Graph-Grammar Based Completion and Transformation of SDL/UML-Diagrams

Position Paper

Ulrich A. Nickel, Robert Wagner

University of Paderborn
Warburger Straße 100
D-33098 Paderborn
Germany
[duke, wag25]@uni-paderborn.de

1 Introduction

In the last years the Unified Modelling Language has become more and more popular. It has been successfully applied in many application domains. Thus, many effort is put into the extension of the UML in order to make it applicable in formerly untypical domains. One example is the development of UML-RT [SGW94]. UML-RT extents the UML by adding the possibility of defining the communication structure of a distributed system. Unfortunately, it lacks of the definition of a precise semantics. Moreover, often there are other graphical specification techniques which are commonly used. In the domain of communication engineering, the Specification and Description Language (SDL) [SDL96] has emerged to a standard. Additionally, the semantics of SDL is formally defined.

At our department, we are working on a project which aims at the development of a seamlessly integrated graphical language for the specification of distributed production control systems. We employ SDL for the specification of the overall communication structure of the system. Whereas the modelling of complex object structures is a typical field of application for the UML. Therefore, we developed a graphical language, which uses SDL block diagrams, UML class diagrams, and UML behaviour diagrams like collaboration diagrams, activity diagrams, and statecharts [KNNZ00].

In a first step, the engineer models the topology of the system. This includes the identification of the participating processes and the definition of the communication channels and all kinds of interchanged signals. In a second step, an initial class diagram is automatically generated out of the SDL block diagram which can now be refined. Additional steps follow until the structure and the behaviour of the system are completely defined.

One of the major problems of such an approach is that there are several ways of mapping one diagram to another. In many case tools the automatic generation is hard coded and cannot be changed, if necessary. Thus, we developed a flexible transformation mechanism which is based on graph-grammars. Thereby, we can formally define the mapping between the different kinds of diagrams, which also allows the automatic checking of consistency.

2 Example

As an example we use a *ServiceMachine* system illustrated in Figure 1. A *ServiceMachine* system offers services to a number of clients. First, each client must open a session by sending the *Login* signal to the *Monitor* process. The *Monitor* process exists from initialization time. The *Monitor* process creates a new *Service* process for each user which has opened a session. Also, a valid communication path between the created *Service* process and the client is established. After the reception of the *Loginack* signal, the user can ask for the service with the signal *Request*. When a user terminates a session with the signal *Logout*, the *Service* process also terminates and informs the *Monitor* process.

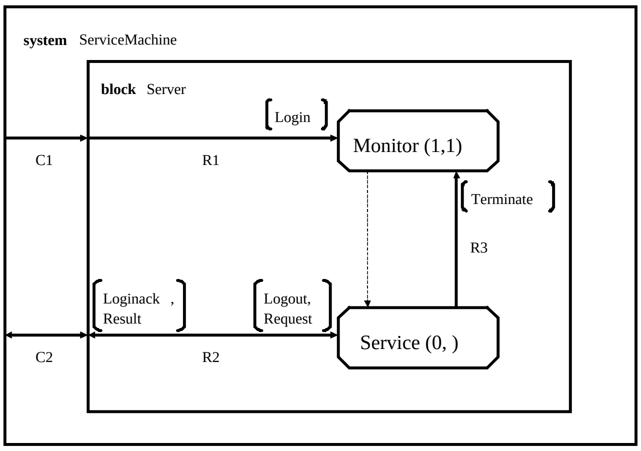

Figure 1, SDL specification of the ServiceMachine system

The mapping of the *ServiceMachine* system to UML is illustrated in Figure 2. Basically, each process is mapped to an active class, e.g. process *Monitor* is mapped to the class *Monitor* and process *Service* is mapped to the class *Service*. Additionally, each class capable of receiving a special signal implements a method named equal to the signal. Communication channels and signalroutes form distinct paths, which are transformed to associations between the communicating process classes.

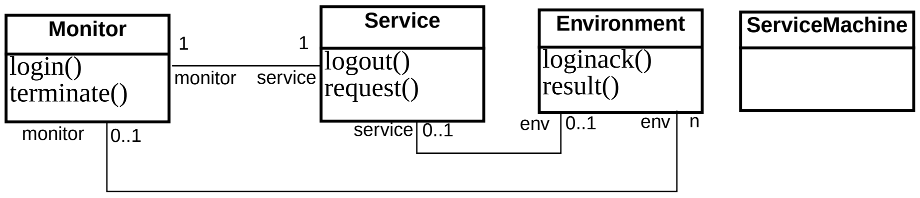

Figure 2, UML class diagram of the transformed ServiceMachine example

The *Environment* class represents the user and handles incoming and outgoing signals addressed to the *ServiceMachine* system. In contrast, the *ServiceMachine* class is responsible for system initialization and process management. This comprises creation and termination of processes and the validation of the maximum number of process instances in the system. This way we guarantee the synchronous semantic of process creation defined in SDL

Since our integrated development environment is capable of editing SDL and UML diagrams [KNNZ00, NNZ00, NZ99], we considered to use story-diagrams to specify the transformation between these two kinds of diagrams. From this story-diagram specification, we will generate Java source code, which can be integrated in the development environment and executed on demand. This way we achieve an automated mapping from SDL block diagrams to UML class diagrams. Note that both the specification and the code generation has to be done only once.

