the shared forest model. All in all, several applications for processing remote sensing data, semi-automatic inventory, simple forest information, forest planning, and wood production planning, and a Virtual Testbed have been developed. Furthermore, web technologies like portals and services provide the model to even more users.

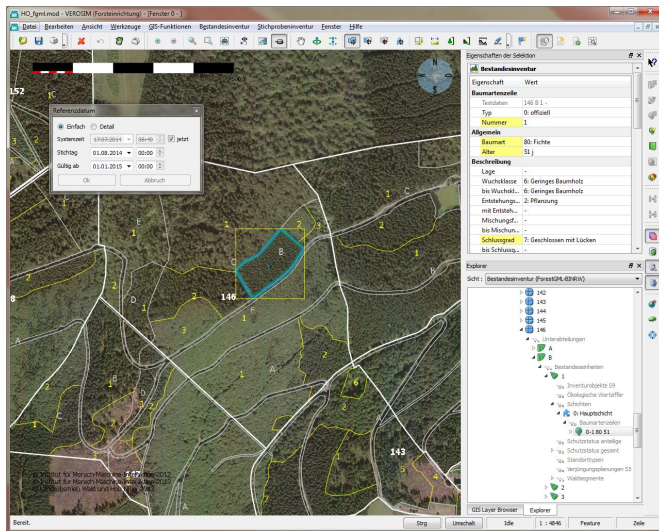


Figure 21. A forest inventory tool accessing a database with a detailed forest stand model.

In particular, the inventory tools have been used and tested in actual measures. In this context, they were evaluated for their real-world applicability – implicitly evaluating the presented approach utilized by these tools as well. Users attested the tools a very good performance and stability.



Figure 22. The presented approach is used for a harvester simulator to access the database managing the highly detailed forest model.

The 4D-GIS is realized using the simulation system VEROSIM and a geo database. The systems are combined by the presented prototype. Typically, inventory data is structured hierarchically to reflect, e.g., administrative units and contains spatial data. The presented approach allows multiple users to collaboratively work with a central database.


The data models employed for the Virtual Forest are all based on GML. A specialized base schema called ForestGML was developed to provide core data constructs to facilitate the Virtual Forest's usage in different areas, states, or administrations. Furthermore, GML as a base schema allows a standardized exchange of data using the corresponding standard web services like WFS (Web Feature Service) or WMS (Web Map Service). This guarantees interoperability between different clients connected to the Virtual Forest database.

VII. CONCLUSION AND FUTURE WORK

We detailedly presented an approach for synchronizing a central database (ExtDB) with simulation databases (SimDB) as a basis for database-driven 3D simulation. After recapitulating our previously published background of the approach, the main contribution of this work is presented: A detailed description of the core method for distributed database synchronization. For each pair of master and replicate copy it manages the state of synchronization – modeled as a state machine. It is based on notifications provided by both databases. On the one hand, they are used to track the changes and schedule transactions for subsequent resynchronization. On the other hand, they are used as receipts to acknowledge transaction execution and to detect change conflicts. Compared to other methods for collaboration in 3D software systems, this approach provides a tight integration of advantages from the database field into simulation technology. To prove its practicability, examples of use from three different fields of application are presented in detail.

In future, we will examine further applications, e.g., from the field of industrial automation. Moreover, a porting of the approach to other database systems than the current prototypes will be reviewed. Finally, the integration of temporal databases will be examined in further detail, especially for valid time, bitemporal, or multi-temporal databases.

ACKNOWLEDGMENT

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.

REFERENCES

- [1] M. Hoppen and J. Rossmann, "A Database Synchronization Approach for 3D Simulation Systems," in DBKDA 2014, The 6th International Conference on Advances in Databases, Knowledge, and Data Applications, A. Schmidt, K. Nitta, and J. S. Iztok Savnik, Eds., Chamoniex, France, 2014, pp. 84–91.
- [2] J. Rossmann, M. Schluse, R. Waspe, and M. Hoppen, "Real-Time Capable Data Management Architecture for Database-Driven 3D Simulation Systems," in Database and Expert Systems Applications - 22nd International Conference, DEXA 2011, A. Hameurlain, S. W. Liddle, K.-D. Schewe, and X. Zhou, Eds. Toulouse, France: Springer, 2011, pp. 262–269.
- [3] 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.
- [4] M. Hoppen, M. Schluse, and J. Rossmann, "A metamodel-based approach for generalizing requirements in database-driven 3D simulation (WIP)," in Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium, ser. DEVS 13. San Diego, CA, USA: Society for Computer Simulation International, 2013, pp. 3:1–3:6.

