A VIRTUAL DESIGN ENVIRONMENT FOR THE PORT OF THE FUTURE

The FAMAS Simulation Backbone Architecture

TRAIL Research School, Delft, November 2002

Authors

Csaba Attila Boer
Erasmus University Rotterdam, Faculty of Economics; Department of Computer Science

Alexander Verbraeck
Delft University of Technology, Faculty of Technology, Policy and Management, System Engineering Department

Yvo Saanen
Delft University of Technology, Faculty of Technology, Policy and Management, System Engineering Department

Hans Veeke
Delft University of Technology, Faculty of Design, Engineering and Production, Transport Technology Group

© 2002 by [Authors] and TRAIL research School
## Contents

### Abstract

1. **Introduction** .......................................................... 1

2. **The Basic Requirements of the Architecture** ............ 4

3. **Overview of the Structure of the FAMAS Backbone Architecture** .... 5

4. **The Technical and Functional Subsystems** .................. 6

   4.1 **The Run Control** ................................................. 6

   4.2 **The Backbone Time Manager** ................................ 7

   4.3 **The Logging Subsystem** ....................................... 7

   4.4 **The Visualization Subsystem** ............................... 8

   4.5 **The Scenario Management** .................................... 9

   4.6 **The Functional Subsystems** ................................. 10

5. **Testing the FAMAS Simulation Backbone Architecture** .......... 11

6. **Conclusion** .......................................................... 13
Abstract

The FAMAS (First All Modes All Sizes) research program contains a number of harbor projects that aim to improve the performance of the harbor processes. One of the subprograms is the FAMAS MV2 project, which aims to create the future container terminals of the port of Rotterdam. The process of designing the future port of Rotterdam is quite complex in the sense that different actors (companies, organizations, research groups, etc.) are involved. Each actor designs and develops simulation models of specific geographical areas or functional subsystems of the port based on its questions, knowledge and experience in that area. In order to have a well-functioning harbor process next to the individual activities of the simulation models realized by the actors the co-ordination between the models must be handled. This consideration implies a complex design multi actor problem.

The aim of the FAMAS MV2 Simulation Backbone project, which is a part of the FAMAS MV2 project, is to provide a virtual design environment where a number of different actors develop specific simulation models that need to communicate and collaborate. The actors therefore have to deal with the coordination of different models. In order to solve the coordination problem between several different simulation models in the harbor environment, we propose a backbone system (which can be considered as the backbone of the virtual design environment) called FAMAS Simulation Backbone Architecture. The FAMAS backbone allows to couple different simulation models implemented by various actors (companies, organizations, research groups, etc.).

Acknowledgements

The authors would like to thank the Connekt organization and the TRAIL research school for supporting and funding the project. They also acknowledge the large amount of work of other researchers: Anko Nagel, Ruud van der Ham, Corne Versteeg, Jaap Ottjes, Mark Duinkerken, Maurits van Schuylenburg, William Rengelink and Shing Wong, who were involved in the FAMAS Simulation Backbone project and helped to reach the final results.

Keywords

Distributed Simulation, Simulation Architecture, Transportation, Port, Reusability
1 Introduction

The FAMAS research and development program has gone into a new phase with the start of the FAMAS MV2 (Maasvlakte) program. The aim of this new program is to conceptualize and design automated container terminals with the connecting Inter Terminal Transport (ITT) Systems by different actors (companies, organizations, research groups, etc.) on the basis of the requirements for Maasvlakte 2 in the Port of Rotterdam for the years 2020 [de Hartog et al., 2001].

In the [de Hartog et al., 2001] a vision is presented for container logistics in the port of Rotterdam for the year 2020, based on market research into the current state-of-the-art in container logistics and the expected developments and trends for the future. The report contains the preliminary investigation of the concept design of this new port (including the area, the equipment, etc.) and performance indicators of the new port. In order to look forward in advance how the future port will operate simulation models can be applied. In the FAMAS research program several simulation models are developed, some at a high level for accessing the overall performance of the port, and some at a detailed level for analysis of specific port functions. This paper addresses the FAMAS Simulation Backbone project.

The FAMAS Simulation Backbone project is meant to create a virtual design environment where these different models can be coupled, can exchange information and can use consistent sets of scenario data. The simulation backbone that has been developed in this project is not an all-embracing simulation model, but an environment where different models can be coupled. The approach follows the rules of enterprise application integration where the attachment of models does not give any problem, provided that the interfaces and functions are well defined.

