Adapting discrete-event simulation tools to support tactical forecasting in the automotive industry

Author(s):
Steinemann, Adrian; Taiber, Joachim; Fadel, Georges; Wegener, Konrad; Kunz, Andreas

Publication Date:
2012

Permanent Link:
https://doi.org/10.3929/ethz-a-007250049

Rights / License:
In Copyright - Non-Commercial Use Permitted
ADAPTING DISCRETE-EVENT SIMULATION TOOLS TO SUPPORT TACTICAL FORECASTING IN THE AUTOMOTIVE INDUSTRY

Adrian Steinemann  
Innovation Center Virtual Reality  
ETH Zurich  
Switzerland  
steinemann@iwf.mavt.ethz.ethz.ch

Joachim Taiber  
The Holcombe Department of Electrical and Computer Engineering  
Clemson University  
United States Of America  
jtaiber@clemson.edu

Georges Fadel  
Mechanical Engineering Department  
Clemson University  
United States Of America  
fgeorge@clemson.edu

Konrad Wegener, Andreas Kunz  
Institute of Machine Tools and Manufacturing  
ETH Zurich  
Switzerland  
{wegener, kunz}@iwf.mavt.ethz.ch

ABSTRACT

The automotive industry is an example of flow production and manufacturing efficiency. Its performance is based on taking the right decisions on strategic, tactical, and operational processes by the responsible engineers. Over the last decade, discrete-event simulation became the tool of choice to verify strategic decisions and tactical measures in the production process. However, the verification process itself is tedious and the usage of the tool is complex. Therefore, the task of verifying suitable solutions is accomplished by simulation experts and limited to high priority projects. For the optimization of operational processes, companies established lean manufacturing initiatives with real-time monitoring systems to measure production processes and to reveal bottlenecks. However, the improvement process is often based on trial-and-error and could be heavily supported by extending the current tool set with discrete-event simulation. Based on a use case of an assembly line in the automotive industry, we propose an approach in which running production environments benefit from simulation experiments without any intensive support by simulation experts. The goal is to extend the current tool set and to simplify the usage of discrete-event simulation tools, so they can be applied by production engineers on a daily basis. This would support their decisions on tactical measures and thus, prevent system instabilities and blocked line conditions.

KEYWORDS

Manufacturing, Discrete-Event Simulation, Line Balancing, Monitoring, Visualization, Interaction, Collaboration, Single Display Groupware

1. INTRODUCTION

Discrete-event simulation (DES) has grown to become the standard tool in the automotive industry to support the evaluation of new production line installations. In the early phases of such factory planning projects, planning and simulation engineers are an important part of the project team and take on the comprehensive responsibility of leading the way towards efficient production structures with their expertise in optimization methods and simulation paradigms. Besides these strategic projects mostly related to early phases of the production process life cycle, DES is also used for major adaptations of existing assembly lines and for specific optimizations related to real-time production. Here, planning and simulation engineers are usually involved as experts with a well-defined task. The expert team goes
through a standardized process of getting familiar with the specific use case, collecting relevant data for the simulation, updating existing models or building up new ones, and verifying and validating them before running the actual optimization experiments. In the end, the team reports the results and makes suggestions on how to proceed. As this process is time-consuming and costly, many companies limit the usage of simulation resources to high priority projects. Nevertheless, the benefit of simulation is evident and thoroughly discussed in the literature.

2. RELATED WORK

Rabelo et al. [17] provide some background on discrete-event simulation and describe its main focus, i.e. the design of new facilities, the evaluation of system improvements, as well as planning and scheduling decisions. They suggest considering performance, robustness, and stability as measures during an optimization process. Additionally, they point out some shortcomings of current enterprise simulation frameworks mirroring hierarchical aspects of an enterprise. In addition, Sanchez [19] discusses the value of robust design and stresses the importance of simulation sensitivity analysis, together with model verification and validation. Whereas, Carson et al. [5] and Swisher et al. [24] cover the field of simulation optimization.

