20th ISPE International Conference on Concurrent Engineering
C. Bil et al. (Eds.)

doi:10.3233/978-1-61499-302-5-19

A Software Architecture to Synchronize Interactivity of Concurrent Simulations in Systems Engineering

Christian BARTELT[a, 1], Volker BÖß[b], Jan BRÜNING[b], Andreas RAUSCH[a], Berend DENKENA[b] and Jean Paul TATOU[a]
[a] *Software Systems Engineering (SSE), University of Clausthal, Germany*
[b] *Institute of Production Engineering and Machine Tools (IFW), Leibniz Universität Hannover, Germany*

Abstract. Due to distributed development of complex technical systems like machine tools, different system components are modeled and simulated in independent program suits. Several standards specify exchange of model data, but communication during concurrent simulations is not standardized yet. Therefore, the SimBus (Simulation Bus) was developed to close this gap. This novel software architecture allows flexible coupling and implementation of existing simulation software suits.

Keywords. System Simulation, Software Architecture, Machine Tool, Multi Domain Simulation

Introduction – Integration of Concurrent running Simulations

Nowadays, design and simulation software systems have become an absolutely essential part in development of complex products. This trend is promoted by concepts for integrated development processes like concurrent engineering or digital factory (German Engineers' guideline VDI 4499) [1]. Established methods like product data management (PDM) or model data standards like STEP (ISO 10303) [2] are part of these efforts to realize continuous data exchange.

Problem of Synchronization of Concurrent Interacting Simulations

Engineering modern CNC machine tools with mechanical, electrical and mechatronic components is a typical example of systems engineering. During the development of machine tools, several R&D departments and suppliers are involved. Every participant uses specialized software for his subject and, depending on the development progress, models with a different degree of detail. To represent the behavior of the machine tool, all sub-models have to interact with each other in a so-called all-in-one system simulation scenario.

Due to different complexity and simulation technologies of partial simulations in such a scenario, it is necessary to represent the system by a distributed simulation.

[1] Corresponding Author.

Therefore, definite simulation modules have to represent several physical and logical system components (Figure 1). The dynamic interaction of modules and the implementation of basic simulation functions, e.g. job control, need a common communication interface, which is binding on all attended simulation systems and tools. In consequence of an integrated development process, it is necessary to realize a scalable and reconfigurable simulation platform. Due to this, each attended module type has to act as a black box with unique functions as specialized interfaces.

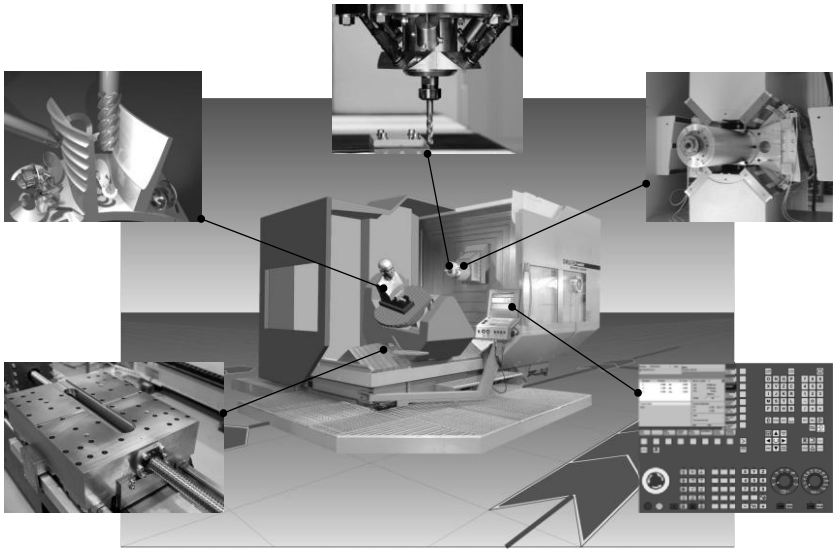

Figure 1: Physical and logical system components of machine tool

The virtual model of machine tool has to be applicable resp. reusable in different phases of machine tool's life cycle like development and process planning. Different models of predefined system components can be realized with a varying degree of detail. A library of available system component's models allows flexible configuration of simulation scenarios, which are fitted for special use cases. E.g. during dimensioning of a customized machine tool, simulation will be mainly focused on mechanical reliability of machine structure and less on precision prediction of tool paths. In this case the user can choose a detailed FEA model of structure's mechanical behavior and a simplified model of numerical control and drive system. In another example, simulations for process planning will be mainly focused on process stability and quality of work piece. Therefore, it would be necessary to choose detailed models, which are able to simulate dynamic behavior of control and drive system to analyze the compliance with tool path.

