Combining Statecharts and Collaboration Diagrams for the Development of Production Control Systems

Position Paper

Ulrich Nickel, Jörg Niere, Wilhelm Schäfer, Albert Zündorf
AG-Softwaretechnik
University of Paderborn, Germany
[duke|nierej|wilhelm|zuendorf]@uni-paderborn.de
D-33095 Paderborn

1 Motivation

Todays manufacturing industry faces big challenges. The market demands a very flexible production process which adjusts quickly to e.g. changing characteristics of the produced goods or different lot sizes and which tries to minimize production costs as much possible. In order to meet these challenges todays production control systems become more and more decentralized [GBFG96]. The building blocks of such a production control system are different, selfacting and computer controlled resources like e.g. switches, shuttles, machines or robots.

One major problem in building those decentralized systems is to develop a detailed and precise plan how those different components interact and in particular how their control software has to behave. There is no specification approach which allows to define, analyze and simulate such a production control system upfront before it is actually built, physically. With regard to the frequent changes, this is of course an important cost factor which should be avoided.

Such a production control system may be regarded as a system of active autonomous objects. These objects communicate with each other to synchronize their concurrently executed activities. We use state-charts [Har84] to specify the behaviour of the objects and SDL [ITU96] to specify the communication aspects. Both are very popular in mechanical and electronic engineering science because they fit the demands for the specification of systems e.g. in the above picture. Many specification tools generate a C-dialect code out of specifications, and the correctness of the specification must be tested by a physical simulation.

We decided to use the FUJABA environment [FNT98, FNTZ99] to specify production control systems. FUJABA is an integrated development environment for UML class diagrams, collaboration diagrams, and Story-Diagrams. The environment allows to generate executable Java code out of specifications and to simulate the execution. Furthermore, the environment is currently extended to state-charts and SDL.

2 Integrating State-Charts and UML Collaboration Diagrams

Figure 1 shows a scenario of a sample factory. The factory is modeled as a flat building without levels and pillars in it. The floor is layered with rectangle shaped fields allowing to address certain positions in the building. The factory contains certain kinds of production places. A production place is e.g. an assembly line, where goods arrive and are loaded on shuttles, or a storage, where goods can be stored.

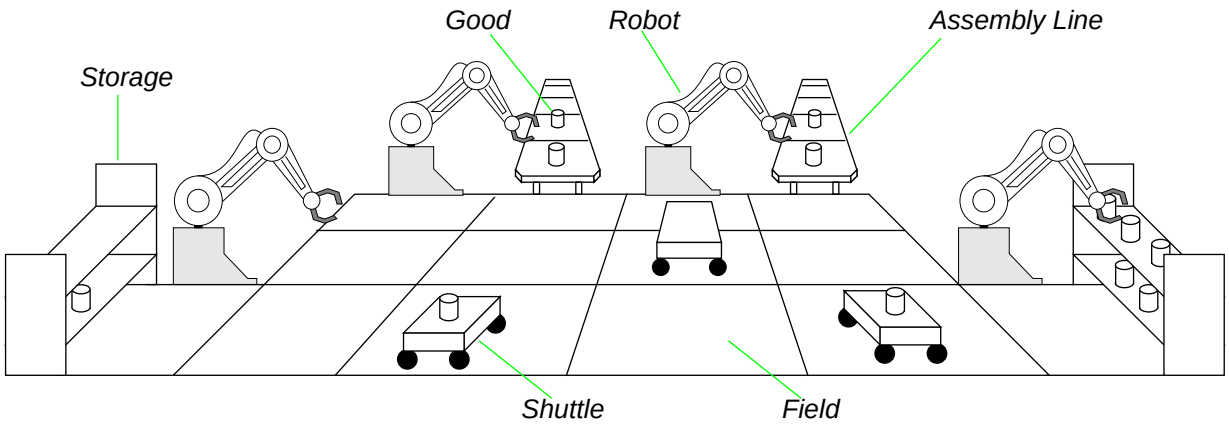

Figure 1 Simple factory example

State-charts provide sophisticated means for the specification of (concurrent) control flows for reactive objects (shuttle, assembly line, storage). However, by purpose state-charts abstract from the complex application specific data structures that build up the concrete states of a system. State-charts do not explicitly deal with values of attributes or with links to other objects nor with the evolution and changes of these object-structures caused by the execution of operations or action methods.

