A Proposal for an Open Source Project to develop a Model-Driven Architecture Tool

A good UML analysis model – a Platform Independent Model (PIM) in MDA terminology – describes an application to be developed in *business terms*. It should not be visible in such a model whether the implementation will be in Java or C# and whether the environment will be .NET or J2EE.

Mainstream UML Tools

One way to transform a PIM into a PSM is to apply a set of rules to the originally platform independent model that transforms it into a platform specific model. Take the example, of a UML model that needs to be transformed into a J2EE specific model. One of the rules may be something like:

> Every UML class in the PIM that needs to be implemented as a Stateless Session Bean will be transformed into three classes in the PSM, implementing respectively the SessionBean, EJBHome and EJBObject interfaces.

The resulting J2EE-specific model may need to be modified by the developer and can then be used as a basis for code generation. The generated code may again need adaptation to platform specific or application specific requirements.

The approach sketched here (or a similar one) is used by most tools that provide J2EE development *wizards* or tools that start to provide MDA compliant features. There are however several major disadvantages:

- The size of a PSM can easily be three times the size of a PIM or more.
- If the PSM is modified the consistency between the PIM and the PSM may be lost.
- The architecture of the PSM will be difficult to identify.

Let us look at a very simple example:

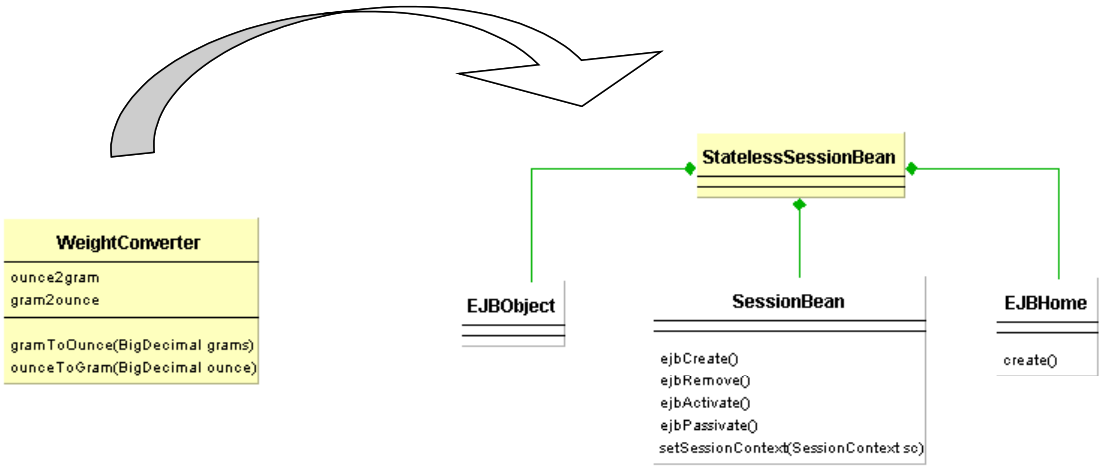

Figure 1 – Mapping a PIM onto a PSM

Figure 1 shows a PIM consisting of a single class that is mapped to the EJB architecture as a *SessionBean.* The PSM would show a *WeightConverter,* a *WeightConverterBean,* and a

WeightConverterHome class. Other artifacts, such as XML to define the configuration would be needed as well. A complete implementation of this PIM would require more PSMs, to provide for the user interface for example. Even for this simple case, the resulting UML for a PSM will become quite complex. Imagine the scenario of a PIM with 100 classes (not very large): the actual EJB architecture would be scattered across a 300 class PSM, and any subsequent change to the architecture would require extensive manual changes or at least repetitive manual re-invocation of reconfigured generation wizards.

Generic PIMs and Generic PSMs as Architecture Models

The technical architecture used to implement a particular model could be specified independently of a business model. To avoid the problem of information redundancy, PSMs need to be capable of expressing the reoccurring design patterns implied by a given architecture. To differentiate between the loose specification of traditional software design patterns and the precise specification of patterns required for automation we define a **texture** *as a pattern specification in an unambiguous UML-based format*. A suitable MDA tool would rely on user-definable, generic PSMs that capture the textures for various architectures. It could be equipped with pre-configured – but customizable – textures for popular architectures: EJB, Swing, .NET etc.

