# Reconciling Scenario-Centered Controller Design with State-Based System Models[*]

Holger Giese, Ekkart Kindler, Florian Klein[†] , Robert Wagner
*Software Engineering Group*
*Department of Computer Science*
*University of Paderborn*
*Warburger Straße 100*
*33098 Paderborn, Germany*
[hg|kindler|fklein|wagner]@upb.de

## ABSTRACT

Scenarios are an effective means for defining the expected behavior of a system during the design and implementation phase. The 'Come Let's Play' approach has demonstrated that scenarios can fully define a system's behavior. In practice, however, the expected behavior defined by scenarios must be achieved in the context of existing components that cannot be changed. Therefore, the scenario-based approach must be reconciled with state-based models. In this paper, we present such an approach for the design of flexible production systems which employs scenarios not only for describing and synthesizing the required system functionality but also for recording observed behavior for analysis or 3D-visualization. We illustrate our approach using an existing material flow system which is a major part of a real production system.

## 1. INTRODUCTION

Scenarios such as Message Sequence Charts (MSCs), Life Sequence Charts (LSCs) or UML Sequence Diagrams provide a suitable abstraction for the communication among systems engineers who need to define the expected behavior of a system during the design and implementation phase.

Scenarios can do much more than only describing a single execution or folded set of executions of a system. They can define the behavior of a system. With their 'Come Let's Play' approach, Harel and Marelly [7] carry this idea to the extreme. Their play engine, and in particular the smart play out mechanism [6], impressively demonstrate that a system can be fully defined in terms of scenarios without providing a model of the underlying components, and that the operational state-based behavior of the components can be synthesized from the scenarios [5, 1]. In all, the 'Come Let's Play' approach represents a major advance: scenarios are the system or, at least, fully define the system's behavior.

In many situations, however, we do not need to define the full system behavior from scratch. Instead, major parts of the system are either reused or consist of pre-fabricated components with known behavior. Thus, scenarios as the means to describe the expected behavior of the new parts of the system have to be incorporated with existing, usually state-based behavioral models. In flexible production systems, for example, there are parts of the system that are fully characterized by the combination of standard components (the plant) and other parts that have yet to be designed for each system (the controller).

In this paper, we discuss how to combine models and scenarios in the design of flexible production systems, in particular in the controller design and synthesis, where scenarios are used to describe the expected behavior of the controller. We will show that scenarios can be used for even more than that: they can be used as a means for the communication between all kinds of tools involved in the development of flexible production systems. For example, scenarios can be used for recording the system's behavior and analyzing its causes or for visualizing observed behavior in 3D (possibly in slow motion) and indicating the sources of misbehavior.

The paper is structured as follows: In Section 2, the domain of flexible production systems as well as our case study are introduced. Our vision for a scenario-centered environment for the development of flexible production systems follows in Section 3. Section 4 presents the elements of our scenario-centered development environment that are already implemented and sketches ideas and the underlying concepts for the remaining parts.

## 2. A FLEXIBLE PRODUCTION SYSTEM

In order to illustrate our ideas, we present a case study of a flexible production control system. The experimental setup implements a manufacturing system for the production of bottle openers (see Fig. 5). The system consists of sever-

[*]This work was developed in the course of the Special Research Initiative 614 - Self-optimizing Concepts and Structures in Mechanical Engineering - University of Paderborn, and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft.

[†]Supported by the International Graduate School of Dynamic Intelligent Systems.

al processing machines, robots, stations, and manual work places. They are connected by a rail-bound material flow system with transportation units, which will be called shuttles in the rest of this paper.

A shuttle is electrically propelled and moves in exactly one direction. It circles around the main loop until the control assigns a production task to it. When a task is assigned to a shuttle, the shuttles moves to the stations and manual work places where the production and assembly steps are executed. If the control does not assign a new task to the shuttle, it will circle around the main loop until it gets one.

The employed production control system consists of PCs on the supervisory control level and Programmable Logic Controllers (PLCs) on the cell level. Higher-level tasks, e.g. planning, order assignment, and coordination of local activities of all controllers are dealt with at the supervisory control level. The PLCs on the cell level are connected to actuators and sensors and are responsible for the control of local components such as stations or robots.

