# Interview on codegeneration.net

*Markus Völter, voelter@acm.org, www.voelter.de*

**This is an interview I gave to Mark Dalgarno of codegeneration.net. I talk about model-driven software development and openArchitectureWare specifically.**

### openArchitectureWare is described as a "tool for building MDSD/MDA tools". Can you say a bit more about this?

Many MDA tools support a certain modelling paradigm (DSL) and/or a defined target platform. While these tools are useful for that particular combination, "real" MDA/MDSD tools let you define your own DSL and let you generate all kinds of textual output. The reason why we say this so explicitly is because our target user group is not primarily the application developer who uses a generator to speed up application development; rather, we address the needs of who we call the domain architect, i.e. the person who builds the code generator. So, as of now, oAW is distributed without any domain-specific transformations, but with a wealth of tools – such as Eclipse plugins for writing code generation templates – for *building* your own DSLs, metamodels, transformations and code generation templates. Thus, oAW is especially interesting for you if you cannot use preexisting generators (or generator cartridges), because you work off the mainstream of J2EE enterprise development.

### What's the history behind the project? How did you get involved?

The project has originally been started by the b+m Informatik in Kiel, Germany. They had a commercial offering called the *b+m GeneratorFramework* which was part of a product line called *ArchitectureWare*. At some point – as a consequence of some pushing of the Open Source community, and supported by b+m's Chief Architect Tom Stahl – they decided to release the product under the LGPL license at SourceForge. The tool was called *open*ArchitectureWare to distinguish its Open Source nature. At that time oAW was basically in version 2.0.

I had been working with Tom Stahl on a project and learned a little bit about (the then still commercial) *GeneratorFramework*. I played around with it a little bit and, in late 2003, I could luckily convince a customer in the US to pay me for learning the tool ☺. I then became one of the first non-b+m committers for the project and started adding features, mainly useful utilities. At the beginning of 2004 I got involved in a project at BMW Car IT where we built a proof of concept of what was going to be the AUTOSAR standard, a distributed component middleware for automotive systems. I was brought in to help with the aspects of modelling and code generation, as well as component middleware. During the 14 months that I stayed in this really fun project, we considerably extended the oAW tool suite to what was going to be version 3. During this phase we created a wealth of utilities, the metamodel generator and also the GEF Editor Generator (something alike

GMF, now discontinued because of GMF). During this phase we also improved the documentation a bit, and the exampleWorkspace was made available.

In February 2005, the group of oAW developers (you can meet the team at http://openarchitectureware.org/staticpages/index.php/team) got together to discuss the future of the project. We came up with really big plans of what version 4 would look like – and then, nothing much happened for a long time. In the last 4 months of 2005 the project took up pace, mainly due to Sven Efftinge, who rewrote the whole thing from scratch. A new version of the Xpand template language was created, including much better Eclipse integration, a workflow engine to control the generation process was added and many other important and useful new features have been implemented. In the beginning of 2006 we started out documentation initiative that resulted in about 250 pages of really good docs for what was now openArchitectureWare 4.0. We releases RC1 on (I think) Feb 1, 2006. 4.0 Final should be out by the time this article is published. My role in version 4 is mainly release manager, tester, doc writer and I have also contributed certain parts to the framework, e.g. the integration with ATL and OSLO, as well as the Recipe framework.

**What's your background, and why do you have an interest in code generation?**

I have been working in middleware and software architecture for a while, at that time maily in the enterprise world, in mid-size to relatively large projects (~100 developers). There were two things that annoyed me: one was that when using typical middleware platforms, you had to do all kinds of things in many places of your system to make it run correctly. There was no "big picture", and it was a lot of work to do. Automating some of these tedious tasks based on "system models" seemed like a good idea. The other thing that made me think was the challenge of how to make sure an architecture could be implemented consistently in large teams with varying levels of developer competence and motiviation. And again, for me the solution is to "implement architecture" in the form of architecture metamodels, DSLs and generators. So these two forces both pushed me into the same direction – which is what we call architecture-centric model-driven software development. It is called "architecture centric", because first and foremost we try to consolidate, formalize and automate the architecture of systems (or system families).

**What is a typical use case for oAW? What technical steps are required from start to finish?**

Whenever you want to build a domain architecture (i.e. model-driven support for a given application domain) there's always roughly the same set of steps you have to go through. This is also largely independent of the tool you're using. So I will describe this generic process, using the oAW tooling as an example. And although I describe the steps in a linear order, in practice you'll execute them in an iterative, incremental fashion!

