Concept-Driven Engineering for Supporting Different Views of Models

Peggy Schmidt
Christian-Albrechts-University Kiel, Department of Computer Science
pesc@is.informatik.uni-kiel.de

Abstract: This paper investigates the the development and evolution of concepts and the management of transformers, which adds semantics to the concepts. We illustrate how concepts, their variants and transformers can be developed via cooperation.

In order to meet increasing user demands for more various software systems, we are revising the software developing process to accommodate mass customisation based upon Concept-Driven Engineering (CDE). CDE is a strategy to application specification and generation of new concepts via transformers. The concept and its transformation rather than the implementation is central to the development process. It allows automation for specification from early stages to executable specification and code generation. CDE continue to be a challenges in building complex software systems that have several variations and options. Software development is based upon a lot of specifications and implementations such as feature models, UML models and code, which are in different formats but share a certain amount of information. CDE is similar to the ideas of Model Driven Engineering (MDE) and Software Product Line Engineering, whereby we use the term concept instead model. A concept can be a model or a specification and is defined on a concept structure and a set of transformers. Concepts are assigned to a set of transformers, which generates new concepts. In this sense also a software system is a concept. One main remark is the management of the evolution of concepts and its transformers. Concept-Driven Engineering supports that requirements of the software do not remain constant during its development time and therefore the specification has to evolved and refined in order to fulfil the new requirements. One remark should lie an the management of the transformers that can be effectively be used on specifications. Against MDE CDE focus on the transformers which carry out the semantics of the specification, resp. the model. It is needful to study how transformers will affect the development process, that means how easy is a transformer to use, resp. to reuse. CDE abilities rely on the detailed transformer designer's knowledge of concept structure and development work flow while considering software system knowledge, software engineering techniques and methods. The aim of CDE is to avoid the development concepts and transformers which are in downstream development incapable to use.

When developers change one concept simultaneously, we need to propagate these changes across all concepts to guarantee them consistent. The process of synchronization propagates changes among specifications in different stages to all involved participants. Exchange of models between local platforms is still a challenging issue. The exchange and

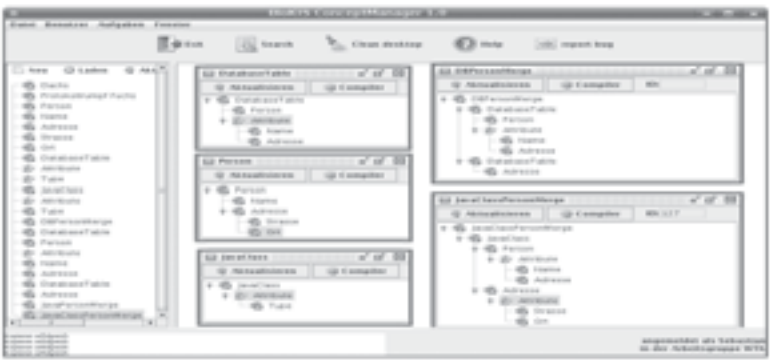

Abbildung 1: ConceptManger for Supporting Concept-Driven Engineering

the synchronization of different local copies between local development systems is so far a tough problem. Tools like subversion support the developers by parallel development, global revisioning, create working copy. The main problem is how to work in parallel and obviate one from overwriting the work of another. Tools like subversion use therefore mechanisms like copy-modify-merge or lock-modify-unlock. We have been developing a tool for collaborative management of concepts, called ConceptManager. The ConceptManager is based upon concept structures. Developers can create concepts. After the activation of a concept all other developers with access rights on this concept can use a concept. When developer works together on a concept we transmit only these parts which were modified and where the other developers are involved. To supports this feature we assign each development task of one developer to a component. Overlayed parts of different developers are connected via ports. Thus a component consists only of a partial copy of the developing software system. This problem faces how to reflect the changes that have occurred in one concept to involved parts of the other developers and in the transformed concepts . An other important feature is that we may have a set of possible modifications on concepts (think on different realizations). The ConceptManger will support the negotiation process between developers and is based upon the ideas of a earlier merge of overlayed concept parts. Additionally we have to pay attention that a transformation can be run on concepts, provided the concepts have compatible concept definitions.

Figure 1 represents a piece of our tool ConceptManager. It shows the concept Person, which consists of the concepts Name and Adress. Later we want to build a database and a java class for this concept. But a database table and a Java class itself are concepts. To get a database and a Java class we have to merge to concepts together (DBPersonMerge and JavaClassPersonMerge). For a given concepts we may have a set of transformers, e.g. to get a database definition script we need a transformer what generates it. Only now the transformer adds semantics to a concept. Other useful feature are the reuse of concepts through the derivation possibility of new concepts from older ones, the search engine, and the definition of constraints on concept structures.