Distributed Information Processing and Rendering for 3D Simulation Applications

Martin Hoppen, Ralf Waspe, Malte Rast, and Juergen Rossmann

***Abstract*—The growing demand for 3D simulation techniques in various application domains leads to more and more specialized tools and complex frameworks. Between homogeneous or inhomogeneous clients, data has to be distributed and synchronized in centralized or decentralized setups. Hardware/Software-in-the-Loop and Co-Simulation are common tasks in virtual prototyping. Load balancing and parallelization is necessary for computationally intensive simulations. Spatially distributed developers and designers collaborate in networked virtual environments. All these different applications impose different requirements on the data distribution and synchronization mechanism. In this paper, we categorize distribution scenarios, their requirements and according synchronization techniques. Four different approaches with different key aspects are presented and compared by means of a reference implementation and several application examples. This overview shall enable the reader to choose the approach best suited for his particular distribution problem.**

***Index Terms*—Data management, distribution, simulation, synchronization.**

I. Introduction

Today, many 3D simulation applications demand some kind of distribution. Thus, synchronization between the different participating components is an issue. This can be realized in different ways. One standardized approach is the High Level Architecture (HLA) [1]. It is especially favorable for interoperability scenarios with different simulation systems. However, it is not the first choice for every type of synchronization problem.

In our daily work in the context of science and industry projects we identified four types of application scenarios that require different types of distribution and synchronization. This publication shall give an overview of these scenarios and the proposed solutions as well as a comparison in between them.

The first scenario is a 3D multi-projection Virtual Reality (VR) system. Here, a single VR scene is distributed onto multiple projection screens. A separate render and simulation client creates each image. The distribution mechanism between these clients must ensure a tight synchronicity between the screens. HLA however is not suitable for such tasks, since there is no central communication hub acting as a federation server and due to the lack of an automatically generated interchange model (FOM). Instead, we developed a simplified synchronization protocol with focus on speed.

The second scenario is a Co-Simulation. Here, two different simulation applications – both experts on their respective field – are interconnected to achieve an overall goal. This is a classic interoperability problem. Thus, HLA is a proper choice as the synchronization of two independent systems benefits from a standardized protocol.

Simulation applications based on huge 3D world models are another type. Such models, e.g., forest or city models are often managed in (geo) databases. Synchronization is needed when multiple simulation clients concurrently use and change such a model. One approach is to use a separate means for synchronization like an HLA RTI (runtime interface). This however carries the risk of divergence between data management (i.e., the simulation model in the database) and communication (RTI). Thus, an integrated approach for data management as well as communication should be favored. In [2], we propose such an approach where a central database is not only used to manage the shared simulation model but also to serve as an active communication hub.

Finally, a fourth type of simulation scenarios are simple interconnections between similar simulation clients. A "no-frills" peer-to-peer approach can be used to simply interchange some values. An example we realized would be the connection between a simulated planetary exploration vehicle and a ground control both realized with the same simulation software. In contrast, an HLA RTI would impose more efforts to integrate.

The rest of this paper is structured as follows: While Section II portrays the utilized reference simulation system, Section III contains an overview of each distribution approach presented in Section I. The approaches are compared in detail in Section IV and exemplary applications are given in Section V. Finally, after presenting related work in Section VI, we conclude and give a summary in Section VII.

II. Reference System

The major prerequisite on the tool level is the use of one single but comprehensive and integrated 3D simulation framework that is able to implement all the methods and support all the processes needed for an integrated prototyping, development and testing environment. The simulation system must support a broad range of applications and usage scenarios. As new usage scenarios require new data structures, the data model must be adaptable to new simulation models, even at runtime. Thus, a meta-data system and a reflection API (like the one available in Java) are necessary.

Manuscript received November 4, 2013; revised January 8, 2014.

The authors are with the Institute for Man-Machine Interaction of RWTH Aachen University, Germany (e-mail: {hoppen, waspe, rast, rossmann}@mmi.rwth-aachen.de).

DOI: 10.7763/IJCTE.2014.V6.870

Such a flexible database can then be used in all kinds of data storage and data manipulation scenarios, not only 3D simulation, but any other kind of simulation. This way, all methods can use the same model that contains (on an equal level) geometric information, as well as, e.g., sensor configurations or controller programs.