Shortcomings of Systems Simulation Infrastructures

Nevertheless typical simulation scenarios are limited to single system components, which are developed independently by different R&D departments. To realize basic software communication between (distributed and concurrently running) simulation components, several open and proprietary standards/architectures existing – generally by using tailored software interfaces regarding bus software architectures. A common use case is coupling Finite Element Analysis (FEA) and Multi Body Simulation (MBS)

systems by using proprietary interfaces of an integrated software suite or by using commercial Computer Aided Control Engineering (CACE) resp. Digital Block Simulation (DBS) software, like Matlab/Simulink [3], [4]. Due to the high degree of specialization, the interaction of these interfaces is mostly limited to bidirectional communication. Beside the predestined way to connect technically the simulation software components by a standard middleware platform, there are two further ambitious challenges – the design of correct interrelations between apriori independent simulation domain models at first and the synchronization of concurrently clocked simulations modules secondly.

1. Related Work – Software Architectures for Integrated Systems Simulations

Looking for architecture patterns to use as starting point when designing a middleware infrastructure for the integration of different concurrently running simulations quickly leads to the High Level Architecture (HLA) [5]–[7]. One main intention of the HLA is a collaborative interaction simulation of different autonomous system simulations (traffic simulation, military tactic simulation etc.). Hence, the HLA is an architecture in which the sender of a request does not need to know his receiver. The HLA is thereby an event-based architecture that supports the emergent interconnectivity of components. For the simulation of machine tool processes, we do not have any problem with the emergent interconnectivity – all simulated machine modules are hard-wired at simulation start-up time.

The other approach arising in the last years in the research communities on integration technology for simulation software is the use of a unified language for system modeling [7], [5]. That means, the description of the whole system could be done in one modeling language like Modelica [8], [9]. Thereby, the simulation of the whole system in an environment like OpenModelica should not pose a problem of models interoperability. Although this approach is ambitious and promising, it remains less practicable in the short term, because its success depends largely on the ability of the unified language to model all aspects of a system from all systems engineering disciplines. However, reducing the need of modeling the whole systems using a unified language to the specification of a standard interface for models interchange and co-simulation between the simulation tools using a unified language seems to be more practicable. Examples of such tentative using the XML description language could be found in [10]–[12]. As explained in [10], [11] the Functional Mockup Interface (FMI) describes a models interchange interface between simulation environments. The intents of the FMI are that a modeling and simulation environment (acting as slave) can generate C-Code of a system model that can be utilized by other modeling and simulation environments (acting as master) either in source or binary form. Therefore, the so-called slave components are not directly coupled with each other but only with the so-called master simulators. Furthermore, a master algorithm that will coordinate all interactions have to be developed. But we are more interested in a system simulation environment in which there is not necessary to choose one simulation tool as a master.

2. A Software Architecture for Coupling Concurrent Heterogeneous System Simulations

The proposed software architecture is driven by a concept for the configuration of simulation scenarios. A basic aspect of this concept implies reusability of simulation modules regarding the execution in any different simulation scenarios – analogous to enabling interconnections between mechatronic modules in the real world. Each module realizes a reactive behavior. After the manual configured interconnection of modules and their initialization/activation, the modules interact self-managed and realize the intended simulation scenario.

In the following subsection the configuration concept for interacting simulation modules is presented in detail. Afterwards the software architecture SimBus that realizes the middleware to manage the interaction between several heterogeneous simulations is explained. In Subsection 2.3 the required interaction scheduling is described in detail.

2.1. Configuration of Simulation Scenarios

The configuration concept considers three engineering levels, which are depicted in Figure 2. At the lowest level generic module interfaces (e.g. controller, integrator, mechanical structure etc.) are described by a formal interface description language. These interfaces are designed in a simulator-independent manner, that means without considering subsequent simulator-specific requirements. Also, modules designed at the next higher engineering level with interfaces from the level above have to be suitable to use in any simulation scenario.