Besides a discussion on measures and achievements improving production throughput, Alden et al. [2] address an approach promoting methodical consistency across an organization. This allows disparate user groups, i.e. simulation engineers and production engineers, to access identical analysis capabilities and thus, to create comparable simulation results.

In their book, Bayer et al. [3] give a comprehensive overview on the application of simulation in the automotive industry. However, despite a high acceptance of simulation tools, they mention drawbacks like insufficient standards, which hinder the efficient implementation of simulation technology. Furthermore, Wood et al. [26] discuss possibilities to reduce human errors in simulation and the importance of standardization to ensure the quality of the results.

In literature, simulation, whether it is related to the automotive industry or specific on optimization, is applied over the whole span from strategic to operational tasks and is always used as a tool for experts. No approach can be found to apply tools as a support for tactical decisions by non-experts, such as e.g. line responsibles.

3. CONTRIBUTION

As mentioned above, planning and simulation engineers take on a comprehensive responsibility in strategic projects related to early phases of the production process life cycle. For production related projects however, these engineers are usually consulted by the project management team and offer their expertise as a service. In general, the closer projects are related to real-time production, the more planning and simulation engineers are replaced by production experts. On the one hand, this means an increase in production expertise, but on the other hand, often also a lack of simulation knowledge (see Figure 1). In addition, during the testing phase and after a set of successfully conducted trials, the project lead is usually transferred to the responsible production team. This process can be described as a change from a strategic towards a tactical and operational focus.

![Figure 1](image-url)
valuable solutions. In current working situations within the automotive industry, this discrete event simulation cannot be accessed by the line responsibles due to missing expertise, but also due to a missing adapted interface for the visualisation of the simulation results.

Based on this in-depth analysis and evaluation of a use case in the automotive industry, we propose an approach that supports production engineers to take tactical and operational decisions. Therefore, we aim to extend their current tool set with a customized environment based on simulation, visualization, interaction, and remote collaboration possibilities. This would extend today’s state of the art with a new system supporting production engineers in their daily work with simulation capabilities without being simulation experts. In section 2, we present our overall approach with its four components. In section 3, we discuss the use case and the relevant processes. Section 4 presents the details on the design of the simulation model, while sections 5 & 6 discuss the software architecture and design. In the end, section 7 presents a prototypical interface and section 8 concludes the paper and points out our future work.

4. OVERALL APPROACH

Based on the analysis of today’s best practices related to simulation in the automotive industry, we see the potential of developing an approach, such that running production environments directly benefit from simulation experiments without an intensive support by simulation experts. Today, the usage of discrete-event simulation tools is mostly limited to high priority strategic projects, due to the need for the continuous involvement of simulation engineers. Although many companies installed monitoring systems to track all occurrences in their production systems, the optimization processes are mostly limited to trial-and-error procedures without any simulation support for the involved production engineers. Thus, our approach distinguishes between projects of high priority requiring extensive and accurate simulation results supported by experts, and constraints out of the daily routine of a production environment where also restricted simulation models are able to heavily support the existing optimization processes. Our approach thus simplifies the usage of DES tools in such a way that they can be applied by production engineers on a daily basis. This supports their decisions on tactical measures like product sequencing and work content distribution and thus, prevents system instabilities and blocked line conditions.

Overall, we focus on the development of a support environment called SAMSON (Simulation Applications for Mechatronic Systems-Engineering over Networks). Its main characteristics are:

- Adaptive simulation models, which can interoperate with the installed monitoring tools and use them as a data source.
- User-friendly interaction methods to support ease of use.
- Suitable visualization capabilities to enable virtual walkthroughs.
- Possibility for live collaboration with remote simulation experts for technical simulation support, with other production sites as methodical reference of a process, or even with development engineers for product specific optimizations.

Figure 2 SAMSON hardware environment to support local and remote collaboration for tactical forecasting.
Altogether, this results in an environment providing a significant improvement potential by aggregating small but continuous process modifications and combining them with the necessary improvement methodology.