A. Model Representation

The tiered data model consists not only of the simulation model itself, but also incorporates a meta-model layer. The meta-model is essential for the flexibility as well as the developer and end user friendliness of the database and the simulation system. The design shown in Fig. 1 is inspired by the Object Management Group (OMG) meta-model hierarchy [3].

The middle layer describes the data model of the simulation. The key idea is the introduction of a micro kernel, the "Versatile Simulation Database" (VSD).

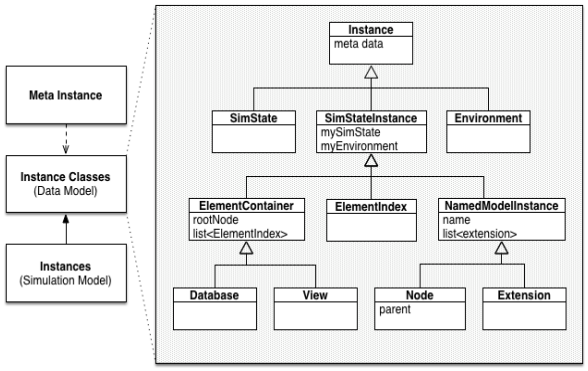

Fig. 1. The core of the "Versatile Simulation Database" (VSD).

In order to be able to retain semantic information and integrate data and algorithms into one single database, the VSD data model is an object oriented graph database [4], consisting of nodes and node extensions. A simplified class hierarchy of the VSD core is shown in Fig. 1. All nodes in the graph database, the database itself and even the simulation environment are derived from a single base class called "Instance". This base class provides mechanisms for inter-instance communication, as well as access to the meta-information system, which allows introspection of class hierarchy, properties and methods. The complete simulation state is held in properties of database nodes and/or node extensions. Due to the active nature of the VSD it is easy to obtain a map of changes of the simulation state starting from an arbitrary point in time, by listening to the messages emitted by the database for property changed, node creation and node deletion. All database entities have an integer identifier generated at runtime, which is unique within one instance of the simulation system.

The uppermost layer is the meta-information system, the basis for persistence, parallel and distributed simulation and communication. It mainly consists of meta-types, meta-instances, meta-properties and meta-methods. In addition to "built-in" classes, it is also possible to generate meta-instances with the corresponding meta-properties and meta-methods during runtime (e.g. for object oriented scripting or new data models). Such "runtime meta-instances" are treated in exactly the same way as the build in meta-instances without any performance overhead in the data management.

B. Micro Kernel Architecture

All simulation functionality of the framework is achieved by creating specialized add-ons, which build upon and interact with the VSD core. Following this approach, the database is able to integrate standard geometric models as well as block-oriented simulation models using input/output connections, the intermediate representation of scripting languages [5] or entirely different types of information like forest inventory data [6] (see Fig. 2).

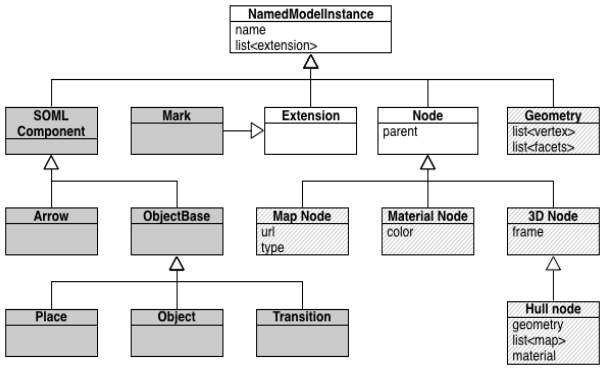

Fig. 2. Exemplary additional classes of the VSD.

The functionality of the micro kernel is extended by various plugins implementing simulation or data processing algorithms, interfaces to hard- or software systems, user interfaces, etc. Using the VSD, the plugins can communicate with the database as well as establish directed communications between themselves. One crucial point is the combination of, and the communication between, different simulation algorithms. This can be challenging in complex scenarios, which incorporate different application domains and require a mutual interaction between the different domains for realistic simulation results.

The basic simulation system architecture is now the basis for our implementation of a novel comprehensive 3D simulation system called VEROSIM, extending the VSD kernel in various directions.

III. Approaches

In this section the different approaches introduced in Section I is presented in detail.

