Utilization of Object-Oriented Models

ILKA PHILIPPOW
Faculty for Informatics and Automation
Ilmenau Technical University
D-98684 Ilmenau/Thür. PF 100565 GERMANY

Abstract: Object Technology has established itself as one of the most significant technologies in software engineering. Object Technology supports the management of very large and complex software systems and software reuse. The modeling process is a main process during software engineering. The OMG has meanwhile standardized a notation for object oriented models. Based on the information within models additional activities can be used for improving the productivity of software development processes. In this paper approaches for enhanced utilization of UML models are described. Activities based on these approaches can be integrated into software development processes. One approach deals with possibilities for model checking. Therefore an additional model, the Object-Process-Model for modeling and testing of dynamic aspects is introduced. The second approach supports to improve software reuse techniques based on the automated instantiation of patterns using extended collaborations.

Key-Words: object oriented modeling process, model checking, Object-Process-Model, software reuse, extended collaborations

1 Introduction
Object Technology – that concerns all of object oriented methods for analyzing, modeling, developing and implementing of software systems which are supported by tools and integrated in an object oriented software development process. The modeling process and, finally the overall model, located in the repository are central elements of software development. Based on the information within the object model improved or new methods can be integrated for increasing the productivity of software development processes. In the following two sections the possibilities of the UML are shortly summarized and a frame model of software development that is focused on the modeling process is introduced.

2 Possibilities of the UML
The UML [1] offers a set of object oriented description elements like class, object, use case and others, which are the fundamental building elements of models. During the modeling process it is necessary to follow two kinds of rules:
• UML-rules concerning the construction of diagrams
• Object oriented rules which are important for a later transformation of models into a programming language (e.g. repeated inheritance is not allowed in Java).
The description elements can be used in several diagrams to represent partial models and various views on the target system like static aspects, dynamic aspects and architectural aspects. The sum of all partial models is the overall model, the complete object model. It is located in an engineering database, the repository. The structure of the repository is determined by the UML metamodel. All models are instances of this metamodel. The metamodel is embedded within a four layer modeling frame, given by the UML. At the lowest level there are concrete elements of an information domain. Elements and their relations have to be described by UML models at the next higher level, using the given diagrams. The next level, the UML metamodel defines the structure of the whole model within the repository. Following the metamodel it is possible to develop and use standardized tools. At the highest level the metametamodel defines a language for describing various metamodels.
It is possible to improve and to extend the UML options creating further models within the given metamodel without great effort to adapt supporting tools (e.g. Object-Process-Model, section 4). But it is also possible to extend the metamodel itself using the metamodel (metamodel for extended collaborations, section 5).

3 Model Centred Software Development Process
A high degree of software reuse is a relevant factor for increasing the productivity of software development processes (SDP). There are already a lot of models for software development processes. However, in this section a software development model is introduced that
is oriented closely to the modeling process and model-based reuse. There are two fundamental kinds of software projects: developing a completely new software system or extending or modifying existing software systems. In the first case, the common starting point concerns use cases, in the second case, the existing software architecture has to be taken into consideration. In both cases a high degree of software reuse can only be reached if it is possible to compare and to get together requirements of use cases and reusable elements of software architectures. This means, use case models, use case instances and models of architecture parts have to be in the foreground. In the following, important properties of SDP are listed that have to be taken into consideration for a model (e.g. see [2],[3], [4], [5]):

- SDP are use case and architecture driven.
- SDP are iterative (activities of various phases have to be repeated).
- SDP consist of increments or milestones (progress of projects has to be visible).
- SDP can be evolutionary (realizing of applications either in a survey-like manner as soon as possible or particular narrow sections almost completely).
- SDB should be event oriented (reacting to new requirements at each level of development).

Software Development Processes (Fig. 1) can be represented by a kind of spiral staircase [6],[7]. Each floor corresponds to one iteration. The results of each step can be interpreted as increments. The people involved in the project have different views of a software project depending on their concrete role in a particular project. View-related problems [described in detail in 6] are not discussed in more detail because they are not relevant for the further explanations, excluding the fact that all people involved have to work with different partial models.

The first four steps are use case oriented, the last four architecture oriented. At each step a main activity has to be carried out. The results of activities can be given almost completely by UML diagrams. The final targets of main activities are detailed use case specification and modeling and the determination and modeling of architectural aspects. The useful diagrams are shown in Fig. 1. For the instantiation of use cases and the definition of scenarios the Object-Process-Diagram (section 4) can be used, also.