In our case study, the used PLCs run much faster than the environment. Hence, signals are always caught. The control system can be seen as a reactive system with a negligibly short reaction time to signals and events stimulated by the environment. Hence, the perfect synchrony hypothesis regarding the interaction between the environment and the controller on the cell level is fulfilled.

In this paper, we neither consider time aspects nor the supervisory level of our production control system. We concentrate on the cell level design and present a simple control task which serves as a running example. A more detailed description of our case study containing further control tasks and a description of the software development process defined in an earlier project can be found in [13, 16].

In our case study, shuttles are equipped with an opto-electronic distance sensor to prevent rear-end collisions with other shuttles. However, the range of the sensor is reduced laterally in order to prevent unintentional stops of a shuttle caused by objects near the track. As a consequence, however, the collision avoidance does not work properly in curves. In order to prevent rear-end collisions in curves, the control has to ensure that only one shuttle enters a curve at a time. This is achieved by a start/stop unit with additional sensor and actuator technology (cf. Fig. 1).

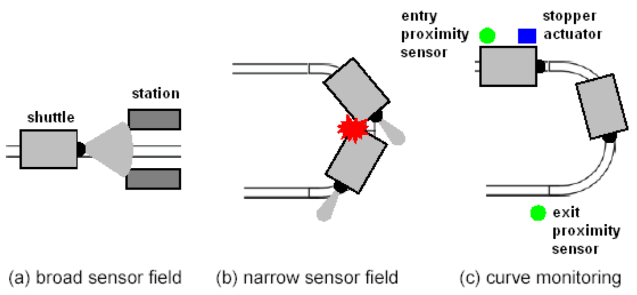

**Figure 1: Curve monitoring for collision avoidance**

When a shuttle enters a curve, the shuttle is detected by an proximity sensor. The control stops all following shuttles by activating a stopper actuator. The stopper is released only if a proximity sensor at the end of the curve indicates a shuttle leaving the curve area.

In order to build a controller for our curve monitoring example, the engineer has to design a controller model manually using some formal specification technique. This is tedious and error-prone. Hence, we provide additional support for validation by means of simulation. For this purpose, we have developed simulation components for the plant model. Note that the simulation components have to be designed only once and can be reused for different plant topologies.

In Fig. 2, the simulation components are specified as classes with boolean attributes representing actuators and sensors. Note that, later on, these attributes will be accessed by the controller model.

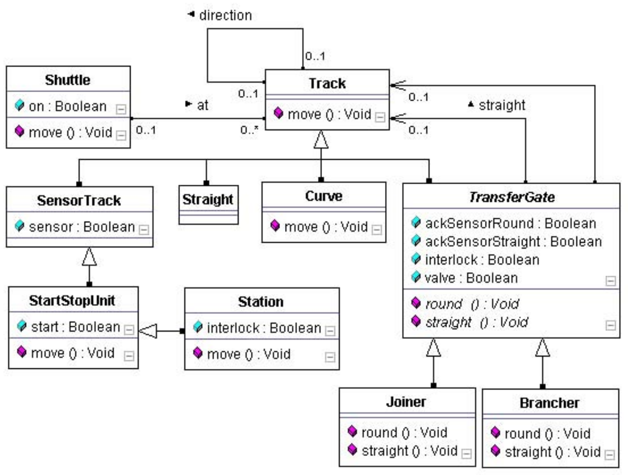

**Figure 2: Static structure of our simulation model**

The operational behavior of the simulation components is specified using state charts and story diagrams. Story diagrams are a combination of activity and collaboration diagrams with graph grammar semantics. They are described in more detail in [3, 21].

These simulation components can be instantiated and connected to an overall plant model which is simulated and visualized by an attached visualizer. This allows the engineer to validate the designed controller model before it is used to control the real plant.