A. Direct Simulation State Distribution (DSSD)

The most direct approach is unidirectional state change propagation from one instance of our reference simulation system to another. The fundamental idea of this distribution method is that it does not require a predefined interchange model.

In the simplest case the two simulation systems share an identical simulation schema and data model, replicated from the same source. Since the node creation order is identical on both systems, each corresponding simulation database entity has been assigned the same identifier. Therefore property changes can simply be transmitted as a value triplet, consisting of the unique id of the node or extension, the name

or id of the property and the new property value. Property values can easily be serialized with the built-in functionality of the database. Optionally, the communication between the simulation systems can be compressed to preserve bandwidth. However, this leads to an increased time in encoding and decoding the data stream.

Direct Simulation State Distribution is not dependent on any specific transportation layer or protocol, and therefore has no fixed propagation speed. Transportation modes currently implemented are TCP, UDP and file stream, with USB and InfiniBand planned for the future. A file stream offers the ability to log and replay complete simulation runs, while standard gigabit Ethernet is used to interactively operate multiscreen 3D projection systems.

However, a completely identical schema and data model with matching ids across all simulation systems is not always achievable or even desirable. For instance, one might attribute certain simulation functions, such as physics, rendering, user input or hardware control, only to a subset of participating computers, thus balancing the overall load across several simulation systems. These systems may run on different hardware and in different simulation configurations. Therefore, a preliminary id matching stage must be introduced to the distribution process.

In a first step the sender compiles and sends a list of its schema, which attributes a unique id to each class and each property available to its simulation database, thus building a meta-instance and meta-property id list. In a second step, the structure of the database is serialized. Nodes are identified by their meta-instance id (shorthand for their class type), their unique instance id and the id of the parent node. Afterwards the synchronization commences with the communication of the aforementioned value triplets.

On the receiver side a list of meta-instances and meta-properties is also compiled. After receiving the corresponding list from the sender, these two lists are compared. If the sender list contains unknown meta-instances or meta-properties these can either be created on the receiver side (without the functionality) or any changes to a node with an unknown meta-instance will be ignored. Next, the ids of the graph database nodes are mapped. This can either be done by matching the meta-instance and the path of a node within the graph [7] or by building and comparing adjacency matrices of the two involved graph databases [8].

B. Selected Property Synchronization (PropertySync)

In contrast to DSSD, which efficiently reproduces the full simulation state from one instance of the runtime DB to another, PropertySync is used to synchronize only a selected set of properties. The main objectives here are flexibility and configurability. It can be used to partition a simulation model onto several machines with varying operating systems, purposes, locations and workloads. Inputs, outputs and shared parts of these sub-models are instances and properties according to section II.A. So-called *sync connections* make use of the publish/subscribe pattern provided by VSD as described above. They are used to realize a cross-platform-synchronization mechanism for simulation models.

Fig. 3 shows the data view of a simulation model with a node prepared for synchronization. The *sync connection* provides a file, TCP server or socket connection. The *object* is marked with a *sync instance*. The *sync property* defines which property of the *object* needs to be synchronized (here: *worldframe*). The *mode* defines whether the property is subscribed (*read*), published (*write*) or both (*read-write*). To provide a maximum of flexibility, the *sync instance* and *property* can be marked with individual *sync id*s, which are added to the message header. Thus, any properties with matching types can be mapped onto each other. Nonetheless, the default behavior is the standard publish/subscribe pattern and can be automatically configured for the user.

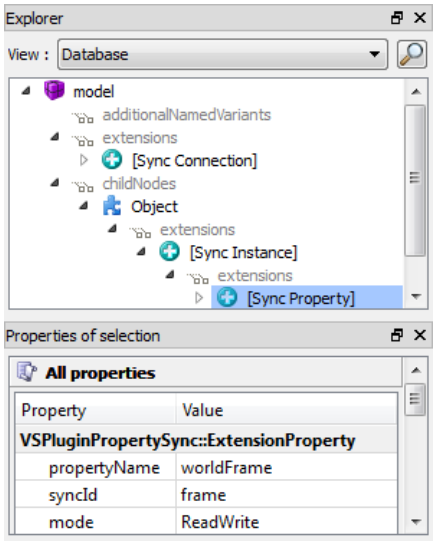

Fig. 3. Data view of a simulation model with a node prepared for synchronization.

