On the Correctness of Model Transformations

Gabor Karsai
Vanderbilt University/ISIS

Flight-critical aviation software today is often being developed using ‘model-based’ techniques. The industry-standard tools for such development are widely available, and they typically include a visual modeling language (supporting dataflow-style and statechart-style modeling paradigms), a simulation engine, and a code-generator that produces embedded code from the diagrams. The modeling languages used are practical, but their semantics is often not defined precisely, to the level of detail used in case of more traditional languages, like Ada. Concurrency issues often complicate language specifications. System verification in practice means simulation using the engine; an activity which cannot be exhaustive, by definition. In spite of these shortcomings, the tools are popular and allow the rapid construction of flight control software.

However, it is unclear how such model-based approaches lend themselves to the certification processes mandatory in aerospace applications. Models offer higher-levels of abstraction for specifying and designing systems, and —if their semantics is well-defined— can be subjected to rigorous formal, often automated, verification. There are many existing effort to couple formal (algorithmic) verification tools to modeling languages. However, the verification of the models does not necessarily imply the verification of the code that is generated from the models. Furthermore, if the design and the analysis modeling languages are different, a generator is needed to bridge this gap, and the generator’s correctness becomes an issue.

Note that existing practical approaches like code reviews and code-level verification seems to fare better than model-based approaches, as the subject of the verification is the final artifact, not an abstraction of it. Additionally, automatically generated code could be hard to understand and analyze in a manual process.

Hence, one crucial ingredient in model-based development of aviation software is the correctness of toolchains that tie modeling, verification, and code generation together. The productivity gains of model-based development cannot be reached unless the toolchain used provides some sort of guarantees that the code satisfies the stringent requirements. Note that these toolchains often include non-trivial transformation/translation steps, including the code generator and model translators that connect the design language used (e.g. Statechart) to the analysis ‘language’ (e.g. SMV). Thus, the correctness of such ‘model transformations’ is an essential question to answer. Otherwise, verification results obtained for the models cannot be carried over to the code generated from the models.

Verifying model transformations, in general, is as difficult as verifying a compiler for a high-level language. However, the domain-specific (read: restricted) nature of the modeling languages, and the fact that model transformations can be formally described using high-level constructs (e.g. graph transformation rules) provides an opportunity for establishing the

correctness of transformations with reasonable effort. The key idea is that the correctness may not be necessary in the most general sense (i.e. for all properties and for all possible models to be transformed), rather, it is sufficient to establish it for specific cases (i.e. for specific properties and for specific models). Thus, arguably, for each instance of the model transformation a certificate can be automatically generated that can be used in the certification process.

Some early results from research on model-to-model transformations indicate that the approach is feasible. As the primary point of interest in behavior preservation across model transformations, we have focused on this issue by establishing rules for a bisimulation relation between the input and the target models. This relation can be explicitly constructed during the transformation, and can be checked with simple linear algorithms once the transformation is complete. However, the approach needs to be extended to other properties (that need to be preserved by the transformations), as well as automatic code generation; which are the subjects of further research.