We may think to represent the whole mechanism of the port in one single simulation model. Due to the complexity of the problem, however this is almost impossible and even if it would be, the designed simulation model would not be flexible and easily manageable in the sense that it would be difficult to incorporate modifications or extensions after completion. Furthermore, we are faced with a multi actor problem, where different companies, institutions, organizations, research groups, etc., handle different parts of the system, and they might not will to share details regarding the implementation and structure of their model. The actors have different ideas, knowledge, and experience from different specific areas and it would be difficult to find a common thinking and implementation mechanism. Therefore it is more advantageous to represent the virtual design environment in a distributed way where each actor can design its own simulation model using its own experience. In spite of this distribution there is a need for a strong interaction between various simulation models. Distributed simulation is an important strategic technology for linking various types of simulation models that are at multiple locations with the aim to create a realistic and complex “virtual world” for the simulation of highly interactive activities [Fujimoto, 2000]. Due to the frequent communication (updating of the event list and information sharing) between the distributed models the distributed system can slow down the speed of the simulation compared to a non-distributed (one big model) system. In this project we want to attain distribution at a reasonable speed. Therefore, we can apply various techniques discussed in [Fujimoto, 2000] that allow to speed up the distributed simulation run.

This approach provides flexibility (it is easier to extend and to maintain the simulation model because a single model focuses on solving small, particular problems), less
investment (there is no need to reconstruct again an already existing simulation model), and reusability (a simulation model can be used in more systems). In a distributed system the simulation models are connected to each other and they need to be coordinated in order to realize the functionality of the system as a whole. In order to achieve an easy distributed coordination the actors can use a standard simulation language. This approach has different disadvantages. Firstly, it costs a lot of money to rebuild in this standard language all the existing simulation models. Secondly the training of the people to this standard language will cost again a lot of money and maybe they do not really prefer the new standard simulation language. Finally, the standard simulation language might not cover deeply enough all the activities in the harbor (specific simulation packages that focus on a certain problem can be more effective). Therefore, in the multi actor problem the package independence is a quite strong requirement.

The problem is therefore, how can we provide a virtual design environment that solves the coordination of different specific simulation models. We solve this problem by introducing a distributed simulation backbone architecture that coordinates the various simulation models that can be coupled to it. In this paper we propose a backbone system, called FAMAS Simulation Backbone Architecture, which allows to couple various simulation models that represent different functional areas of the harbor and are developed by diverse actors (companies, organizations, research groups).

The article is structured as follows. In Section 2 we present the basic requirements of the FAMAS MV2 Backbone project. Then, in Section 3 we briefly introduce the FAMAS Backbone Architecture. Section 4 gives a comprehensive presentation of the technical and functional components of the Backbone Architecture. In Section 5 we discuss the experiments conducted for testing and proving the functionality of this architecture. Finally, in Section 6 we draw the conclusions.
2 The Basic Requirements of the Architecture

The aim of introducing the backbone architecture is to allow for coupling different simulation models and coordinating them. As we focus on systems that are continuously tested, maintained and developed, the virtual design environment, that contains several different interacting models, should be flexible and must allow for the reusability of the simulation models. The following requirements must be achieved by the FAMAS Backbone Architecture [Boer et al., 2002]:

- Distributed execution. The multi actor approach considers distributed simulation models, that is, the models can be executed or operate at geographically distributed environments (different computers, different locations, different countries). As models are distributed we have to apply distributed execution of them.
- Effective communication. Due to the distributed execution and the need for coordination of the activities, the communication plays an important role.
- Standalone and distributed testing possibility. The simulation models developed by different groups can be tested both in a standalone environment (in the environment in which was developed) and in a distributed environment.
- Package independence. The architecture must allow for the combination of simulation models implemented in different simulation packages (e.g. Arena [Kelton et al., 2001], eM-Plant [Technomatix Technologies, 2002], etc.) and programming languages (C++, Java, etc.).
- Structure transparency. The transparency of the architecture helps the modeler to couple the simulation models effortlessly.
- Hierarchical structure. The hierarchical design and development gives the possibility to couple first a model in a rough form (black-box model) and later implemented in more detail (white-box model).
- Industrial applicability. The backbone architecture can be used by different industrial partners, which are involved in developing simulation models.