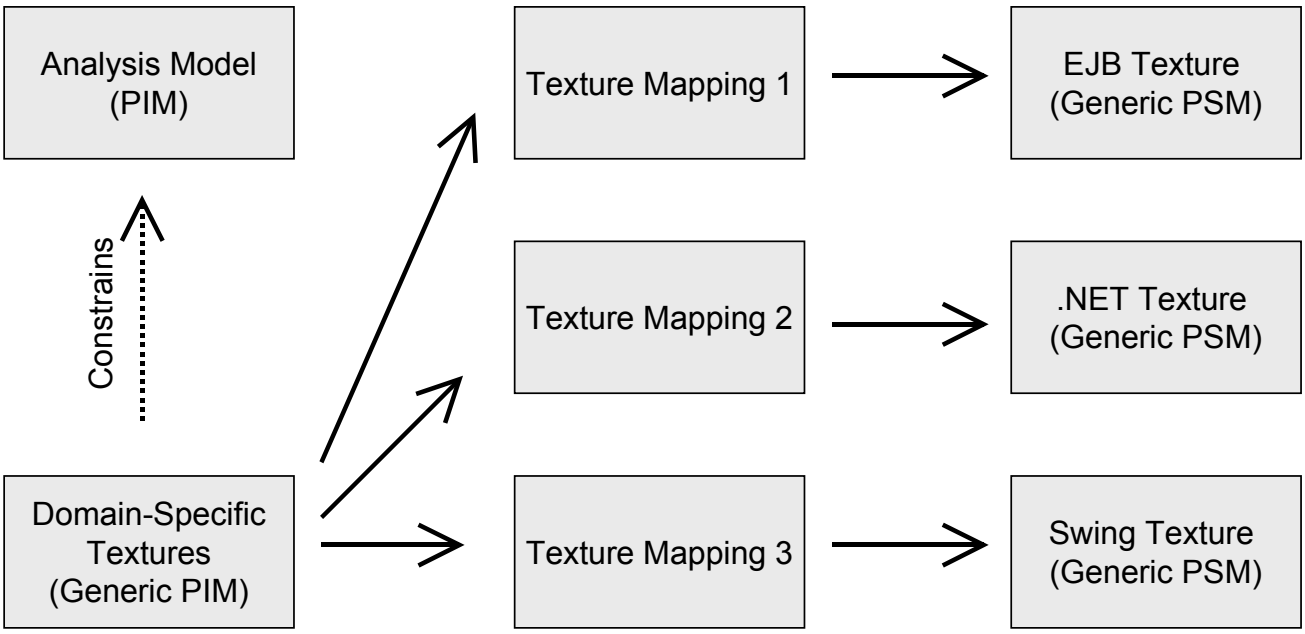

Figure 2 – Texture Mappings

Generic PSMs would have the advantage of being **domain independent**. What we are still missing is the binding of a given PIM (typically in a domain-specific notation) to a domain-independent, generic PSM. This requires the application of the concept of textures to PIMs. Besides the specific PIM that contains the domain-specific analysis model, a generic PIM is required to define the allowable textures in specific PIMs.

One way of looking at textures is as a set of constraints that have been added to the meta-model. The binding between PIMs and PSMs can then be expressed as **texture mappings** [1] – *mappings between textures in a generic PIM and textures in a generic PSM*. Texture mappings provide all the information required to automate the implementation of a given PIM in a given architecture as defined in a generic PSM.

Using this setup, the same PIM could be implemented using different architectures or programming languages. It would mean that we can develop analysis models and architecture models independently. The act of developing an application for a specific platform would mean that the developer has to define a texture mapping.

Texture mappings allow us to keep a level of indirection between architecture model and analysis model. They are used to map each construct in a generic PIM to corresponding constructs in a generic PSM. We should be able to use our architecture for other analysis models, and we could decide to implement our analysis model using some other architecture.

To return to our example, there will never be a *WeightConverterHome* UML class in a PSM, just a binding in the texture mapping that says that for certain stereotypes of classes in the PIM, the SessionBean texture should be applied. The code generation templates associated with the generic PSM will then provide a *WeightConverterHome.java* class.

Building an MDA Tool

We propose to develop a tool providing UML modeling facilities [including meta-modelling facilities as envisaged in UML 2.0] and a good template based code generation engine together with a suitable implementation of Textures. Such a tool would provide a developer with significant advantages over current tools:

- The ability to define domain-specific notations for PIMs
- The ability to develop PIMs independently of architecture models
- In the true MDA spirit, both investments in domain modeling and platform dependent expertise could be harvested and maintained.
- A developer would be able to develop special purpose architecture models, including associated code generation templates quickly and easily.
- Textures and texture mappings would be quick to change, allowing agile development methods to be applied.
- Customization for specific projects or for company standards would be easy.

There is sufficient experience with template based generation of code, see [2] and [3], that can be the basis for developing the needed flexibility and customizability of such a tool.

The implementation of *textures* and *texture mappings* in a tool still needs invention and creativity. This will be a worthy challenge and could result in a true step forward for MDA-based development tools.

References

[1] This is a proposal made by Jörn Bettin in an unpublished communication and constitutes an evolution of earlier work that is sketched in http://www.isis.vanderbilt.edu/oopsla2k1/Papers/Bettin.pdf.

[2] Ghica van Emde Boas, Architecture Based Code Generation from UML Models, Workshop on Generative Programming, OOPSLA 2001.

[3] http://www.research.ibm.com/journal/sj/392/vanemdeboas.html

Ghica van Emde Boas
Bronstee.com Software & Services
emdeboas@bronstee.com

Jörn Bettin
SoftMetaWare
jorn.bettin@softmetaware.com