So, first you start by understanding the domain. If you don't know the domain for which you want to build a generator, you can't build one. You try to structure the domain, find rules and exceptions and you often start conceptualizing by writing down a glossary or some such. Then you formalize it even further into a metamodel, often still on the flipchart. Note that there's an interesting corrollary: the process of building a domain architecture actually helps you to understand the domain, and also helps you to structure it – since if it's unstructured, you can't build a (reasonable) domain architecture for it. Ok, so now you got the metamodel on a whiteboard. The next step is to make it tool usable. So you take a modelling tool (any of the UML tools supported by oAW or EMF, if you're in an EMF environment) and "draw" the metamodel. Using the respective oAW or EMF tooling, you generate implementation classes for the metamodel. You then feed these classes into openArchitectureWare. They are used at generator-runtime to hold the model as instances of the metaclasses.

The metamodel also includes constraints. So, in addition to defining the metaclasses, their properties and relationships, you also need to add constraints on how their instances can be combined into valid models. In oAW you write down the constraints either in our Checks language (which looks a lot like OCL and has nice, metamodel-sensitive editor support for Eclipse) or in "real" OCL using the OSLO plugins.

At this time, you will want to come up with an example model (or set of models, if you use separate models for the different viewpoints of a system). You can use UML, XML, textual notations, EMF instances, etc. for modelling – we support a wide variety of UML tools. You'll instantiate the model and see whether the constraint checks "catch" invalid models.

You can now go about writing transformations. There are two kinds of transformations. Model-to-Model and Model-To-Code. As of today, M2C is still way more important in MDSD, so I'll cover that first. For M2C transformation you use oAW's Xpand template language (for which I'll provide details further below). Again, it comes with a really nice Eclipse plugin for writing those. If you need additional properties or relationships in you metamodel that are not in the metamodel itself (because you might only need them for some of your templates) you can use our xTend language to add those to the metaclasses without touching the metaclasses themselves (much like the extension methods in C# 3.0).

In case you need M2M transformations you can go about this in at least three different ways: First of all, you can code the transformation in Java. If you have a set of nice libraries (as supplied with oAW 4 Classic) this is actually much more productive to do than you would expect. Alternatively, you can also use oAW's Wombat language. This is a small functional language for concisely specifying M2M transformations. Both of these approaches work with EMF based models as well as with what we call oAW Classic (using oAW 3's meta meta model and conventions). If you're in an EMF world, you can of course use all the transformation technologies available for EMF, including TEFKAT, MTF and ATL. For ATL we supply an integration with our workflow engine.

Ah, yes, I forgot: Of course you have to tie together model instantiation, constraint checking, model transformation and code generation. You do this by writing an XML file based on our (really simple) workflow schema. You can then run the workflow either from Eclipse (Run As... oAW Workflow) or from ant.

### How much effort is involved in building new generators?

This is really hard to tell because it depends on so many factors, among them your understanding of the domain and the target architecture, your proficiency with oAW, the amount of change to the domain architecture over time etc. etc. In general, I typically give two numbers (and those exclude the time you'll need to get into oAW): The effort required to create a domain architecture in addition to manually developing a technical prototoype for your architecture is about 25%. So, if you want to build a generator for a system for which you'd normally build a technical prototype, then you'll need to calculate 25% more. And of course, if you use the domain architecture in subsequent (sub-)projects, the relative number is reduced. The second number I give is: if you don't see the benefits of using MDSD after 4 weeks into the project, something's wrong. So you have to make sure you're taking an incremental approach to MDSD so that at (more or less) any time the benefits are accessible to the project.

### How would you migrate to oAW from a legacy project?

You mean from a project that has been developed without any model-driven stuff? Hm. Unlike AOP, where you can sometimes "retroffit" aspects to existing software, this does not work in the model-driven world. So, migrating to a model-driven approach requires that you re-develop some of the legacy system. You'll typically take only a part of the overall legacy system for your migration to model-driven development. That part should ideally be well understood, exhibit some degree of variability. Since the transition to MDSD can be considered some kind of reengineering (you formalize things, describe the structure and variabilities and automate some of the development) you want to be sure that the respective part of the legacy system is something that you'll use extensively in the future. Technically speaking, it is fairly easy to take existing code and "templatize" it. The question is, however, what's the point? If you take "bad" legacy code and create a generator that can (re-)create the same legacy code, you don't win much. So you'll always have to do some refactoring, and also, when defining the metamodel and DSL with which you describe the models, you have to make sure that you can express all the variabilities that you want to cover with the generator. You templates have to be adapted accordingly. And this is not *that trivial* anymore.