From scientific point of view introducing, designing and developing a virtual design environment that satisfies the above mentioned requirements would provide an innovation, as it would offer a solution for complex design problems when multi actors are involved. Besides, the approach has also societal relevance, as it introduces an effective method for designing the future port of Rotterdam. This method entails cost reduction as users can stay with their methods and simulation packages, and reuse developed models or sub-models if possible. This approach increases the effectiveness of individual design and development in the sense that it lets different groups to stay with their own way of thinking and to use their experience. As this project aims to solve the coordination of the different models developed in this way, if a standard connection and coordination method could be developed this would reduce the costs and time of coupling new models to the system in the future. Due to the fact that the virtual design environment consists of several simulation models and the coordination between them, it provides an easy way for maintenance because if changes are needed it is not necessary to change the whole system, but only the model(s) involved. In the next section we are going to give an overview of the structure of the FAMAS Simulation Backbone Architecture from technical point of view.
3 Overview of the Structure of the FAMAS Backbone Architecture

This section briefly describes the technical design of the FAMAS Backbone Architecture.

The structure of the backbone architecture consists of technical and functional components (see Figure 1). The technical components of the backbone system are:

- Run Control
- Backbone Time Manager
- Logging
- Visualization
- Scenario Management

The user defined simulation models are the so-called functional components. The overall system that consists of technical and functional (non-technical) subsystems is called a federation, where the subsystems that are connected to the backbone are the federates. The federates (technical and functional) that form a federation communicate by means of messages. The messages are sent and received by the standard socket mechanism with the TCP/IP protocol. The backbone works as a multiple client-server model; each subsystem is a client of all the other subsystems, but each subsystem is also a server, which can be addressed by all the other subsystems. In this way direct communication between subsystems is supported which minimizes communication over the backbone compared to sending all information through a central component (e.g. Run Control), which would act in this case as a gigantic switching board.

Each message consists of four fields, separated by a forward slash. The standard format is: Sender/Receiver/MessageType/Parameters/

- **Sender** is the name of the sender subsystem, e.g. Run Control
- **Receiver** is the name of the receiver, e.g. BBTM
- **MessageType** is used to identify the message, e.g. NextEvent
- **Parameters** is a list of parameters representing the contents of the message, e.g. the next event time
4 The Technical and Functional Subsystems

4.1 The Run Control

The Run Control is the main controller of simulation studies in the FAMAS Backbone Architecture. As can be seen in figure 2 the Run Control subsystem has a direct connection both with the technical and the functional subsystems.

We can distinguish three important activity phase of the Run Control subsystem: initialize and start a simulation run, activities during a simulation run and stop and close a simulation run [Boer et al., 2002]. In order to start a simulation run in the backbone architecture, we must start the Run Control subsystem and indicate which scenario we will run through a scenario object. The scenario object contains the values of important parameters of individual simulations, or common parameters that are shared between subsystems (see section 4.5). It also might contain information about the simulation treatment such as the simulation length. The main function of the Run Control is to execute a simulation run according to the specified scenario. Further activities that are performed by the Run Control subsystem during a simulation run involve:

- **Consistency checking**: the Run Control subsystem periodically checks every subsystem that is connected to the FAMAS Backbone System,
- **Information serving**: the Run Control provides useful information to other subsystems, such as the address of other subsystems, variable values, etc., from the Scenario Manager,
- **State logging**: sending special state information about the architecture to the Logging subsystem (see section 4.3).

The simulation process can be stopped through the Run Control subsystem in two different ways: stopping may be required by a subsystem at any point of time, or the run may be halted because of the run length specified in the scenario expires. The Run Control subsystem is implemented in Java 2 SDK Enterprise Edition and is based on a Client-Server Multi Threaded architecture.
4.2 The Backbone Time Manager

Another core element of the FAMAS Backbone Simulation architecture is the Backbone Time Manager (BBTM). Its main function is to synchronize the simulation time among different simulation subsystems. For the time being the architecture supports only discrete event based simulation.

