Merge of models: an XMI approach

Author: Antonio Martini

Supervisor: Lars Bendix

Parallel working of several developers gives many advantages in a software development process, but it causes also problems: among them, there is double maintenance. To avoid this problem, developers often have to integrate their works with the latest version to be able to release their own version which includes the previous changes as well. This work is called merging process: the developer mainly has to find changes among his own version, the last version on the repository and, in case, the common ancestor. Often, his changes conflict with those added by others, so these conflicts have to be resolved. This task (the merging process) is quite important and hard to be done, so it should be carried out frequently and carefully: consequently, it requires a set of tools to be well performed.

In the code centric development, we find a lot of good text-based tools which help managing the merge task. Lately, industries are increasing the use of Model Driven Development, creating models to auto-generate code. Nevertheless, the environment is not yet mature enough to support adequately the parallel work of the developers. Unfortunately, moving from a code centric development strategy to a model centric one showed that former textual-based merge tools do not work appropriately with models. In fact, models are serialized using the standard XMI: a language which creates documents containing structured data. Therefore, the comparison of text lines is not the best choice anymore, as a little change at the syntax and semantic level could correspond to several changes on the text level. Consequently, we need a more sophisticated solution in order to find, compare and resolve conflicts between model files, changing for example the granularity of the unit of comparison from the text line to the node of a tree. Model merge tools are not precise enough either, since they have some problems such as detecting too many false positives and false negatives, or not merging considering the smallest possible element, but just raising a conflict if the same top level object is modified (too coarse granularity of unit of comparison). Moreover, they are all oriented towards interactivity, which means that the developer has to follow the entire merge process, conflict by conflict. Furthermore they have to choose "on the fly" among (probably) wrong alternatives provided, instead of looking for the connections between them, creating an "ad hoc" solution.

The aim of this thesis is to investigate the feasibility of a merge process for models using only the XMI serialization. We take three XMI files representing three models (the common ancestor and the two changed versions) and we provide a new file representing a merged XMI.

The study of the XMI language has highlighted some problems, first of all, the fact that the same model could be represented by different XMI productions (different patterns of serialization and different versions), which means that we cannot compare a set of any XMI files. Furthermore, some patterns contain more model semantic information than others, or it is differently represented which means that we can make assumptions when dealing with a certain pattern, and these are not valid when dealing with another one. These considerations forced us to define some restrictions which have to to be satisfied in order to consider this work valid.

We proposed a 5-step merge process which takes as input three XMI files and provides as output a new XMI valid file that represents the merged XMI tree. Such a process makes a diff of the files relying on unique identifiers (avoiding the expensive job of matching).

The proposed algorithm, once all changes are obtained, finds conflicts among them. The algorithm works deeply, which means that two changes are in conflict only if they involve the same smallest structural element (fine-grain unit of comparison) and widely, which means that it should find every direct conflict. The algorithm warns also about syntax violations and distance-1 possible non-direct conflicts; it could be extended in a way to be able to warn also distance-n non-direct conflict. The algorithm merges non-conflict changes, which means that if two changes are not in direct conflict, they are applied correctly with respect with XMI syntax.

The output is represented as a file in which we can find trace about all changes (also those in conflict), so the algorithm runs in batch-mode. In fact, it reports about all ordinary changes, unsolvable conflicts, syntax violations and possible problems. To handle such reports, three approaches are used: annotations to label a changed element, alternatives which show all possible solutions for a conflict, and warnings that report about violations or context-related problem. Since such mechanisms are not supported by the XMI language, we propose the use of the extension XMI tag and XMI comments. We also provided a specification which could be verified and extended by further research.

Discussing outputs of the algorithm (which should be verified on a wider set of examples), we can observe how this XMI approach suffers from the lack of semantic information, which leads to a lack of warranty about correct model semantic output.

Finally, the algorithm at present is not completely environment-independent, since it needs to analyze a set of XMI files which are necessarily serialized using the same pattern, and it works better on a specific pattern. However, once the XMI language had found homogeneity in the specification and in the implementation performed by tool vendors, we showed how we can provide, without any further information (other than the three provided files), a preliminary XMI valid merge which includes all information about changes, conflicts and some model-domain problems which could be elaborated subsequently by the developer or by further tools.