I mean, you have to consider why you would want to migrate a legacy system to a model-driven development paradigm. Typically you want to do this because you want to evolve the system, so you want to improve the "evolvability" of the code, or it's internal structure. So the effort you have to invest to make it "model-driven" is probably well-spent, since helps you achieve those goals of improving the structure of the system.

So, in general, the more relevant question, in my view, is: how do you adopt MDSD for something you have up to now done manually, repeatedly? Here I really want to stress again the architecture-centric thing. Instead of trying to build DSL for your business logic, you should first formalize and automate the technical architecture. Several reasons for that: first of all, it relieves developers from again and again writing the tedious technical code that is so typical for many of today's platforms. Second, in many shops the technical architecture is actually the most-reused aspect of the system, so automating it pays off quickest. Finally, the developers themselves are the *domain experts* – i.e. you don't have to design the DSL with a real customer: you developers are their own customers. Later, once you have a stable and well-automated basis, you can cascade business logic DSLs on top of the architecture centric one.

Finally, let's not forget that MDSD alone does not solve a problem: you should also take into account iterative, incremental development as well as testing. In some ways, these two can be considered a precondition to successful MDSD.

### Where is oAW being used? What types of project?

oAW is used in all kinds of domains – specifically, outside the mainstream. Examples include J2EE-based banking and insurance web applications, Spring-based applications, Eclipse-based Rich client apps in various domains, C/C++ based realtime systems in the automotive domain, data management and conversion in C++/Java/Python-based radio astrononomy systems, .NET applications, Enterprise SOA architecture, etc. So we have a heterogeneous user community – which also makes it quite interesting.

The size of the project obviously varies from simple one-man shows up to really larger projects, where typically a subset of the developers take care of building the domain architecture and the rest use the domain architecture to speed up application development.

### What are the perceived and actual benefits you should realise by using oAW?

I think I've mentioned some of these above. In general, MDSD makes for more efficient development – most people readily understand this. However, the benefits of structuring and thereby improving your software quality (for example by harmonizing the architecture and separating technical from functional concerns) is actually at least as important. You get these benefits specifically, if you build your own domain architecture (as opposed to downloading a cartridge and just using it). This is why we really recommend people to build their own metamodels, DSLs and transformations! If you just download something from the web you're in the 4GL/CASE trap: the tool might do 80% of what you need easily, but you'll have a hard time tweaking the remain 20% into the tool.

oAW specifically really make domain architecture development relatively easy. We are convinced that we have the best template language in the world (although out competitors might disagree with that ☺). Our Eclipse integration, specifically the statically type checked templates, is a real productivity booster. Our workflow engine really does the job

well. And finally, we have really good documentation since version 4, and also good support in our forums at openArchitectureWare.org

**What is the size of the user community?**

This is really hard to tell, we don't have actual numbers. Here are some facts: we typically have several new discussion threads per day on our forum. Myself and other folks who do professional consulting for oAW really have enough work to do. On conferences, when we do oAW sessions and workshops, we always have quite a number of people.

I think we are probably the number two Open Source MDSD/MDA tool after androMDA, we are probably number one for projects where people *build* their own domain architectures as opposed to downloading pre-built cardtridges. However, as I said, I have no numbers to support this statement. It is based on the feedback I get from people.

**Are there any published case studies I can refer to?**

There's certainly the presentation at EclipseCon 2005 that resulted from the BMW Car IT project at http://www.voelter.de/data/presentations/EclipseCon.pdf. We don't have more case studies online yet, but we are working on changing this.

**What was the thinking behind moving the project to the umbrella of Eclipse in January 2006? (Say a little about the GMT project and possibly about the relationship to other tools – similar or different approaches.)**

The GMT project actually serves as an umbrella project for various projects in the MDSD world. Among other things, it also includes the ATL transformation engine as well as the MOF2Text project for code generators. So one of the reasons of migrating to Eclipse was the potential synergies among these tools – for example, we already integrated the ATL engine into oAW. There are of course also a number of interesting tools outside of GMT, Apart from EMF, them GMF (Graphical Modelling Framework) project is probably the most interesting one since it allows DSL creators to (relatively) easily build graphical editors for EMF-based metamodels.

