Customer-Oriented Systems Design through Virtual Prototypes

Stephan Flake, Christian Geiger, Wolfgang Mueller,
Volker Paelke, Waldemar Rosenbach
C-LAB, Paderborn University
Fuerstenallee 11, 33102 Paderborn, Germany
{*flake, chris, wolfgang, vox, bobka*}*@c-lab.de*

Juergen Ruf
Wilhelm-Schickard Institut
Tuebingen University
Sand 13, 72076 Tuebingen, Germany
ruf@informatik.uni-tuebingen.de

Abstract

Rapid prototyping based on 3D models is well accepted for several applications. This article addresses the application of animated virtual 3D prototypes for the development of computer-based systems supporting early collaboration of the system designer with the external customer. Our methodology seamlessly integrates illustration through 3D animation with the main tasks of computer-based real-time systems development, i.e., implementation and verification. The approach is outlined by the example of the design of a flexible manufacturing system.

1. Introduction

In the recent past, rapid prototyping with virtual 3D models became well accepted for various applications in the areas of mechanical engineering and construction. Virtual prototypes, e.g., piping arrangements in nuclear power stations and ships, avoid time consuming and costly generation of physical mock-ups. Though virtual 3D models have their place in those domains, their application is less accepted in the field of general systems development.

However, 3D animation has great potential to significantly enhance communication with the external customer through the entire development process, since existing design languages, i.e., textual programming languages and visual diagrams like UML, do not give external customers easy access to behavior/function of the system under design. Therefore, customers cannot give immediate feedback on the functional correctness of the specified system.

In contrast, animated 3D models as a design representation can significantly enhance customer feedback throughout the development process. We introduce a refined collaborative development process which is tailored to current design practice and integrates state machine based implementation and verification of real-time properties with 3D animation.

Starting with requirements, we develop scenarios which are transformed into an agent-based description. From the agent-based model we derive a 3D representation and an UML-based implementation. The latter gives a first executable model and can be used as input for design verification. 3D animation can serve as visualization for several purposes. While simulation can control the animation of the 3D model, a model checker can generate traces of counter examples that trigger animation sequences. In order to seamlessly integrate tools for analysis and animation, our system is based on the animation library i4D for rapid prototyping of animated 3D models.

The remainder of this article is structured as follows. The next section sketches the general and the refined collaborative customer-centered design process. Section 3 introduces the example of a flexible manufacturing system which is used as a running example. Section 4 presents our approach in detail with a brief outline of the applied tools. The final section closes with a summary and a conclusion.

2. Collaborative Systems Design

The general systems design process can be roughly divided into specification, implementation, and analysis. Requirement analysis and specification take place in the very early phase of systems design. This phase requires a most frequent communication and information exchange of the designer with the customer in order to agree to a first contract. With this contract the designer tries to implement the required system. Practical experience shows that it makes sense to contact the customer from time to time during systems design in order to minimize the deviation from the requirements. Deviations can be due to different interpretations, conflicting specifications, or not implementable specifications. To avoid this, Repenning proposes a model for a collaborative development which integrates the customer's feedback in the very early phases of the process [8]. Figure 1 shows our interpretation of this model.

The model is based on the principle of bidirectional ob-

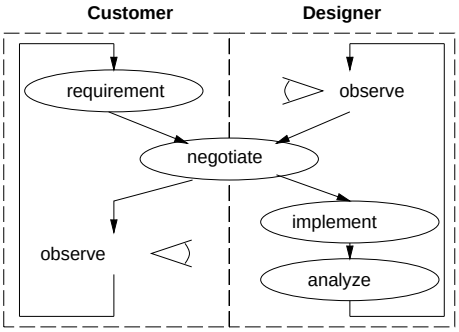

Figure 1. Collaborative Systems Design

servation. First, the designer tries to understand the customer's requirements. The following negotiation reduces these requirements to the technical realizable subset. This can be seen as the first contract between designer and customer. In the ideal case, the customer observes the designer during implementation and analysis and collaboratively tries to refine the requirements for the next iteration of the design process. The biggest problem in current practice is that it is often not possible for a customer to understand the details of the implemented functions. The designer often faces the problem to communicate the details of a function to a customer. This is mostly done by the means of graphical user interfaces which partly emulate the required function as it is understood by the designer. When this is not sufficient, more advanced techniques from multimedia authoring can be applied. State-of-the-art presentations employ photographs and textual descriptions, sometimes augmented with video and sound. However, it is difficult to present complex technical systems with inherent three-dimensional geometric properties by these medias. Customers must be enabled to experience the features and failures of the current system prototype in a realistic way in order to influence the design process. An obvious solution to overcome this problem is to use interactive 3D simulations and animations, which provide rich exploration of the prototype, and animation techniques for dynamic content [7].

