# Workshop on Generative Computing

## Workshop on Generative Computing, Model-Driven Architecture (MDA) and Model-Driven Software Development (MDSD)

Medieninformatik-Forum, 1.7.2004, Hochschule der Medien Stuttgart

Prof. Dr. Edmund Ihler

Prof. Walter Kriha

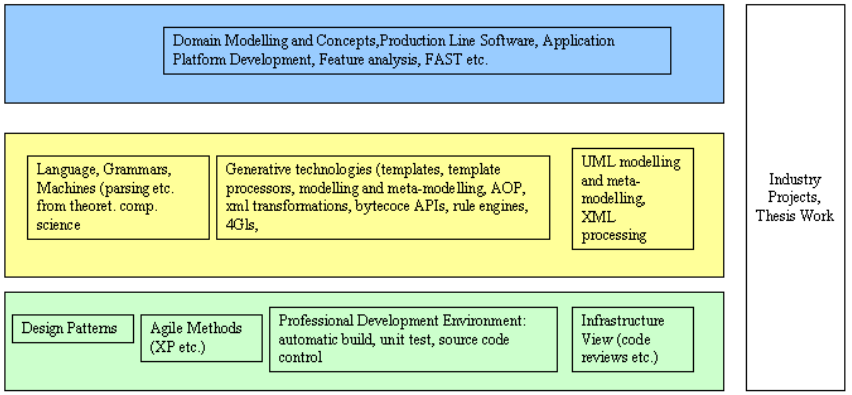


## Agenda

1. Generative Computing at HDM – What we do
2. Model-Driven Architecture in Practice (Markus Reiter, Joachim Hengge, Softlab/HDM)
3. Automatic Business Process Composition using Semantic Technology (Christoph Diefenthal, IAO/HDM)
4. Metamodelling in Smalltalk (Claus Gittinger, Exept AG)
5. Using Generative Computing in a large scale EAI framework (Marcel Rassinger, e2e basle)

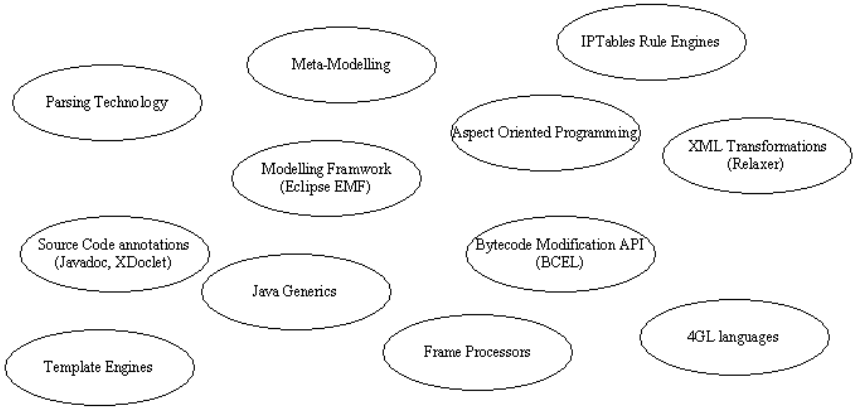


## Generative Computing at HDM




Production Line oriented software development needs a lot of supporting knowledge and technology

# Generative Technologies (this term)




**From each technology we learned different things, e.g. to recognize limitations (when do you need a model or a meta-model?). We started with generation, moved to modelling and ended with parsing and AOP.**



