Course overview - Generative Computing

Look at the resource section at the end for links.

Course Overview

"For learning to become pemanent it must pass through stages of exploration, play and interestingly, retelling of what is being learned. People understand and remember new things, which is to say they master them, only after actively exploring them. Humans learn in action" (quote taken from Mannings Publication Co. and their "in action" series)

The lecture will follow this advice and take you from simple code generation techniques to advanced meta-modelling using examples and practical exercises, presentations and discussions and of course theoretical work.

Long Term Course Structure

The course "generative computing" has a different focus every summer term:

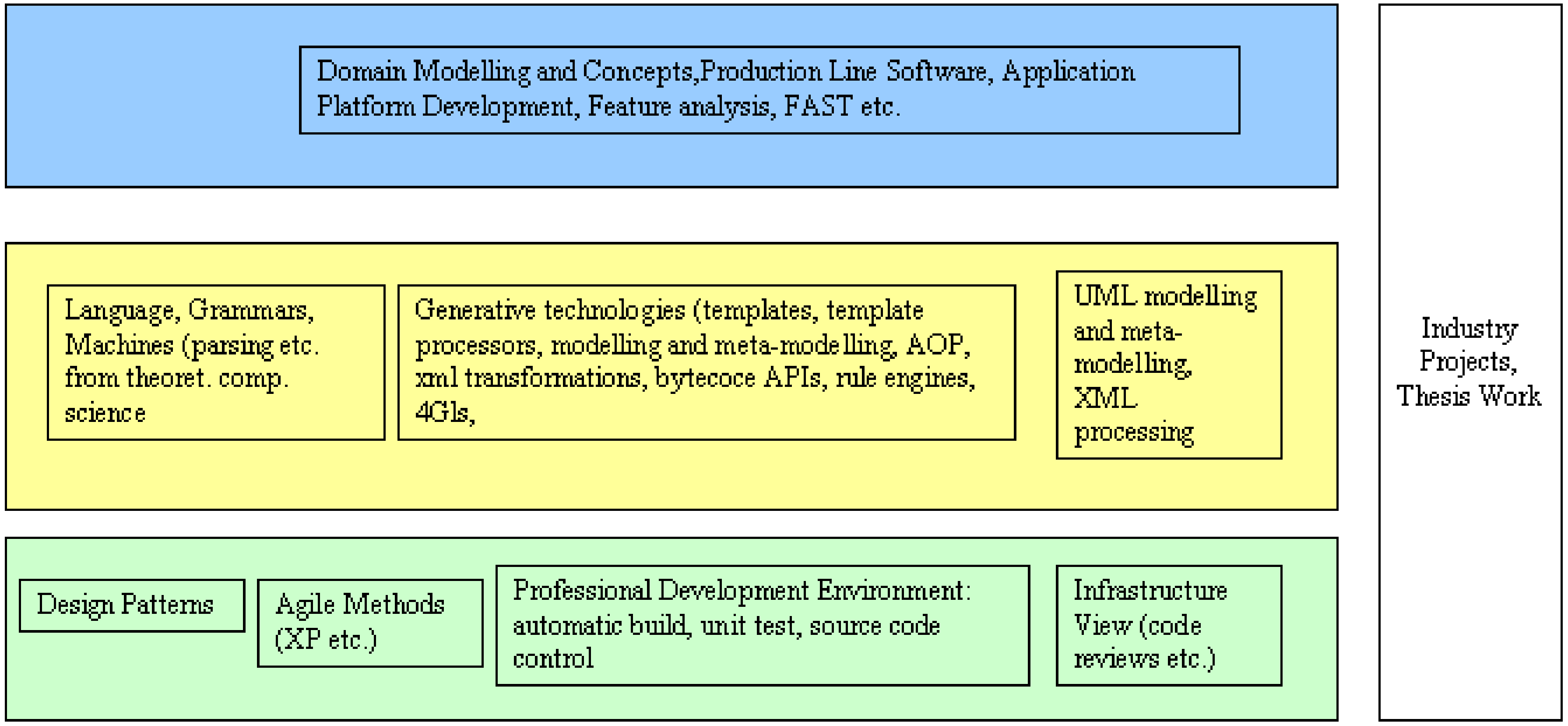
Automated, professional devlopement environment and methodology	Covers automated builds, source code control, extensible IDEs like eclipse (plug-in approach), collaborative features like wiki's, bug reporting etc. The technical means are embedded into a concept of agile development with continuous integration, unit test and so on. The flexibility goals of generative computing are achieved through automated scripts (e.g. ant), plug-ins and simple code generation (source code completion, IDE templates, Javadoc/XDoclets)
Code Generation Technologies	Covers advanced code generation technologies (templates, transformational, meta-programming) and the theory behind (frame processing). Code generators will be used and examined. Goal is to enable participants to build their own generators. Models and meta-models play a central role as input for generation. Domain engineering is shown to give participants a complete picture of CG but is not explained with its theoretical foundations.
Domain Engineering	Covers feature analysis, Domain Specific Languages, production line architectures etc. How are conceptual models created? What is the connection between component architectures and domain