The time manager currently implements two ways of synchronization, namely conservative and real-time synchronization. During conservative synchronization only one model is considered as “current” at any moment. All applications (models and backbone subsystems including Run Control) that are defined in the scenario of an experiment are considered participants and must be scheduled on the time-axis before a simulation run can start. The basic principle for synchronizing the participants on the same time-axis, using a conservative mechanism, is as follows. Each participant is assumed to send its first future event time (next event) to BBTM. BBTM selects the participant with the event with the smallest time and gives permission to perform this event. After performing the event the participant again sends its first future event time to BBTM and so on. Participants sending the same event time are handled according to the FIFO-sequence: the one who sent its event time first is allowed to proceed first. Nevertheless, using a conservative synchronization approach the simulation run is slowing down, but there are some advantages that we must take into account [Veeke et al., 2002]:

- Support for concurrent development (gaining time in designing complex simulation models);
- Support of platform independence;
- Full reproducibility can be maintained in a given start-up sequence;
- The same experimental environment can be used by different research projects.

In order to increase the speed of the simulation we have to use several approaches that can optimize the conservative synchronization. One of them is to minimize the communication by offering each model a time horizon and conditions under which it can act autonomously without consulting the BBTM.

The real time synchronization supports the experimenting with real equipment and/or control. When using real equipment, it is not possible to work in conservative mode as the time for real equipment cannot be stopped. Therefore, this synchronization is interval based [Verbraeck et al., 2000].

The BBTM is implemented in Borland Delphi environment. The BBTM has some capabilities for logging the connections and the frequency of communication with the subsystems that have communicated with the BBTM. Furthermore, the clock time and simulation time can be shown, as well as the ratio, which is 1.0 for real-time performance, and larger than one for faster performance.

4.3 The Logging Subsystem

The main purpose of the Logging subsystem is to collect data in a global data storage. Besides using the Logging subsystem, the components of the FAMAS Backbone System can store their information locally as well. We distinguish three mechanisms that can be used in the Logging subsystem: Log everything, Log only relevant data and Log at the end. These mechanisms differ from each other by the quantity and quality of the log of the information.

The logging data are stored in a fixed relational database. The data sent by various subsystems can be stored in one or more tables in the database. We distinguish two
important messages that will be sent to the Logging subsystem: one for initializing the table, the other one for sending information to the table. All the subsystems in the Backbone architecture can use a table initialized by a subsystem in order to store the specified type of information. At the end of the simulation run the Run Control is the last subsystem that is closed down in the backbone simulation architecture. The Logging subsystem waits until the last moment for the logging information. This means that it closes down immediately before the Run Control subsystem.

An additional application has been developed in Visual Basic for reviewing and analyzing the data. This application, called FAMAS Logging Viewer enables easy access to data by selecting the database and defining the required table.

4.4 The Visualization Subsystem

The visualization subsystem as it is discussed in [Boer et al., 2002] has two important goals:

- **Visualize the state of the simulation.** This enables the user to combine a number of views from different models or systems in one overview screen. The visualization may include snapshot screens, animation, statistics of various performance indicators and status views of equipment and processes.
- **Present the simulation results during and after the run.** In this case the visualization subsystem can be considered as an animation subsystem that can present as many instances of 2D or 3D viewing as necessary. There are possibilities to animate the whole distributed system in one screen or just part of it in different screens.

The presentation of the world consists of a fixed background and placed upon that, dynamic figures. It is up to the participating subsystems to fill the background and show the figures. The insertion of graphical objects to the background needs to be done only once and so it minimizes the communication. Defining objects in the background is very appropriate for non-moving elements. The Animation Subsystem collects the background objects and each time a screen needs to be refreshed, all background objects are drawn first.

**Figure 3: Screenshots of two different views of the map of port of Rotterdam**

Figures are used to show arbitrary shapes. Shapes are specified as wire frames with their own origin. The coordinates are specified first, and then the planes of the shape in terms of these coordinates. To increase the ease of use of the Animation Subsystem, a number of shapes can be predefined. It is to be expected, that different FAMAS.MV2 projects will use the same kind of equipment with the same appearance (e.g. AGV’s, Carriers and quay cranes).
During the simulation the camera can be moved interactively by using the buttons at the top of the animation screen (Figure 3). The user can move the camera in X, Y or Z direction and also turn it in all three directions. The visualization subsystem is implemented to be as general as possible, it is intended to wait for commands to show figures in a two or three-dimensional world.

![Figure 4: 2D animation offered by eM-Plant](image)

Besides the global animation offered by the Visualization Subsystem, there is a possibility to use the animations of the individual simulation models as well (Figure 4). In this manner we can have a global animation that animates the process of the whole model on a high level and we can have animations provided by the simulation packages that animate the detailed process of the submodel.