C. Database Synchronization (DBSync)

As mentioned in the introduction, we also developed a database-driven approach for distributed 3D simulation [2]. Here, a central external database (generically called ExtDB) is used for data management, communication and versioning, providing a Central World Model (CWM) as introduced in [9]. It manages a simulation model comprising static (e.g., buildings or trees) as well as dynamic (e.g., cars or helicopters) parts. Simulation clients connect to ExtDB to jointly use the shared model. It also works as an active communication hub by providing a change notification mechanism just like VSD. This way, e.g., updates to a car simulated by one client are distributed to all others. Hence, the whole simulation is represented by the shared model's sequence of states. Using a temporal database [10], all these changes are versioned yielding a queryable simulation log for archiving and debriefing. In our current prototypes the object-oriented data management system SupportGIS Java (SGJ) [11] is used as an ExtDB implementation.

To efficiently access the shared simulation model within ExtDB, we developed a flexible approach for database synchronization. Here, the runtime database of the simulation system (generically called SimDB) is synchronized with ExtDB. For SimDB we use the aforementioned VSD.

Synchronization is realized on three levels: Schema, data, and functional level. Once during system startup, the schema description from ExtDB is synchronized to SimDB yielding a

schema mapping (see Fig. 4) so both systems "speak the same language." During runtime, data conforming to the matched schema can be replicated from ExtDB to SimDB. Local changes to these replicates can be resynchronized to ExtDB using the data mapping (also see Fig. 4) in between. They are detected using the state change notifications of VSD as mentioned above.

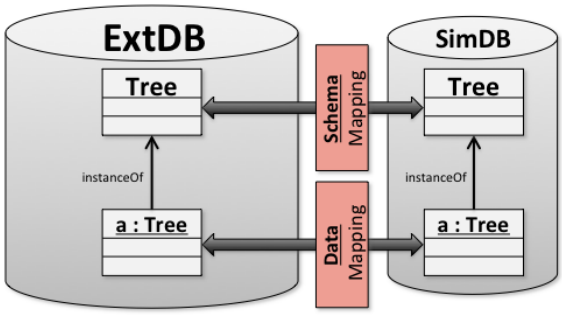

Fig. 4. Schema and data mapping of the database synchronization approach.

ExtDB notifies other clients of these changes so they can adopt them within their respective replicate copies of the same data. Fig. 5 shows an example where a door object in ExtDB is replicated to two simulation clients' SimDB instances. The door is opened in client #1 where SimDB locally emits a change notification. The change is resynchronized to ExtDB were a time-stamped version of the door's previous state is stored before the change is notified. Subsequently, client #2 will adopt the door's new state in his replicate copy.

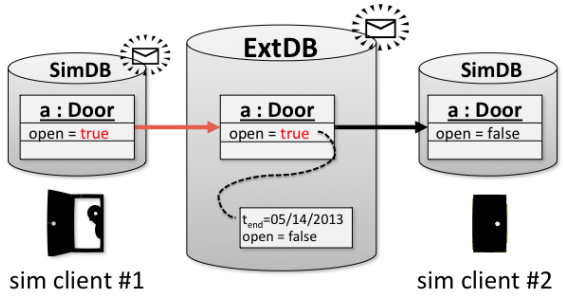

Fig. 5. Example of database-driven communication including versioning.

Especially when using simulation models with standardized data schemata like CityGML [12] or SEDRIS [13] some semantics or functions of data objects need to be synchronized as well. This is done using functional data synchronization during runtime. Here, specific objects represented in ExtDB's schema are translated into functional equivalents interpretable by the simulation system. A simple example would be a material description in SEDRIS translated to a material description known by the simulation system so it can be used for rendering.

To make the approach portable to other SimDB and ExtDB implementations, generalized database requirements were defined [14]. They are based on techniques known from Model-Driven Engineering (MDE) [15] and follow a similar approach like Model-Driven Interoperability (MDI). Here, both databases' meta-models (i.e., their abstract syntax) are required to be matched by model transformations, e.g., using ATL [16]. By requiring additional model transformations to and from the UML [17] as a pivot meta-model, UML structures and semantics can be used to give a generalized method specification of the database synchronization approach presented above [18].

