Ramp-Up and Maintenance with Augmented Reality in Development of Flexible Production Control Systems

Prof. Dr.-Ing. Jürgen Gausemeier[1]; Prof. Dr. Wilhelm Schäfer[2];
Dipl.-Wirt.-Ing Raimund Eckes[1]; Dipl. Inform. Robert Wagner[2]

[1]Heinz Nixdorf Institute
University of Paderborn, Germany
[2]Department of Computer Science
University of Paderborn, Germany

Abstract
One of the problems in today's manufacturing industry is long ramp-up time of production systems. This is due to long testing of hardware in combination with the not yet tested control software. Although validation by simulation helps to identify most of the errors, the remaining errors must be removed during the ramp-up. In this paper we show how the ramp-up is supported by an Augmented Reality tool that visualizes states depending on different domains (mechanics, electrics, software) in the real manufacturing system. This approach is part of a methodology for the integrated development of control software for flexible production systems.

Keywords
Augmented Reality, Ramp-Up, Maintenance, Production Control Systems

1 INTRODUCTION

One of the problems in today's manufacturing industry is long ramp-up time of production systems.

For producing companies shortening the ramp-up time facilitates an earlier market entry and therefore shortens time-to-market which in turn provides advantages in market shares and higher revenues, especially for high-tech products [1].

One reason for the time-consuming ramp-up process is long testing of hardware in combination with the not yet tested control software. During the ramp-up the issues of all the different involved domains and their interaction must be considered concurrently. This determines the complexity and duration of the ramp-up process. In particular, faults in control software of highly automated stations cause longer cycle times or collisions [1].

The aim is to advance the methodology for the development of flexible production control systems in order to support the ramp-up process [2]. This is done by using Augmented Reality to visualize online states of components of the production system and the control software.

The main research on the implemented methodology was performed within the project ISILEIT (Integrative Specification of Distributed Production Control Systems for the Flexible Automated Manufacturing), sponsored by the German Research Foundation (DFG - Deutsche Forschungsgemeinschaft) within the Priority Programme (Schwerpunktprogramm) “Integration of Software Specification Techniques for Applications in Engineering”. The project was carried out in cooperation with computer scientists and engineers.

The methodology is briefly described in the following section. After that the AR-tool supporting the advanced methodology is presented. The subsequent section discusses the application of the AR-tool within the ramp-up process. Because the AR-tool supports the ramp-up and the maintenance processes the usage in the maintenance process is not mentioned explicitly. The paper ends with a short conclusion.

2 DEVELOPMENT OF FLEXIBLE PRODUCTION CONTROL SYSTEMS

In this section we provide a brief overview of our modeling approach which was developed in the afore mentioned project. A more detailed description can be found in [2] and a graphical overview of our software development process is presented in [3].

2.1 Methodology

The software development process for our case study is defined by the following phases:

Analysis Consolidation Phase. Based on the results of a previous informal requirements-gathering phase, which is beyond the scope of this paper, the topology of the planned production control system is modeled in this phase. Modeling

includes the identification of the number and types of participating processes as well as the definition of communication channels and different kinds of interchanged signals.

Design Phase. In the next step, the initial (UML) class diagram of the desired system is derived. Here, each process(type) is represented by a class in the class diagram. In addition, each signal received by a process creates a signal method in the corresponding class.

Reactive Behavior Modeling Phase. Now we have to define how each process will react on these signals. Thus, for each process class, a statechart must be provided describing the general process behavior. These statecharts should cover at least all signals that are defined by the corresponding process class.

Action Modeling Phase. Statecharts specify in which states a certain process reacts to a particular signal. In response to a signal, a process might change its state and execute some additional activities. These activities might again include complex computations. These complex computations might employ or modify complex object-oriented data structures in order to reflect the surrounding world or the execution of manufacturing plans for certain products. For the specification of such complex computations we use UML-like collaboration diagrams with precisely defined operational semantics based on graph-grammar theory [3]. Consequently, we use collaboration diagrams to specify complex control flows of methods employed as actions within statecharts.

Frequently, one will need more than a single collaboration diagram to model a number of object structure modifications. Therefore, we combine statecharts (and activity diagrams) with collaboration diagrams yielding a powerful visual specification language. Basically, we allow the use of collaboration diagrams as the specification of activities instead of just pseudo-code statements.