# What we learned

1. Code annotation is easy. Mostly for implementation specific add-ons (deployment descriptors or documentation). XDoclet provides extensible framework. Granularity limited by what parser recognizes.
2. Template engines are simple to use. No model required. Can be too powerful (see Terrence Parr on language restrictions for template engines). Required features are e.g. reading and reacting on model attributes and dynamic sub-template inclusion.
3. Frameprocessors need a definition of slots, i.e. a good knowledge of hot/cold spots of the domain. They keep template results in an in memory AST and delay code generation (serialization) till the end. Easy, powerful and no model required. Breaks down when granularity (slots) no longer sufficient.
4. Modeling Frameworks read several model formats (UML, XML etc.) and transform it into an instance of a meta-model (ecore for eclipse). The meta-model needs a meta-meta model which can use the same classes. A M3 model is not always available or tools do not use it. Granular capture of features is possible. Lots of transformations on the same level.
5. Bytecode manipulation APIs require a lot of VM know-how but allow transparent addition of features. Interesting but cumbersome and hard to use.
6. Aspect Oriented Programming uses e.g. bytecode manipulation to add functionality. Granularity is only limited by what the Aspect parser support. In theory every target language statement or terminal could become a pointcut for extension. Incredibly powerful but cries out for modelling of the aspects to avoid total chaos. Needs integrated view of all aspects.
7. XML processing with Relaxer: fast, easy. Needs access to meta-data in generated java classes, e.g. to support an editor which needs to offer legal options.
8. Grammars, languages and automata are useful constructs to understand parsing and generation issues.
9. 4GL languages are incredibly effective. (16K->400K->16MB generated code) but are really domain languages and therefore have a built in acceptance problem outside the specific domain.



# What we tried

1. Building a Struts aware generator which reads the Struts model and generates/validates artifacts (you can see it later at media night presentations)
2. Gerneration of CSV file readers from Eclipse JET – driven by an XML model of the CSV format
3. Generation of a notepad like editor with Swing GUI using a frame processor (XML based)
4. Adding logging statements statically and and load time using bytecode generation API (bcel)
5. Lexical scanning and parsing using ANTLR and self-defined grammars
6. Investigate IPTABLES rules producing generators
7. Generate glue code from annotated Java (Javadoc, XDoclet)
8. Using EMF and GEF to create a model of a forum application which can be filled in.
9. MDA frameworks, genetic algorithms etc. just for the fun of it....



# What we didn‘t touch

**Everything that has to do with domain analysis, production line software etc. This will be handled in the next lecture on generative computing.**

**Model-Driven Architecture (MDA) in details. We just didn‘t have the time (;-)**

# What we would like to know

1. What could a really powerful programming language do? Is it true that a domain language needs to be different from the implementation language?
2. What about interpreting a model instead of generating code from it?
3. UML and XML – what is their relation? Can UML really describe domain concepts? Is XML only a good serialization format?
4. Is there a proven way to generate meta-models and meta-meta models and to use them for code generation?
5. Can we build an editor for eclipse EMF that understands M3 MOF and edits M2?



# Surprise 1: Theoretical Computer Science is useful!

1. Teams working with generative computing need a good understanding of languages, grammars, parsing and automata to be effective. (see Markus Voelter on MDSD in Object Spectrum June 2004)
2. Theoretical computer scientists change the way they where teaching the subject: „Uwe Schoening, Ideen der Informatik“ is readable both for computer science students and practicioners.



# Surprise 2: Model-Driven Software Development

1. A multi-paradigm approach (domain analysis, meta-modelling, model-driven generation, template languages, agile software development, open source infrastructure) from: Jorn Bettin, Model driven software development, june 2004
2. Compared to MDA relaxes several principles: not bound to UML/MOF, DSLs may be non-graphical. No tools based round-trip philosophy
3. MDSD patterns like: do not work on the serialization format, extract infrastructure, two (or more) phase generation to ease transformation and allow adaption to different platforms.
4. Provides technology and workflow to implement production line architectures

Markus Voelter and Jorn Bettin are about to release a book on this subject. Markus Voelter will give us a detailed introduction in fall here at HDM.



# And now have fun with:

1. Model-Driven Architecture in Practice (Markus Reiter, Joachim Hengge, Softlab/HDM)
2. Automatic Business Process Composition using Semantic Technology (Christoph Diefenthal, IAO/HDM)
3. Metamodelling in Smalltalk (Claus Gittinger, Exept AG)
4. Using Generative Computing in a large scale EAI framework (Marcel Rassinger, e2e basle)