After the controller model is validated and the production system is built, the ramp-up phase starts. We support this phase by automated PLC-code generation from the controller model. Additionally, we use Augmented Reality (AR) technology to augment the engineer's field of view with additional information about the state of the plant and the control software. From the difference between operating states of the control software and the real plant the engineer can deduce failures of the system.

Although the current approach has many advantages like e.g. validation by simulation, automated code generation and ramp-up support, it requires good domain knowledge as well as advanced software design skills. Since engineers are more used to thinking in scenarios, we propose to extend our modeling approach as described in the following section.

## 3. VISION

In order to illustrate the idea of our approach, we discuss the different tasks of an engineer when developing a controller. Figure 3 gives an overview on the artifacts and tasks.

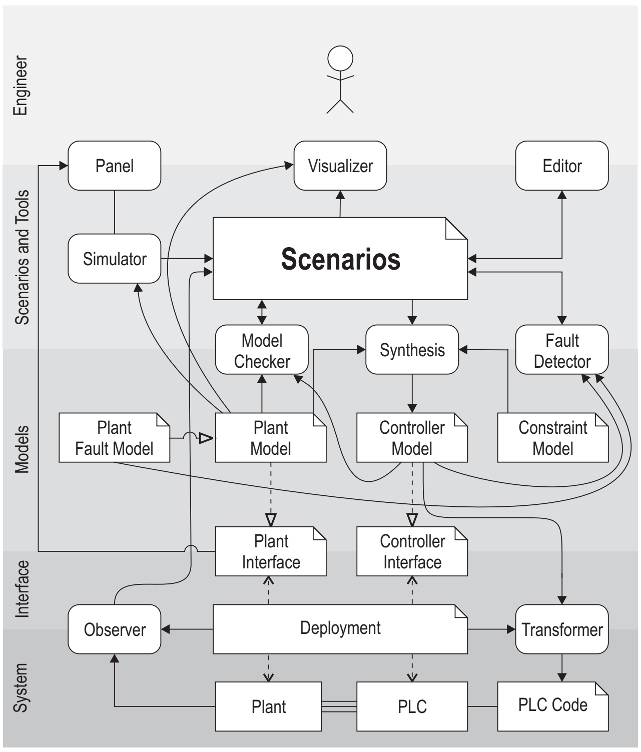

**Figure 3: The vision: An overview**

The core parts are the interfaces of the plant, i. e. its actuators and sensors and the interfaces of the controller. We assume that there already is a construction plan for the plant and a deployment descriptor mapping all sensors and actuators in the interfaces to the physical sensors of the PLC and the plant. Moreover, we assume that there is a model for each component of the plant and that we have some constraints for the components of the model, which are states that should never be reached – formally expressed by state formulas. In our example, we have the constraint that there should never be two shuttles in a curve at the same time.

Now, an engineer starts developing the code for the PLC. For specifying the controller, an engineer can play with the model in a way similar to Harel's and Marelly's play-in mechanism. To this end, the engineer sees a panel with all actuators and sensors, which allows him to interact with the plant (resp. its model). These interactions will be recorded in a scenario and, at the same time, the behavior will be animated in a 3D-visualization. Note that the behavior resulting from these interactions comes from simulating the model of the plant – in contrast to Harel's approach, where such models do not exist. In order to animate the behavior in 3D, there must be some additional information, which we call visualization model (cf. PNVis [8]). At the end of this stage, the engineer has provided some scenarios to the system, which he might also edit by hand. Some scenarios might be marked as inadmissible behavior.

An example of a scenario is shown in Fig. 4. It could be a part of a specification of the control of a curve segment of the plant, where the StartStopUnit guards the entry of the curve segment and the SensorTrack monitors the sensor at the end of a curve.

