Integration of $\mathcal{Z}$-Based Semantics of OO-Notations

Jürgen Ebert, Roger Süttenbach
University of Koblenz-Landau, Institute for Software Technology
{ ebert, sbach }@informatik.uni-koblenz.de

Abstract. This paper shows how an integrated and formalized description of the abstract syntax and the semantics of the notations of object-oriented methods can be produced by extending EER/GRAL descriptions of the syntax by an operational semantics notated in $\mathcal{Z}$.

Keywords: operational semantics, EER/GRAL, object-oriented methods, declarative modeling, abstract syntax, integrity conditions, $\mathcal{Z}$.

In order to formalize the semantic aspects of an object-oriented method its (textual and visual) languages have to be described precisely. Here, we develop an operational semantics based on abstract syntax. Therefore, we will present the description of the abstract syntax at first.

1 Abstract Syntax

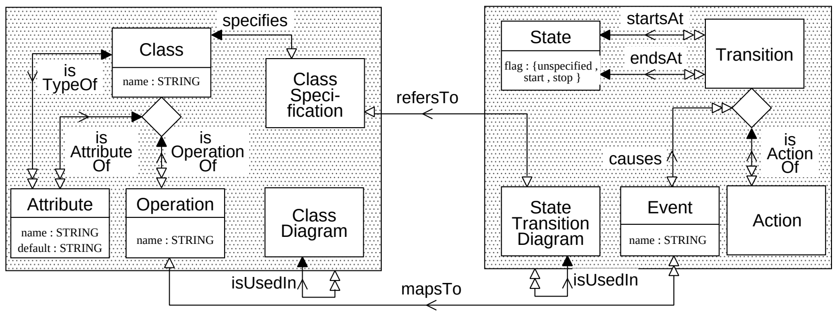
The abstract syntax of a notation used in an object-oriented method can be described by a *metamodel*. The set of instances of this model is the set of *abstract syntax graphs* of the documents (usually diagrams) which are used for modeling real systems using that method.

For every document there is one corresponding syntax graph. This graph can be checked whether it is correct with respect to the metamodel or not. A diagram is a correct diagram if its corresponding syntax graph is an instance of the metamodel of the method.

We use the *EER/GRAL approach* of modeling to achieve a formal description of syntax graphs as described in [EWD+96]. Here, graph classes – sets of graphs – are defined using *extended entity relationship (EER) descriptions* [CEW94] which are annotated by integrity conditions expressed in the $\mathcal{Z}$-like *assertion language GRAL* (GRAph specification Language) [F97].

Example: Figure 1 is an example of an EER/GRAL description which shows a small part of the Booch metamodel [SE97]. Roughly speaking, it defines two kinds of diagrams – ClassDiagrams and StateTransitionDiagrams. Class diagrams describe the class structure of a system and state transition diagrams describe the sequences of operations which are possible for a given class.

In the example, GRAL predicate CD4 demands that the names of classes in



For G **in** *Booch* **assert**

$$\ldots$$

$\mathsf{CD4}:\quad \forall\, c_1, c_2 : V_{Class} \mid c_1 \neq c_2 \bullet c_1.name \neq c_2.name\ ;$

$$\ldots$$

$\mathsf{STD9}:\quad \forall\, e : V_{Event};\ o : V_{Operation} \mid e \rightharpoonup_{mapsTo}\ o$

$\quad \bullet\ e \rightharpoonup_{isUsedIn} \leftharpoonup_{refersTo} \rightharpoonup_{specifies} \leftharpoonup_{isOperationOf}\ o\ ;$

Fig. 1. Metamodel of the Booch Method (Part from [SE97])

a system have to be unique, and GRAL predicate STD9 demands that an event which is mapped to an operation of a class must belong to the state transition diagram of the respective class[1].

The EER/GRAL description of the notations underlying a method formalize their (context free and context sensitive) syntax. It is proposed to formalize different notations separately, in order to integrate the descriptions into an overall description afterwards. The approach has successfully been applied to the Booch method [SE97] and to OMT [ES97].

2 Semantic Description

The semantics of a notation can be described by an operational semantics based on abstract automata, called *semantic bases*. Such an automaton has *states*, which describe possible configurations, and a *transition relation*, which describes possible transitions from one configuration to another. Furthermore, there is an *initial state* from which the computation starts.

A semantic basis corresponds to some $\mathcal{Z}$-text and some $\mathcal{Z}$-schemata. We propose to describe the semantic basis SB_N of a graphical notation N in four parts:

- some global definitions,
- a schema $Config_N$ to model the states,
- a schema $Init_N$ to model the initial state, and
- a schema $Delta_N$ to model the transition relation.