The specification of application specific object-structures is a well known application area for graph grammars, cf. [Roz97]. Basically, a graph grammar rule allows the specification of changes to complex-object-structures by a pair of before and after snapshots. The before snapshot specifies which part of the object-structure should be changed and the after snapshot specifies how it should look like afterwards, without taking care of regarding how these changes are achieved. While graph grammars are appropriate for the specification of object-structure modifications, they lack appropriate means for the specification of control flows.

To overcome these problems, we propose to combine state-charts and graph-rewriting rules. We use state-charts (and activity diagrams) to specify complex control flows and graph rewriting rules to specify the entry, exit, do, and transition actions that deal with complex object-structures. In order to facilitate the use of graph rewriting rules for object-oriented designers and programmers, we propose to adopt UML collaboration diagrams as a notation for object-structure rewriting rules.

Figure 2 shows an example for the combination of state-charts (activity diagrams) and object-structure rewrite rules within an UML collaboration diagram. The left half of Figure 2 shows a cut-out of the state-chart of class shuttle, namely its fetch state. The initial substate of state fetch is startFetch. In our example, the do-action of state startFetch is specified via an object-structure rewriting rule that shows three objects, this, f, and al. The this object is attached to the Field object f via an at link and the

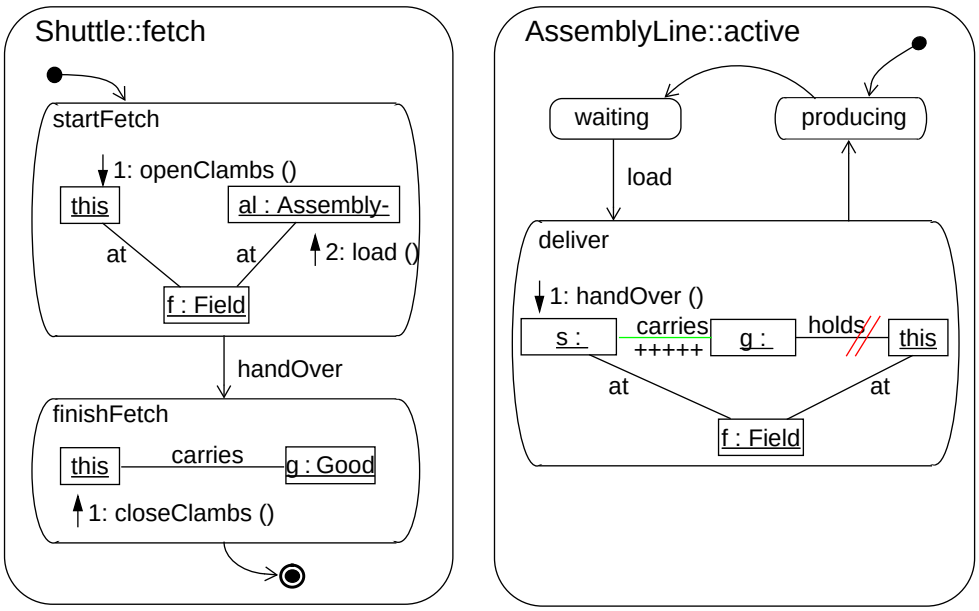

Figure 2 Refinement of state fetch

same holds for the AssemblyLine object al. We interpret this, al and f as variables and the shown links as logical constraints. Such a diagram is executed by binding the specified variables to concrete object instances such that all specified constraints are fulfilled.

In our example variable f is bound to the field object that is connected to the current shuttle object (bound to variable this) via an at link. In turn, al is bound to the AssemblyLine object that is attached to that field object via an at link. Once all variables are bound to appropriate object instances the specified change effects and method invocations are executed. State startFetch shows no object-structure changes but two method invocations (the do-action of state startFetch is a pure collaboration diagram specified in UML notation). Thus, in step 1 startFetch calls method openClambs on the object this, i.e. the current shuttle. In step 2, startFetch calls method load at the AssemblyLine al. After that the do-action terminates and state startFetch is ready to accept the next event. Let us assume, that the AssemblyLine object bound to variable al is in the state waiting, cf. the right-hand side of Figure 2.