3 Graph-Grammar Based Transformations

The specification of changes to application specific object-structures is a well known application area for graph grammars [Roz97]. A graph rewrite rule describes the changes to an object-structure 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 how this changes are achieved.

While graph grammars are appropriate for the specification of object-structure modification, they lack appropriate means for the specification of control flows.

To overcome this problem we introduced UML activity diagram as control flow notation for graph rewrite rules, cf. [JZ98, FNTZ98]. In order to facilitate the use of graph rewrite rules for object-oriented designers and programmers, we additionally adapted UML collaboration diagrams as a notation for object-structure rewrite rules. For this combination of activity diagrams and collaboration diagrams we use the name *story-diagrams*.

We implemented a case tool called Fujaba[1], which uses an SDL/UML abstract syntax graph (ASG). A part of the ASG is presented in Figure 3. For more details about our integrated meta model of SDL/UML see [KNNZ00].

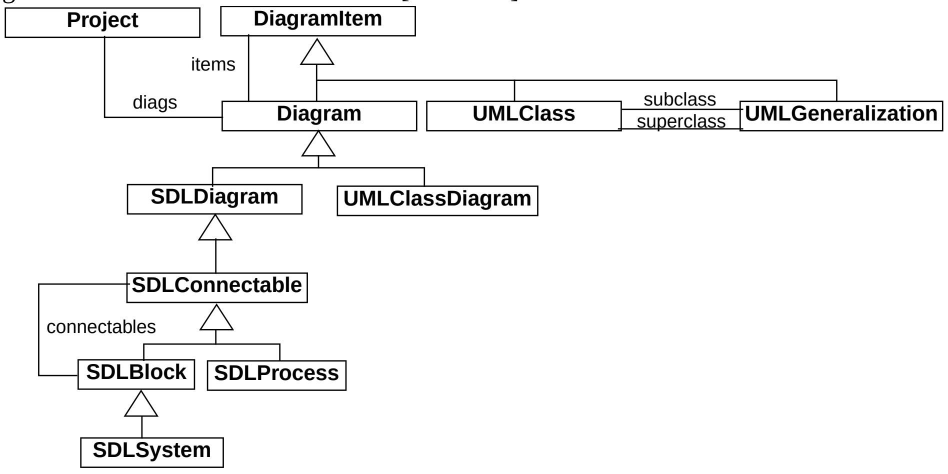

Figure 3, SDL and UML Metamodel used in Fujaba

[1] From UML to Java and Back Again. See http://www.fujaba.de for further information.

In our SDL model, blocks and processes are treated as own subdiagrams. Thus, both model elements, *SDLBlock* and *SDLProcess*, inherit all properties from *SDLConnectable*, which is a kind of *SDLDiagram*. Each top level system is a kind of block and therefore inherits from *SDLBlock*. Each *SDLBlock* can hold via the *connectables* association either further subblocks or processes, but not both.

With some knowledge about the object-structure used in Fujaba, we can specify the transformation by using story-diagrams. Figure 4 illustrates a sample transformation of a SDL system to UML classes. Because of lack of space, the transformation of paths and signals is not demonstrated.