Activities represent real action instructions within a project. They have to be determined depending on concrete properties of particular projects. They contain pre- and postconditions, people involved, and relevant documents. Each activity of the top level is refined by further activities of lower levels. Due to the different sub-activities, it is sometimes hard to distinguish between analysis and design tasks.

The steps belong to four different processes that are serial or parallel active during the software development. These processes are:

- Requiring (finding and notation of use case types)
- Modelling (static, dynamic, implementation aspects)
- Conversion (notation of model information in programming languages)
- Introduction (configuration, installation, coaching).

Fig. 1 UML Oriented Model for Software Development
Single activities of these processes can be carried out serial or parallel during software development. In early iteration there are more activities for requiring and modeling, in very late iterations activities for conversion and introduction. In all iterations the modeling process and its results are relevant:

- to compare requirements with models and to reuse models,
- to integrate late requirements into models,
- to generate model-based source code,
- to qualify models during conversion and introduction
- to reuse models.

Taken all this into consideration the modeling process is the central process of software development. This UML oriented model for software development has the modeling process itself as its central point.

The proposed process can be adapted by specific activities to support software development using the information in the different models. Such extended or additional activities aim to use information of the overall model for improving the productivity of software development. Usual kinds of model utilization are model based documentation and code generation. Therefore, it is very necessary to create correct models. A concept for checking UML models is introduced in section 4. Another kind of model utilization are model-based concepts for reuse. In section 5 a concept for the automated framework application based on UML models is explained.

4 Checking of UML Models
The UML semantic and syntactical definition is relatively simple and offers high flexibility of use. Especially the last fact may be the reason for the wide acceptance in practice. Presently, the evolution of UML follows different ways. One way is an extended formalization of UML to improve the applicability of automated tools. A critical success factor of this way is the acceptance in practice. In large projects to be realized in practice there is a need for supporting common programming style throughout a project. In our concept we try to support the common developer in minimizing failure during the modeling process. We have developed model checking components for three aspects:

- Consistency check and correspondence with UML and object oriented rules
- Evaluation of partial models with regard to later comprehensibility and reuse
- Simulation of particular dynamic aspects.

For checking consistency, two components are realized:

Firstly, the input check: During the input check it is examined if it is possible and allowed to add a particular element in context of already existing elements. Note that logical consistency can be injured temporary, for example an object does not belong to a defined class yet. For Input check factory patterns that are proposed by Gamma [8], are used. The necessary rules are defined and factory methods are integrated into the interface of a repository. They ensure consistency according to UML rules. Secondly, there is a component for checking consistency when a partial model is considered to be finished. This check concerns not only UML rules but also object-oriented rules of particular programming languages. After checking consistency we get model structures prior to implementation in a programming language. Two further components have been developed for evaluation of diagrams. Both components aims to assure understandability of models for a later reuse of models. The first component evaluates the graphical comprehensibility of diagrams. This is done by a fuzzy inference mechanism which is adapted to all different UML diagrams. The second component evaluates properties of model elements like the deeps of class hierarchies or the number of attributes. For this purpose we have adapted well known source code related metrics [9] to object oriented models. The components for checking consistency and evaluation are described more in detail in [10], [11].

In this paper the simulation of dynamical aspects is described. The simulation is focused on the interaction behaviour of objects. During the execution time, several processes can be activated depending on the state of the own object or the state of other objects involved. Failure can occur if a modeled object state can never be reached or if there are conflict situations. Therefore it is important to test the modeled interaction behaviour of objects by model-based simulation. To check the behaviour it is necessary to describe pre- and post conditions for the execution of methods and to simulate the execution of methods and to observe the pre- and post conditions. There is no fitting UML diagram for this purpose. Therefore, a diagram for an additional model, the Object-Process-Model (OPM) [10] has been developed. It enables to simulate the dynamic behaviour at model level and to test behaviour accordance between model and model-based implementation.
The elements of the OPM are shown in Fig. 2. Elements of the OPM are processes and object references, belonging to particular classes. Objects possess attributes, the values of attributes represent a concrete state that can be used to formulate different pre- and post conditions. The conditions have to be written at the edges. Attributes are assigned to condition types. There are four condition types. For example, an attribute of the enum type can contain one value from enumeration list <EL>. Conditions can be formulated as operations using type related operators. It is possible to refine objects and processes. So we get hierarchical structures. Fig. 3 shows an example of an OPM. The pre- and post-conditions are used to perform an automated simulation transformation from Object-Process-Models into Petri-Nets. Simulation and analysis are performed on Petri-Net level, but the user of the tool system gets the results at the level of the Object-Process-Model. If modifications of the OPM are necessary, the simulation will have to be repeated. After simulation test and model-based implementation of the software, a testing code can be generated by using the information within the OPM. The testing code can be integrated into the source code and executed. If necessary, the source code or the model should be modified to reach a balance between them. After balancing of model and source code, the test code can be removed.