Long Term Course Structure (Continued)

configurations?

The course is also part of current research activities in the area of development and runtime environments for mobile and embedded systems. It is also tightly related to the seminar "design patterns" (simple patterns, architectural patterns, pattern use e.g. to support extensibility and performance in eclipse, generative patterns etc.)

Courses on Generative Computing/Model Driven


Domain Modelling and Concepts,Production Line Software, Application Platform Development, Feature analysis, FAST etc.
Language, Grammars, Machines (parsing etc. from theoret. comp. science
Generative technologies (templates, template processors, modelling and meta-modelling, AOP, xml transformations, bytecoce APIs, rule engines, 4Gls,
UML modelling and meta-modelling, XML processing
Design Patterns
Agile Methods (XP etc.)
Professional Development Environment: automatic build, unit test, source code control
Infrastructure View (code reviews etc.)
Industry Projects, Thesis Work

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

Goals for "code generation"

1. Learn how and when to use generative technologies.
2. Learn the techniques behind generative approaches. When do you need a model? How do you combine components/frameworks with code generation classes? Finally, how do you BUILD a code generator.
3. Understand why code generation is so important in todays J2EE or .NET environments
4. Learn the necessary tools: template processors, code generators, compiler-compilers, XSLT-engines
5. Learn to perform domain analysis and create meta-models and models.
6. Finally learn how to create flexible software by different means: frameworks, components, generation or mixtures of all these technologies.

It is my firm believe that people who really understand code and generation techniques will automatically realize the importance of models and meta-models. The first generative techniques we will use do not require a model. But as we go along we will experience that dreadful statement "I can't fix it - it's not in the (meta-)model!" more often.

Roadmap

1. Why code generation? Introduction to generative technologies
2. How to perform a domain analysis, create a model/meta-model and build a code generator - experiences from a thesis. (With participation from the industry)
3. Simple generation: Java Code Annotation, Java generics, Java doc, XDoclet
4. Template Processors: Eclipse JET and what you can do with it. The theory behind frame processors.
5. Model driven generation: Eclipse Modelling Framework. Advanced code generation with this framework. We will use it and also take a look at its implementation.
6. XML/XSL based code generation: using Relaxer, Schematron etc. to transform XML Meta-data into source code. Why is the functional language approach so successful in generation? Advantages of "duck modelling approaches (see Sean Mc Grath in the resources section)
7. Compilation: how ANTLR works. Grammar, recognizers and AST. How to build a network protocol using a grammar. Or how to validate input using a compiler.
8. Domain and software analysis for a large industry project. Shows how a large piece of software is analyzed and chances improvement - either through code generation or organizational processes - are detected. Shows how models and meta-models are a natural consequence of such an analysis. (With participation from the industry)

Roadmap (Continued)

Includes commonalities and variations, techniques for generalizations.

9. Meta-modelling (MOF etc.) and Model-Driven Architecture. How to create meta-models and how they can be used in generative computing.
10. Architectures and dependency management. Some theoretical work on how to separate business logic from presentation needs in templates. How to decouple components and interfaces. Externalization of interfaces and implementations using inversion-of-control type architectures.

Read, discuss and try

The seminar will be held as follows:

1. For every session some ADVANCE reading is required.
2. The topic will be presented and discussed - based on the literature
3. We will go into the lab and try it.
4. Groups of students will pick a specific technology, use and investigate it and prepare a presentation (theory and practical work) for it.