D. HLA

Our simulation system is also capable of interacting with other systems via HLA. This is achieved by attaching specific sender and/or receiver extensions to the nodes whose properties are distributed within the HLA federation. A central HLA communication node – referenced by each sender/receiver – is responsible for communication with the federation and for translating the native data into a FOM compatible format (and vice versa). Again, the state change message of the database is used to obtain data necessary for sending these changes to the HLA federation via the communication node. Changes from the federation are redirected to the corresponding receiver extensions, which in turn update the properties. To avoid possible update loops, message emissions are blocked while updating a property.

IV. Comparison

The four approaches can be compared with regard to different aspects.

A. Basic Idea

The basic idea or main purpose of the DSSD approach is load distribution. I.e., rendering and simulation tasks shall be distributed over a set of clients typically at the same site. The PropertySync technique is rather used for a simple synchronization of individual property values, e.g., to pass values through a distributed I/O network. HLA is a protocol for cooperative distribution between different sites and clients with a focus on interoperability. Finally, the DBSync approach's main idea is to provide a Central World Model for persistence, communication, and versioning.

B. Mode / Topology

DSSD, HLA, and DBSync use a client-server architecture. In DSSD, the topology can be arbitrarily configured but is fixed during runtime. In contrast, HLA has a publish-subscribe model allowing clients to dynamically connect and disconnect from its RTI. Similarly, clients can connect and disconnect to the central database in the DBSync approach. In contrast, the PropertySync approach has a peer-to-peer structure where individual properties are synchronized between a pair of clients.

C. Clients

DSSD, HLA, and DBSync conceptually all support an arbitrary number of clients. For DSSD, these clients must be homogeneous (currently only VEROSIM clients are supported). In contrast, HLA and DBSync support heterogeneous clients as long as they support the respective protocol or API. PropertySync however can only be used to connect two homogeneous clients (currently VEROSIM).

D. Schema / Data

The DSSD approach works with distributed heterogeneous schema and data. I.e., schema and data do not have to be identical at every client. PropertySync goes one step further

by even allowing to synchronize properties independent of their schema – only the data types have to match. HLA combines a central common schema (the FOM) with a heterogeneous set of schemata for each client (SOM). Finally, the DBSync method uses a central schema within ExtDB that is synchronized to all clients and manages a central master-copy of the shared simulation model.

E. Replication

Only two of the presented techniques rely on replication. DSSD presumes a large percentage of existing replication of the simulation model using files or other techniques. Using DBSync, schema and data are replicated. Data replication however can be done selectively (i.e., only parts of the simulation model can be replicated to a client). In contrast, HLA and PropertySync do not presume any replication.

V. Applications

A. Virtual Reality and Multiscreen 3D Simulation

With the help of the DSSD approach we can easily transfer any 3D simulation model from a desktop workspace onto a stereo multiscreen setup, without any further adaptation necessary by the simulation expert. Each screen is powered by two computers (one for each eye) and additional computers are responsible for the simulation and user interaction. On each computer the simulation is started in the appropriate configuration and the model is loaded and executed automatically.

The result is a comprehensive development and testing environment based on simulation technology, a Virtual Testbed (see Fig. 6).

Fig. 6. Interaction with the virtual Testbed.

The Virtual Testbed concept is a key technology in the emerging field of eRobotics [19], because Virtual Testbeds can act as a central focal point in multi-disciplinary development projects. The additional screen space leads to a more immersive experience for the user.

B. Self-Localization Unit

In the context of the research project SELOK, an algorithm for the self-localization of mobile robots in unstructured environments called VisualGPS was developed [20]. A local map of landmarks is constructed from sensor information (laser scanners, stereo cameras) and localized in a global map of landmarks, usually derived from aerial survey data, using a particle filter. The approach is generic and can be applied to trees in the woods or stones and craters on planetary surfaces. A multi-purpose mobile self-localization unit was constructed (see Fig. 7), which contains an industrial computer running the simulation system VEROSIM that features all necessary algorithms and holds the environment model. Another instance of VEROSIM is running on a ground station, which controls the mobile robot as well as the localization unit. In this scenario, all data flow is realized using PropertySync.

Fig. 7. Integration of 2D laser scanner, stereo camera, IMU and industrial computer to a localization unit.

