A Database Synchronization Approach for 3D Simulation Systems

Martin Hoppen, Juergen Rossmann

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

Abstract—**Equipping a three-dimensional (3D) simulation system with database technology provides many advantages: Simulation models can be managed more efficiently than with files, temporal databases can be used to log simulation runs, and active databases provide a means for communication. Thus, we use a central database to share simulation models from different fields of application from space missions to forestry. To enable real-time access, each simulation client caches the model to its local runtime simulation database. For that purpose, each pair of databases must be synchronized. After a synchronization on schema level, each client replicates data on-demand. In this publication, we present an approach that uses both databases' notification services to keep master copies in sync with their replicate copies. State machines are used to model the approach.**

***Keywords*–Database Synchronization; 3D Simulation; Distributed Database.**

I. INTRODUCTION

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 field 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 [1], 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 local, real-time capable runtime database. 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. Its specification however 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 [2]. They incorporate methods adopted from Model-Driven Engineering (MDE) [3] and allow to use the concepts of the Unified Modeling Language (UML) to give generalized method specifications for the different components of the overall approach [4]. Thus in this paper, 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 rest of this paper is organized as follows: In Section II, the foundations of the database-driven approach for 3D simulation are recapitulated. Section III summarizes the system requirements and the applied approach for method specifications using the UML metamodel. Both sections pave the way for the main Section IV where we present the notification-based synchronization approach. In Section V, exemplary applications are shown and Section VI presents some work related to our own. Finally, in Section VII, we conclude our work and present some future work.

II. 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 [5] 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 [5] 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.

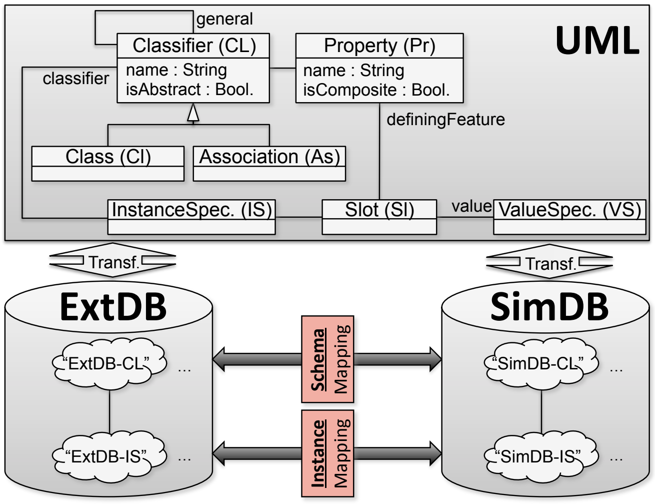

Figure 1. Principle structure of the approach for database-driven 3D simulation.

The two databases are synchronized on schema and data level. During the former, the schema description is transfered 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 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 IV.

III. System Requirements

To generalize the approach system requirements were identified [2]. The aim is to make it universally available for different implementations of ExtDB and SimDB. A general compatibility of the two databases' modeling concepts is stipulated using both their metamodels. A database's metamodel represents its abstract syntax (modeling concepts). Their compatibility can then be expressed with a model transformation, e.g., using the ATL Transformation Language (ATL) [6].

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 [5]. 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.

IV. Notification-based Database Synchronization

Following the definition in [5], the presented scenario, i.e., the combination of SimDB and ExtDB, would be a distributed database (DDB). Similar to a distributed database management system (DDBMS), our approach aims at transparency of the distribution. 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. In contrast, a classical DDB is accessed as a whole from the outside and the DDBMS hides away its distributive nature. 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 [4]. 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 ore 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. Differences in between such a pair are resynchronized periodically but not synchronously. Thus, we have a similar scenario as described in [7] 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.

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 [5]. 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.

We therefore 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) [7]. 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.

A. Change Tracking

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. 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 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. 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.

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.

Altogether, instance mapping entries (i.e., pairs of master and replicate copy) can be seen as to reside in a certain state of synchronization. This can be modeled as a state machine in statechart notation [8] for each object's or link's instance mapping entry. For objects, this state machine is given in Figure 2 (it is similar for links). It may be in a synchronous state (*Synced*), a *Loading* or *Unloading* state, a state representing its absence or non-management (*NonManaged*), or a transaction state (*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 5.