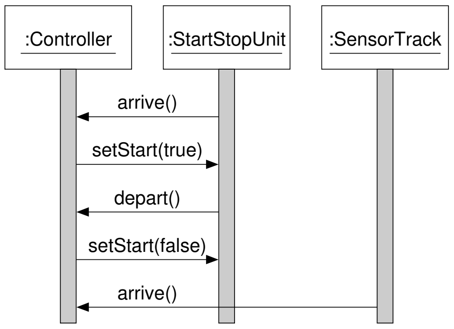

**Figure 4: A scenario**

In the next step, controller synthesis algorithms can be used to synthesize a controller that allows the scenarios provided by the engineer and additionally meets the constraints attached to the models of the plant; in our example, it would guarantee that there are no two shuttles on a curve segment at the same time. The resulting controller could be simulated along with the model in order to validate and to visualize it; or the controller could be analyzed and some additional properties could be verified by a model checker. Once the engineer is convinced that the controller is correct, the code for the PLC can be be automatically generated from the controller model (taking into account the deployment mappings).

Then, the real plant can be run with this PLC code. An observer can record the behavior of the plant and store it as a scenario, which can be inspected by the engineer or animated in a 3D-visualization. Typically, some of the observed behaviour will be wrong, which we call failure behaviour. The failures could result from not providing enough scenarios as input to the controller synthesis algorithm (i. e. from a wrong or incomplete specification), or it could be due to faults in the hardware[1].

At this point, we can use the recorded scenario for another purpose. We assume that we have a refined model of the plant that covers all possible hardware faults and the resulting failure behaviors. Then, a fault detector, which uses techniques from model checking, could identifying how particular scenarios (that are impossible in the ideal model) could arise from faults. The output are more detailed scenarios including the faults that resulted in the misbehavior. These could be visualized in order to allow the engineer to fix them in the hardware. This way, the ramp-up procedure of a new plant could be significantly speeded up because hardware faults that resulted in failure behavior could be identified more quickly. This could even be combined with augmented reality technology in order to visualize faults in the real plant.

[1]A fault is a defect (in the hardware) that might or might not have negative effect on the behaviour of the system. When the fault results in an unexpected or unspecified behaviour, this is called a failure [11].

## 4. REALIZATION

In this section, we give an overview of the existing tool support for the integrated design of production control systems. We also discuss the needed conceptual and tool-related extensions in order to be able to realize our vision for production control system development in the near future.

### 4.1 Existing Tools

The provided tool support is based on the integrated environments FUJABA TOOL SUITE [18], COMPONENTTOOLS [4], and PNVIS [8].

The simulation components for the *Plant Model* are specified using the FUJABA TOOL SUITE. They consist of UML object diagrams, corresponding class diagrams, state charts and story diagrams (cf. Section 2). These simulation components are connected to a concrete plant model according to a particular plant topology. The plant topology is created using a tool called LONTROL [16]. In the future, we plan to use a more general tool for this purpose. It is called COMPONENTTOOLS and supports building a system from library components with underlying models.

The *Controller Model* is built using the same specification techniques as for the plant model. Note that up to now the controller model has to be designed manually. For this purpose FUJABA includes editors as well as a code generator.

From the plant and controller model, FUJABA generates executable code for the *Simulator*. In order to simulate the entire production system on standard hardware, the generated code is pure Java. Hence, the simulation is platform independent.

The simulator is equipped with a *Visualizer*, which combines geometry information with 3D models and renders them in real-time using Virtual Reality (VR) technology. The screenshot in Fig. 5 shows such a 3D model of our complete case study. This visualization stems from an earlier project, where the control software was hard coded into the simulation. In order to become more flexible, we plan to connect the PNVIS tool to our simulator.

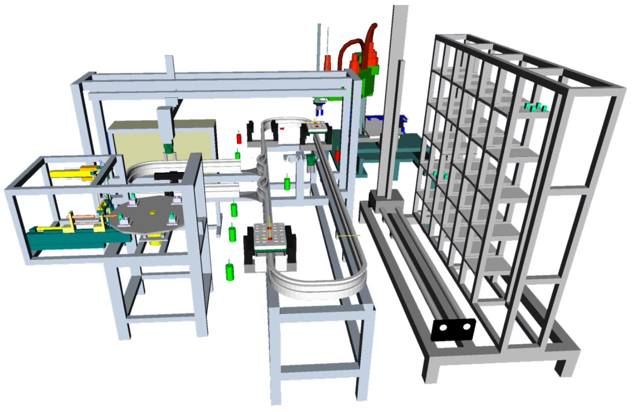

**Figure 5: 3D visualization of the simulation**

After a successful validation, we ensure a correct implementation by generating the PLC-code automatically from the controller model [16]. This *Transformer* module is also part of the FUJABA TOOL SUITE.