Participants have to be fluent in design patterns, have implemented software systems or applications and must be able to understand source code by reading it. Some technologies involve languages other than Java or XML (like Ruby or C++). XSL know-how is also a big benefit.

Note

Most important: if you cannot spend time to read articles or books during this term: don't decide to join this course.

Theoretical Background

During the course a couple of theoretical concepts are needed. Everybody participating will have to pick up something and present it to the class. Things to cover are:

1. **Domain Analysis**
2. **Feature Analysis**
3. **Frame Processor Technology**
4. **Grammar and Language Recognizers (Parse trees and abstract syntax trees (AST)).**
5. **Meta-programming**
6. **Model serialization using XMI.**
7. **Meta-modelling using the Meta-Object-Facility of OMG**
8. **Functional languages**
9. **Cross-cutting concerns (Aspect Oriented Programming)**
10. **Byte Code manipulation (e.g. JDO)**
11. **Roundtripping and artefact preservation (Generation/manual manipulation)**
12. **Generalization Techniques. Variations and Commonalities. Scope definitions.**
13. **Problem domain vs. solution domain. Separation of languages.**
14. **Generative Design Patterns**

Theoretical Background (Continued)

15. Interpreters and Simulations
16. Data-Driven Design (game design, automotive computing)
17. Component technology, inversion-of-control patterns, API evolution

Things to try

This is a list of possible applications for code generation. Participants can decide which one they would like to use and present.

1. Using the Java metadata interface to generate code. Show how Java code annotations can be used.
2. Using XDoclet to generate EJB artifacts (entity beans, facades and business delegates, value objects etc.)
3. Creation of an iptables generator from user input. Chose whatever model/code generation technology you find appropriate. Take a look at existing generators for firewall scripts first.
4. Take a look at the source code behind the JET emitter framework in eclipse. Describe architecture and patterns and show how they are used.
5. Analyse the ecore meta-model of Eclipse EMF. How does it relate to the UML meta-model?
6. Use the eclipse EMF to model and generate something.
7. Use ANTLR to define the grammar for a network protocol and generate the code templates for it.
8. Use JavaCC to create a compiler for a self-defined language.
9. Investigate Frame processing technology and show how a public domain frame

Things to try (Continued)

processor is implemented

10. Use a MDA tool to generate something (andromeda or any other available MDA tools is OK)
11. Take a Java bytecode modification package and use it to build a simple code interceptor. This is an early stage of an AOP engine.
12. Get the aspectJ engine and implement something. Explain the basics of aspect-oriented programming. Do not use logging as an example (;-)
13. Take an executable UML package - if one is available - and use it.
14. Find out how a debugger for a template processor could be implemented (for JSP, JET or whatever). Why do generative approaches frequently pose a debugging problem?
15. Take a close look at generators for games (scenes etc.)
16. Use AspectJ to generate log4j statements automatically before and after every method.
17. Build a little recognizer (lexical and syntax level) by hand using the "Building Language Recognizers by hand" chapter from Terrence Parr.
18. Try to create test cases automatically from a model. What parts of an application can be generated?

Things to try (Continued)

19. Desing a small Domain Specific Language for a domain.
20. Perform a commonalities/variations analysis for an intended production line in some business area.
21. Build an interpreter. Connect the interpreter to a framework.
22. Speculate on the role of meta-data in autonomous computing.

Required Reading for the next time

Introduction to Model-Driven-Software-Development, by

1. **Jorn Bettin, Process Implications of Model Driven Software Development Objectspectrum Article [http://www.softmetaware.com/process-implications-of-mdsd.pdf]. This article is also available in German [http://www.sigs.de/publications/os/2004/MDD/bettin_MDD_2004.pdf]**
2. **Jorn Bettin, Model-Driven Software Development: An emerging paradigm for Industrialised Software Asset Development [http://www.softmetaware.com/mdsd-and-isad.pdf]. This article contains the major forces and patterns behind MDSD. This is required for the third session!**

Resources

The resources cover freely available information as well as excellent books right to the topic. All entries are commented to let you know what a paper or book is all about. I also expect participants to use this literature in case of questions.

Free Information on generative computing

The following information is freely available and taken together is an excellent introduction to the subject.