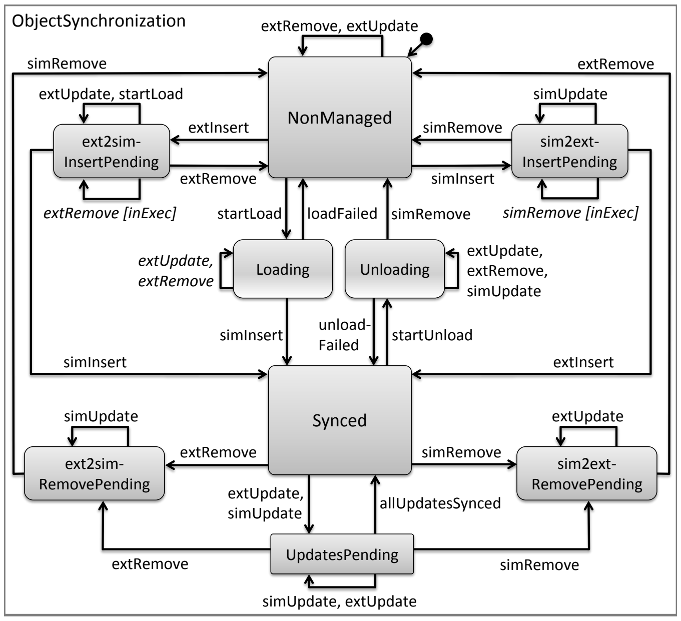

Figure 2. Synchronization states of an object's instance mapping.

An exemplary chain of events depicted in Figure 3 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 4. Here, a sim2extInsert transaction is scheduled for the new object. When the transaction is executed (see Subsection IV-C), 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.

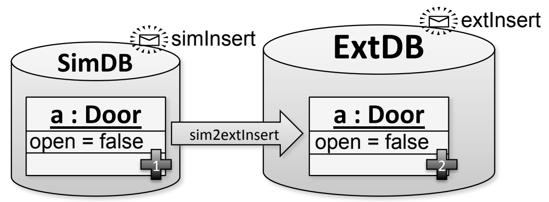

Figure 3. Exemplary insertion of a door object into SimDB and subsequent synchronization to ExtDB using a sim2extInsert transaction.

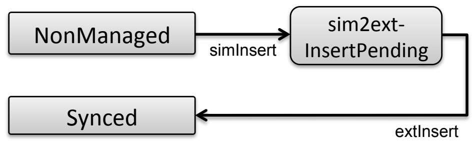

Figure 4. Excerpt from Figure 2 for the state transitions accompanying the exemplary insertion depicted in Figure 3.

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.

Property changes are modeled in Figure 5. The UpdatesPending state encapsulates a sub state structure for managing property updates. 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$_i$). 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), 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 IV-B). 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 changes. 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 2).

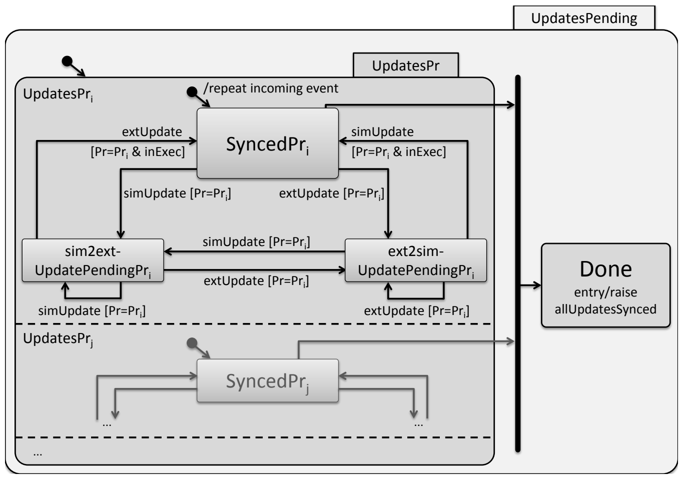

Figure 5. Sub structure of state UpdatesPending from Figure 2 for property updates.

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.

B. Change Conflict Handling

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.

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 IV-C). 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 $T1$ commits. So, while native transactions of the utilized database management systems (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 IV-C).

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.

One example are updates (Figure 5). A property's update transaction can be examined separately as updates of different properties are independent from each other. Table I lists an exemplary sequence of events for some integer property and the associated actions, statemachine 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 statemachine'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 statemachine 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

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

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

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).

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.

C. 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.

V. APPLICATIONS

Using the presented approach, different kinds of applications have already been realized as shown in Figure 6.

In a city scenario, a central database (ExtDB) manages a shared simulation model with a city, a helicopter, and a car. Two simulation clients are connected with their respective synchronized SimDB and each control a vehicle. Changes (e.g., the movement of the car or the helicopter) are distributed using the methods presented in this paper. Furthermore, all changes are automatically archived using a temporal ExtDB. This provides an integrated log for the development of the simulation model's dynamic properties over time and allows for subsequent replay, analysis, debriefing, and archiving. 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) [9], 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.