C. Virtual City Simulation

A drive through a virtual city model stored in a central database was realized using the approach presented in Subsection III.C (Fig. 8). Here, a central SGJ database contains a SEDRIS-based desert village (data: RDE) as static parts of the shared simulation model. It also comprises the dynamic parts of the simulation: A helicopter and a car. Two VEROSIM simulation clients are connected to SGJ. Client #1 runs on a standalone PC connected to a television screen and controls the car. Client #2 controls the helicopter. To drive its multi-projection screen it uses the DSSD approach. Changes to either vehicle are logged and communicated via the central SGJ. For debriefing, a simulation run can be replayed using a VEROSIM instance that accesses the versioned data from the shared simulation model.

In the context of the research project Virtual Forest the same techniques are used to manage forest model data.

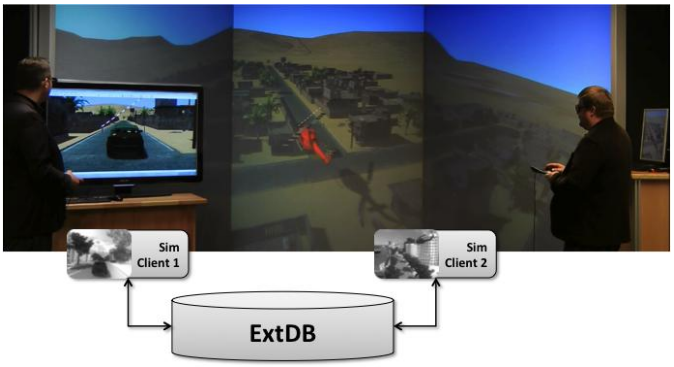

Fig. 8. Database-Driven virtual city simulation.

D. MATLAB Co-Simulation

Based on HLA, the simulation system VEROSIM was connected to the MATLAB Toolbox [21] to obtain a standardized and easily configurable way to establish a co-simulation. In a test scenario we used MATLAB to calculate the behavior of a system consisting of damped harmonic oscillators and used VEROSIM to present the results in 3D. Furthermore, any property changes in VEROSIM relating to the oscillators (such as stiffness) were sent to the MATLAB model.

VI. Related Work

The use of HLA in distributed discrete event simulation is detailed in [22]. In distributed virtual environments (DVE) a more direct approach is generally favored. Reference [23] gives an overview of different synchronization algorithms used in various DVEs. This paper also describes a method somewhat similar to DSSD. The method does not explicitly propagate state changes of the simulation system, but commands that in turn may be used to alter the state of the target system.

As described in [24]-[26], in multiscreen systems distributed rendering is also quite common. However, here the rendering alone (not the simulation) is distributed.

Regarding Subsection III.C we found different approaches that combine 3D software systems with databases. However, none of them provides a comparable integration of data management, collaboration, versioning, and flexible schema support. Some only use a database to store simple object positions [27]. In contrast, others store the complete scene data like in our approach. Some of those also support a distribution mechanism for collaboration between the clients [28]-[32], while others only support single user or read-only usage [33], [34]. Different data schemata are only supported by few systems [30], [32], [34]-[36]. Finally, data versioning is only provided using proprietary data schemata [33] or on file level using a CVS-like approach [35], [36].

VII. Conclusion

An evident conclusion that can be drawn from this work is that different simulation problems require different approaches for distribution and synchronization. In particular, we outlined how the standard HLA-based approach may not suit every scenario. High-performance synchronization between the clients of a 3D multi-projection screen benefit from a more direct approach we call DSSD (Direct Simulation State Distribution). A "no-frills" synchronization between single properties can more simply be realized using the presented PropertySync method. Finally, collaborative scenarios with huge shared simulation models benefit from a database-driven approach to distribution. Nevertheless, classic interoperability scenarios like a Co-Simulation between heterogeneous simulation systems still do benefit from the standardized HLA protocol.

Prototypical implementations based on a Versatile Simulation Database (VSD) as well as various exemplary applications practically prove the advantages of our diversified approach.