Also, all our support tooling was already integrated into Eclipse, so this was also a force that pulled us into this direction. Of course, another reason was the increased visibility for oAW. Being on Eclipse.org is certainly helps popularity. And we also felt a little bit honoured when Jean Bezivin asked us to join, so we didn't hesitate ☺

**Who's involved in the Eclipse work?**

Currently, I am the only Eclipse.org committer, so I am playing the proxy there. However, this is absolutely not going to remain that way. As soon as it becomes clearer where GMT is going (it is probably going to become a part of the Eclipse Modelling top level project), we will make sure we'll add two or three more committers. We are a very active project and we really need more people that can contribute directly.

**The project sponsors are all based in Germany. Any plans to widen the community of sponsors? What types of support can these sponsors offer?**

The fact that the sponsors are based in Germany is mainly a consequence of the history of oAW as I explained above. We are obviously trying to widen the community. All our documentation is in English, out support forum is English, too. As soon as there'll be non-German speaking developers, we'll change the core-developers list to be in English also.

With regards to sponsors, we have to be specific what this means. Obviously, there is b+m who provided the original oAW. Then there are a number of companies (oose.de, MID, b+m and itemis) who specifically sponsored the documentation effort. More sponsoring happens for example by itemis who pay Sven for some of his time when working on oAW. There are a number of other companies who have allowed us to use things that we created in projects paid for by those companies as part of oAW. The prime example for this kind of sponsoring is certainly BMW Car IT.

So, with regards to the kinds of support sponsors can offer, there is the full range:

- You can simply support some of the developers by paying them for developing oAW
- You can contribute additional modules, docs, patches, etc. directly, eventually becoming a core contributor to oAW
- You can hire some of the developers (or the companies behind them) to work on your projects, and allow them to make some of the (non-business-critical) stuff available as Open Source.

So, just as with most Open Source projects, some kind of backing by somebody who has money is important. oAW wouldn't be where it is today if this had not been the case in the past.

**What are the key features of oAW? Can you say a bit more about the xPand template language and the template editor? (Could include a screenshot here)**

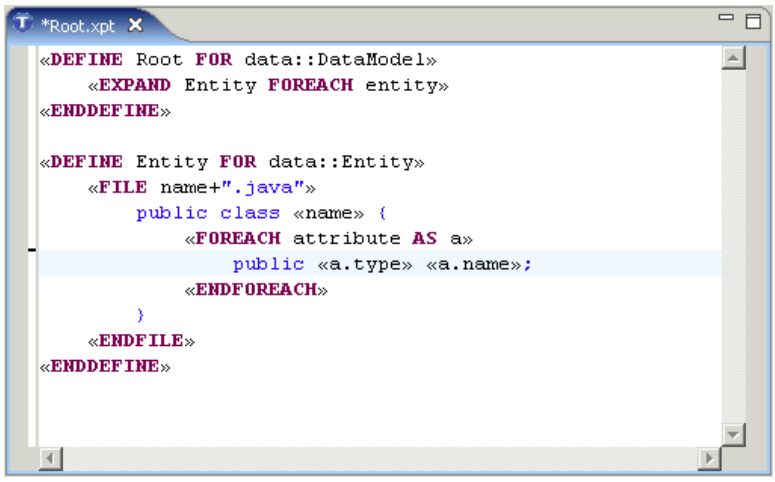
Here's a list of the core features:

- The workflow engine precisely controls the generator's workflow, as specified in an XML workflow definition.
- With a suitable instantiator, oAW can read any model. Currently, we provide out-of-the-box support for EMF , various UML tools (MagicDraw, Poseidon, Enterprise Architect, Rose, XDE, Innovator, …), textual models (using JavaCC or antlr parsers as frontends), XML, Visio as well as pure::variants variant configuration models.
- Can generate any kind of output, using a powerful template language called Xpand that supports template polymorphism, template aspects and many other