On the other hand, designers require additional tools for their system implementation, for instance model checkers that automatically check required system properties for validity. The seamless integration of expressive 3D presentation techniques and formal verification methods requires a refinement of the previously described design process. We propose a structured design process based on an adequate conceptual model and supported by tools for visualization and analysis.

Due to our experience, support for computer-based concurrent systems in early design phases is often based on agent-oriented conceptual models since the agent metaphor has shown to be well applicable for conceptualizing aspects for those purposes [5]. While the conceptual model of agents with goals and complex behavior is suitable to informally capture first ideas, it lacks precision in order to be directly applicable as input to formal verification like model checking which requires a state-oriented implementation [2].

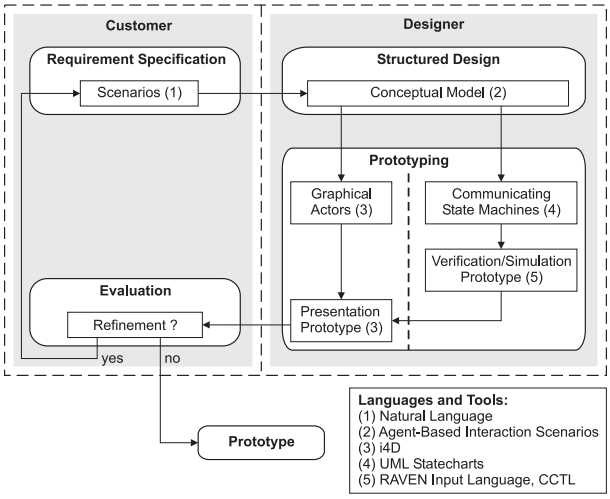

Figure 2. Design Approach

The design process for system development in virtual environments needs to address interactivity, bridge the large conceptual gap between external customer's view and technical implementation, support extensive user involvement, handle fuzzy requirements, and provide a base for formal verification techniques and expressive 3D illustration approaches. Our approach is illustrated in Figure 2 and combines techniques of scenario generation, hierarchical decomposition, and iterative prototyping. We outline our approach in more detail in Section 4 based on the running example introduced in the next section.

3. Manufacturing Case Study

The holonic manufacturing system (HMS) was introduced as a test case by the IMS Initiative [12]. An HMS is based on the notion of a holon which denotes a building block of a system for complex production lines. Hierarchically composed holons cooperate along the lines of basic rules to achieve a goal or objective. A single holon is understood as an autonomous cooperative building block which mostly consists of a information processing and a physical part. Holonic system structure is based on the principles of self-similarity and self-configuration which form self-adaptable and thus fault-tolerant networks of holons.

Our HMS example is composed of a set of different stations and a transport system as it is illustrated by the virtual

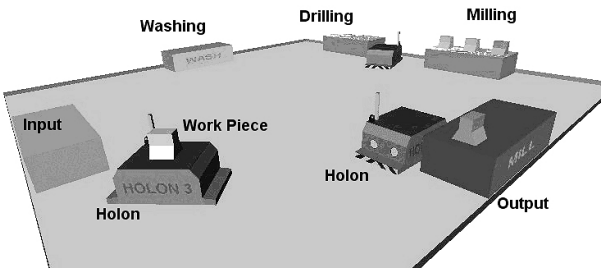

Figure 3. Screenshot of a 3D HMS Simulation

3D model in Figure 3. The transport system consists of a set of AGVs (Automated Guided Vehicles). An AGV is an autonomous vehicle which moves parts between working stations. Parts enter the system at an input station. AGVs take parts to different subsequent manufacturing stations where they are transformed by milling, drilling, and washing. Finally, parts are taken to an output station and leave the system.

