Abstract

Unified Modeling Language (UML) provides several diagram types viewing a system from different perspectives. In this research, we exploit logical relationships between different UML models. We propose operations to compare, merge, slice and synthesize UML diagrams based on these relationships. In the formal demonstration we show how statechart diagrams can be synthesized semi-automatically from a set of sequence diagrams using an interactive algorithm called MAS. We also demonstrate how a class diagram, annotated with pseudocode presentations of key operations, can be synthesized from sequence diagrams, and how class diagrams and sequence diagrams can be sliced against each other.

1. Introduction

UML [4] has become an industrial standard for the presentation of various design artifacts in object-oriented software development. Different diagram types of UML can be used to view a system from different perspectives or at different levels of abstraction. Therefore, UML models of the same system share information and are mutually dependent. Changes in one model may imply changes in another, and a large portion of one model may be synthesized on the basis of another.

Automated support for synthesizing one UML model from another can provide significant help for the designer. Such synthesis operations help the designer to keep the models consistent, to speed up the design process, and to decrease the risk of errors. Typically, every iteration phase increases the information present in the design. The new information must be somehow translated and merged to existing models. Usually this is done manually, which is often tedious and can easily lead to mismatches and inconsistencies due to human errors.

In object-oriented analysis and design (OOAD), dynamic modeling aims at the description of the dynamic behavior of objects using statechart diagrams. Examples of interactions can be given as sequence diagrams. Automated support for constructing statechart diagrams from sequence diagrams provides considerable support to the designer. It helps her to quickly shift from constructing example sequence diagrams to tuning the statechart diagrams. The designer should add the information implied by the sequence diagrams also to the static model (class diagrams), or check that the static model conforms to the sequence diagrams. Class diagrams are often annotated with pseudocode descriptions of operation implementations. Such descriptions can be inferred from sequence diagrams as well.

In this demonstration we show examples of synthesizing one UML diagram from another. We also show how set theoretical operations (e.g., union) can be applied for two diagrams of the same type. Our implementation platform, the Nokia TED [8], is a multi-user software development environment that has been implemented at the Nokia Research Center. TED supports most of the UML diagram types and a reasonably large subset of the UML metamodel as a component library, thus allowing us to operate easily directly with the model repository.

2. An interactive approach to synthesize statechart diagrams

Fully automatic statechart diagram synthesis algorithms generalize information given in sequence diagrams, i.e., the resulting statechart machine accepts more paths through the modeled system than those represented as the sequence diagrams [2]. The generalization is often the desired affect. However, in some cases a synthesized statechart machine might also accept unwanted or erroneous paths and thus be "overgeneralized". Applying the synthesis algorithm to an incomplete set of possibly inaccurate sequence diagrams may result in an overgeneralized state machine. The synthesis algorithm presented in this demonstration avoids undesired results caused by overgeneralization. To achieve this goal, the algorithm asks the designer for guidance during the synthesis process, when needed. MAS [3] is an
algorithm that synthesizes UML statecharts diagrams from sequence diagrams. It models the synthesis process as a language inference problem and uses Angluin’s [1] framework of minimally adequate teacher to infer the desired statechart diagram with the help of the user. Being a minimally adequate teacher requires that the designer can answer two kinds of simple questions:

1. she must decide whether a given behavior is possible in the system she is implementing (the membership queries),

2. she must accept or reject the output statechart diagram, and moreover, if she rejects the diagram, a counterexample must be given (the equivalence queries).

The counterexamples can be positive or negative. A positive counterexample defines additional behavior that should also be included in the statechart diagram. A negative counterexample defines behavior that is included in the conjectured statechart diagram, but that should not be allowed.

3. Synthesizing annotated class diagrams

The synthesis of an annotated class diagram consists of three steps: class diagram synthesis, operation description synthesis, and pseudocode synthesis.

A sequence diagram is transformed to a class diagram in terms of the standard UML metamodel [4]. We translate elements of a sequence diagram, such as classifier roles and messages, to corresponding elements of a class diagram, such as classes, associations, and methods. In addition to this fairly straightforward transformation, we define a set of heuristics that suggests more elaborate constructs, like composition, interface hierarchies, and multiplicities, to the user. These suggestions are uncertain by nature, but still plausible [7].

To produce operation descriptions, an automatic state machine synthesis algorithm is used to generate state machines from a set of sequence diagrams for a selected operation call [2]. The algorithm for generating pseudocode takes a state machine, interpreted as a deterministic finite automaton (DFA), as its input and generates, when possible, a pseudocode procedure represented by the particular DFA. Finally, the pseudocode is displayed as a note attached to the operation in the synthesized class diagram.

4. Combining model operations

The above model operations are integral parts of a larger framework of model operations used for checking, merging, slicing and synthesis of UML models [7]. We divide model operations to (1) basic operations that apply set theoretical operations (e.g., union or intersection) for two diagrams of the same type and (2) transformation operations that take a UML diagram as an operand and produce another diagram of another type as its result (e.g., synthesizing a class diagram from a sequence diagram). Combining different model operations and applying them subsequently, various software design goals can be met. For instance, to check that a set of sequence diagrams is legal from the point of view of a static model (i.e., a class diagram), the user can synthesize another class diagram from the sequence diagrams and apply a difference (basic) operation by “subtracting” the synthesized class diagram from the original one. The parts of the synthesized class diagram that are left indicate structures that are not defined in the original static model.

We are currently studying visual scripting mechanisms [5] for tying these model operations together.

Acknowledgements. This work has been financially supported by TEKES, Nokia, Neles Automation, Plenware, Sensor Software Consulting, and Acura Systems.

References