1. annotations in Tiger [http://www-128.ibm.com/developerworks/java/library/j-annotate1/index.html]. More on Java annotations with examples [http://ramnivas.com/blog/index.php?p=15]
2. (logging with AOP) [http://www-128.ibm.com/developerworks/library/j-cwt03085/?ca=dnt-610] how about comparison with autonomous computing logging?
3. The Code Generation Portal [http://www.codegeneration.net] contains good articles on theoretical and practical problems of code generation. It also offers a list of articles and generators [http://www.codegeneration.net/tiki-index.php?page=MainArticlelist]
4. Kathleen Dollard on generating code for .NET describes her experience with code generation on the windows platform. [http://www.codegeneration.net/tiki-read_article.php?articleId=47].
5. Angie - a frame processor. Architecture of a frame processor [http://www.d-s-t-g.com/neu/media/pdf/facts_e/dlt21474.pdf]
6. Markus Voelter on Metamodelling (german) [http://www.voelter.de/data/presentations/metamodelling-paper.pdf]The author also wrote the book on containers and components (J2EE patterns) and there is lots of

Free Information on generative computing (Continued)

good information at his site, even for code generation in embedded control.

7. Gilad Bracha, Generics in the Java Programming language Article on the new Java 1.5 generic types [http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf]
8. Tell, Don't Ask. [http://www.pragmaticprogrammer.com/ppllc/papers/1998_05.html] Sound advice on interface design, class coupling problems etc. Ask yourself: which of those problems are still relevant once a generative approach is used?
9. From David Flanagans site a short glossary on the key terms of Java generics: Java generics glossary [http://www.davidflanagan.com/blog/000027.html#more] As most of you know, I believe in reading code to improve your programming skills. The book from David Flanagan (Java By Examples) is now in its 3rd edition and contains valuable source code which makes you understand Java mechanisms.
10. Dave Thomas of The Pragmatic Programmers, LLC on generative technologies, Ruby etc. With many examples, tips and pitfalls covered. Actually not a paper but an interview. with a very experienced software developer. [http://www.codegeneration.net/tiki-read_article.php?articleId=9]
11. Mark Pollack, generating source code using Java Doc [http://www.javaworld.com/javaworld/jw-08-2000/jw-0818-javadoc_p.html]
12. Jack Harrington, Code-Generation Techniques in Java, An excellent introduction to the subject [http://www.onjava.com/lpt/a/4133]. Explains code and model driven generation and the whole process.

Free Information on generative computing (Continued)

13. Terence Parr, Enforcing Model-View Separation in Template Engines Draft submitted to www2004 [http://www.cs.usfca.edu/~parrt/papers/mvc.templates.pdf] Excellent discussion of template technology and how it interfaces with source code components. Dsicusses general separation between templates and code and which techniques violate this principle. Interestingly the author admits that his own engine is getting always closer to functional language principles even though he claims to be not of the functional camp. What does this say about model2+ architectures which use XSL/XSLT to generate web pages?. This paper will be the base for a discussion on grammar and the whole software technology around "web" programming (JSP, ASP, PHP, XSL and all those template based things)
14. Terence Parr, Building Recognizers by Hand [http://www.antlr.org/book/byhand.pdf]. An easy to read introduction to parsing. Parr shows how to code a top-down recursive parser and takes you through all the steps, starting with mixing scanning and parsing and later separating the steps.
15. The ANTLR Homepage [http://www.antlr.org]. Home of the ANTRL compiler generation tool (former PCCTS). Find excellent literature on language recognizers etc. from Terence Parr (look at his work in progress and download the two pdf files "Building Recognizers by Hand" and "Language".). Take a look at the "getting started" document. And finally browse through the FAQ section if e.g. you need to understand the difference between a parse tree and an AST.
16. Antlr faq on What's the difference between a parse tree and an AST?