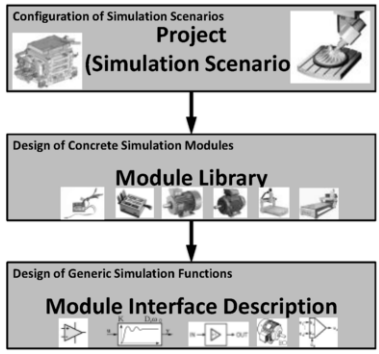

Figure 2. Configuration Levels

Therefore, one can realize or instantiate concrete modules to use in a specific simulator (e.g. Beckhoff controller or Siemens controller with the same generic controller interface etc.). All specified modules are collected in a global module library, which can be used to define concrete simulation scenarios by interconnecting modules from the library. The third engineering level – simulation project definition – is dedicated to concrete scenario specification. At this level, the engineer would, accordingly to his simulation scenario, select a set of modules from the module library and would interconnect them regarding the corresponding interface descriptions from the lowest level.

2.2. An Architecture for Integrated Simulation of Mechatronic Components

Our proposed software architecture describes a virtual communication bus that interconnects the different simulation modules (c.f. Figure 3). The virtual bus is realized by a middleware, which provides a set of communication management services. These services are necessary to coordinate the interactions between the simulation modules. Indeed, the simulation modules and their interconnectivity realize the container design pattern and are thereby able to concurrently execute simulation tasks based on domain specific models. Each of these containers includes one simulator and one SimBus Adapter. For the communication between containers, a SimBus Module Interface is provided.

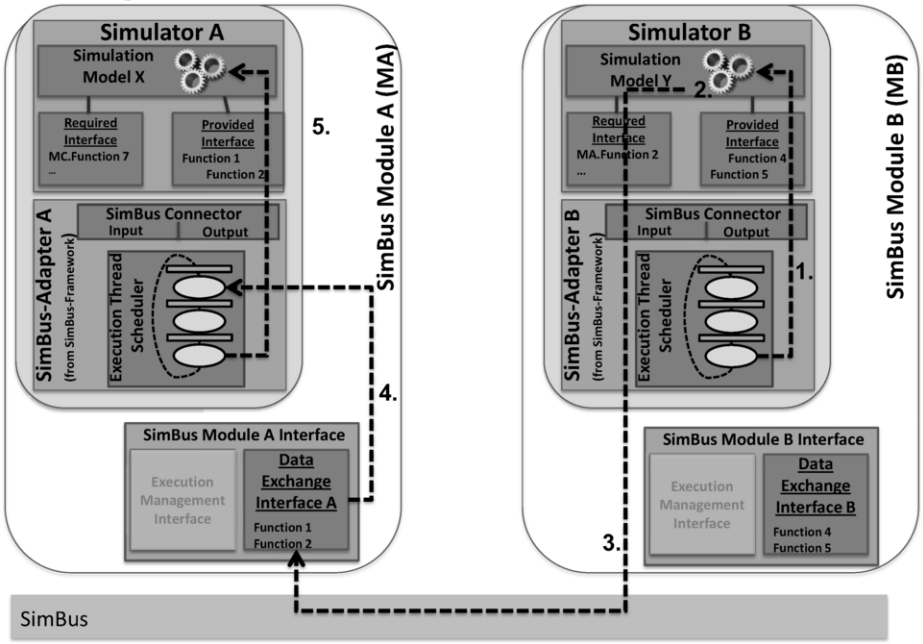

Figure 3. SimBus Architecture – Simulator Interaction

Let's consider an example of a simulation involving two simulators (Simulator A and B in Figure 3). Normally, simulators are based on tool specific software technologies (e.g. Multi Body Simulation, Finite Element Analysis, Digital Block Simulation etc.) and are able to simulate domain specific models, each simulator having a "provided" and a "required" interface. All providing interface functions are implemented by a model (e.g Model X and Y in Figure 3), which is simulated by the simulator (e.g. a controller model in Matlab Simulink). Required interfaces describe functions that must be called at external modules to execute the model simulation.