Stations have an input and an output buffer for incoming and outgoing parts. Once having located an item at its output buffer, a station

1. broadcasts a request for delivery to all AGVs
2. receives replies h_i from idle AGVs
3. selects one AGV h_i
4. notifies AGV h_i for its acceptance, and notifies all other AGVs for their rejection.

On the other hand, each AGV h_i

1. is idle until it receives a request for delivery from a station s_j
2. sends a reply to s_j on request of s_j
3. moves to s_j on notification of acceptance from s_j
4. takes an item from the output buffer of s_j
5. asks the next destination station s_k ($s_k \neq s_j$) for permission to deliver the item
6. moves to s_k on notification of acceptance from s_k
7. unloads the item at the input buffer of s_k
8. moves to a parking position and returns to step 1.

4. Collaborative Development through 3D Animation

This section outlines how the refined design pocress illustrated in Figure 2 is applied to the case study of the previous section. After conceptual modeling, we discuss implementation and verification.

4.1. Conceptual Modeling

System behavior can be meaningfully described as a collection of representative interaction scenarios similar to UML's sequence and collaboration diagrams based on the agent-based metaphor. Consider the following example: For a station in the context of the case study, the goal to perform a transport has been identified. Figure 4 illustrates the main steps to achieve the goal, given as plan 1. Complex steps can again be goals, for which separate scenarios have to be developed. Note that in general several plans to achieve a goal are possible. For instance, consider the subgoal for selecting an AGV. The station could choose between different plans to determine an AGV, e.g., the nearest, the most reliable, or the least used AGV.

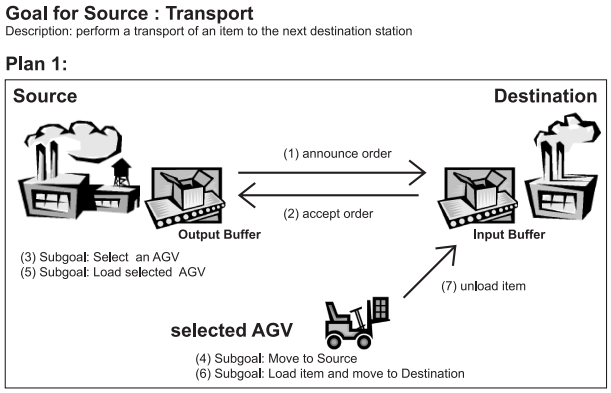

Figure 4. Interaction Scenario for Transports

The scenarios should provide representative examples of the application objects, their individual behavior, and interaction. Scenario analysis refines the scenarios into structured situation descriptions that detail interactive, communicative, and behavioral content. In the design process, the scenarios are progressively refined to define communicative goals of agents and behavior for their achievement by deliberative means. The designer takes these descriptions to identify relevant agents and behaviors. The behavior can then be transferred to an adequate 3D presentation model and a precise state machine based implementation model (e.g., UML StateCharts [6]) for simulation and formal verification.

The resulting agents and behaviors have a direct mapping to the state machine based implementation and the actor-based i4D animation framework.

An i4D application consists of a number of actors placed on a stage. Actors include conceptual objects like lights and cameras, visual 3D objects, and even software elements without a visual representation. Actors can be hierarchically structured and have attributes that describe their properties. They perform actions (e.g. animations, sending messages, sound) by continuously modifying their attributes.

Incoming Command	Reply
createMT wash 90.0 50.0 90.0	wash1
startMT wash1	ok
...	
createAGV 50.0 50.0 270.0	agv1
startAGV agv1	ok
moveAGVtoOutBuffer agv1 in	ok
startAGVhandOver agv1	ok
moveAGVtoInBuffer agv1 mill	ok
startAGVhandOver agv1	ok
moveAGVtoParking agv1	ok
...	
getAGVnextObstacle agv1	24.2 agv3

Table 1. Sample Animation Sequence

The 3D actors are connected to the system implementation through an interface. In the context of our case study, a Tcl/Tk script with a TCP/IP socket interface acts as a server for handling incoming animation commands. Table 1 shows some sample commands for creating objects and manipulating actuators (e.g., moving around, loading parts). Moreover, information from virtual sensors can be retrieved, e.g., AGVs can use animated "ultrasonic sensors" to determine the distance from obstacles. Each command is associated with an animation sequence, e.g., StartAGVHandOver starts an animation that loads an AGV with an item at a station. More details of specification and realization of the interface is given in [1].