Verification. The system design model will be used for code generation. When it reaches a mature state, it will be verified. This proves the correctness of the specified control software. The verification is done by formal model-checking techniques based on ASM and AsmL [4].

Code Generation Phase. Once all aspects of the system are specified, our development environment (FUJABA, www.fujaba.de) is able to generate a complete and executable implementation from the class and behavior diagrams. This implementation is used for the simulation phase that follows. Furthermore, as will be explained later, generation for PLCs (Programmable Logic Controller) is also supported.

Simulation Phase. One of the main problems in today's manufacturing industry is long down-times of assembly lines due to long testing phases of newly installed software. Our approach allows the simulation of the production process beforehand in order to shorten down times of physical assembly lines caused by software reconfiguration.

Ramp-Up and Maintenance Phase. Once the manufacturing system is installed, it has to be put into operation. This process is called ramp-up. The aim is a fully-functional production system with tested control software. It is followed by a maintenance phase which lasts as long as the system is operating. The aim is to provide an uninterruptible operating process. Causes for interruptions have to be analyzed and eliminated as fast as possible. The analysis of causes for an interruption generates down-time for the manufacturing system.

The main focus of this paper is on the ramp-up support realized with an AR-tool (Augmented Reality). However, before the AR-tool and its application are presented, two steps of our methodology are explained in more detail since they build the basis for this kind of support.

2.2 Reactive Behavior Modeling

In the following, we exemplify the specification of the control software for a transfer gate used in an experimental setup in the mechanical laboratory for computer-integrated production. Within our material flow system two types of transfer gates can be identified: those that branch out of a track (Brancher) and those that merge two tracks into one (Joiner). Although they are used differently, they are identical in construction and operate on the same principle. Figure 1 shows a schematic overview of a transfer gate.

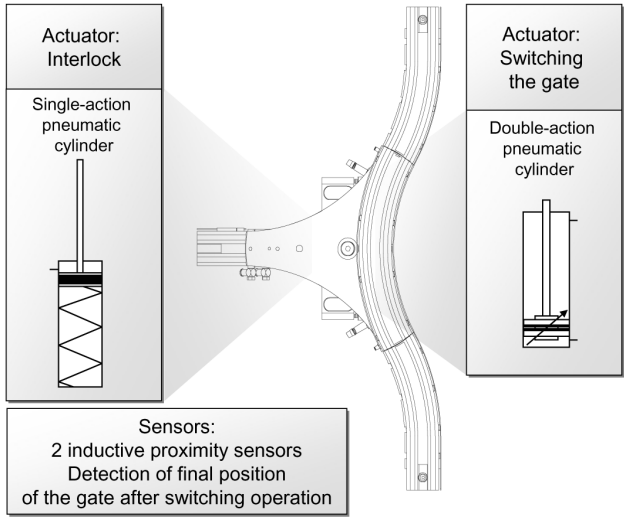

Figure 1. Transfer Gate

A transfer gate consists of an interlock, a double-action pneumatic cylinder, and two proximity sensors, one for each switching direction. For safety reasons the transfer gate is generally locked. The first step in switching into another direction is to disengage the interlock by activating the single-action pneumatic cylinder. Now the transfer gate

can change its direction by turning into the new position. This is done by the double-action pneumatic cylinder. When the switching operation is completed, the final position is detected by one of the proximity sensors. After that, the pressure-controlled mechanical interlock has to be re-engaged. The occurrence of a switching failure can be recognized only by an absent sensor trigger after a predefined timeout.

The control of this behavior is specified by the statechart depicted in Figure 2.

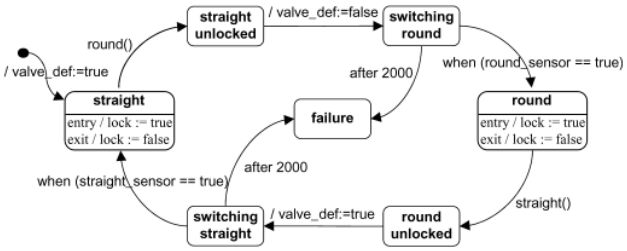

Figure 2. Statechart of the transfer gate controller

The specified statechart switches the transfer gate between the straight and round directions. Initially, the transfer gate is switched to the straight direction and fixed by a mechanical interlock. When the *round()* event is received, the interlock is disengaged by the exit action *lock:=false*, and state *straight unlocked* is entered. After that, the trigger-less transition is fired and the appropriate action *valve_def:=false* is executed. This action triggers a pneumatic cylinder responsible for turning the transfer gate into the round direction. The state *switching round* is left if either the proximity sensor announces that the switching process is completed or if a timeout of 2000 milliseconds occurs. In case of a timeout, a failure state is entered. In the other case, the switching process was successful. Thus, state *round* is entered and the interlock is re-engaged. Now the transfer gate can be switched back to the straight direction by sending a *straight()* event. The switching is performed analogous to the described switching process for the round direction.