First of all, our proposed approach requires an in-depth analysis of the specific use case. This analysis needs to identify the critical production processes, their parameters, the constraints on these processes, and the general objectives of the company as related to these processes. The next step is to adapt existing simulation models to the specified environment and to tune them towards a tactical forecasting of the identified objectives. Finding the right level of detail and leading the simulation model towards an adequate and robust solution is a cyclical process. The implementation procedure thus consists of the description of the project to clarify the purpose of the simulation, the acquisition of data out of the monitored production system, the development of the model itself, and its verification. Once this process is concluded, it is possible to restrict the number of variables, and thus to provide the possibility of executing simulation experiments on a non-expert level, to connect this tool to an optimization engine, and to analyze its results for identifying valuable improvement scenarios. However, the in-depth analysis and the adaption of the simulation model are only the first of a number of steps needed to reach the objective of efficiently supporting production engineers.

The second keystone of our approach is to concentrate on an intuitive interaction concept and front-end application as known from Single Display Groupware (SDG) environments such as Microsoft Surface [13], ThinSight [8], or MightyTrace [9]. These tabletop systems replace the interaction using keyboard and mouse by the usage of pens, Tangible User Interfaces (TUI), and direct touch with fingers. Furthermore, they adapt the Graphical User Interface (GUI) to match their multi-user capability and the intended ease of use. As discussed by Stewart et al. [23] and Shen et al. [20], this enables the user to concentrate on the discussion without being distracted by handling the system. Based on the interaction concept of such SDG environments, our goal is to develop an application which is designed, in our specific case, along the process of balancing production lines considering the given production steps and their assignments to various stations. Thus, the simulation suite used by experts like simulation engineers operates in the background and provides just the needed information directly to the newly designed front-end application. This front-end application only provides a subset of the simulation suite’s input capabilities, which are relevant to the production engineer. Together with intuitive interaction devices it is thus adapted to the requirements of tactical decisions for the production line.

Figure 3 3D Blender model [4] of a production line
The third fundamental component of SAMSON is the virtual representation of the actual production floor. This representation needs to be implemented as an interactive environment which shows all relevant occurrences and events of the production line (e.g. carriers, products …). Using a game engine like the one of Blender [4], simulation tool events can be translated into movements within a virtual environment, and thus the simulation runs can be augmented into a spatially correct representation. In addition, the game engine offers the possibility to perform virtual walkthroughs of the production floor. This supports the usual inspection walkways of production engineers and thus enables the possibility to compare simulated situations with well-known occurrences from everyday routine. Since the production engineers are not used to the abstract user interface and data representation of DES software, this eases detailed discussions on the introduction of new products, the distribution of new work content, or line balancing measures by concretizing the results of simulations, conceptual trials, and experiments (see Figure 3).

Bringing the intuitive interaction concept, the front-end application, and the visualization together, we have a collaboration environment in which several users interact with a system offering direct control to a simplified simulation front-end and an interactive virtual environment. The SAMSON hardware
equipment is similar to known systems like BUILD-IT [18]. The components are a tabletop multi interaction device for navigation within the bird’s view of a 3D-world and the configuration of simulation experiments, and a viewer’s perspective in this virtual environment being visualized on a vertical screen. This setup is also combined with a video conferencing solution offering the possibility to meet with remote partners (see Figure 2). Unlike normal video conferencing, all partners will see the same situation on the horizontal and on the vertical side screens as well, and they are also able to interact with the objects to explain certain solutions and ideas during the net-based team sessions. For instance, it would be possible to place a virtual camera in the scenario in such a way that both sides see a certain position of the production line, which might be very relevant for the ongoing discussion. Especially for the globally distributed production in the automotive industry, such scenarios are very helpful, since the required expertise could be easily connected from another location without impeding the short-term decision process within the tactical planning scenario.