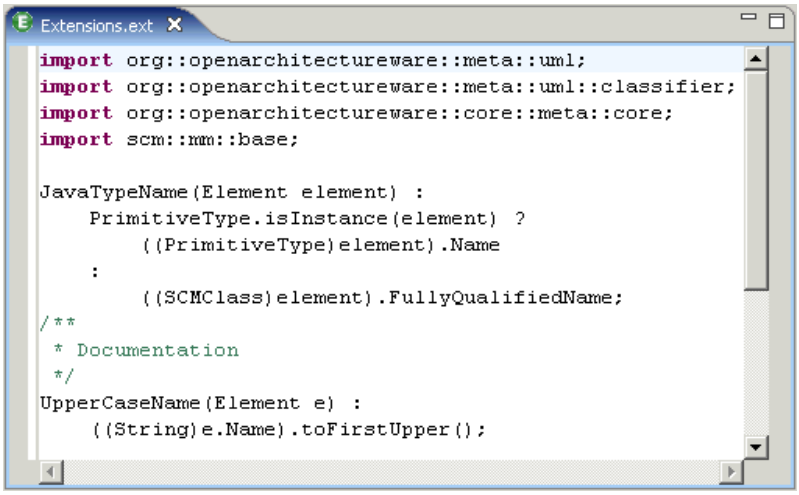
advanced features necessary for building non-trivial code generators such as optional typing of model elements and sorting and filtering of element sets.

- Explicit domain metamodel: the metamodel of the problem domain is represented as Java classes. Metamodels can be built using either Eclipse EMF or oAW's own metamodel generator that creates those Java classes from metamodels rendered in UML. The generator already comes with the most important UML metaclasses, which can be reused to build UML profiles.
- Various ways of checking model constraints: a semi-declarative checking using Java APIs, a proprietary declarative constraints checking language as well as optional OCL integration based on the Kent OCL framework.
- Support for multiple models: you can read several models as part of one generator run. These models can use different concrete syntaxes (one could be UML, another XML and a third one Visio). The generator then "weaves" these models together to form a comprehensive, all-encompassing model. Constraints can be checked over model boundaries; references among models can be established.
- Model-to-model transformations can be implemented using the Wombat language, a textual, functional transformation language. There are two very important properties of this language: first, it has a very concise and powerful syntax. Second, it can transform between the various metametamodels, *e.g.,* you can transform a model defined with the oAW-classic metametamodel into an EMF model! Alternatively you can integrate third party transformation languages such as ATL. An Adapter is provided.
- The Recipe Framework allows you to define validation rules for artefacts created outside of the generator (such as manually written subclasses). During code generation, these rules can be instantiated; later, the Eclipse IDE will read these checks and verify them. This will help to guide developers beyond the modelling/generation stage.
- Complete modular design: You can enhance or extend any of the framework components as you see fit

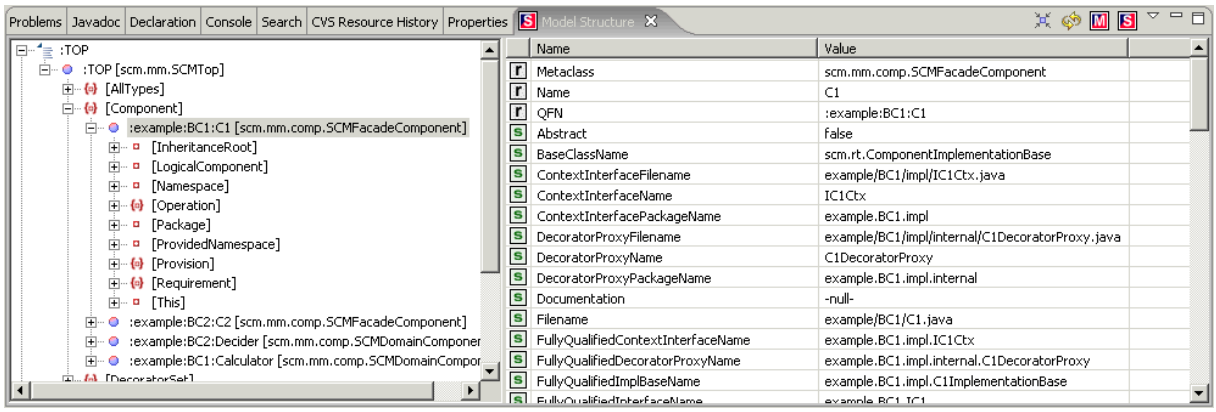
openArchitectureWare comes with a number of Eclipse Plugins that help make development more efficient. Here are a couple of screenshots. The following one shows the Xpand template editor. In addition to syntax highlighting, it provides metamodel-aware code completion facilities as well as static error checking.