### 4.2 Required Extensions

A key requirement for our scenario-centered approach is the *Panel*, which will allow the engineer to play in scenarios. Using code generation combined with a set of user interface templates for different sensor and actuator types, such a control panel can be derived from the *Plant Interface Model* and integrated with the *Simulator* generated from the *Plant Model* in a straight-forward way. Templates using information from the 3D model used by the *Visualizer* would even allow the seamless integration of the control panel into the simulated VR environment.

The appropriate *Editor* for revising played-in scenarios depends on the formalism used to express them. Taking the respective expressive power of different notations and the preferences of the engineer users into consideration, we intend to implement the editor based on FUJABA's support for UML 2.0 sequence diagrams, extending it where necessary.

*Synthesis* will be integrated into FUJABA seamlessly to allow immediate inspection of the generated state charts. Unfortunately, none of the existing approaches for the synthesis of operational state-based behavior from scenarios [5, 1, 9, 12, 17, 20]) supports the integration of existing state-based models. State-based approaches for controller synthesis [15] in turn require state-based constraints rather than scenario-like input. However, both scenarios and state-based models can, in principle, be mapped to temporal logic formulae. The existence of approaches for synthesis from temporal logic specifications, such as [14] for linear time logic (LTL), [2] for computational tree logic (CTL), or [10] for the $\mu$-calculus, therefore suggests the intended synthesis is indeed possible. We still need to work out a specific solution, though.

The *Observer* will generate scenarios in a way similar to our play-in mechanism, but use the activations of the actual sensors and actuators as its input. In the case of deviations from the specified behavior, the engineer can pass the recorded scenarios to a *Fault Detector* in order to generate a scenario identifying the source of the observed failure behavior. The *Fault Detector* achieves this by using the refined *Plant Fault Model* and model checking techniques to generate witnesses. For the implementation of a proof of concept, we intend to use the MCIE [19] library. As a more direct approach to fault diagnosis, redundant or dedicated additional sensors could also be added to the plant in order to detect equipment failures and system states that violate constraints.

## 5. CONCLUSION AND FUTURE WORK

The presented vision of a scenario-centered development environment for flexible production systems suggests that scenarios in combination with state-based models can serve as the basis for the automated development of the control functionality. In addition, scenarios can further be used to analyze an existing plant and identify possible sources of unexpected or faulty behavior.

Future work will include the adjustment of existing synthesis techniques for mixed specifications consisting of scenarios and state-based models as well as tool support for the recording of plant behavior and the succeeding analysis.

## 6. REFERENCES

[1] Y. Bontemps and P. Heymans. As fast as sound (lightweight formal scenario synthesis and verification). In *In Proc. of the 3rd International Workshop on Scenarios and State Machines: Models, Algorithms, and Tools (SCESM04) ICSE Workshop W5S Edinburgh, UK*. IEE, 2004.

[2] E. M. Clarke and E. A. Emerson. Design and Synthesis of Synchronization Skeletons using branching time temporal logic. In *Proceedings of the IBM Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer Verlag, 1981.

[3] T. Fischer, J. Niere, L. Torunski, and A. Zündorf. Story diagrams: A new graph rewrite language based on the unified modeling language. In G. Engels and G. Rozenberg, editors, *Proc. of the* $6^{th}$ *International Workshop on Theory and Application of Graph Transformation (TAGT), Paderborn, Germany*, LNCS 1764. Springer-Verlag, 1998.

[4] A. Gepting, J. Greenyer, E. Kindler, A. Maas, S. Munkelt, C. Pales, T. Pivl, O. Rohe, V. Rubin, M. Sanders, A. Scholand, C. Wagner, and R. Wagner. Component tools: A vision of a tool. In *Proc. of the 11th Workshop on Algorithms and Tools for Petri Nets (AWPN), Paderborn, Germany, September 30 - October 1, Tech. Rep. tr-ri-04-251*, pages 37–42, September 2004.