In another scenario, a planetary landing mission is simulated. During descent, a database-managed, shared model of the planet's surface (i.e., an object-oriented map) is created by different components in a distributed approach. Subsequently, the same map can be used for (simulated) navigation. All system components benefit from using and building up the same shared model with a consistent schema, standardized interfaces, and an integrated communication infrastructure using the presented approach.

As a last example, a forest model is extracted from remote sensing data and other geo data sources. Here, the approach is used by the various stakeholders in the forest sector to collaboratively generate, update, refine, analyze, simulate, and

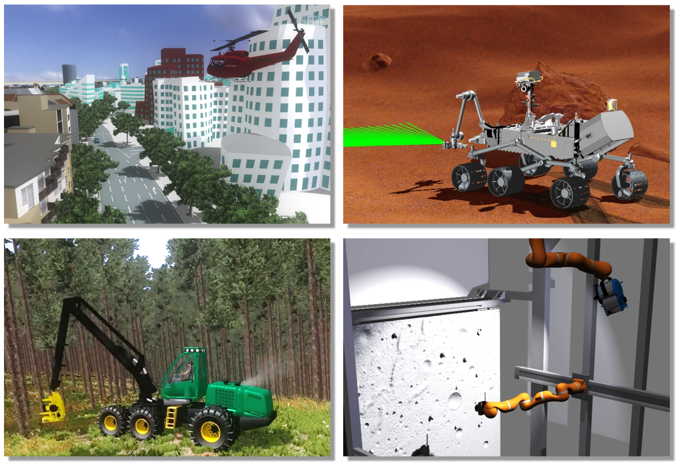

Figure 6. Different applications realized using the presented approach.

simply use the highly detailed forest model managed by an ExtDB component. Instead of directly accessing this central database, the presented approach decouples clients from the utilized technology of ExtDB by only accessing data from their local SimDB database. Furthermore, the very same data schema can be used throughout the applications reducing ”friction losses” due to (offline) data conversions.

VI. Related Work

Regarding database synchronization for 3D simulation systems and similar software only few approaches can be found. In [10], 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 [11]. 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 [12], [13], a Virtual Reality (VR) system is combined with an OODBMS to provide VR as a multi-modal database interface. In [14], 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 commited 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 [15] is based on the multi-user Virtual Environment system DeepMatrix [16], extended by a relational DBMS back-end providing persistency. Clients allow for so-called shared operations like ”rotate object” that are send 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 [17], 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 [18]. 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, [10] comes close but lacks details and is only a proposal without known implementations. The ghosts in [11] may suffice for rendering but are to restricted for sophisticated simulation applications. Furthermore, not all objects are managed by the database. In [12], [13], [14], [17], only VR-specific data and operations are supported. [15] does not manage the model data itself using the database. Finally, the approach in [18] 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. [19] 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 [20]. A pivotal supermodel is used to transform schema as well as data. In [21], the same system is extended to provide runtime transformations with read-only access. A similar approach is taken in [22] using a proprietary pivotal graph-based representation. [23] 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.

VII. Conclusion and Future Work

We 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 in detail: The core method for 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. Different applications already prove its practicability.

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.

REFERENCES

[1] 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.

[2] 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.

[3] M. Brambilla, J. Cabot, and M. Wimmer, Model-Driven Software Engineering in Practice, ser. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2012.

[4] 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.

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

[6] 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.

[7] T. Connolly and C. Begg, Database systems: a practical approach to design, implementation, and management, internatio ed. Pearson Education (US), 2009.

[8] D. Harel, "Statecharts: a visual formalism for complex systems," Science of Computer Programming, vol. 8, no. 3, Jun. 1987, pp. 231–274.

[9] Simulation Interoperability Standards Committee (SISC), "Standard for Modeling and Simulation High Level Architecture (HLA) IEEE 1516," 2000.

[10] E. V. Schweber, "SQL3D - Escape from VRML Island," 1998. [Online]. Available: http://www.infomaniacs.com/SQL3D/SQL3D-Escape-From-VRML-Island.htm

[11] 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.

[12] Y. Masunaga and C. Watanabe, "Design and implementation of a multi-modal 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.

[13] 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.

[14] 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.

[15] 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.

[16] G. Reitmayr, S. Carroll, A. Reitemeyer, and M. G. Wagner, "Deep-Matrix - An open technology based virtual environment system," The Visual Computer, vol. 15, no. 7-8, Nov. 1999, pp. 395–412.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] P. Berdaguer, 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.