Thus, the load method invocation on al generates a load event that causes al to switch to activity deliver. In this case we deliver is named an activity, because there are only triggerless outgoing transitions and to underline the parallelism between state-charts and activity diagrams. Activity deliver is again specified by an object-structure rewriting rule. In activity deliver variable this represents the current assembly line. Deliver determines its field f, the shuttle s attached to field f and a good g it holds. The activity shows two object-structure changes. First, the holds link connecting variable this and g is canceled by two small lines. This is executed by deleting the corresponding link. Second, the + symbols attached to the carries link connecting variables s and g indicates that such a link is created. Finally, activity deliver calls method handOver at shuttle s and terminates. On termination of activity deliver, the outgoing triggerless transition fires and the corresponding assembly line object switches to activity producing. Once our shuttle object receives the handOver event the corresponding transition fires and activity finishFetch is triggered. finishFetch just checks whether the shuttle actually carries a good and then it closes its clambs and terminates. Thereby, the terminal state is reached and the whole fetch activity terminates.

The FUJABA environment is able to generate Java code for such behavioural specifications[1]. For the static aspects of a system UML class diagrams are used (details, see [FNT98, FNTZ99]). The advan-

1. To be honest, currently only activity diagrams and collaboration diagrams are supported. State-charts are scheduled for August 1999.

tage of FUJABA in comparision to other tools is the opportunity to validate a specification with a graphical simulation environment before it is physically tested.

3 Simulation of the system

Figure 3 shows the graphical debuging tool of FUJABA, called Dobs (dynamic object browsing system). On the right-hand side, the screenshot shows a cut-out of the current object structure of the running program. There are four field objects (f1-f4). Each field is linked to its neighbours. The assembly line (a6) holds an instance of the class Good (g7). Both, the assembly line and the shuttle s5 are linked to the same field. Now, the shuttle can take the good to carry it to the storage. Therefore, the method load of the assembly line a5 has to be executed. In our example, the user has selected the assembly line and the tool displays the public methods of class AssemblyLine on the left part of the screenshot.

