Tool and Model Interoperability: Now!

Janne Luoma, Juha-Pekka Tolvanen
MetaCase
{janne | jpt}@metacase.com

The systems and software development industry has seen several attempts at tool integration. Although the worst days of data silos are behind us, much work remains. For example, despite 10 years of development of XMI, UML users still can't reliably exchange models between tools. There are no quick fixes: creating one tool is a challenge, making integration possible between unrelated tools is even harder. Previous attempts have also suffered from tool vendors seeking competitive advantage: those with the largest market share may not even want true interoperability. The challenge has never been completely a technical one.

To achieve interoperability today three things must happen. First, tools must offer ways for companies to define and extend the information stored by models, to improve compatibility with other tools and maintain links to them. Second, tools must enable integration based on existing standards, and allow users to build their own integration with both file-based and programmable interfaces. Finally, companies should select and apply only tools that provide those features. In that respect, all tool users can influence the state of the art.

A possible solution has appeared on the tooling side, and is seeing increasing popularity and adoption: Language Workbenches. These tools allow their users to customize how the models are stored, created, edited, checked and they also offer various interfaces. For example, almost all Language Workbenches allow organizations to define their own generators to read the models and output code and also other artifacts, such as file-based integration formats. Thus, users of these tools can never suffer from vendor lock-in, as their users can always take their data and save it to another format.

Language Workbenches differ in their support for integration. The simplest level is support for integration within the Language Workbench – between models, between different languages, and between users – yet even there some tools have problems. The level of true inter-tool integration can be even worse, and users should check to see what facilities there are for calling the tool from the command line and providing a file-based integration format as an argument, and for calling the functions of the running tool from an external tool. To achieve a wide range of interoperability the interfaces should enable distribution and support at least the major platforms and programming languages.

The models in the Language Workbench from MetaCase, MetaEdit+, can be accessed from both Eclipse and Visual Studio at the same time, and the same models can be used for generating code for different targets, e.g. C, Java/Android and C#/Windows Phone. The resulting code can be traced back to models for debugging, update and animation, using an open interface to the tool's functions. These integrations are freely available and extendable. With the same objective, a file-based XML approach is provided covering not only the data in models but also their representation and language definitions (metamodels).

Often the life-cycle starts with less formal data like requirements (e.g. in Excel) or with existing code (e.g. basic data types and definitions in plain files). Interoperability of tools can leverage these by importing the existing assets. Models in design tools can also be integrated with analysis, testing, and simulation tooling. For example, MetaEdit+ has been integrated with tools like Simulink, Labview, Uppaal, SPIN, Absolut, T-VEC and Conformiq. This integration is proven on an industrial scale, with large models (hundreds of thousands of model elements) and hundreds of developers. Integration does not need to be static, but can even be used at runtime to annotate models with performance and execution data.

By providing open access to model data and tool functions, tools can take part in a vendor, language and platform neutral environment, where users are free to pick the best tool for each job. That access forms a solid basis for the integration standards of today, and for the development of new standards like OSLC tomorrow.