4.2. State Machine Based Implementation

A state machine based implementation is derived from agent behavior. For each agent, reactive behavior has to be identified in detail and implemented as states and state transitions.

Figure 5 illustrates how the behavior of a station input buffer is divided into a negotiation part (Acceptor) and an actual loading part (Loader) which both run in parallel. That state machine can serve as a first executable implementation which can be used for simulation. Events in the simulation control the 3D animation through the interface. This application can be used as a first feedback to the customer.

On the other hand, the implementation can be processed further to an input for formal verification, e.g., for model checking. Figure 6 gives an example for RIL code which is applicable for the RAVEN model checker. The example demonstrates that StateCharts can be more easily understood than RIL code.

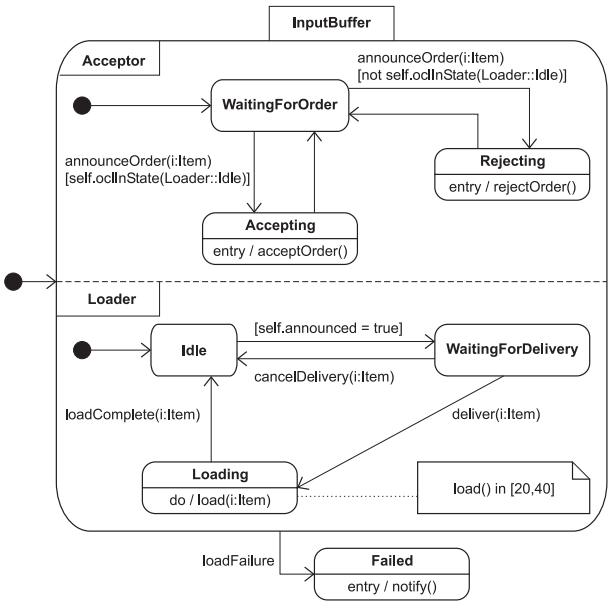

Figure 5. UML StateChart Example

4.3. Formal Verification

Symbolic model checking is well established in the formal verification of small systems, in particular in the field of electronic system design [2]. Compared to classical simulation, the most notable benefit of model checking is the completely automated ellaboration of the complete state space of a system.

The primary input for model checking is a system model given by a set of finite state machines. As a second input, model checking needs a formal specification of required properties. The requirements are typically given as formulae based on temporal logics. Once having specified a property as a temporal logic formula f_i, a model checker can directly answer the question "Does a given model satisfy property f_i?" by either true or false. On request, a model checker typically generates counter examples when the model does not satisfy a specified property. A counter example demonstrates an execution sequence that leads to a situation that falsifies the property, which is very helpful for detailed error analysis.

4.3.1. Real-Time Model Checking with RAVEN. For systems verification, we apply the RAVEN model checker [9] that supports verification of real-time systems. In the following, we give a brief introduction to RAVEN.

In RAVEN, a model is given by a set of time-annotated extended state machines called I/O-interval structures. I/O-interval structures are based on Kripke structures with [min,max]-time intervals and additional input conditions at their state transitions. A more detailed and formal specifi-

```
MODULE acceptor
SIGNAL state : {waitingForOrder,rejecting,accepting,failed}
INPUTS announced   := global_newOrderForMachine
       loadFailure := global_loadFailure
       idle        := (loader.state = loader.idle)
DEFINE rejectOrder := (state = rejecting)
       acceptOrder := (state = accepting)
INIT   state = waitingForOrder
TRANS
|- state=waitingForOrder
   --  loadFailure                      --> state:=failed
   -- !loadFailure &  idle & announced --> state:=accepting
   -- !loadFailure & !idle & announced --> state:=rejecting
                                       !-> state:=waitingForOrder
|- state=rejecting
   -- loadFailure    --> state:=failed
                     !-> state:=waitingForDelivery

... // some more transitions omitted
END
```

Figure 6. RIL Code for Acceptor

```
AG(
   (acceptor.state = acceptor.accepting)
    -> AF[99](
               (loader.state = loader.waitingForDelivery) &
               AX (loader.state = loader.loading)
              )
  )
```

Figure 7. Sample CCTL Formula

cation of I/O-interval structures is presented in [11].