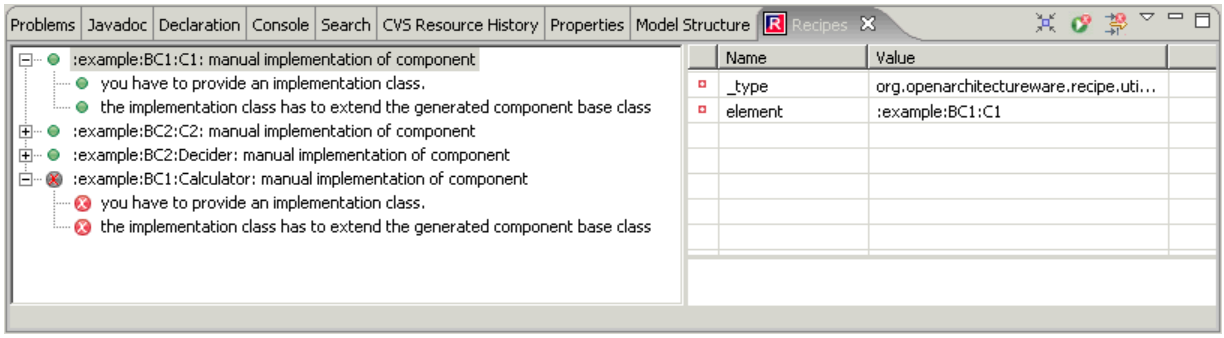

You can also define metamodel extensions, *i.e.,* properties on metaclasses, that are available in the template language. They are defined externally to the metamodel implementation classes.




A model structure view helps you view and understand models that have been parsed from UML or other sources. For EMF based models this is not necessary, since the EMF-supplied editors provide exactly the same functionality.




The checks verified by the recipe framework can be rendered nicely in an Eclipse view. Changing something in the manually written source code will update the checks in real time.




**Can you give examples of when it would be useful to weave multiple models together?**

Almost always. There are two kinds of situations. I like to call them paritioning and subdomains. Partitioning is rather straighforward. Consider a really big application that consists of hundres of components that are described using XML files. You don't want to have these hundreds of component descriptions in one model file. So the generator has to read a set of model files and still build one large model. So in the case of partitioning, it's simply about breaking a large model into several smaller model files.

Subdomains are more interesting. You have to use them if you'll model various aspects of your system using different DSLs, specfically if you have to use different concrete syntaxes. Consider the generic web application consisting of service definitions and a business data model, a persistence mapping, a form flow definition and form layout. You cannot sensibly model all those aspects with, say, UML. Just try to do form layout in UML, it does not work. So you might want to define services and the business data model with UML class diagrams, the form flow using UML state diagrams, the persistence mapping using XML

(assuming you don't use Hibernate & friends) and the form layout with XML. In that case, your generator needs to load all of these different models (some XML, some UML/XMI) and "weave" the models together. It is essential to understand that usually it is *not* possible to load the various models sequentially in multiple generator runs, because, in order to check the consistency and also to generate sensible code, several of the aspects are needed. For example, to generate the forms, you need the form flow definition, the form layout as well as the business data structures. So the generator needs to load all the asepects – or subdomains – in one generator run, weave the models, and generate the application.

Of course, there's the philosophical discussion whether the example I just gave is *one* model that is spread into several files, and accordingly, *one* DSL with *one* concrete syntax that has those different parts or whether it's several models, several DSLs and serveral concrete syntaxes. While this is certainly an interesting topic to discuss about in a conference workshop, it's irrelevant from a practical perspective. De facto, it's different artifacts that your generator has to load simultaneously and weave.

Note that another distinction between these two kinds of multi-model is that in the first case, you have a number of partitions that use the same metamodel (it's just a big model broken down into chungs) whereas in the second case, your subdomains use different metamodels (and usually, different concrete syntax).

**The Recipe framework can be used to check artefacts defined outside the generator. Could this support migration from a legacy system?**

So let's first briefly explain what the Recipe Framework is. Consider the following scenario: Your model contains components. The components in the model specifically describe how the components communicate with other components, i.e. it contains provided and required ports as well as the definition of the interfaces with which these are associated. The model does not contain any component implementation code. So, you have to manually write the implementation code for the components in the model. A best-practice way of doing that is to let the generator generate a base class that contains all the aspects that can be derived from the model (e.g. correctly typed proxies for the required ports), and the developers have to extend from that generated base class and overwrite/implement the methods implementing the behaviour. Here is where the Recipe framework can help you: the problem is, that a normal generator *cannot* ensure that developers correctly extend the generated base class and overwrite certain methods since this stuff is done manually and therefore beyond the control of the generator – it has already terminated by then! So what you can do with the recipe framework is that during code generation, you define certain constraints which must be evaluated against all of the generated and manually written code. These constraints are written to a file by the generator, and after the generator terminates, the IDE (in our case, Eclipse) loads these constraints and checks them. A nice view provides interactive feedback about the status of these constraints. So in our example, the constraints would evaluate to false as long as you