The SimBus Adapters support the communication between modules via the bus and schedule the execution of requested functions within the simulator. A SimBus Adapter consists of a SimBus Connector and an Execution Thread Scheduler. The connector implements a wrapper that requests functions of the simulation model via a tool-specific software interface (output) and prepares responds from external modules for the executed simulation model (input). The Execution Thread Scheduler manages

external function requests that are implemented in the simulation model. The scheduling mechanism is described in detail in Subsection 2.3.

Supported by this architecture, engineers can use independently developed simulation models based on heterogeneous software simulators (Simulink, ASCET, CAx-tool, CutS [13]) using the same software container. But the connector that wraps these simulators is implemented differently due to the fact that simulation tools use different simulation models and have different interface for integration (dll, plugins etc.).

In Figure 3 the representative processing of a request is depicted. Module B initiates a request towards Module A. In Step 1, the Execution Thread Scheduler processes an external request of Function 4 within Module B. It calls the corresponding simulator function supported by its tool-specific connector. The implementation of Function 4 needs the external service of Function 2 from Module A during its execution (Step 2.). Therefore Simulator B calls the function at Module A using the input wrapper of its SimBus Connector in Step 3. Then the request of Function 2 is queued in the first "Waiting State" of the Execution Thread Scheduler of Module A in Step 4. Finally – if the Function-2-request is switched in the third "Running State" – the Execution Thread Scheduler calls the corresponding function on Simulator A using the output wrapper of its SimBus Connector.

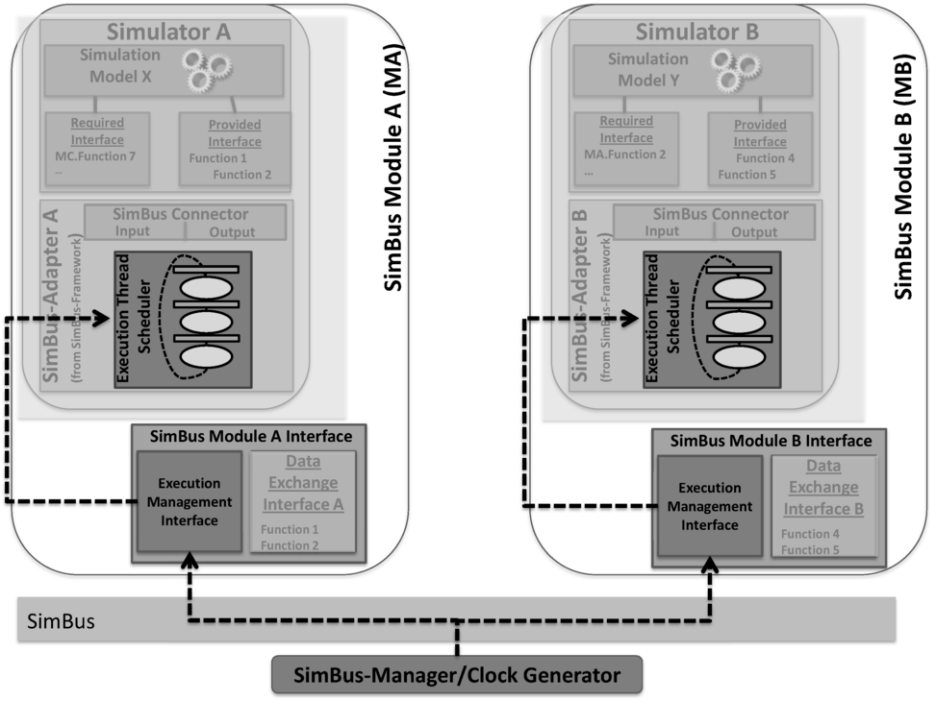

Figure 4. SimBus Architecture - Generic Module Scheduling

Beside the simulation-model-specific Data Exchange Interface, each module realizes the same Execution Management Interface (c.f. Figure 4. SimBus Architecture - Generic Module Scheduling). The Execution Thread Scheduler controls the processing of requests on the simulator. It synchronizes the processing of asynchronous requests between different modules via the bus. Each external request processing by the