For property specifications on a given set of I/O-interval structures, the temporal logics Clocked Computational Tree Logic (CCTL) is applied [10]. CCTL formulae are composed from propositions denoting predicates in combination with boolean connectives and time-annotated temporal operators. Timing interval specification is a significant benefit of CCTL when being compared to the classical Computational Temporal Logics (CTL) [2].

In the context of RAVEN, I/O-interval structures and a set of CCTL formulae have to be specified by means of the textual RAVEN Input Language (RIL). A RIL specification contains (a) a set of global definitions, e.g., fixed time bounds or frequently used formulae, (b) the specification of parallel running modules, i.e., a textual specification of I/O-interval structures, and (c) a set of CCTL formulae, representing required properties of the model.

Figure 6 gives a code fragment of the RIL model for our running example. For property specification, consider the following example. One requirement in our case study is that the input buffer of a station must not be blocked for too long in order to ensure sufficient continuous workload, i.e., each accepted delivery request must be followed by actually loading an item at the input buffer not later than 100 time units after acceptance. Due to the dependency on other modules, in particular the AGVs, it is not obvious whether the model satisfies this property. The designer has to specify a corresponding CCTL formula as it is shown in Figure 7. Additional property specifications for the manufacturing case study can be found in [3].

4.3.2. Presentation of Results. If a CCTL formula is proved to be incorrect, a counter example execution run can be generated. Execution runs are given by time-annotated sequences of state changes. RAVEN invokes a built-in waveform browser that lists all variables and their states over time; but only boolean variables can be visualized (Figure 8).

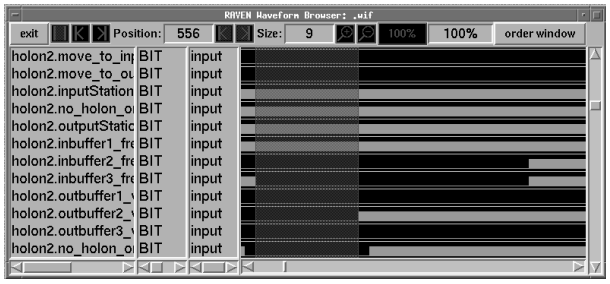

Figure 8. Waveform Browser

That waveform representation has two major drawbacks. On the one hand, it is hard to interpret when the internal bit representation has no direct correspondence to the high level objects in the implementation, e.g., when the behavior given by 5 states and 15 state transitions of a vehicle is modeled by more than 50 variables. On the other hand, it is hard for uneducated designers to interpret the results. To overcome the limits of waveform representation, we provide an intuitive application-dependent animated 3D model to visualize sample model execution runs for advanced illustration of counter examples. A screenshot of the case study scenario is shown in Figure 9.

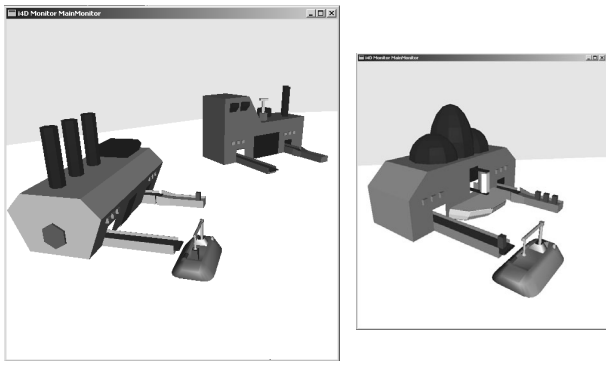

Figure 9. Screenshot of the Case Study

4.4. Animation Environment

For modeling and animation, we use the component-based high-level graphics library i4D [4]. Figure 10 illustrates the architecture of the i4D system. The central unit of

this system is the management component that is responsible for organization and execution of the different i4D components.