In the future, we plan to improve the DSSD approach by minimizing the time and bandwidth required to generate, transmit and evaluate the schema and the current state of the sending VEROSIM system. This should lead to a significant speed-up of the simulation start. Furthermore we want to introduce a mixed TCP / UDP protocol, which enables multicasting of non-essential information. Regarding the DBSync approach, we are thinking about prototypes for other database systems for ExtDB, e.g., PostgreSQL. Another interesting research field is the combination of the different approaches. As shown above, this has already been done for DSSD and DBSync. It may likely be evaluated in more detail to identify other benefiting scenarios.

Acknowledgment

Parts of this work were developed in the context of the research projects Virtual Forest, SELOK and iBOSS-2. 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. SELOK/iBOSS-2: Supported by German Aerospace Center (DLR) with funds of the German Federal Ministry of Economics and Technology (BMWi), support code 50 RA 0911 (SELOK) and 50 RA 1203 (iBOSS-2).

References

[1] *Standard for Modeling and Simulation High Level Architecture*, IEEE 1516, 2010.
[2] M. Hoppen, M. Schluse, J. Rossmann, and B. Weitzig, "Database-driven distributed 3D simulation," in *Proc. the 2012 Winter Simulation Conference*, 2012.
[3] I. Kurtev and K. V. D. Berg, *MISTRAL: A Language for Model Transformations in the MOF Meta-Modeling Architecture*, pp. 139–158, 2005.
[4] M. Gyssens, J. Paredaens, J. V. D. Bussche, and D. V. Gucht, "A graph-oriented object database model," *IEEE Trans. Knowl. Data Eng.*, vol. 6, no. 4, pp. 572–586, 1994.
[5] J. Rossmann, M. Schluse, and R. Waspe, "Combining supervisory control, object-oriented petri-nets and 3D simulation for hybrid simulation systems using a flexible meta data approach," in *Proc. the 3rd International Conference on Simulation and Modeling Methodologies, Technologies and Applications (Simulation Tools and Platforms) - Simultech 2013*, 29-31 July, 2013.
[6] J. Rossmann, M. Schluse, A. Buecken, P. Krahwinkler, and M. Hoppen, "Cost-Efficient semi-automatic forest inventory integrating large scale remote sensing technologies with goal-oriented manual quality assurance processes," in *Proc. IUFRO Division 4 Conference Extending Forest Inventory and Monitoring over Space and Time*, Quebec City, Canada, May 19-22, 2009.
[7] C. D. Godsil and G. Royle, *Algebraic Graph Theory*, Springer, 2001.
[8] S. Roman, "Advanced linear algebra," Springer, 2007.
[9] E. Freund, M. Müller, and J. Rossmann, "Data storage and flow control in automation systems by means of an active database," in *Computational Intelligence for Modelling, Control & Automation '99. Intelligent Image Processing, Data Analysis & Information Retrieval*, Vienna, Austria: Amsterdam: IOS Press; Tokyo: Ohmsha, 1999, pp. 235–240.
[10] R. Elmasri and S. B. Navathe, *Database Systems: Models, Languages, Design, and Application Programming*, 6th ed., Prentice Hall International, pp. 1155, 2010.
[11] SupportGIS Java. [Online]. Available: http://www.supportgis.de
[12] CityGML. [Online]. Available: http://www.citygml.org
[13] SEDRIS. [Online]. Available: http://www.sedris.org
[14] M. Hoppen, M. Schluse, and J. Rossmann, "A metamodel-based approach for generalizing requirements in database-driven 3D simulation (WIP)," in *Proc. the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium*, 2013, pp. 3:1–3:6.
[15] M. Brambilla, J. Cabot, and M. Wimmer, "Model-Driven software engineering in practice," Morgan and Claypool Publishers, 2012.
[16] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, "ATL: A model transformation tool," *Sci. Comput. Program.*, vol. 72, no. 1–2, pp. 31–39, Jun. 2008.
[17] Unified Modeling Language (UML). OMG. [Online]. Available: http://www.uml.org/
[18] M. Hoppen, M. Schluse, and J. Rossmann, "Database-driven 3D simulation - a method specification using the UML metamodel," in *Proc. 11th International Industrial Simulation Conference ISC 2013*, 2013, pp. 147–154.
[19] J. Rossmann and M. Schluse, "Virtual robotic Testbeds: A foundation for e-robotics in space, in industry - and in the woods," *2011 Dev. E-systems Eng.*, pp. 496–501, Dec. 2011.
[20] J. Rossmann, C. Schlette, M. Emde, and B. Sondermann, "Advanced self-localization and navigation for mobile robots in extraterrestrial