5. USE CASE DESCRIPTION

In their work, Williams et al. [25] and Patel et al. [14] present the usage of a simulation based on two different use cases in the automotive industry. Williams et al. discuss the processes of engine pre-assembly and the preparations needed to distribute the products to the final-assembly plants. Patel et al. illustrate the final process system, which is an important part of the quality assurance in the automobile manufacturing process. Since our use case is based on an engine final-assembly in an OEM plant, it matches the applications of the above mentioned papers. To protect the confidentiality of the assisting company, all mentioned numbers both in the text and on the pictures are hypothetical, but illustrate the main challenges for an improvement process.

The engine final-assembly is physically located ahead of the power train assembly and the marriage area, where the power train joins the car body which is proceeding along the main-assembly. The engine line consists of four major processes. These are the sequencing out of the local storage matching the sequence on the main-assembly line, the core task of assembling the random mix of several engine types and available options, the final quality inspection, and the interruption-free supply of the subsequent power train assembly. All needed steps take place in a machine-aided, but manually operated flow production on mainly single process assembly stations. Furthermore, several FIFO (first in – first out) buffers split the line into individual segments to support system stability. Nevertheless, as these buffers do not allow any re-sequencing, engine and power train assembly are still directly connected to the main-assembly, and thus imply the need for sequence synchronization (see Figure 4).

Detailing this production flow, the following operations take place: at the very beginning, one pre-assembled engine is taken out of a transport rack holding several engines at a time. Using a gantry crane, the engine is lifted onto a carrier, which guides it through the assembly line. Within the first section, parts like the transmission and the power steering pump are mounted onto the engine. In the last section, the assembly is completed with the programming of the engine control unit and the quality inspection. The carrier then moves through the final buffer and delivers the engine to the power train assembly. Once the engine is delivered, the carrier travels back to the start of the line and queues up to receive the next engine. Throughout the whole line, the carriers move automatically to the next station controlled by their integrated distance measurement system. Nevertheless, once the carrier is at the station, it will not move until the worker releases it by pulling a trigger. In case of exceeding the predefined cycle time margin, the carrier beeps to signal an ‘overcycle’. If the carrier is released before reaching the cycle time margin, the work step is considered as an ‘undercycle’. Additionally, from a modeling point of view, stations are considered to be ‘working’, ‘blocked’, ‘starved’, ‘paused’, or ‘failed’.

![Figure 4 Illustration of production flow.](image-url)
The actual work content is variable through several engine types, available options, changes in the product mix, product adaption, new types running into production, or through running out processes. Several weeks in advance, the production planners begin to calculate expected demands, which are finally fixed in a specific sequence a couple of days before the actual start of production. Given these demands or sequences, the line balancing can be adapted. This means that the given work content is spread as uniformly as possible over the assembly stations considering worst case scenarios, option configurations, and given constraints through machine dependent manufacturing tasks. Once the production is started, the line manager is responsible to control the system based on their experience and expert knowledge. Thus, the line manager needs to walk through the line and keep an eye on possible indicators for production constraints. A short-term reaction on this operational level could be to double the manpower on a specific station to avoid or overcome a blocked line condition. On a tactical level, the line manager uses the installed monitoring tools and the provided data. This leads to a typical procedure of finding bottlenecks and constraints, analyzing buffer effectiveness, as discussed by Li et al. [12], and to improve the overall equipment effectiveness by reducing speed losses, down time losses, and quality losses.

With regard to this use case and its high variance of work content per product on each assembly station, a standardized tool, supporting production engineers in their daily challenges, becomes indispensable. This tool would have its highest impact evaluating line balancing configurations and supporting engineers on tactical decisions to prevent system instabilities.

