An agile modeling environment for partial models

Pär Emanuelson
Ericsson Softlab AB
Wallenbergs gata 4
SE-583 35 Linköping
Sweden
+46 13 235717
par.emanuelson@softlab.ericsson.se

ABSTRACT
Three major features are proposed for modeling tools to be suitable for use in agile processes. Advantages are described. The current state of UML tools and some of our experiences are described with regard to these features.

Keywords
Modeling, UML, model editor, partial models, incremental development, development process, tool support.

INTRODUCTION
The use of modeling in software design is increasing, since models capture important aspects of a system that programming languages do not. Models may capture for example requirements and architectural aspects and models can be executable as well.

Agile processes that promote early testing and design iteration are getting more and more attention since many organizations have experienced problems with processes that promote heavy specification writing before any executable results are produced.

The modeling environments of today do not support agile processes well and especially not early and incremental testing. With the current environments:

- Testing is prevented until rather large and complete models have been designed.
- The designer's effort is spent on fixing irrelevant inconsistencies instead of thinking about the problem to be solved.
- Time is wasted on recreating the test environment after each change to the model.

We think current modeling tools are unproductive and that modeling should be done with constant feedback, in the form of analysis and testing. This paper will concentrate on the testing aspect. We think that models can and should be tested early in the development and we will present the following features that we think are needed.

- Direct execution of models.
- Execution of partial models.
- Execution is incremental and ongoing in parallel with the development of the model.

Such a modeling environment would support a more productive development process and make stronger tool support possible, for example:

- Development can be driven by (1) adding new modeling elements, and (2) filling in the gaps, which can be automated and be done by asking the tool for inconsistencies and then eliminating these. These phases interchange.
- Development is done in two stages (1) get a model that works (2) make the model execute efficiently. These two stages can be applied both in a macro and micro perspective.

Integration of modeling and XP has been promoted in the "extreme modeling" [1] and "agile modeling" [2] communities. Executable models are promoted in [3] and [4]. The contribution of this paper is specifying certain qualities that the environment for executing models should have to be really productive. This goes beyond basic problems of being executable at all which involves finding modeling constructs that are enough well-specified to be executable. The qualities we propose include the treatment of partial models and incremental execution. We do not know of any modeling environments that offer such functionality. In the programming world, Lisp and Smalltalk environments generally provide this but C++ and Java do not.

THE MODELLING ENVIRONMENT
Direct execution
Direct execution means that the model can be executed in the modeling environment with minimal involvement of interrupting and disturbing events, which would prevent the user from thinking about the model and the expected results of execution.

This means of course that the model contains all the information needed for execution such as action language code for methods. To perform code generation to files that have to be edited before execution is a major disturbance. The model does not have to be interpreted per se, but there should be no visible compilation step that may produce irrelevant errors or require irrelevant information.

Generation of a target language program is a way to make the model execute faster, which should be a separate part of the design process.

Partial models
A model under development is seldom 100% complete and consistent. A modeling tool should accept this and allow execution although there are missing and conflicting parts in the model. This should be the case regardless of the choice of target programming language, which do require varying degree of completeness.

The tool should for example allow models to contain calls to procedures, which have no definition. As long as these procedures are not invoked during the execution there is no problem. If such a procedure is invoked a dynamic error can be issued.

It is not trivial for a user to fix a model such that it does not contain undefined procedures. When defining a procedure the user will define calls to other procedures that are not defined and if the user then tries to define these the same problems will arise again. The user may have to define dozens of procedures before getting down to the bottom.

The example of a missing procedure can be extrapolated to missing classes, methods, attributes, exceptions, constants, libraries, modules etc.

Executing a model with missing types is more complex, but can also be much more rewarding for the user. In short dynamic typing can be used for executing models and static type checking can be used for code generation.

The benefit of allowing execution of partial models is that the designer can concentrate on developing the model step by step and make frequent tests that she is on the right track. She is not disturbed by irrelevant error messages and can defer fixing these inconsistencies until later when she can ask the tool to display these and begin to fix them.

Incremental execution
An execution is started when we start the model editing session and is then going on in parallel with the development of the model. When we make a modification of the model, the tool might have to do the corresponding changes to the execution. For example when we add an attribute to a class, this attribute is added to the instances in the execution as well.

CURRENT UML TOOLS
The current UML 1.4 version is too restricted for executable models. Typically UML tools generate code stubs into files. These files are then hand edited and action code added in the target language. These files can then be compiled and then executed.

Some commercial UML tools have proprietary additions that can be used to make executable models. As far as we know none of these tools allow execution without code generation to a programming language (normally C or C++) and they do not provide any of the three major features described in this paper.

There is an experimental tool, which provides direct execution, similar to what we describe here [3].

Models in the coming UML 2.0 version will be executable. One may expect that some tools will offer direct execution and execution of partial models but probably very few if any will offer incremental execution.

PREVIOUS EXPERIENCES
We have experience from the design and implementation of a development tool for an object oriented, strongly typed, graphic based expert system language, with the three major features described in this article. This was a 30+ man-year effort, which was used by 100+ users. That experience showed that partially defined models can be handled very well and that development can be very interactive and fast even when the language is statically typed.

Some UML tools provide an API towards the model, which can be used to conveniently fetch any information from the model that is needed for execution, this API also provides model updates. By using such an API an interpreter can execute in parallel with the model editor and information can be exchanged between the two worlds.

We have experience from doing a large UML profile for a telecom programming language, where we used such an API and the UML and Rational Rose customization features to make a customized model editing and code generation environment [5].

CONCLUSIONS
We have described a modeling tool, which we think is very different from current tools. We think that our tool supports agile modeling very well, which the current tools do not.

REFERENCES
1. Website, online at http://www.extrememodeling.org/

2. Website, online at http://www.agile-modeling.com/

3. Marko Boger, Toby Baier, Frank Wienberg, Winfried Lamersdorf, Extreme Modeling, in Extreme Programming and Flexible Processes in Software Engineering - XP2000.

4. Ivar Jacobson, “UML all the way down”, talk at UML World 2001.

5. Pär Emanuelson, Tony Olsson, Jerker Wilander: The UP Project Experience, Ericsson Conference on Software Engineering (ECSE99), Copenhagen, Denmark, 1999.