didn't implement the subclass (typically obeying a certain naming convention) and have it extend the correct superclass. As soon as you create it, the constraints are re-evaluated and the errors go away. So, in short, the Recipe framework can provide developer guidance wrt. to the programming model *beyond* the runtime of the generator. This really quite helpful for larger projects, where generated and non-generated code have to be integrated using certain well-defined patterns.

To answer your question wrt. legacy system migration: yes, you could use the Recipe framework to make sure that the those aspects of a legacy system that are already migrated to a model-driven development style are compatible with the "rest" that is still the legacy code. I have to admit, though, that I did not yet try to use it in this way.

### What are the main benefits of release 4 over previous releases?

I think most of this can be read from the above feature listing; to summarize:

- EMF support
- Enhanced xPand language (e.g. aspect oriented templates)
- Checks and xTend languages
- M2M Transformation using Wombat
- Better control of the process using the workflow engine
- Much better IDE integration
- Recipe Framework
- Significantly enhanced documentation and samples
- And some more support for additional UML tools
- In general better code structure, improved modularity, better APIs

Since we still support the oAW approach based on the "old" metamodel generator and also the "old" XMI instantiators, I think there is no reason for new projects to stay with oAW 3. We are told by users that using Version 4 is much simpler.

### What are the limitations of the current toolset?

In the core, there's the problem that the EMF and Classic integration is not perfect. For example, the definition of metamodels in an EMF based scenario uses EMF's facilties while in the Classic case you'll use our own metamodel generator. Also, the instantiators that read the XMI from various UML tools can only instantiate Classic models; we cannot yet create Ecore instances from these XMI files (unless, of course, the tool produces EMF-based XMI output or uses Eclipse's UML2 project).

### What are the barriers to adoption?

One issues is, of course, that we don't have a "market place" of domain architectures (or building blocks, such as metamodels, templates, etc.) for mainstream platforms. Basically, this means that you have to build your own generator. Of course, in many situations this is what you want to do anyway. But sometimes it would be useful to just reuse already existing cartridges for Spring, J2EE or Hibernate. The fact that we don't provide these is certainly something that slows down adoption. Another aspect is – and that is probably intrinsic to this kind of tool – that a powerful tool does have a significant learning curve, so you have to invest some time to get into oAW.

And then there are all the obstacles to adoption of MDSD in general. Technically I think the most pressing problem is the pain you have to go through to come up with custom editors for your DSL – textual or graphical. As a consequence, many people use XML or UML + profiles, the two "generic" modelling syntaxes, for rendering their models. This has many negative consequences. In the Eclipse world, the GMF project will help to solve this problem for graphical editors. I am still waiting for a TMF project, that allows you to quickly generate textual editors for EMF metamodels.

Finally, there are organizational issues when adopting MDSD, i.e. you have to adapt your development process, your team structur, your architecture awareness, etc. These things can be hard to change in existing organizations.

### Is there a roadmap for future releases?

We don't yet have a formal roadmap yet – we are working on it, so keep an eye on eclipse.org/gmt/oaw. However, we have a number of features that we want to provide in subsequent versions, after 4.0. These include

- Ability to instantiate Ecore models from our XMI instantiators
- Of course, a wide range of incremental improvements for the IDE support, including (limited) support for refactoring and F3/Ctrl-G support.
- A generic model explorer/editor that can explore/edit all kind of (EMF and non-EMF) models
- Some more extensions to the AO facilities of the generator, as well as lazy evaluation support for regions in templates
- More pre-built Recipe checks

**How does he see MDSD and oAW in particular being accepted in the developer community?**

Hm. This is an interesting question. Let's first look at MDSD in general. I think that the adoption of MDSD and MDA is not as rapid as one could have guessed from the original hype. Although I have to say that I am involved in a number of interesting projects that do use MDSD to great effect – and I can't imaging to be able to run (most of these) projects without MDSD any time soon. I think "simple MDSD" is used quite often, where, for example, people generate annoying boiler plate code for a "bad" infrastructure. However, MDSD really becomes interesting once you have technology independent programming models, multiple viewpoints in your models to automate packaging and deployment and sophisticated model validation. I don't see too many folks doint that. Also, currently MDSD is often quite UML centric; this can result in awkward modelling syntaxes or outright rejection of MDSD since people don't like UML. I think that projects like GMF – the Eclipse project for generating graphical editors for EMF based metamodels – will bring about a big change.