6. SIMULATION MODEL DESIGN

As mentioned before, many simulation projects are related to high priority strategic projects and are conducted with the continuous involvement of simulation engineers. Nevertheless, as the resulting models are specifically designed to support a limited optimization task (e.g. restructuring of an assembly line), in most cases, they do not support the daily use by production engineers. On the one hand the models are not necessarily designed to simulate generic work content shifts or sequencing variances, and on the other hand they only implement parameter settings on an expert level for extensive and accurate simulation runs. Thus, once such a high priority simulation project is completed, the resulting models need to be simplified to an adequate level of detail and modified towards the main use case of the daily business on a non-expert level. Otherwise, the models could not be used to offer a better basis of decision-making during daily operations and the efforts of model building would remain limited to the high priority projects. This change of objective would also heavily impact the prevailing analysis and improvement processes. Production engineers could analyze daily challenges (cf. below listed items) themselves without spending lots of resources for simulation projects:

- Different viewpoints and possibility to better understand and get a feeling for the behavior of the line itself and to recognize the main reasons of particular events.
- Use provided information from production planners to simulate line balancing configurations based on the upcoming production demands.
- Find ways to raise system stability and robustness against blocked line conditions through flexible distribution of work content.
- Evaluate new line concepts to minimize constraints influencing overall equipment effectiveness.

Using the software tool ‘Plant Simulation’ from Siemens (former eM-Plant) [15], we focused on a model design reflecting the actual situation on the line. Our goal was to create a standardized model. First of all, it is crucial to determine the model’s appropriate level of detail depending on the intended application. This is the key to find the right balance between potential and simplicity of a simulation model.
sequencing and line balancing through work content distribution. These are driven by the goal to fulfill the scheduled production volume and deliver the engines to the power train assembly within the predefined tact time using as little resources as possible.

The engine final-assembly offers all the data which is needed to feed a discrete-event simulation model. On the one side, this is planning data which can be used as an input for tactical forecasting experiments. As previously mentioned, this data covers production volumes and demands known up to one month ahead of schedule, real production sequences, engine type specific work content distributions, option lists, buffer sizes, and production volume targets. Al-Aomar [1] classifies this data as controllable factors contributing to variation in simulation outputs. On the other hand, the installed monitoring system provides live-data which is used as initial and boundary conditions or for verification of the tactical forecasting model. Here, examples are real-time status information of all assembly stations and present buffer levels during start-up, running conditions, and shut down (compare with artificial factors described by Sanchez [19]).

The intended application of sequence variation and line balancing through work content distribution requires a representation at a high level of detail. Based on the discussed data set, we therefore decided to implement a model which represents every single station and all existing buffers. However, the work content on each station is summarized and strictly limited to production time input with an underlying distribution which ensures the model’s flexibility (see Figure 5).

Our goal was to represent the line behavior in as much details as possible. Thus, we put a lot of effort on its modeling. On the one hand, the system behavior is reflected in a method set handling the carriers’ movements entering and exiting stations or buffers. On the other hand, the work time variance on each station is represented by an engine type and option sensitive distribution based on historical data records, described as random factors by Al-Aomar [1]. The movement handling methods together with the work time distribution build the core of the simulation model.

Subsequently, a proper verification process is needed taking historical product sequences, adjusting the system variables, and running trials comparing the results with the real values of buffers, production volumes, process lead times, and exit rates towards the power train assembly.

In the end, projected demands, product mixes, and lists of engine types and options can be used as input data. Furthermore, the simulation model allows adjusting the system variables like the number of carriers, travel times between stations and through buffers, buffer capacities, or shift models to fine-tune the model and to replicate the historical data on hand. The goal is to support production engineers during tactical fine-tuning procedures of their production line by an output which provides them with a robust basis for decision-making.

Therefore, we implemented a first graphical user interface (GUI) (see Figure 6) which includes a subset of changeable parameters. This selection covers the essential inputs like settings for product sequences or demands, system variables like number of transporters, buffer capacities, travel times …

Figure 6 Menu window as graphical user interface covering the essential inputs for setup variables (production sequences, demands, number of transporters, buffer capacities, travel times …)
Furthermore, we want to use the simulation data generated by the DES software to animate a virtual representation of the actual production floor. Therefore, we need a way to collect the information of any object movement within the system, to serialize it, and to transfer the information to the SAMSON intercommunication software, which will be presented in the next section.