- [5] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*, ser. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2012.
- [6] OMG, “Unified Modeling Language (UML).” [Online]. Available: <http://www.uml.org> 2014.12.01
- [7] M. Hoppen, M. Schluse, and J. Rossmann, “Database-Driven 3D Simulation - A Method Specification Using The UML Metamodel,” in 11th International Industrial Simulation Conference ISC 2013, V. Limère and E.-H. Aghezzaf, Eds., Ghent, Belgium, 2013, pp. 147–154.
- [8] E. V. Schweber, “SQL3D - Escape from VRML Island,” 1998. [Online]. Available: <http://www.infomaniacs.com/SQL3D/SQL3D-Escape-From-VRML-Island.htm> 2014.12.01
- [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] 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.
- [11] 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.
- [12] 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.
- [13] 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.
- [14] G. Reitmayr, S. Carroll, A. Reitemeyer, and M. G. Wagner, “DeepMatrix - An open technology based virtual environment system,” *The Visual Computer*, vol. 15, no. 7-8, Nov. 1999, pp. 395–412.
- [15] 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.
- [16] 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.
- [17] P. A. Bernstein and S. Melnik, “Model management 2.0: manipulating richer mappings,” in Proceedings of the 2007 ACM SIGMOD International Conference on Management of data - SIGMOD ’07. New York, New York, USA: ACM Press, Jun. 2007, pp. 1–12.
- [18] P. Atzeni, P. Cappellari, and P. Bernstein, “Model-Independent Schema and Data Translation,” in *Advances in Database Technology - EDBT 2006*, ser. Lecture Notes in Computer Science, Y. Ioannidis, M. Scholl, J. Schmidt, F. Matthes, M. Hatzopoulos, K. Boehm, A. Kemper, T. Grust, and C. Boehm, Eds. Springer Berlin / Heidelberg, 2006, vol. 3896, pp. 368–385.
- [19] P. Atzeni, L. Bellomarini, F. Bugiotti, and G. Gianforme, “A runtime approach to model-independent schema and data translation,” in Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, ser. EDBT ’09. New York, NY, USA: ACM, 2009, pp. 275–286.
- [20] A. Smith and P. McBrien, “A Generic Data Level Implementation of ModelGen,” in *Sharing Data, Information and Knowledge*, ser. Lecture Notes in Computer Science, A. Gray, K. Jeffery, and J. Shao, Eds. Springer Berlin / Heidelberg, 2008, vol. 5071, pp. 63–74.
- [21] P. Berdager, A. Cunha, H. Pacheco, and J. Visser, “Coupled Schema Transformation and Data Conversion for XML and SQL,” in *Practical Aspects of Declarative Languages*, ser. Lecture Notes in Computer Science, M. Hanus, Ed. Springer Berlin / Heidelberg, 2007, vol. 4354, pp. 290–304.
- [22] R. M. Fujimoto, “Parallel and distributed simulation,” in Proceedings of the 31st conference on Winter simulation Simulation—a bridge to the future - WSC ’99, vol. 1. New York, New York, USA: ACM Press, Dec. 1999, pp. 122–131.
- [23] K. S. Perumalla, “Parallel and distributed simulation: traditional techniques and recent advances,” Dec. 2006, pp. 84–95.
- [24] R. Elmasri and S. B. Navathe, *Database Systems: Models, Languages, Design, And Application Programming*, 6th ed. Prentice Hall International, 2010.
- [25] SEDRIS, “SEDRIS.” [Online]. Available: <http://www.sedris.org> 2014.12.01
- [26] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, “ATL: A model transformation tool,” *Science of Computer Programming*, vol. 72, no. 1-2, Jun. 2008, pp. 31–39.
- [27] T. Connolly and C. Begg, *Database systems: a practical approach to design, implementation, and management*, 5th ed. Pearson Education (US), 2009.
- [28] D. Harel, “Statecharts: a visual formalism for complex systems,” *Science of Computer Programming*, vol. 8, no. 3, Jun. 1987, pp. 231–274.
- [29] CityGML, “CityGML.” [Online]. Available: <http://www.citygml.org> 2014.12.01
- [30] Simulation Interoperability Standards Committee (SISC), “Standard for Modeling and Simulation High Level Architecture (HLA) IEEE 1516,” 2000.
- [31] Verein Deutscher Ingenieure (VDI), “VDI 2219 - Information technology in product development Introduction and economics of EDM/PDM Systems (Issue German/English),” Düsseldorf, 2002.
- [32] U. Sandler, *Das PLM-Kompodium: Referenzbuch des Produkt-Lebenszyklus-Managements (PLM compendium: reference book of product lifecycle management)*. Berlin: Springer, 2009.
- [33] J. Rossmann, M. Schluse, R. Waspe, and R. Moshammer, “Simulation in the Woods: From Remote Sensing based Data Acquisition and Processing to Various Simulation Applications,” in Proceedings of the 2011 Winter Simulation Conference, S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, Eds., 2011, pp. 984 – 996.
- [34] J. Rossmann, T. Steil, and M. Springer, “Validating the Camera and Light Simulation of a Virtual Space Robotics Testbed by Means of Physical Mockup Data,” in 11th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS), 2012, pp. 1–6.
- [35] J. Rossmann, C. Schlette, M. Emde, and B. Sondermann, “Discussion of a Self-Localization and Navigation Unit for Mobile Robots in Extraterrestrial Environments,” *Artificial Intelligence*, 2010, pp. 46–53.
- [36] J. Rossmann, B. Sondermann, and M. Emde, “Virtual Testbeds for Planetary Exploration: The Self-Localization Aspect,” in 11th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA), 2011, pp. 1–8.
- [37] Y.-H. Yoo, T. Jung, M. Langosz, M. Rast, J. Rossmann, and F. Kirchner, “Developing a Virtual Environment for Extraterrestrial Legged Robots with Focus on Lunar Crater Exploration,” in The 10th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS), 2010, pp. 206–213.
- [38] J. Rossmann, M. Schluse, and A. Bücken, “The virtual forest - Space- and Robotics technology for the efficient and environmentally compatible growth-planing and mobilization of wood resources,” *FORMEC 08 - 41. International Symposium*, 2008, pp. 3 – 12.
- [39] J. Rossmann, M. Hoppen, and A. Bücken, “Semantic World Modelling and Data Management in a 4D Forest Simulation and Information System,” *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XL-2/W2, 2013, pp. 65–72.