[1] The predicate STD9 is an example for *regular path expressions* in GRAL. In this case the long predicate states that there exists a path from e to o consisting of four edges with the respective type and direction.

We will give an example by describing a semantics of both parts of our meta-model.

A ClassDiagram is used in Booch's method to describe what objects can be found in a system. The universe of objects is given by the set OBJECT. An object is either existent on its own (like e.g. the integers, strings, etc.) in which case it has a value assigned to it by the partial function $\mathit{objectValue}$, or it may be an instance of a – user-defined – object class given in the diagram.

$$[\mathit{OBJECT}, \mathit{VALUE}]$$

$$\begin{array}{|l}
\mathit{objectValue} : \mathit{OBJECT} \mathrel{\text{⇸}} \mathit{VALUE} \\
\mathit{classOf} : \mathit{OBJECT} \mathrel{\text{⇻}} V_{\mathit{Class}} \\
\hline
\langle \mathit{dom}(\mathit{objectValue}), \mathit{dom}(\mathit{classOf}) \rangle \ \textsf{partition}\ \mathit{OBJECT}
\end{array}$$

Objects of classes may change over time. An object carries the attributes of its class and each attribute has a (changing) value, which is again an object (given by the type of this attribute).

$$\begin{array}{|l}
\hline
\mathit{Config}_{\mathit{CD}} \\
\mathit{objects} : \mathbb{F}\,\mathit{OBJECT} \\
\mathit{objectAssign} : \mathit{OBJECT} \mathrel{\text{⇸}} (V_{\mathit{Attribute}} \mathrel{\text{⇸}} \mathit{OBJECT}) \\
\hline
\mathit{objects} \subseteq \mathit{dom}(\mathit{classOf}) \\
\mathit{objects} = \mathit{dom}(\mathit{objectAssign}) \\
\forall\, o : \mathit{OBJECT};\ c : V_{\mathit{Class}} \mid c = \mathit{classOf}(o) \\
\qquad \bullet\ (\mathit{dom}(\mathit{objectAssign}\ o) = c \leftharpoonup_{\mathit{isAttributeOf}} \\
\qquad\qquad \wedge\ (\forall\, a : V_{\mathit{Attribute}};\ c_1 : V_{\mathit{Class}} \mid c \leftharpoonup_{\mathit{isAttributeOf}} a \leftharpoonup_{\mathit{isTypeOf}} c_1 \\
\qquad\qquad \bullet\ c_1 = \mathit{classOf}((\mathit{objectAssign}\ o)(a))) \\
\hline
\end{array}$$

The initial configuration of a class diagram is an arbitrary instantiation for each class of the diagram.

$$\begin{array}{|l}
\hline
\mathit{Init}_{\mathit{CD}} \\
\mathit{Config}_{\mathit{CD}} \\
\hline
\end{array}$$

Since a class diagram alone has no restrictions on the changes of its instantiations the transition relation $\mathit{Delta}_{\mathit{CD}}$ is reduced to a very simple one.

$$\begin{array}{|l}
\hline
\mathit{Delta}_{\mathit{CD}} \\
\Delta\mathit{Config}_{\mathit{CD}} \\
\hline
\end{array}$$

A StateTransitionDiagram describes the state space of a given class, the events that cause a transition from one state to another, and the actions which are triggered by these events. The Config of such a diagram is the actual state.

$$\begin{array}{|l}
\hline
\mathit{Config}_{\mathit{STD}} \\
s : V_{\mathit{State}} \\
\hline
\end{array}$$

The inital state of the semantic basis corresponds to the start state of the state transition diagram.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Init}_{STD}} \\ \hline
\textit{Config}_{STD} \\ \hline
s.\textit{flag} = \textit{start} \\ \hline
\end{array}$$

A *Delta* is made due to an event which enables the transition and triggers the action.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Delta}_{STD}} \\ \hline
\Delta \textit{Config}_{STD} \\
e? : V_{Event} \\
a! : V_{Action} \\ \hline
\exists\, t : V_{Transition} \bullet s \leftharpoonup_{startsAt} t \rightharpoonup_{endsAt} s' \ \wedge e? \rightharpoonup_{causes} t \leftharpoonup_{isActionOf} a! \\ \hline
\end{array}$$

3 Semantic Integration

At this point, we have only given a syntactic integration. For example, a StateTransitionDiagram always refersTo a ClassSpecification and therefore to a Class. We will now describe the semantic integration of both diagrams. In order to do this we use some *meta operations* which operate on semantic bases and adapt them depending on the extended semantics of the integration.