You know, I think the code generation aspect of MDSD is a solved problem and many people use code generation in one or another flavour. But thinking in models, using models to represent various viewpoints at various levels of abstraction, using sophisticated model validation or building your own editors/IDEs is still something of a niche topic. Reasons could be that the tool support isn't perfect yet – although tools *are* workable today! – and that such an approach is quite different from how many people develop software today, technically, as well as from a process point of view.

I am pretty sure though, that MDSD is going to grow steadily – maybe slower that some might wish.

Wrt to oAW: I think I elaborated on that quite a bit already above. In general, I think the adoption goes really well – we still have a publicity problem outside of Germany/Europe. But those people who used oAW, especially the Version 4 Release Candidates, are really happy – the feedback is quite encouraging.

**How do you see the difference between MDSD and MDA or Software Factories?**

Let's first look at MDA. MDA, in my view, is a specialization of the generic MDSD approach. For several aspects of MDSD, the OMG proposes certain standard solutions, e.g. MOF as the meta meta model, UML + profiles as the de facto modelling standard (yes I know that you can use all languages that are defined in terms of the MOF – but tool support de facto restricts you to UML + profiles), QVT for model-to-model transformations, OCL for constraints, and the notion of PIMs and PSM for organizing and categorizing your models. Note that, of course, where it makes sense, MDSD folks will use MDA standards. However, oftentimes the standard is a bit big and clumsy, which is why pragmatics typically use only subsets. Examples of this approach can be seen in EMF (which implements EMOF, a subset of the MOF), various constraint languages that

implement subsets of OCL or various M2M Transformation languages that are more or less similar to QVT. Also, the de-facto restriction to UML is not something I find particularly useful.

Software Factories are a little bit different. This is because they have a broader scope. Basically, SF is Product-Line Engineering the Microsoft way. So SF provide tools to organize the various viewpoints that are required when developing a software product of a particular kind, specifically, you can customize your Visual Studio IDE to make development of that particular kind of application more efficient. This includes custom wizards, custom help, custom project structure and – this is where it gets model-driven – custom DSLs. So DSLs are *just one part* of the Software Factories initiative. That part, however, is conceptually compatible with MDSD. Microsoft has their own standard meta meta model (called MDF, for meta data framework). The DSL Tools extension to Visual Studio let's you "draw" metamodels using MDF, you can also define a concrete, graphical syntax for your DSL. Transformation tools to generate code are also included.

Note that there are other similar approaches such as Model-Integrated Computing (MIC, put forward by the people in the technical computing area, specifically, UVanderbilts's ISIS), Language Oriented Programming (LOP, advocated by JetBrains and their MPS tool), Domain Specific Modelling (DSM, a term mainly used by Metacase) and of course Generative Programming (GP, popularized by Krzysztof Czarnecki and Ulrich Eisenecker).

So, while all of these approaches have different history and focus and thus do certain things differently, they are all basically the same in that you define a DSL, use that DSL for describing models and then process these models (transformation, generation, analysis and simulation).

**What other projects are you working on? Can you say a little bit about your forthcoming book?**

Sure. The book is of course the hightlight, especially for the readers of this interview. It is a significantly updated translation of the German book on Model-Driven Software Development that I wrote with Tom Stahl last year. You'll find details at www.mdsd-book.org. It is a quite practical guide to MDSD covering topics from metamodelling over how CBD and SOA goes together with MDSD, we look at the development process and many other facets. Go to the web site to take a look at the table of contents.

Another project that I am working on with a couple friends is Software Engineering Radio. This is a podcast (i.e. MP3's distribued via RSS) for the professional software developer. We have interesting tutorial sessions on topics such as middleware, software architecture, concurrency, programming languages, etc. Also, we feature interviews with the "big names" of the community such as Doug Schmidt, Eric Evans, Gregor Kiczales or Ted Neward. We have already done a couple of Episodes on Model-Driven Software Development, by the way. Check it out at www.se-radio.net, it's great stuff.

My final project – as every year during summer – is of course to work less and spend more time in my sailplane (see http://www.voelter.de/flying/pictures.html) ☺