### 4.5 The Scenario Management

A scenario completely defines a simulation run of a distributed model. Each scenario has a unique identifier. The Run Control subsystem is responsible for the interpretation and execution of a scenario object.

A scenario object consists of three parts: A Variable Declaration section, a Scenario Script Section and an Initialization Script Section.

1. **The Variable Declaration section** defines the values of parameters for the simulation run. The variables can be global, local, static or dynamic. A static variable cannot change during the simulation run, while a dynamic variable can be updated during the run, allowing other models to be informed about the new value.

2. **The Initialization Script section** defines the 'set-up' of a simulation run, it identifies which technical and functional subsystems are used.

3. **The Scenario Script section** finally defines how the simulation process is executed. It can define events in (simulation-)time that the Run Control subsystem must execute. The Run Control behaves like a participating simulation model that has to react to events on the global event calendar.

The Scenario Creator is a separate user-friendly tool implemented in Borland Delphi, which creates, stores (in a single file) and retrieves simulation scenarios.
4.6 The Functional Subsystems

The functional subsystems (or functional components) in the FAMAS Backbone System are the ‘core’ of the simulation study itself [Boer et al., 2002]. The problems that need to be analyzed by the simulation study are described in the functional components. These are intended to be general and can be developed by different groups in different simulation packages or programming languages. These components should run as standalone models in their environment where they are built or should couple to the backbone system without or with minimal user interference. In order to combine different simulation models effectively and easily, standardization and ease of use are also extremely important.

In order to prove the concept of generality of this architecture and to carry out the assigned aim for the technical subsystems we use different environments (Java, Delphi, Visual Basic and Visual C++) and for the functional subsystems three simulation packages, namely TOMAS, eM-Plant and Arena.

In order to connect such a subsystem in an arbitrary simulation package to the backbone, we have to be sure that the simulation package can communicate with the FAMAS backbone system. In many cases these commercial simulation packages are closed, and sometimes it is difficult to get the needed internal information, such as the time of the next event. Therefore, first it is essential to create interfaces for the subsystems that connect the simulation language or programming language to the backbone. For simulation languages, we have implemented these interfaces as DLLs that act as a kind of wrapper to the simulation package: on one side, they interface to the simulation language, and express the information in the standards of the simulation package, at the other side they conform to the backbone language and structure (see Figure 2). The DLLs and methods of the interfaces are seen as an important part of the backbone system, although they run on the same computer as the corresponding simulation package.
5 Testing the FAMAS Simulation Backbone Architecture

The idea behind the simulation backbone project is to create a harmonized simulation environment that links different simulation models into one experimentation and demonstration environment. Several research groups develop different simulation models of the new part of the port that will host the container terminals. Based on their experience, these modeler groups develop their own simulation models in different commercially available simulation packages or programming languages. The advantages offered by simulation packages increase the effectiveness of the individual simulation studies but also pose a problem for the entire project. The final results can only be achieved when simulation models of different areas in the port are combined into a larger simulation model.

The backbone structure has been tested with the MICL (Maasvlakte Integral Container Logistics) model. The MICL model is a rough model that contains the overall functionality of the entire Maasvlakte container port on a high abstraction level. The ITT (Inter Terminal Transport) functionality has been taken out of the MICL model, and in the MICL model, incoming and outgoing messages have been defined to indicate the need for an ITT vehicle, and the arrival of an ITT vehicle at the destination terminal. Now, the so-called ITTMICL model can be coupled to the backbone separately, as long as it can handle the two mentioned messages. Internally, the ITTMICL model can be as simple or as complex as the modeler wants.

Several versions of the ITTMICL model have been developed: one in TOMAS – containing the original functionality, as the MICL model had also been developed in TOMAS. Another model with flexible lay-outing has been developed in eM-Plant. In this model the ITT AGV’s can have any desired functionality, because they have been modeled as separate objects. As could be expected, the TOMAS – TOMAS connection via the backbone worked fine. Furthermore, the TOMAS models interface very well with all technical subsystems.