[5] D. Harel and H. Kugler. Synthesizing State-Based Object Systems from LSC Specifications. In *Proc. 5th Int. Conf. on Implementation and Application of Automata*, volume 2088 of *Lecture Notes in Computer Science*, pages 1–33. Springer Verlag, 2001.

[6] D. Harel, H. Kugler, R. Marelly, and A. Pnueli. Smart play-out. In *Conference on Object Oriented Programming Systems Languages and Applications, OOPSLA '03*, pages 68–69, October 2003.

[7] D. Harel and R. Marelly. Specifying and Executing Behavioral Requirements: The Play In/Play-Out Approach. *Software and System Modeling (SoSyM)*, 2003.

[8] E. Kindler and C. Páles. 3D-visualization of Petri net models: Concept and realization. In J. Cortadella and W. Reisig, editors, *Application and Theory of Petri Nets 2004,* $25^{th}$ *International Conference*, volume 3099 of *LNCS*, pages 464–473. Springer, June 2004.

[9] I. Krüger, R. Grosu, P. Scholz, and M. Broy. From MSCs to Statecharts. In F. J. Rammig, editor, *Distributed and Parallel Embedded Systems*, pages 61–71. Kluwer Academic Publishers, 1999.

[10] O. Kupferman and M. Y. Vardi. $\mu$-Calculus Synthesis. In M. Nielsen and B. Rovan, editors, *Proceedings of the on 25th International Symposium Mathematical Foundations of Computer Science (MFCS 2000), Bratislava, Slovakia*, volume 1893 of *Lecture Notes in Computer Science*. Springer Verlag, August/September 2000.

[11] J. C. Laprie, editor. *Dependability : basic concepts and terminology in English, French, German, Italian and Japanese [IFIP WG 10.4, Dependable Computing and Fault Tolerance]*, volume 5 of *Dependable computing and fault tolerant systems*. Springer Verlag, Wien, 1992.

[12] E. Mäkinen and T. Systä. MAS - an interactive synthesizer to support behavioral modeling in UML. In *Proceedings of the* $23^{rd}$ *International Conference on Software Engineering (ICSE 2001), Toronto, Canada*, pages 15–24, May 2001.

[13] U. Nickel, W. Schäfer, and A. Zündorf. Integrative specification of distributed production control systems for flexible automated manufacturing. In M. Nagl and B. Westfechtel, editors, *DFG Workshop: Modelle, Werkzeuge und Infrastrukturen zur Untersttzung von Entwicklungsprozessen*, pages 179–195. Wiley-VCH Verlag GmbH and Co. KGaA, 2003.

[14] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the sixteenth annual ACM symposium on Principles of programming languages, Austin, Texas, United States*, 1989.

[15] P. Ramage and W. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1), January 1987.

[16] W. Schäfer, R. Wagner, J. Gausemeier, and R. Eckes. An engineers workstation to support integrated development of flexible production control system. In H. Ehrig, editor, *Integration of Software Specification Techniques for Applications in Engineering*, volume 3147 of *LNCS*. Springer-Verlag, 2004.

[17] S. Uchitel and J. Kramer. A workbench for synthesising behaviour models from scenarios. In *Proceedings of the 23rd international conference on Software engineering*, pages 188–197. IEEE Computer Society, 2001.

[18] University of Paderborn, Germany. *Fujaba Tool Suite. Online at http://www.fujaba.de/.*

[19] University of Paderborn, Germany. *Model Checking in Education. Online at http://www.upb.de/cs/kindler/Lehre/MCiE.*

[20] J. Whittle and J. Schumann. Generating statechart designs from scenarios. In *Proceedings of the 22nd international conference on on Software engineering June 4 - 11, 2000, Limerick Ireland*, 2000.

[21] A. Zündorf. *Rigorous Object Oriented Software Development.* University of Paderborn, 2001.