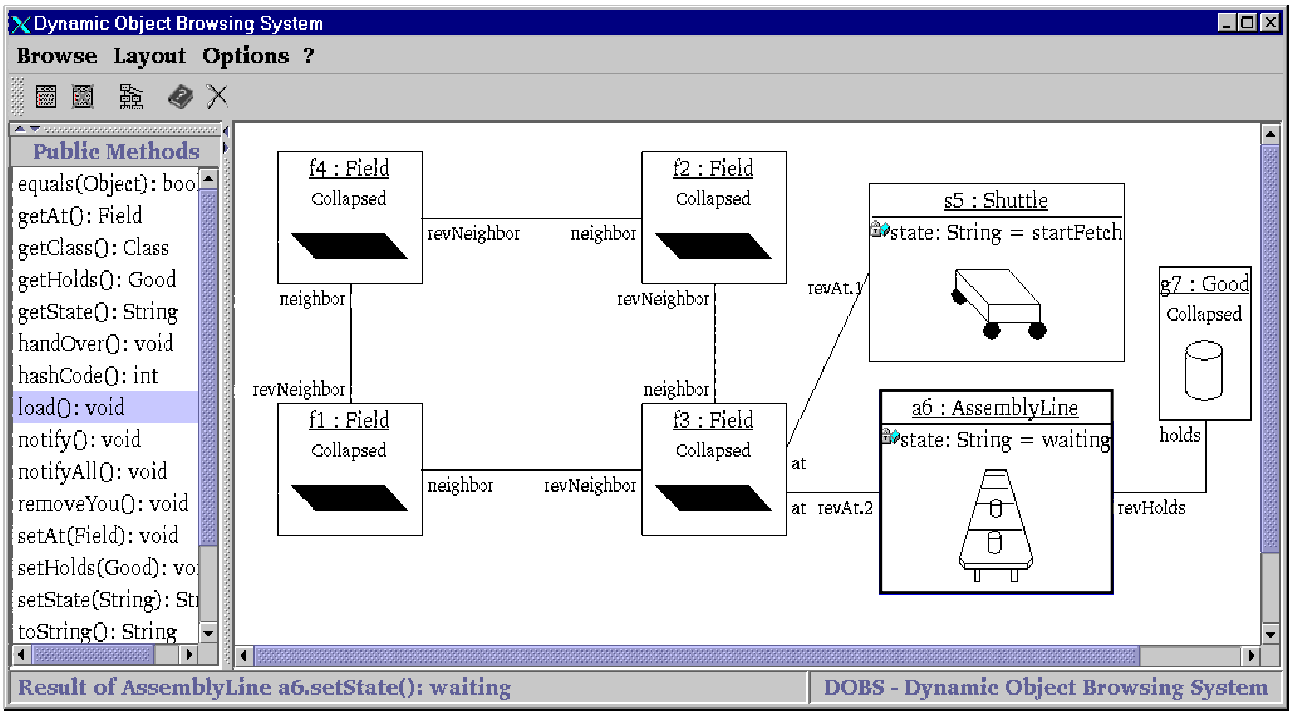

Figure 3 Dynamic Object Browser

Now, the user can invoke the method AssemblyLine::load() and the dynamic objects browser displays the changed object structure after the execution has terminated. Then, the good will have a carries link to the shuttle and the holds link to the assembly line will be deleted.

We plan to employ FUJABA for the specification, validation/simulation, and control of complex production control systems. Thus, it is indispensable to provide a simulation environment which runs automatically and allows for real time aspects. Additionally, a more sophisticated graphical representation should be implemented. The user must be able to check the states and event queues of all objects and the tool has to show the event/message flow between these objects. Last but not least, we have to animate the specified system to validate, whether it works as intended.

4 Conclusions and Future Work

FUJABA has been developed since November 1997. The first ’release’ version has supported an editor for UML class diagrams and UML activity diagrams and object structure rewriting rules. It comprises a code generator and a basic consistency analyser.

Dobs, the Dynamic Object Browsing System, has become part of FUJABA in the beginning of 1998. The assistance of Design Pattern [GHJV95] recognition and instantiation is also a part of the environment (not mentioned in this paper).

As current work FUJABA is enhanced by state-charts and SDL. For both an editor, a consistency analyser, and a code generator are scheduled for autumn 1999 and spring 2000, respectively.

The described extentions of Dobs up to a graphical simulation environment are also current work and are scheduled for next year. Since the physical parts of our production control system e.g. shuttles, have to be programmed in Neuron-C and not in Java, the code generator must be accomodated for those dialects. We assume, next generation shuttles, switches, assembly-lines, toasters, televisions, etc. have an integrated Java-Chip so that the specifications can be executed directly.

References

[FNT98] T.Fischer, J. Niere, L. Torunski. *Design and Implementation of an integrated Development Environment for UML, Java, and Story Driven Modeling*, Master Thesises, Paderborn 1998 (in German)

[FNTZ99] T.Fischer, J. Niere, L. Torunski, A. Zündorf. *Story Diagrams: A New Graph Grammer Language based on the Unified Modelling Language and Java*. to appear in Proceedings of TAGT '98 (Theory and Application of Graph Transformations), LNCS, Springer 1999

[GBFG96] J. Gausemeier, H.-J. Buxbaum, S. Förste, G. Gehnen. Decentral Control Architecture for Modular Flow Systems. *Proceedings CAD/CAM Robotics and Factories of the Future*, London, England, 14. - 16. August 1996.

[GHJV95] E.Gamma, R.Helm, R. Johnson, J.Vlissides. *Design Patterns*. Addison-Weslay, 1995.

[Har84] D. Harel. *Statecharts: A visual approach to complex systems.* CS84-05, Department of Applied Mathematics, The Weizmann Institute of Science, 1984

[ITU96] ITU-T Recommendation Z.100. *Specification and Description Language (SDL).* International Telecommunication Union (ITU), Geneva, 1994 + Addendum 1996

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen. *Object-Oriented Modelling and Design*. Prentice Hall, Englewood Cliffs, N. J. 07632, 1991.

[Roz97] G.Rozenberg (ed). *Handbook of Graph Grammars and Computing by Graph Transformation*. World Science, 1997.