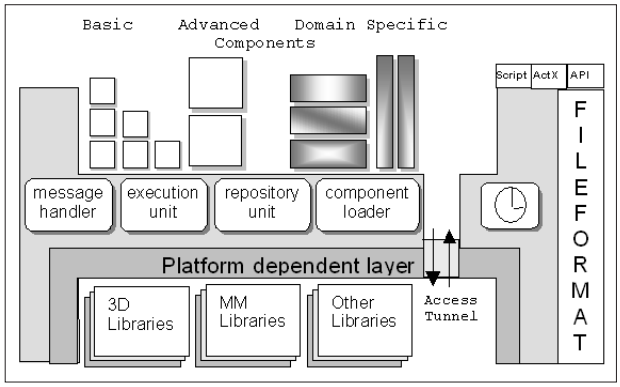

Figure 10. i4D Structure

i4D is based on a framework that allows to manage different components in a flexible way and provides a structured and open interface for external (third party) components. Therefore, a number of different systems can be seamlessly integrated (e.g., simulation systems, multi agent systems). i4D runs on NT and Linux and uses the Open GL graphics standard. A programming interface for C++ and a script binding using Tcl/Tk is supported for application development. A TCP/IP interface can be used to communicate with other applications.

5. Summary and Conclusion

We have presented a refined model for a collaborative systems development process. The refined process covers current practice as it is applied in collaborative computer-based systems design and focuses on state-based systems. Our approach integrates external representation (3D animation) into the design process for computer-based concurrent systems.

A first evaluation of our environment has been conducted using the example of a flexible manufacturing system of considerable complexity. However, in order to draw general conclusions further evaluation is required.

While i4D is well suited for 3D visualization, creating the interface still requires too many manual interactions, as it has to be built from scratch. We are currently investigating a basic set of domain-oriented commands that are dedicated for several applications.

Acknowledgements

We gratefully thank Malte Beyer for modeling the case study scenario in 3D and Tim Schmidt for implementing the animation prototype in i4D. Christian Reimann was involved in the implementation of the i4D kernel. This work is supported in part by the DFG project GRASP within the DFG Schwerpunktprogramm 1064 ”Integration von Techniken der Softwarespezifikation fuer ingenieurwissenschaftliche Anwendungen”.

References

[1] A. Braatz, S. Flake, W. Mueller, and E. Westkaemper. ”Prototyping einer Fahrzeugsteuerung in virtueller 3D-Umgebung”. In *Simulation und Visualisierung 2000*, Magdeburg, Germany, March 2000. (in German).

[2] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA, 1999.

[3] S. Flake, W. Mueller, U. Pape, and J. Ruf. ”Analyzing Timing Constraints in Flexible Manufacturing Systems”. In *International NAISO Symposium on Information Science Innovations in Intelligent Automated Manufacturing (IAM’2001)*, Dubai, March 2001.

[4] C. Geiger, V. Paelke, C. Reimann, and W. Rosenbach. ”A Framework for the Structured Design of VR/AR Content”. In *ACM Symposium on Virtual Reality Systems and Techniques, VRST*, Seoul, Korea, October 2000.

[5] H. Nwana and D. Ndumu. ”A Perspective on Software Agents Research”. *Knowledge Engineering Review, Cambridge University Press, Cambridge, MA*, 1999.

[6] OMG. *UML Unified Modelling Language Specification, Version 1.3*. Object Management Group, March 2000. URL: ftp://ftp.omg.org/pub/docs/formal/00-03-01.pdf.

[7] V. Paelke. ”Systematic Design of Interactive Illustration Techniques for User Guidance in Virtual Environments”. In *Proceedings of IEEE VR 2000*, March 2000.

[8] A. Repenning and T. Summer. ”Agentsheets: A Medium for Creating Domain-Oriented Visual Languages”. *IEEE Computer*, 28(3), 1995.

[9] J. Ruf. ”RAVEN: Real-Time Analyzing and Verification Environment”. *Journal on Universal Computer Science (J.UCS), Springer, Heidelberg*, February 2001.

[10] J. Ruf and T. Kropf. ”Symbolic Model Checking for a Discrete Clocked Temporal Logic with Intervals”. In *Conference on Correct Hardware Design and Verification Methods (CHARME)*, Montreal, Canada, October 1997.

[11] J. Ruf and T. Kropf. ”Modeling and Checking Networks of Communicating Real-Time Processes”. In *Int. Conf. on Correct Hardware Design and Verification Methods*, Bad Herrenalb, Germany, 1999. Springer-Verlag.

[12] E. Westkaemper, M. Hoepf, and C. Schaeffer. ”Holonic Manufacturing Systems (HMS) - Test Case 5”. In *Proceedings of Holonic Manufacturing Systems*, Lake Tahoe, CA, February 1994.