For the specification of the entire system, further controller statecharts are needed, e.g. statecharts to control the stopping and starting of shuttles at stations or before transfer gates. Requirements or corresponding system functionality is then expressed by a number of different statecharts. The sheer number of these statecharts and their interplay makes it hard to check manually whether the system functionality is defined correctly. As an example consider a requirement like “a shuttle never enters a transfer gate if the transfer gate is currently switching its direction”. Our approach enables automatics verification of such kind of safety-critical requirements using model checking [4].

After a successful verification, the entire system model needs to be implemented. In our approach, the defined precise semantics of the used models allows us to generate the code from the model automatically. In the following section we give a short overview of the generation of executable code from the statechart in Figure 2.

2.3 Code Generation

There are different programming languages for PLCs, each intended for a specific application domain and based on the background of the control engineer who uses them. In order to achieve more conformity of the different notations the standard IEC 61131-3 [5] was developed. Each of the standardized languages covers different abstraction levels. Instruction List (IL) for example is a low level assembly language very close to hardware programming, whereas Sequential Function Charts (SFC) describe the sequence of a PLC program as a state transition diagram. For the automatic generation of PLC-code out of an object-oriented specification, we adapted our code generation mechanisms to produce Structured Text (ST). Structured Text is a notation similar to PASCAL. It is a higher-level language than IL and provides more structurizing and organizing constructs such as if-then-else-conditionals and while-loops. However, as a classical procedural programming language, ST does not support typical object-oriented concepts like inheritance or polymorphism. Thus, a direct mapping of a statechart to an object-oriented implementation, e.g. the state-pattern [6], is not possible. The mapping of object-oriented concepts has to be managed explicitly as described in [3]. For statecharts, we omit this overhead by using a simpler translation based on switch-case statements.

When implementing a statechart on PLCs, two problems arise. The first problem is moving from an event-based statechart world into a signal-based PLC-universe. In contrast to events, signals exist the whole time and are handled by the PLC as Boolean values. To overcome this problem, an event can be associated with the rising and falling edges of a corresponding signal. For example, a rising edge can be detected comparing the signal between two cycles. If the signal was low in the previous cycle and is now high, a rising edge is detected.

Second, due to the cyclic execution of the three phases of the PLC, signals and events from the environment can occur simultaneously. Even worse, if a signal does not hold for at least the maximum amount of time needed for a cycle, one cannot be sure that the PLC will ever read the signal. The solution to this problem is to use PLCs that are fast enough.

In our experimental case study the used PLCs run much faster than the environment. Hence, signals are always recognized. The control system can be seen as a reactive system with a negligibly short reaction time to signals and events stimulated by the environment. Thus, the assumption of the perfect

synchrony hypothesis [7] regarding the interaction between the environment and the controller is fulfilled.

The behavior of a statechart can be implemented by simple switch-case constructs. The piece of code in Figure 3 is part of the generated program for the statechart shown in Figure 2. It gives a short overview of the translation in Structured Text for the states *switching round* and *round*.

We declare an integer variable state to keep the current state of the statechart. Each state is encoded by an integer which is handled in a case statement. Events and signals are translated to Boolean variables in the symbol table. The symbol table provides a mapping between the program and the sensors/actuators plugged into the PLC. In each state, the outgoing transitions of that state are treated.

```
VAR state: INT := 1;              /* state = "straight" */
  timer: TIMER;
  expired: BOOL := FALSE;
END_VAR;

CASE state OF
  ...
  3:/* state = "switching round" */
    timer(IN:=TRUE, TV:=T#2000ms);/* start or update timer */
    expired := NOT timer.Q;       /* test for expired timer */

  IF (round_sensor = TRUE) THEN /*when(round_sensor==true) */
    timer(IN:=FALSE, TV:=T#2000ms);/* stop timer */
    state := 4;                    /* state = "round" */
    lock := TRUE;                  /* entry action */

  ELSIF (expired = TRUE) THEN      /* after 2000 */
    timer(IN:=FALSE, TV:=T#2000ms);/* stop timer */
    state := 7;                    /* state = "failure" */
  END_IF;

  4:/* state = "round" */
  IF (straight = TRUE) THEN     /* event: straight() */
    lock := FALSE;              /* exit action */
    state := 5;                 /* state = "round unlocked" */
  END_IF;
  ...
END_CASE;
```

