Dynamic Modeling in OOram

Geir Magne Høydalsvik and Jon Harald Holm

University of Trondheim
Norwegian Institute of Technology
Department of Computer Systems and Telematics
UNIT-NTH/IDT, N-7034 Trondheim, NORWAY.
geirmag@idt.unit.no

Abstract

This position paper outlines the research effort regarding dynamic modeling capabilities in OOram (Object-Oriented role modeling and analysis). This research has focused on partial descriptions of objects, using the role abstraction. The behavior of each role has been described by statecharts – which are synthesized to create a wide range of object specifications.

The rest of this paper is organized as follows: Section 2 explain role models as an objects abstraction technique. Section 3 explain the use of Harel statecharts and SDL-92 PDL. Section 4 gives some concluding remarks.

2 OOram

OOram separates between roles (why), types (what), and classes (how). Types and classes denotes the traditional distinction between specification and implementation – found in a wide range of methods. The role concept is somehow unique to OOram. In OOram roles act as building blocks for generic structures that can be configured in various ways to achieve flexible and reusable software.

Roles are organized into role models. Role models can be included in other role models by a set of synthesis operations (similar to the inclusion and refinement operations found in [5]).

The relationship between roles and types is as follows: A type may play different roles in different role models, and a role may be played by many different types of objects. Different role models can be integrated by letting instances of types (objects) play multiple roles, one for each role model. A type can be informally defined as the external observable syntactic and semantic properties of an object.

A role becomes a particular view on an object which makes it possible to model the collabora-

1TASKON AS, Gaustadalleen 21, N-0371 OSLO 3, NORWAY.
tion structure independently of any object that will populate the structure. A basic role diagram is depicted in Figure 1. Two roles called A and B shown as large circles. A path between two roles means that a role may “know about” the other role so that it may send messages to it. A path is always terminated by port symbols at both ends. A port symbol may be a single small circle, a double small circle or nothing. Nothing means that the near role does not know about the far role. A single circle (p) represents that an instance of the near role (A) knows about none or one instance of the far role (B). A double circle (q) represents that an instance of the near role, knows about none, one or more instances of the far role.

Synthesis of role models can be achieved by taking two role models as a staring point, then start merging one or more roles from one model with one or more roles from the other (parents). The characteristics of the new model (child) is a combination of the characteristics of the two originating ones (i.e. signatures, and behavior).

3 Dynamic Modeling

The pre-existence of the OOram method imposed some initial requirements on dynamic extensions. The following was found to be essential:

1. Encapsulation: Each role should have a separate dynamic description.
2. Message passing: Roles should interact by means of messages.
3. Synthesis: Dynamic descriptions should be synthesizable.

Besides these requirements it was clear that some kind of State Transition Diagrams was preferred because of its graphical representation, understandability, and customer familiarity. For similar reasons some kind of Timing/Signal Diagrams was preferred for protocol specifications.

Dynamic modeling in OOram evolved through several stages. First, it was Signal diagrams depicting important scenarios in a role models. The obvious drawback with this approach was the reliance of informal descriptions of message signatures. Later, State Transition Diagram (STD) was added to describe the dynamic behavior of roles (similar to the Booch’s method). However, two major shortcomings was revealed:

1. An explosion of states when two FSM’s was synthesized.

2. FSM’s expressive power was too limited.

The first problem was attacked by using Harel statecharts, the second by introducing a SDL-92’s Process Description Language.

3.1 Statecharts

When two roles are synthesized, it is possible that the corresponding STDs will be independent in the synthesized state diagram. But, because of the synthesis, the independent STDs can be affected by the same messages. To be able to model this, the independent STDs must be combined, causing an explosion of states and transitions.

Statecharts have some well known advantages with respect to STDs, most important Aggregated states, Generalized states, and Abstracted states. By using these features we found that the explosion of states and transitions could be avoided, and manual changes to the resulting diagram were possible.

We only used a subset of the Harel statechart notation. The most important restriction is that another state cannot be used as a condition (i.e. role in state), this because of the encapsulation property of roles. Events and Actions in a statechart were annotated with the port (message→port),
because collaborators are restricted to specific ports.

The dynamics of each role and each role model is described by a Statechart. Synthesis of roles imply a synthesis of the corresponding statecharts. For each role, synthesis of states is done recursively. If states from different diagrams represent the same abstract state, they are merged into one state, having transitions from both diagrams.

Two statecharts are independent if none of the transitions in either of them influence transitions in the other.

1. Synthesis of two independent statecharts results in a new state containing these two statecharts as orthogonal (AND) statecharts.

2. Synthesis of two dependent statecharts is done by adding states and/or connecting states by transitions.

Some alternative combinations are depicted in Figure 2.

Allowed changes after a synthesis operation (conformance rules):

1. Any state from the parent diagram can become a generalization (XOR) or split into orthogonal parts (AND).

2. A transition can be moved between states in a child diagram, and a transition can be removed if the associated events and actions are moved to other places in the child diagram.

3. New states or transitions can be introduced into the child diagram.

4. New states or transitions cannot be introduced into the parent diagram.

The essential of these four rules is that no changes can be made to a diagram that has been used in a synthesis, and that the child diagram must be a specialization of the parent diagram.
3.2 SDL-92 PDL

Statecharts do not specify actions in any detail, only messages sent and received. To be capable of specifying Attributes and Methods, we adopted the SDL-92\(^2\) PDL (Process Description Language). The PDL can shortly be described as graphical symbols depicting sequences, selections, and iterations of state transitions. A transition is a sequence of the following symbols: old-state, input, task, output, new-state. A task may assign variables and call procedures.

The approach taken was to use statecharts during the Role Modeling stage, and to provide for conversion to SDL-92 PDL during the Type Specification stage. In this way we both supported the flexibility needed in the experimental stages of analysis and synthesis, and the added expressive power needed during detailed specification.

Because SDL-92 could not capture all the facilities offered by statecharts directly, we added some features and notation and named it Extended Process Diagrams (EPD). There are also some restrictions on the use of statecharts, most important that the statechart symbol H* cannot be mapped to our EPD. With these modifications it is possible to do a direct mapping from statecharts to EPDs.

4 Conclusion

Simple role models is combined by merging two role models in a controlled manner, adding an extensive dynamic specification would obviously complicate the synthesis process. However, by using the statecharts formalism, we could reach an acceptable solution by doing a semiautomatic synthesis in much the same way as before. When a type specification is created, the synthesis process ends, and we can add more details without worrying about combinations. We thus adopted the SDL-92 PDL as an abstract specification of methods (i.e. state transitions). The choice of SDL-92 was pragmatic – an alternative, more declarative solution is to use Objectcharts [3]. However, we find that some more experiences with the existing solution must be achieved before changing the approach.

References


\(^2\)SDL-92 – Functional Specification and Description Language, CCITT standard, the language include object-oriented features.