The eM-Plant model has been connected to the backbone using the specially developed eM-Plant backbone wrapper. For eM-Plant the wrapper is so generic, that no special programming in the wrapper needs to be done to handle the two messages that are directly exchanged with the MICL model. The eM-Plant – TOMAS interface via the backbone works fine. The eM-Plant wrapper also takes care of the communication with the BBTM, Run Control, Visualization, and Logging technical subsystems. This was tested and proved to work very well, except for one point. Because the future event list cannot be read in eM-Plant versions 4 and 5, a truly conservative time mode is not possible. Instead, the timing is based on small time-steps that are exchanged with the BBTM. A simpler implementation of the ITTMICL model has also been made in Arena. Here, the layout of the terminal structure is fixed, and cannot be parameterized through initialization files. Furthermore, the intelligence for the two messages to be exchanged with the MICL model is programmed into the Arena backbone wrapper. The Arena wrapper also takes care of the communication with the technical subsystems. Because the code for the eM-Plant and the Arena wrappers is for 90% the same (only the specific interface with the simulation language differs), the tests immediately showed positive results. The timing is based on small time-steps here as well, although the Arena event list can be read from the C++ wrapper. This is a point for later extensions.
For each of the combinations (MICL-TOMAS, MICL-eM-Plant, and MICL-Arena), simulations were run to look at speed, possible problems, and differences. The tests proved to be positive for all aspects.

In the tests, it was shown that the functionality of the backbone architecture worked as intended. Especially for TOMAS and eM-Plant models, the coupling is easy and straightforward. For Arena, a little more effort needs to be done because the C++ Arena wrapper needs to be tailored for specific needs for each project. Probably this can be brought ‘closer’ to Arena by moving this functionality into the VBA part of Arena.

On 19th of April a final demonstration test has been done for different groups from different companies involved in the FAMAS project. The test was presented in the SimLab, where all the technical and functional subsystems run on separate computers.
6 Conclusion

The tests show that different simulation models can be easily coupled to the backbone structure by forming a virtual design environment. The technical components make it easy and straightforward to set up a complex federation of multiple simulation models. The architectural structure makes it possible to reuse all kinds of models in the simulation studies, even on the level of resources such as crane models or AGV models.

Simulation languages that can already be coupled to the backbone are Tomas, eM-Plant and Arena. Arena still needs a little extra work, but that can be improved in the future. The wrappers used to connect models implemented in these languages to the backbone are built in a way, that other simulation languages can also be interfaced to the backbone, as long as they allow for a DLL to be integrated with the simulation environment.

The MICL tests with the separate ITT model show that the concept of the backbone works both technically and functionally. The technical subsystems provide added functionality, while the interfacing of functional simulation models to the backbone can be done with no (TOMAS), negligible (eM-Plant), or minimal (Arena) effort.

Concluding this article we evaluate the FAMAS Backbone Architecture based on different criteria:

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Current backbone status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flexibility</td>
<td>Models can run within the backbone or outside; the environment can be extended with more models in multiple packages.</td>
</tr>
<tr>
<td>Manageability</td>
<td>Model changes can be done within a model, without changing other models; scenarios can be set at one entry; data is collected at one single point; visualization is distributed and centralized.</td>
</tr>
<tr>
<td>Innovation</td>
<td>For multiple parallel simulation projects, platform provides possibility to experiment in a uniform environment (scenario, output), without obliging projects to use a certain package. The multi-screen visualization enables demonstrations of complex multi-model simulations. However, models have to be developed yet.</td>
</tr>
<tr>
<td>Industrial applicability</td>
<td>Build on top of standard protocols and an architecture following rules of enterprise application integration.</td>
</tr>
<tr>
<td>Flexible modeling detail</td>
<td>The simulation models can be built at a high or detailed level and interfaced to the backbone, even in a mixed situation.</td>
</tr>
<tr>
<td>Quality of visualization</td>
<td>It offers a global 3D animation (visualization subsystem) and the animations provided by the simulation packages.</td>
</tr>
<tr>
<td>Easy to control</td>
<td>Due to the technical components (especially Run Control) the distributed system is controllable with minimal effort.</td>
</tr>
<tr>
<td>Cost</td>
<td>Linking to the backbone requires some adjustment in the simulation model, but reduces effort for output analysis, visualization and scenario management. Furthermore for the entire project it reduces the effort spent on basic model development.</td>
</tr>
</tbody>
</table>

Further activities will focus on the comparison of our approach with the HLA (High Level Architecture), which is an IEEE/DoD standard for interfacing simulation models in the military field, and allowing for coupling it to the backbone. Furthermore, besides the simulation models we would like to analyze how the FAMAS Backbone Architecture supports the coupling of real equipments and controls.
References