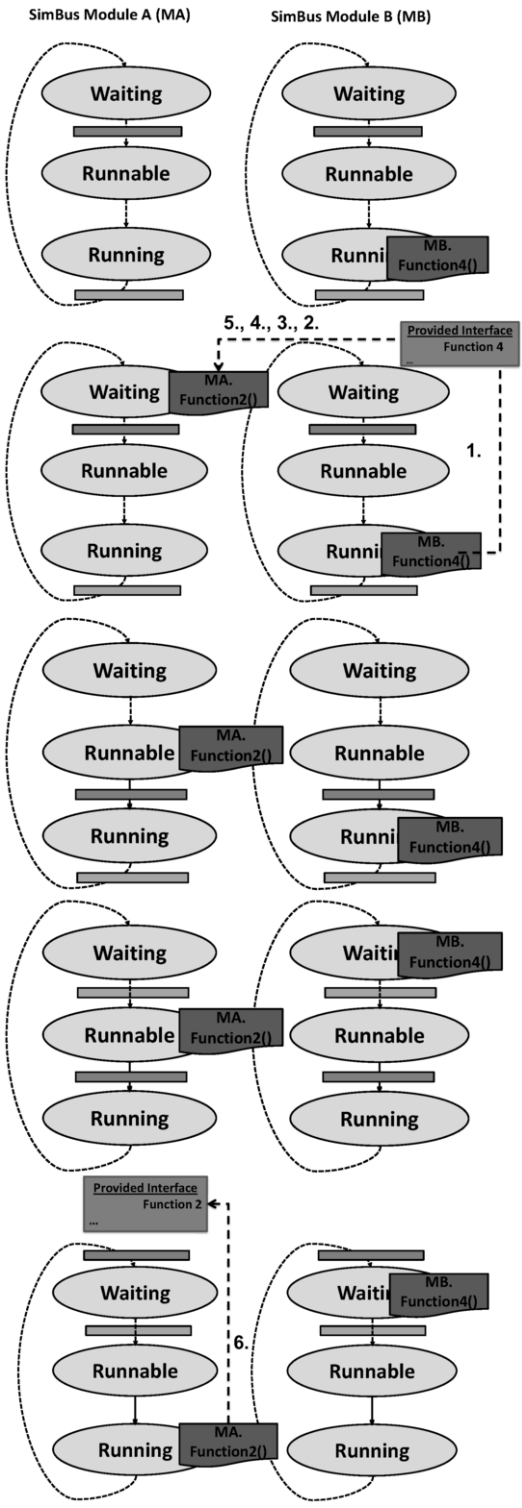

Figure 5. Processing within Execution Thread Scheduler

Execution Thread Scheduler can assume one of the three states – "Waiting", "Runnable", and "Running" (c.f. Figure 5). The state changing is controlled by the SimBus management component called SimBus Manager. This component triggers the state change of all requests by a broadcast signal via the bus to all modules. This mechanism is further described in detail in the following subsection.

SimBus –Synchronized Interaction between Executed, Heterogeneous System Models

As previously mentioned, the SimBus platform provides a master component called SimBus Manager, which assumes the role of a scheduler at the execution time span of the simulation modules. In other words, the SimBus Manager broadcast via the execution management interface an activation signal to all modules connected on the bus (c.f. Figure 4). Receiving this signal means for each module that it may execute pending jobs. The communication between the SimBus Manager and the modules on the bus corresponds to the control flow on the bus.

We differentiate between the control flow and the data flow (data exchange between the simulation components). This separation is useful since we allow the simulation components to communicate to each other via function calls without using the SimBus Manager as transfer-buffer and by controlling the function execution in components with the SimBus Manager, we ensure a deadlock free execution of the models in a distributed environment in that remote function calls between

modules are asynchronous (non-blocking) and each initiated remote function call runs in its own thread until the call is completed. Thus, 10 calls of the same functions initiate 10 different threads. Processing a remote function call at the receiver side assume that the corresponding thread is first send in the state "Waiting", secondly in state "runnable" and finally in the state "running". The function execution takes place only in the running state. If the execution cannot complete because some input data is not yet available (according function calls are initiated), the thread is sent in the waiting state, meaning that the thread sleeps until its input data is available. Each component gets its input data by calling the corresponding producer in an asynchronous manner; it may continue its execution or wait for the response if the requested data is needed immediately. The main advantage of our proposed software architecture is a decentralized data communication between components. Each component requests its input data when needed. The progress of the component execution is then control by a master component, which assumes the coordination of the concurrent execution.

