MetaCase

Nokia Siemens Networks Case Study

"Domain-Specific Modeling solution makes development **significantly faster and easier** than the old manual coding practices," Jari Lehto, Nokia Siemens Networks

Every architect wants developers to follow the architecture's rules while creating and maintaining applications. If that does not happen, the planned architecture erodes and making changes become costly and difficult. This becomes an even bigger challenge when the development team grows or is located on different sites.

While guidelines, manuals and training help, they unfortunately cannot guarantee that the architectural guidelines and rules are really followed. The amount of reviews and code style checks can only be increased so far before they become counterproductive. Most importantly, these checks only occur after the code has already been written.

Nokia Siemens Networks

THE SOLUTION

Good engineering practice tells us that the earlier the architectural rules are followed, the better. Ideally this happens right at the design stage, and is maintained throughout the later stages. This ideal can be achieved by codifying the architectural rules into the very language with which applications are designed and built.

Since the architects already know the rules, they are the best candidates to define such a domain-specific modeling language. The concepts and rules in the language guide the work of the developers, help them create good applications quickly, ensuring the architecture is followed, and avoid creating things that would not work or would be inefficient. The code generators, also defined by the architects, ensure that this quality is maintained all the way through to the finished product.

At Nokia Siemens Networks architects have defined a domain-specific language in this way, encoding the architectural rules and constraints of a particular telecommunication platform into the modeling language and related generators.

The modeling language was built with MetaEdit+, automatically giving NSN a top-class editing environment for the models. Importantly for NSN, MetaEdit+ also provides the functionality needed to work effectively with models on an industrial scale, such as reusing models, refactoring and replacing model elements, organizing and handling large models, and multi-user access.

RESULTS

Having architectural rules built in to the language and generators gives several benefits:

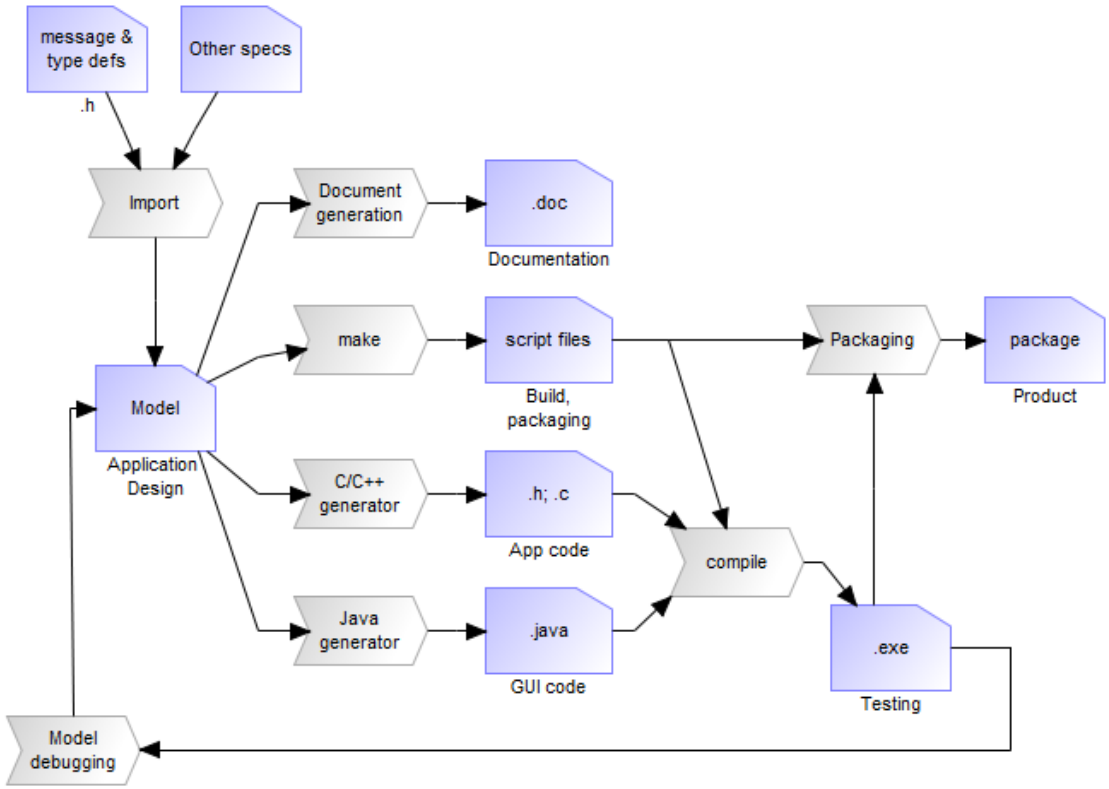
- Developers follow the architecture rules and constraints. This is particularly valuable in large organizations and long term projects, coping well with organizational changes.
- Changes in the architecture are easier to introduce as architects can change the rules by changing the modeling language and generators. This uses the unique ability of MetaEdit+ to update existing applications automatically to the new language (and architecture). Developers can also get reports on places where the application specifications could be changed to take better advantage of the new architecture.
- Productivity increases because the code generators automate a large portion of the development work.
- Changes to operating systems, libraries and frameworks are easier to introduce because models are made directly with architecture and domain concepts rather than plain code concepts. When something in the implementation platform changes, the models remain the same, and the entire change can be made by one person in the generators.
- The model integration facilities of MetaEdit+ support true reuse, not just string references, reducing the need for diff & merge.
- In addition to code, generators for documentation, metrics etc. keep all the deliverables up-to-date and consistent with the single source, models.

THE BIGGER PROCESS

Application design and application implementation are parts of the wider development process: starting from requirements and integrating with legacy code, right through to testing and product packaging. For this purpose the DSM solution for was extended to cover several other phases of the development process as illustrated below.

First, the import of basic data and message definitions was implemented using a dedicated parser that reads the legacy files and imports them to a library in the modeling tool. The parser was implemented using MetaEdit+ Reporting Language, the same language used to build generators.

With the definitions imported, application developers could refer to legacy data directly while modeling: imported data would thus behave like any other model element. In addition to generating the C and Java code (for the GUI) a number of other generators were also provided, such as document generation and model checking. Generators were also defined to produce packaging and make files. It was also beneficial that generators can be used for model-level debugging and testing: application designs are then animated along the program execution.



CONCLUSION

Compared to the earlier manual coding practices, the DSM solution makes development significantly faster and easier. Training needs are also reduced, as application developers don't need to master the architecture details and underlying framework. Equally importantly, the architectural rules codified in the modeling language and code generators improve the quality of the applications, better guaranteeing that the architecture and coding rules are actually followed. These are significant improvements, especially when considering the amount of resources needed to implement the automation: one man-week.

YOUR NEXT STEP

Visit us at http://www.metacase.com to see how MetaEdit+ can speed up your software development!

MetaCase
info@metacase.com
www.metacase.com