environments," *Computer Technology and Application 2*, no. 5, pp. 344–353, 2011.
[21] HLA Interoperability for MATLAB models – HLA Toolbox™ [Online]. Available: http://www.mak.com/products/partner-products/hla-interoperability-for-matlab-models.html
[22] R. Fujimoto, "Parallel and distributed simulation," in *Proc. 2001 Winter Simul. Conf.*, 2001.
[23] Z. Bartosiewicz, "Control theory with singular state-space constraints," *Journal of Mathematical Systems Estimation and Control*, vol. 8.1, pp. 147-150, 1998.
[24] M. Repplinger, A. Löffler, D. Rubinstein, and P. Slusallek, "URay: A flexible framework for distributed rendering and display," Technical Report 2008-01, Universität des Saarlandes, Saarbrücken, 2008.
[25] G. Humphreys and I. Buck, "Distributed rendering for scalable displays," in *Proc. the 2000 ACM/IEEE Conference on Supercomputing*, IEEE Computer Society, 2000.
[26] P. Yin, X. Jiang, J. Shi, and R. Zhou, "Multi-screen tiled displayed, parallel rendering system for a large terrain dataset," *IJVR*, vol. 5, no. 4, 2006.
[27] T. Manoharan, H. Taylor, and P. Gardiner, "A collaborative analysis tool for visualisation and interaction with spatial data," in *Proc. the Seventh International Conference on 3D Web Technology*, 2002, pp. 75–83.
[28] E. V. Schweber. (1998). SQL3D - Escape from VRML island. [Online]. Available: http://www.infomaniacs.com/SQL3D/SQL3D-Escape-From-VRML-Island.htm
[29] S. Julier, Y. Baillot, M. Lanzagorta, D. Brown, and L. Rosenblum, "Bars: Battlefield augmented reality system," in *Proc. NATO Symposium on Information Processing Techniques for Military Systems*, 2000, pp. 9–11.
[30] C. Watanabe and Y. Masunaga, "VWDB2: A network virtual reality system with a database function for a shared work environment," *Information Systems and Databases*, pp. 190–196, 2002.
[31] 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 *Proc. 21st International Conference on Data Engineering Workshops (ICDEW'05)*, 2005, pp. 1259–1259.
[32] K. Walczak, "Dynamic database modeling of 3D multimedia content," in *Interactive 3D Multimedia Content*, W. Cellary and K. Walczak, Eds. London: Springer London, pp. 55–102, 2012.
[33] A. Vakaloudis and B. Theodoulidis, *Spatiotemporal Database Connection to VRML*, 1998.
[34] D. Schmalstieg, G. Schall, D. Wagner, I. Barakonyi, G. Reitmayr, J. Newman, and F. Ledermann, "Managing complex augmented reality models," *IEEE Comput. Graph. Appl.*, vol. 27, no. 4, pp. 48–57, 2007.
[35] Dassault Systèmes. (2008). ENOVIA V6 technical advantages whitepaper [Online]. Available: http://www.3ds.com/fileadmin/PRODUCTS/ENOVIA/PDF/WHITE-PAPERS/PCSWhitepaper-0807_final_July_29.pdf
[36] Autodesk. (2005). Best Practices for Implementing Autodesk Vault [Online]. Available: http://images.autodesk.com/adsk/files/best_practices1.pdf

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

R. Waspe studied theoretical physics at Queen Mary College, University of London, UK. Between 2001 and 2005 he was a research assistant at the Institute of Robotics Research (IRF) in Dortmund, and between 2005 and 2006 software developer at EFR-Systems GmbH in Dortmund. Since 2006 he is team leader at the Institute of Man-Machine Interaction at the RWTH Aachen University.

M. Hoppen studied computer science at the University of Bonn, Germany. Since 2007 he is a Ph.D. student and research associate at the Institute for Man-Machine Interaction of RWTH Aachen University (Germany) where he works on combining 3D simulation systems with databases.

M. Rast has studied physics at the University of Bonn, Germany and is currently working as a research associate at the Institute for Man-Machine Interaction of RWTH Aachen University since 2008. His main field of interest is the integration of physics process simulation for virtual prototyping.