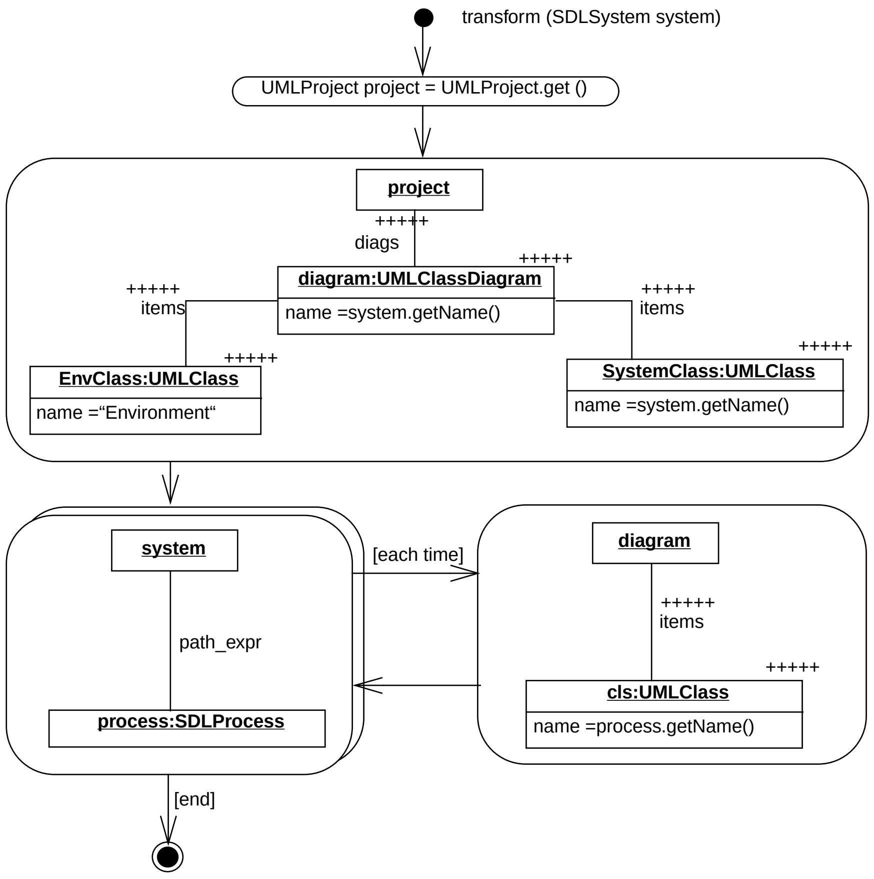

Figure 4, Story-diagram for the transformation of SDL processes to UML classes

In the first activity, the current project is retrieved and stored in the local variable *project*. This is achieved by a simple call to *UMLProject.get ()*. In the next activity, a new *UMLClassDiagram* node *diagram* named equal to *SDLSystem* diagram is created (shown by the *'+++++'* annotation and the assertion *name=system.getName()*) and a link between the project and the created *UMLClassDiagram* node is inserted. In the same manner the classes *EnvClass* and *SystemClass* are created

Next, the flow changes to the next activity. In this activity, a path expression *path_expr* is used to find all nodes of type *SDLProcess*. For each process found, the transition annotated with [*each_time*] is executed. There, a new *UMLClass* node is created and receives the same name as the considered *SDLProcess* node. After successful creation, the control is given back to the path expression and the next *SDLProcess* node is being searched for. If all *SDLProcess* nodes have been considered, the stop activity is reached and the (partial) transformation is complete.

4 Conclusions and Future Work

In this position paper, we presented a graph-grammar based approach for the automatic completion and transformation of SDL and UML diagrams.

Currently, we implement these concepts in Fujaba. This case tool has been developed at our department since November 1997. It supports the described parts of SDL and UML and is able to generate executable Java-code.

As future work, we plan to develop a system for the intelligent management of consistency rules. The system will take into account that some consistency checks depend on each other. Moreover, it will allow the user to classify inconsistencies as recommendation or potential inconsistency, for example.

5 References

[FNTZ98] T. Fustier, 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 6th Int. Workshop on Theory and Application of Graph Transformation (TAGT), Paderborn, Germany*. Springer Verlag, 1998.

[JZ98] J.H. Jahnke and A. Zündorf. Specification and Implementation of a Distributed Planning and Information System for Courses based on Story Driven Modeling. In *Proc. of 9th International Workshop on Software Specification and Design, Ise-Shima, Japan*, pages 77–86. IEEE Computer Society Press, 1998.

[KNNZ00] H.J. Köhler, U. Nickel, J. Niere, and A. Zündorf. Integrating UML Diagrams for Production Control Systems. In *Proc. of the 22th Int. Conf. on Software Engineering (ICSE), Limerick, Irland*. ACM Press, 2000.

[NNZ00] U. Nickel, J. Niere, and A. Zündorf. Tool demonstration: The FUJABA environment. In *Proc. of the 22th Int. Conf. on Software Engineering (ICSE), Limerick, Irland*. ACM Press, 2000.

[NZ99] J. Niere and A. Zündorf. Using Fujaba for the Development of Production Control Systems. In *Proc. of Int. Workshop and Symposium on Applications Of Graph Transformations With Industrial Relevance (AGTIVE), Kerkrade, The Netherlands*, LNCS. Springer Verlag, 1999.

[Roz97] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation*. World Scientific, Singapore, 1997.

[SDL96] International Telecommunication Union (ITU), Geneva. *ITU-T Recommendation Z.100: Specification and Description Language (SDL)*, 1994 + Addendum 1996.

[SGW94] B. Selic, G. Gullekson, and P. Ward. *Real-Time Object Oriented Modeling*. WILEY, 1994.