In Figure 5 an exemplary extract of request processing within the Execution Thread Scheduler is depicted. Following the interaction between modules in Figure 3 the processing of the Function 4 request is shown. At the initial stage of the figure, the execution of Function 4 is in the state "Running". This means that the processing of Function 4 within Simulator A is active. During execution of Function 4, Function 2 of Module A is requested (Step 2., 3., 4., 5. in Figure 3). This Function request is queued in the "Waiting" state of the Execution Thread Scheduler of Module A as is depicted in the second stage of Figure 5. Afterwards the execution of Function 4 is idle until it receives the requested data from Module A. When all execution threads in state "Running" are terminated or idle, the SimBus Manager sends the state change signal. The third and fourth stage show the state changes of Function 2 and 4 execution in both modules. At the final fifth stage, the request of Function 2 calls its implementation within Simulator A using the SimBus Connector.

3. Evaluation by Machine Tool Simulation

The developed software architecture was implemented based on a CORBA-like platform using [14] ICE. The implementation provides a scheduling service (SimBus Manager) and a framework that contains an abstract module container (c.f. Module A, B in Figure 3). Furthermore a SimBus Connector was implemented for all required simulation tools (Simulink-Connector, CutS-Connector etc.). At runtime each instantiated module hosts a certain connector to communicate with the tool-specific simulator. Within the abstract module container, the Execution Thread Scheduler and the universal Execution Management Interface is implemented. The Data Exchange Interface was designed for each module type (lowest level in Figure 2). The design of Data Exchange Interfaces is based on the Interface Description Language of ICE. Using this interface description, skeleton code to bind the interface on the Execution Thread Scheduler resp. the SimBus Connector can be generated automatically by the ICE-tools. For the SimBus Manager a user interface was implemented. With this tool the user can initiate/start all required modules and can control the simulation step by step by broadcasts on the SimBus. To realize the upper level of Figure 2, a scenario configuration editor was implemented based on the Graphical Modeling Framework of Eclipse/EMF. With this configuration editor the user can create new simulation scenarios using a predefined library of simulation modules.

The use case "process machine interaction" was built up for evaluation of the shown approach. Four modules, based on machine tool sub-systems described in the first Section, were defined in this scenario setup. As shown in Figure 6, we have separate modules for the physical and logical sub-systems numerical control, a separate position controller, a structural model and material removal representing the results of machining. Objective of this scenario setup is the simulation of process machine interaction based on real tool paths.

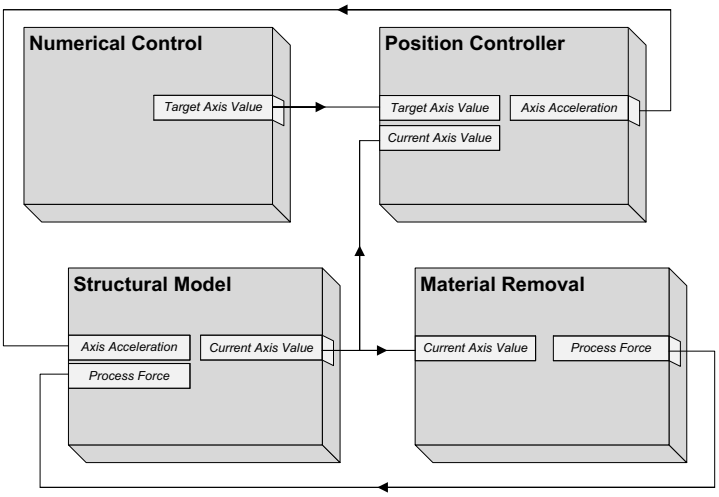

Figure 6: Use case "process machine interaction"

The abstract modules were realized in several software implementations resp. models with different degrees of detail. Thereby, each module is represented by two or three exchangeable models using the common module interfaces. Due to this variety of available models in this example, it is possible to configure the shown scenario setup in 36 different ways without formal restrictions of compatibility. The number of alternatives increases with the number of models representing each module. Regarding to quality of simulation performance, the user has to ensure that the configuration of simulation affords adequate results.