To integrate the semantics given above we use the idea that each user-defined object in the system gets additionally assigned a state of its corresponding state transition diagram. We put this idea into action by using the meta operation

$$\underline{\textit{addToSchema}} : \textit{Schema} \times \textit{Declaration} \times \textit{Predicate} \rightarrow \textit{Schema}$$

which extends a given schema in its declaration and its predicate part:

$$\begin{array}{l}
\underline{\textit{addToSchema}}(\textit{Config}_{CD}, \textit{objectState} : \textit{OBJECT} \nrightarrow V_{State}, \\
\quad \textit{dom}(\textit{objectState}) = \textit{objects} \\
\quad \wedge\, \forall\, o : \textit{objects};\ s : V_{State} \mid \textit{objectState}(o) = s \\
\qquad \bullet\, s \rightharpoonup_{isUsedIn} \rightharpoonup_{refersTo} \rightharpoonup_{specifies} (\textit{classOf}(o)))
\end{array}$$

Using this integration, the initial configuration specification and the delta specification have to be adapted in order to reflect the fact, that states of the objects shall behave according to the corresponding state transition diagram.

$$\begin{array}{l}
\underline{\textit{addToSchema}}(\textit{Init}_{CD}, , \\
\qquad \forall\, o : \textit{objects};\ s : V_{State} \mid \textit{objectState}(o) = s \bullet \textit{Init}_{STD})
\end{array}$$

$$\begin{array}{l}
\underline{\textit{addToSchema}}(\textit{Delta}_{CD}, , \\
\quad \forall\, o : \textit{objects};\ s, s' : V_{State} \mid \textit{objectState}(o) = s \wedge \textit{objectState}'(o) = s' \\
\qquad \bullet\, (\exists\, e? : V_{Event};\ a! : V_{Action} \bullet \textit{Delta}_{STD}))
\end{array}$$

These three changes lead to a modified version of SB_{CD} which reflects the integrated semantics.

4 Conclusion

These short (and simplified) examples should have shown, that a formal $\mathcal{Z}$-based semantics of object-oriented (oo) methods might be achieved:

1. The formalization of the *abstract syntax* of the notations of oo-methods can be made properly and concisely by using the EER/GRAL approach.
2. These metamodels have several parts, which are then *syntactically integrated* into one model. This is usually done by introducing a relationship or by adding generalizations between two entities of different diagrams.
3. The *semantics* of the individual notations may be described operationally by defining semantic bases.
4. When considering the *semantic integration* these semantics may be combined by meta operations on the semantic bases.

Here still a lot of work has to be done. Up to now about seven different meta operations on semantic bases have been identified [F96] and applied in examples. These operations, when combined, enable us to realize the semantic integration in several cases. We hope that some more experience will help to understand the main approaches for integration using this technique.

It is interesting to note, that syntactic and semantic integration seem to be only loosely coupled. We expected that the choice of a syntactical integration operation in step 2 above would determine the semantic integration operation in step 4. But we did not find such a result. The meta operations used to integrate the semantic bases are only determined by the intention of the considered object-oriented method but not by the syntactic approach.

Acknowledgement
The authors thank Alexander Fronk for his work on the Subject.

References

[CEW94] Carstensen, M. ; Ebert, J. ; Winter, A. ; *Entity-Relationship-Diagramme und Graphklassen.* in german Koblenz: Universität Koblenz-Landau, Report, 1994.

[EWD+96] Ebert, J. ; Winter, A. ; Dahm, P. ; Franzke, A. ; Süttenbach, R. ; *Graph Based Modeling and Implementation with EER/GRAL.* In: B. Thalheim [Ed.]; 15th International Conference on Conceptual Modeling (ER'96), LNCS 1157, p. 163-178. Berlin: Springer, 1996.

[ES97] Ebert, J. ; Süttenbach, R. ; *An OMT Metamodel.* Koblenz: Universität Koblenz-Landau, Fachbericht Informatik 13/97, 1997.

[F97] Franzke, A. ; *GRAL 2.0: A Reference Manual.* Koblenz: Universität Koblenz-Landau, Fachbericht Informatik 3/97, 1997.

[F96] Fronk, A. ; *Software Engineering: Semantik von Entwurfssprachen.* Master's thesis in german. Koblenz: Universität Koblenz-Landau, 1996.

[SE97] Süttenbach, R. ; Ebert, J. ; *A Booch Metamodel.* Koblenz: Universität Koblenz-Landau, Fachbericht Informatik 5/97, 1997.

This article was processed using the LaTeX macro package with LLNCS style