An elegant solution is to attach an observer code to every single transporter and to all products entering the system, which will generate an event as soon as the observed position parameter changes. This event then executes a script which reads the necessary object information: ID, type (product or transporter), and its location (x and y coordinates, station numbers for transporters, and transporter numbers for products). The script then serializes this information and passes it to the TCP socket tool of the Plant Simulation suite [15] which has been connected with the SAMSON intercommunication software beforehand. The TCP socket then sends off the serialized package containing the specified data.

7. INTERCOMMUNICATION SOFTWARE

The SAMSON environment is based on the components of simulation, visualization, interaction, and collaboration. Therefore, we need an intercommunication concept to exchange data between the discrete-event simulation suite (Plant Simulation [15]), the virtual environment (Blender Game Engine [4]), and the interactive hardware (MightyTrace [9]). Although remote collaboration partners are connected with a standard video conferencing solution for the video and audio channel, the intercommunication software still must support the connection on a data level to synchronize multiple SAMSON environments. In the end, the overall goal is to enable a seamless and intuitive interaction.

Based on the programming language Python [16], we designed SAMSON as central server software (SAMSONintercom) with the ability to communicate bidirectionally with its several client modules at the same time (see Figure 7). The SAMSONintercom module therefore uses the Python package ‘threading’ which constructs higher-level threading interfaces. The client modules are divided into simulation, visualization, and interaction pieces (SAMSONsimulation, SAMSONvisualization, and SAMSONinteraction). In case of a multi-site collaboration, the central server piece supports also the communication with multiple visualization and interaction modules. On the other hand, the simulation module remains a unique component and builds a unit with the central server module.

The communication within the SAMSON environment takes place as a bidirectional exchange of objects using TCP sockets. These objects are centrally generated within the SAMSONintercom module based on the factory method pattern and stored within a central object repository, as well as broadcasted to all clients, and stored within the clients’ local object repositories. The communication protocol is based on serialization using the Python package ‘cPickle’ which takes care of the serialization and de-serialization of Python objects.

Once the SAMSONintercom module is started, it takes care of handling incoming requests from any type of client. For each client, it starts a new thread and calls the implemented ‘run’ function of the thread server. This thread server then manages the connected clients, the object factory, and the central object repository.

The object factory builds all requested objects along predefined templates. These templates support the object types ‘environment’ (related to the resolution of connected hardware), ‘simulation’, ‘visualization’, and ‘interaction’. The basic template contains the variables ‘Factory Version’, ‘ID’, ‘Name’, and ‘Thread ID’, as well as ‘set’, ‘get’, ‘serialize’, and ‘de-serialize’ functions. On the other hand, the object repository is implemented with the functions ‘new’, ‘set’, ‘get’, and ‘remove’. Furthermore, it holds and manages a dictionary of all existing objects. As an example, we want to illustrate the communication and performed procedures within the SAMSON environment, when a new engine is entering the simulation model.
• The simulation client requests a new simulation object at the communication server. Once this request is identified by the server, the object factory generates a new simulation object with a unique ID. This object is sent back to the requesting client. The client updates the object with the latest information, stores the object in the local repository, and sends an update to the server. Once this update is identified by the server, the object is copied into the central object repository, and the server broadcasts the update to all other connected clients. Finally, these clients store the object in their local repository and eventually trigger some internal functions dependent on the received update.

8. VISUALIZATION CLIENT

As mentioned before, we decided to use the Blender Game Engine as the tool of choice to build up the virtual environment. Blender [13] is a free open source 3D content creation suite which supports modeling, rendering, and animation. It also includes a game engine supporting collision detection, dynamics simulation, and a Python scripting API. Based on this API we implemented the SAMSON visualization module as a master script which is connected to the corresponding and previously designed virtual model. First, this master script starts the TCP communication with the SAMSON intercom server module. Usually, TCP sockets are implemented as blocking sockets. Thus, the execution of a script stops at the point where the socket is instructed to receive data and waits for the communication from the connected counterpart. As the Blender Game Engine works with an internal tick rate and repeatedly triggers the implemented Python scripts, any software running within a Blender session has to be developed as absolutely non-blocking. Otherwise, the executed Python scripts containing ‘while true loops’ or blocking sockets would lead to an interruption in the execution routine and freeze the Game Engine.