Figure 3. Example for generated PLC code

Timer events are handled by built-in timer functions. If a transition is enabled, the new state is set and the appropriate exit and entry actions are executed. Note that the presented program is executed once in each cycle. Thus, it is the body of an implicit loop-forever statement.

3 RAMP-UP

In this chapter the realization of Augmented Reality based ramp-up is described.

3.1 Ramp-up procedure

The ramp-up process includes the installation of the hardware, testing the hardware in combination with the control software and the initiation of the manufacturing system. Until now there has been no support for testing the interaction between the control software and the hardware.

The first issue that can be tested is the mechanical functionality of the hardware. The functionality of actuators and sensors can be tested by operating them manually. The connections of electrical components to the PLC are usually implemented by means of a field bus. Therefore, the connection to the PLC can only be tested when the PLC is in operation.

The first time the control software can be tested with the real hardware is after finishing the installation. Testing the control code is done by means of a test specification. The expected behavior is compared with the actual behavior. This ends up in a kind of trial and error testing, because if an error occurs the source of the error is not directly identifiable. The test specification cannot include every possible error and the possible reasons for the faults that cause the error. The source of the error can be a defective installation of mechanical components, incorrect cabling, incorrect field bus I/O configuration, or faults in the control software.

3.2 Architecture

In order to detect these faults while the control code is executed by the PLC, we developed an AR-tool which enables the online visualization of states of the manufacturing system. These states belong to different domains, namely mechanics, electrics and software. The architecture of the application within the ramp-up process is shown in Figure 4.

In the presented architecture the main components are the integrated development environment FUJABA, the PLC, the manufacturing system hardware and the AR-tool. As described in Section 2.2, the control software is specified by statecharts using FUJABA. This includes the mapping of variable names to input and output ports of the PLC. This mapping is named I/O-configuration. FUJABA generates the control code that is loaded into the PLC. For the different states that describe the behavior of the system a current state is included in

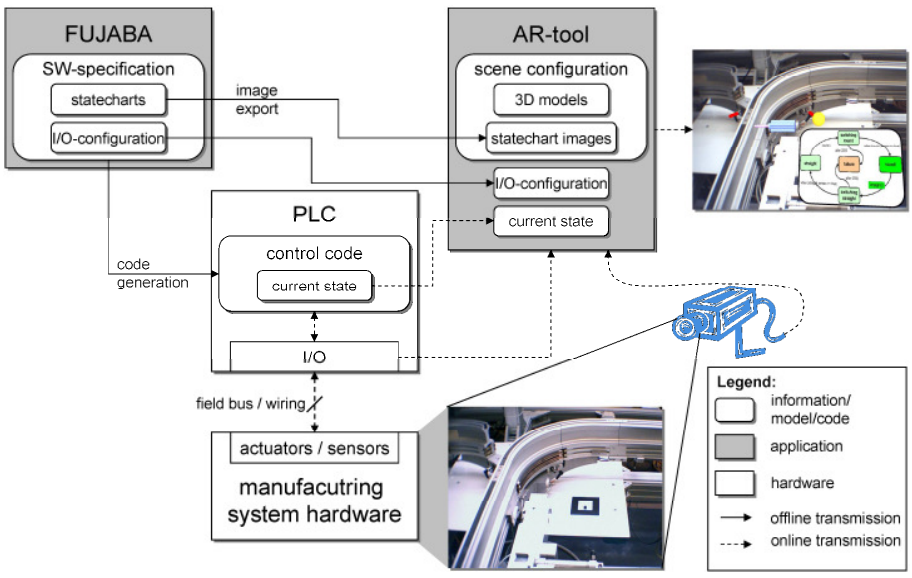

Figure 4. Architecture

the control code. According to this current state variable images of the corresponding statechart are exported. This means that each value of the current state variable is represented by a visualization of a statechart. The PLC executes the generated code including the current state variable, which changes its value according to the state of the control software. The I/O ports of the PLC and the current state variable are read online to detect the current states of the manufacturing system. The current state variable is visualized by the corresponding statechart image.