**Data Types for Conditions:**
- **Int** attribute can contain one integer value
- **Enum<EL>** attribute can contain one value from enumeration list <EL>
- **List<EL>** attribute can contain one or more different values from <EL>
- **Set<EL>** attribute can contain one or more different and equal values from <EL>

**condition:** attribute related operation
The described concept of UML model checking is implemented in the commercial tool otw 2.4. [11], [12].

5. Model Based Application of Frameworks
Reuse makes it possible to reduce the time of development, and to improve the quality by using software which has already been tested. It is possible to reuse source code in the form of modules, functions, classes or components or results of former software developments like models or patterns. Frameworks allow to reuse both, code and models or patterns. A framework [14] is a reusable generic unit for a problem domain. It contains the knowledge of how a problem can be solved in the particular domain. Solutions in a domain have a lot of identical parts, which are represented by frozen spots of frameworks. You can create different applications by modifying the so called hot spots of a framework. Hot spots are predefined places for adding a specific functionality. Object oriented frameworks determine the basic class structure and object interaction model. However, in practice there are a lot of problems. The main problems are missing know-how and long training time, caused by poor or not existent description and documentation. To support framework development and application you need a framework oriented method which not only points out a way to develop a framework but also includes a unified description of framework development, application and documentation. This description will be used to provide guides and tools for framework application.

Based on framework specification and definition of relevant hot spots, a framework has to be described by packages and collaborations [13]. Packages represent the architecture components of a framework. Interactions within one component or between components are described by collaborations. Collaborations are used to describe hot spots and how to handle the framework application. The adaptation of hot spots for a special purpose can be done by means of recipes. Each recipe consists of a number of application steps and each step can actively be supported by tools. To give active support during framework application it is necessary to find a way for an automated instantiation of modified hot spots. UML-Collaborations are useful for describing design patterns. In collaborations, model elements like class, attribute or component can be defined as parameters. For an automated hot spot instantiation it is necessary to include parameters for objects and methods, too. Parameters mark modification points for adapting frameworks. Recipes determine how parameters can be adapted. During framework application the application developer chooses fitting recipes. The instantiation of extended collaboration is automatically done based on the recipe. The framework user does not need detailed knowledge of the function and internal structure of the framework.

Fig. 4 shows the metaclass model of extended collaboration. The grey classes are common UML classes. The package contains all model elements of framework. It refers to all relevant diagrams. Collaborations describe structure and interaction of hot spot elements. The white classes are extended metaclasses. They contain all information that is necessary for tool supported automated instantiation of collaboration.

Parameters refer to changeable elements. They serve as placeholder for concrete elements. Symbolic parameters refer to elements for which only a modification of name is allowed. During automated instantiation of the collaboration parameters of chosen recipes are replaced by concrete values. It is in this way that adapted models of application are created. For this
purpose, an algorithm was developed and integrated in a tool. In detail, the method for framework development, documentation and examples for extended collaborations are described in [14]. The concept of extended collaboration for automated pattern instantiation is implemented in the commercial tool ost 2.4. [12]. Framework development is a very complex process. Typically this is an architecture oriented, incremental, iterative and use-case-driven process. The developers must not only have a good knowledge about software techniques but also knowledge of the problem domain. They have to develop a useful framework metamodel and application recipes. A successful framework application depends on the quality of the application recipes and tool support. The important tasks of the developer are the instantiation of architecture and design patterns and idioms, and for further application the definition of correct hot and frozen spots and useful application recipes. This requires enough knowledge about the particular problem domain. That means domain specialist and domain engineering techniques should be integrated into the development process [15].

6 Conclusion
The modeling process has to be considered as central process of software engineering. The overall model is a fundamental basis for additional methods and tool support. In this paper, an UML oriented model for software development that is oriented closely to the modeling process and model based reuse was introduced. Additional concepts and methods can be integrated as sub activities to increase the productivity of software development, e.g. the proposed model checking concept and the concept for automated instantiation of frameworks based on extended collaborations. Frameworks can be integrated within software product lines. For developing product lines a very important problem is how to find the domain relevant common and variable aspects, and their traceability. To solve this problem it is necessary to include methods of domain engineering into the software development process. Presently we are working on finding an approach to connect description methods for domain engineering with object oriented models. This work is part of a research project with Siemens Corporation, Munich.

References:
[8] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns – Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995