Now, once the TCP connection is established and up and running, the script is ready to exchange serialized object messages according to the predefined protocol with the SAMSON intercom module. Thus, besides of the ‘send’ and ‘receive’ functions, our master script implements also a local object repository which holds and manages a dictionary of relevant objects. Therefore, the object repository interprets the incoming messages and executes the implemented functions ‘newObject’, ‘setObject’, ‘getObject’, and ‘removeObject’. Afterwards, the object repository notifies the affected virtual objects within the Blender’s built-in messaging system.

Besides the built-in tick rate and the messaging system, the Blender Game Engine is also based on a structure using ‘logic bricks’. These logic bricks are divided into the types ‘sensor’ (reacts on any change of the observed parameter and triggers the associated controller), ‘controller’ (waits for incoming triggers, combines them in case of multiple sensors, and triggers the associated actuators), and ‘actuator’ (performs actions like moving objects). Based on these logic bricks, we implement every single virtual object within the 3D Blender scene with a ‘message’ sensor to enable the connection to the above mentioned messaging system. Thus, every single virtual object is able to listen for their messages. Whenever the ‘message’ sensor receives a message, it triggers the appended controller (see Figure 8). To implement the actual movements of carriers and products received from the discrete-event simulation tool, we currently work on specific action scripts. These scripts would be triggered by the controller and thus implement the ability to initiate predefined actions like moving a carrier to the next station.

9. CONNECTING SINGLE DISPLAY GROUPWARE AND BLENDER

Since the publications of the first multi-touch interaction system using frustrated total internal reflection (FTIR) [7] to track interactions, and subsequent adaptations like [22], the public attention for the field of Single Display Groupware (SDG) and multi-touch interaction started to rise. Subsequently, many do-it-yourself tutorials on how to build low-budget multi-touch systems have been published, and the number of lifestyle and home entertainment demo applications, like sorting picture galleries and playing games, started to grow. Lately, former SDG systems with multi-interaction capabilities like DiamondTouch [6], reacTable [10], or Microsoft Surface [13] have been followed by two promising systems, i.e. ThinSight [8], and MightyTrace [9]. Both are integrating their tracking technology into
commercial liquid crystal displays (LCD), and thus drastically reduce their space requirements. Some recently published work like [27] also conveys a trend to support industrial-oriented tasks. As mentioned before, within the SAMSON environment we aim to support both, local and remote collaboration and to benefit from existing hardware infrastructure for multi-user interaction. Regarding the circumstances of distributed meetings, we aim to support the separated teams with a virtual line representation in the connected environment, its available interactive surfaces, and all occurring interactions. On the other hand, typical engineering sessions cover a large number of grabbing and displacing interactions to rearrange a scene. Thus, we concentrate on a combination of a 2D and 3D scene representation.

Based on the MightyTrace [9] tracking technology, we thus implemented a prototype which acts as a direct interface between the chosen SDG hardware system and the 3D creation suite Blender [4]. This prototype does not yet support the interaction with the Blender Game Engine, and thus, is not yet part of the previously presented SAMSON software either. The interface is based on the standard Blender API and implements static and dynamic associations between real and virtual objects, i.e. between tangible user interfaces (TUI) or pens and their virtual counterparts, or with other objects like machines and conveyors in case of a dynamic association.

The basic tasks of the interface are to exchange interaction data with the SDG system, to maintain an actual list of all active objects with their ID, position, and state, and to modify objects of the attached Blender scene. Therefore, we divided the interface into the components ‘Protocol Parser’, ‘Object Repository’, and ‘Modifier Scripts’ (see Figure 9).