4. Conclusions

A flexible integrability of several heterogeneous simulators to holistic machine simulations requires a suitable middleware platform that manages the interactivity between simulated machine modules. But common used (event-based) reference architectures for simulator integration (e.g. HLA) are not overcome this challenge. For this reason a software architecture which schedules the interactivity between concurrent running but pre-connected simulators has to be researched. In the previous sections such software architecture – Simulation Bus (SimBus) – was described in detail. This architecture was implemented as a middleware platform for a flexible configuration of simulation scenarios based on a predefined pool of simulation modules. Subsequently a

representative simulation scenario – process machine interaction – was deployed on that platform and was successfully evaluated.

5. Acknowledgments

This research work was supported by "Niedersächsisches Ministerium für Wissenschaft und Kultur" (NMWK) within the Project "Pro³gression – Diligent Production", subproject "FleXimPro – Flexible software architecture for the integrated simulation of manufacturing processes of hybrid machine tools".

References

[1] O. Sauer, M. Schleipen, und C. Ammermann, „Digitaler Fabrikbetrieb. Virtual Manufacturing", in *4. ASIM Fachtagung: Simulation in Produktion und Logistik, Integrationsaspekte der Simulation: Technik, Organisation und Personal, 7. und 8. Oktober 2010*, Karlsruhe, 2010.

[2] ISO10303-1:1994, „Industrial automation systems and integration—product data representation and exchange - Part 1: overview and fundamental principles. International Organization for Standardization", 1994.

[3] M. Zaeh und M. Hennauer, „Prediction of the dynamic behaviour of machine tools during the design process using mechatronic simulation models based on finite element analysis", *Production Engineering - Research and Development*, Bd. 5, S. 315–320, 2011.

[4] C. Brecher und S. Witt, „Simulation of machine process interaction with flexible mulit-body simulation", in *Proceedings of the 9. CIRP Intenational Workshop on Modeling of Machining Operations*, 2006, S. 171–178.

[5] D. Chen, L. Wang, und J. Chen, *Large-Scale Simulation: Models, Algorithms, and Applications*, 1. Aufl. CRC Press, 2012.

[6] J. S. Dahmann, F. Kuhl, und R. Weatherly, „Standards for Simulation: As Simple As Possible But Not Simpler The High Level Architecture For Simulation", *SIMULATION*, Bd. 71, Nr. 6, S. 378–387, Jan. 1998.

[7] „Pitch - HLA Tutorial". [Online]. Available: http://www.pitch.se/hlatutorial. [Accessed: 14-März-2013].

[8] „Modelica and the Modelica Association — Modelica Association". [Online]. Available: https://www.modelica.org/. [Accessed: 21-März-2013].

[9] R. Kossel, W. Tegethoff, M. Bodmann, und N. Lemke, „Simulation of complex systems using Modelica and tool coupling", in *The 5th International Modelica Conference*, 2006, S. 485–490.

[10] T. Blochwitz, M. Otter, J. Akesson, M. Arnold, C. Clauß, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, und others, „Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models", in *9th International Modelica Conference, Munich*, 2012.

[11] O. Enge-Rosenblatt, C. Clauß, A. Schneider, P. Schneider, und O. Enge, „Functional Digital Mockup and the Functional Mock-up Interface–Two Complementary Approaches for a Comprehensive Investigation of Heterogeneous Systems", in *8th International Modelica Conference, Dresden*, 2011.

[12] V. Böß, J. Brüning, und B. Denkena, „Standardized Communication in

Simulation of Interacting Machine Tool Components“, in *Concurrent Engineering Approaches for Sustainable Product Development in a Multi-Disciplinary Environment: Proceedings of the 19th ISPE International Conference on Concurrent Engineering*, 2012, S. 825–836.

[13]B. Denkena und V. Böß, „Technological NC Simulation for Grinding and Cutting Processes Using CutS“, in *Proceedings of the 12th CIRP Conference on Modelling of Machining Operations*, Donostia-San Sebastián, Spain, 2009, Bd. II, S. S. 563–566.

[14]ZeroC, Inc., „Internet Communications Engine (Ice)“, *Welcome to ZeroC, the Home of Ice*, 2013. [Online]. Available: http://www.zeroc.com/. [Accessed: 10-Apr-2013].