The PLC is connected to the manufacturing system hardware by means of a field bus. The PLC reads the input and sets the output in cycles. The processing of the input ports (connected logically to sensors) and the respective setting for the output ports (connected logically to actuators) is done by the control software. The I/O-configuration is used to transform the I/Os from the PLC to the states of the 3D models.

The scene configuration within the AR-tool consists of the 3D models and the loaded statechart images that correspond to the current state variable. The 3D models represent states of components that can be observed by the engineer using AR for example, the state of an inductive proximity sensor or a pneumatic cylinder that is built into a switch.

Using the AR-tool the engineer is tracked to determine his viewpoint. His view is augmented with the online information of the manufacturing system, i.e. with states of the components and the state of the control software (see Figure 5). This is done by augmenting the view with 3D models and images in correct position and orientation. The behavior of the sys-tem results from the combination of states and their changes in the course of time.

3.3 Application

As follows from the example in Section 2.2, the switching operation is completed when the final position is detected by one of the proximity sensors. The occurrence of a switching failure can be recognized only by an absent sensor trigger after a predefined timeout.

Since the double-action pneumatic cylinder is built into the gate and can not be seen from the outside, the AR-tool displays it as a 3D model. In the same way the position and detection states of the proximity sensors are represented by 3D models.

There are two cases for the application of the AR-tool which need to be distinguished. These cases