After setting up a socket connection to the controller software of the SDG system, the protocol parser takes care of interpreting the received messages. The implemented transfer protocol is based on event messages with the information of the event type (device activated, position update, state update, device deactivated), the device ID, the device’s state (button pressed, hovering), and the device’s position on the interactive surface.

The object repository is responsible to keep track of the pens and TUIs on the SDG system, and to store their positions, their states, and their associations with virtual objects within the 3D scene. Additionally, the repository can be extended to hold a history of performed interactions, and thus to support modifier scripts with a need to interpret previously performed interactions.

Finally, the modifier scripts implement the direct connection to the attached Blender scene. Therefore, they offer functions to associate or disassociate TUIs and virtual objects. This association can be either static or dynamic.

The static association is used to visualize any interaction performed on an interactive surface. Thus, the Blender scene shows a virtual SAMSON collaboration environment with a realistic representation of pens and TUIs (see Figure 10). In the end, we get a collaboration environment with virtual objects moving around as if guided by an invisible hand. In combination with the implemented paint function, we even get the same strokes as the user draws at the real SDG system.
match the location of the performed pick event with a nearby virtual object.

10. CONCLUSION AND OUTLOOK

Although discrete-event simulation is widely used in the automotive industry, most simulation projects are related to early phases of the production process life cycle or restricted to support only high priority optimization projects with a tactical or operational focus. Nevertheless, the benefit of simulation is evident, although the process is time-consuming and costly. To support the optimization of operational processes, however, many automotive companies established lean manufacturing initiatives using real-time monitoring systems which measure production processes to reveal bottlenecks and production constraints. As here, the actual improvement process is often based on trial-and-error. This process can be heavily supported by extending the current tool set with discrete-event simulation to improve the overall equipment effectiveness.

Based on a use case of an assembly line with high sequence and work content variance in the automotive industry, we presented the SAMSON environment, which is based on the integration of restricted simulation models for a non-expert level, and which aims to support simulation experiments within the improvement process performed by production engineers. The fundamental components of SAMSON are the interoperability with the installed monitoring tools using them as a data source, the user-friendly interaction methods to support the ease of use, suitable visualization capabilities to enable virtual walkthroughs, and the possibility for live collaboration with remote team members. In combination with the adequate improvement methodology, this environment opens up significant improvement potential through continuous process modifications.

To conclude our paper, we want to point out our next steps. The SAMSON intercommunication software builds the core of the environment and is based on the Python programming language [16]. As the seamless integration is an important issue, we plan to compare the discrete-event simulation tool Plant Simulation from Siemens [15] with the open source project SimPy [21]. SimPy is an object-oriented and process-based discrete-event simulation language based on standard Python and thus represents the counterpart to SimTalk which is used within Plant Simulation [15]. Since the DES tool connected to SAMSON only runs in the background, we do not depend on any front-end GUI at all. Furthermore, we will concentrate on the development of the actions scripts to animate the basic movements of the virtual objects within the Blender scene to represent the actions received from the discrete-event simulation. In addition, we will focus on the intuitive front-end application mentioned in section 2, which represents an important part in the work flow of the line balancing through work content distribution. On the other hand, we plan to extend the prototype for the connection with SDG systems to work also with the Blender Game Engine. Thereby, we plan to implement the new TUIO protocol [11] for the communication between connected hardware systems and the visualization software. Once the whole SAMSON environment is up and running, we will conduct several user studies to measure how well the process of line balancing is supported, and how good users rate the system.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the efforts and the commitment of Mark Peter Staiger, Simulation Engineer at BMW Group. He constantly supported model design and assisted with his simulation knowledge. Furthermore, we want to thank Russell Roman, Donnie Pittman, and their team at BMW Group most sincerely for their support and the readiness to share their expert knowledge of production. Furthermore, we want to point out the implementation efforts and valuable input of Raphael Das Gupta. During his bachelor thesis, he extended the existing prototype between Blender and the SDG system MightyTrace, and implemented the dynamic association of objects.

REFERENCES