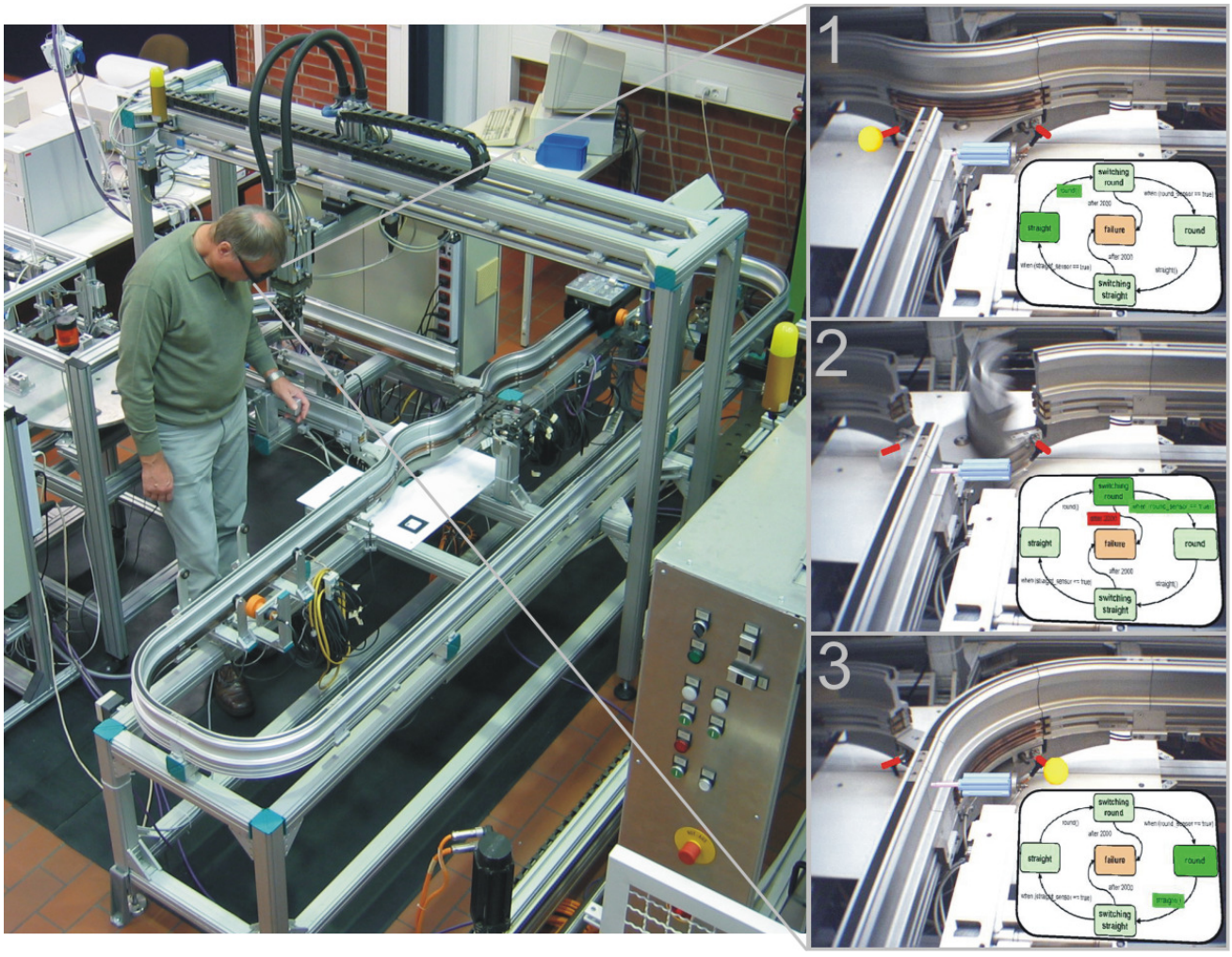

Figure 5. Application of Augmented Reality in the flexible manufacturing system

consider states of the real hardware, the "augmented" states and the states of the control software.

In the first case the "augmented" states displayed do not suit the real hardware states. For example, a proximity sensor detects (measurement or signal) that the transfer gate changes its direction but the corresponding "augmented" state does not. The conclusion is a defective installation of the hardware.

The second case is that the displayed statechart does not suit the behavior of the hardware and the "augmented" states. This leads to the conclusion that there is a fault in the software specification. For example, if the timeout period is specified too short, the software will always change to the failure state while switching. Another example is the mix up of sensor events in the specification, namely mixing up *straight_sensor* and *round_sensor* in Figure 2.

The application of the AR-tool is shown in Figure 5 where a transfer gate changes its direction. The corresponding states are displayed online in the engineer's view in the numbered order. The current state of the control software is highlighted within the statechart and can be compared with the behavior and the states of the hardware components.

4 CONCLUSIONS

In this paper we discussed how Augmented Reality can be used to support the ramp-up process. The AR-tool augments the engineer's view with the states of the corresponding hardware and software, i.e. it integrates the different domains. Faults in the installation and the control software can be identified more easily. The AR-tool enhances the perception and understanding of the interaction between control software and controlled hardware.

5 REFERENCES

[1] WIENDAHL, H.-P.; HEGENSCHEIDT, M.; WINKLER, H.: Anlaufrobuste Produktionssysteme. In wt Werkstattstechnik online, volume 92, H. 11/12, 2002

[2] GAUSEMEIER, J.; SCHÄFER, W.; WAGNER, R.; ECKES, R.: An Engineer's Workstation to Support Integrated Development of Flexible Production Control System. In: Ehrig, Hartmut (Hrsg.): Integration of Software Specification Techniques for Applications in Engineering Bd. 3147. Springer Verlag, 2004 (LNCS)

[3] NICKEL, U; SCHÄFER, W.; AND ZÜNDORF, A.: 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

[4] RAMMIG, F; KARDOS, M.: Model based formal verification of distributed production control systems. In: Ehrig, Hartmut (Hrsg.): Integration of Software Specification Techniques for Applications in Engineering Bd. 3147. Springer Verlag, 2004 (LNCS)

[5] International Electrotechnical Commission, Technical Commitee No. 65. Programmable Controllers - Programming Languages, IEC 61131-3, 1993

[6] GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J.: Design Patterns: Elements of Reusable Object Oriented Software. Addison-Wesley, Reading, MA, 1995

[7] BERRY; G.; GONTHIER, G.: The esterel synchronous programming language: Design, semantics, implementation. Technical report, Ecole Nationale Superieure des Mines de Paris, 1988

6 BIOGRAPHY

Jürgen Gausemeier is professor and head of the Computer Integrated Manufacturing research group at the Heinz Nixdorf Institute, University of Paderborn.

Wilhelm Schäfer is professor and the head of the Software Engineering research group, Department of Computer Science, University of Paderborn.

Raimund Eckes is research associate at the Computer Integrated Manufacturing research group, Heinz Nixdorf Institute, University of Paderborn.

Robert Wagner is research associate at the Software Engineering Group, Department of Computer Science, University of Paderborn.