Free Information on generative computing (Continued)

[http://www.jguru.com/faq/view.jsp?EID=814505]. Find out why an AST is useful. Combine this with the MDSD rule of doing translations always on the meta-model (i.e. not on the concrete syntax)

17. Excellent slide and code for compiler construction on .NET [http://dotnet.jku.at/courses/CC/]
18. Hook up an interpreter to your application. [http://www.javaworld.com/javaworld/jw-03-2005/jw-0314-scripting.html]
19. Ashley J.S. Mills, ANTLR Tutorial [http://supportweb.cs.bham.ac.uk/documentation/tutorials/docsystem/build/tutorials/antlr/antlrhome.html]. This tutorial to ANTLR is just one of a series of excellent tutorials from the University of Birmingham. Also available are pieces on XDoclet, Ant, Docbook etc.
20. XDoclet homepage Sourceforge location [http://xdoclet.sourceforge.net/using.html]. Nice examples and download.
21. Gregor Kiczales, Andreas Paepke, Xerox Corp. 1996 Open Implementation and Meta-object Protocols [http://www.parc.xerox.com/oi/] On Reflection and Meta-Object-Facility Protocols, Meta-programming, Open Implementation. Still very good if you need to grasp the concepts of meta-classes and what to do with them. Shows how you could implement lots of things which we call design patterns today right in a proper programming language using meta-object protocols. Learn to understand

Free Information on generative computing (Continued)

reification, reflection, introspection, injection etc..

22. Chris Holmes, Andy Evans, A Review of Frame Technology. Covers other generative techniques (AOP etc.) as well. Includes classifaction of approaches to code generation and an extensive bibliography.
23. Sean Mc Grath, Duck modelling in commercial IT systems. Propylon article [http://www.propylon.com/news/ctoarticles/040224_duckmodeling.html]. Talks about the problems with grammar based approaches for data model creation. Expects dynamic languages (python, ruby etc.) to take over long term because of increased business flexibility.
24. Secrets, Hot Spots and Generatlizations: Preparing Students to design software families. By H.Conrad Cunningham et.al, Univ. of Mississippi. Report on a class on software families etc. Shows the problem to teach very abstract concepts to students with little epxerience in large projects. Good bibliography with titles on variability analysis etc. Use a framework approach to teach generalization.
25. Adding rules to applications [http://www-128.ibm.com/developerworks/library/ac-able2/?ca=dnt-67]by Jeff Pilgrim et.al. How to use a rule engine (ABLE) in your application.

Architectures and Dependency Management

1. >Evolving Java-based APIs [http://www.eclipse.org/eclipse/development/java-api-evolution.html] by Jim des Rivieres, OTI. Tells you what you need to do to allow binary compatibility. Even when using generative technologies in many cases you have to connect to existing packages and components.
2. Jorn Bettin Complexity & Dependency Management: Creating an environment for Software Asset Evolution and Software Mass Customization [http://www.softmetaware.com/complexity-and-dependency-management.pdf]
3. Floyd Marinescu, Examining the Validity of Inversion of Control [http://www.theserverside.com/news/thread.tss?thread_id=31676]
4. Martin Fowler, Inversion of Control Containers and the Dependency Injection pattern [http://martinfowler.com/articles/injection.html]
5. Terence Parr, Enforcing Model-View Separation in Template Engines Draft submitted to www2004 [http://www.cs.usfca.edu/~parrt/papers/mvc.templates.pdf] Excellent discussion of template technology and how it interfaces with source code components. Dsicusses general separation between templates and code and which techniques violate this principle. Interestingly the author admits that his own engine is getting always closer to functional language principles even though he claims to be not of the functional camp. What does this say about model2+ architectures which use XSL/XSLT to generate web pages?. This paper will be the base for a discussion on

Architectures and Dependency Management

grammar and the whole software technology around "web" programming (JSP, ASP, PHP, XSL and all those template based things)

Domain Analysis and Software Production Lines

1. Linda Northrop (SEI) Software Production Line Framework. [http://www.sei.cmu.edu/productlines/framework.html]

And some more on Model-Driven Software Development

If you can read german I'd suggest to read the article by Markus Voelter on MDA in the July/August 2004 issue of object sprectrum magazine. It provides an overview to current issues with MDA. Download from Voelter's homepage [http://www.voelter.de/data/articles/MDSD.pdf] or a short introduction to MDSD by Dave Frankel [http://www.bptrends.com/publicationfiles/04%2D04%20COL%20MDSD%20Frankel%20%2D%20Bettin%20%2D%20Cook%2Epdf]. The MDSD homepage [http://www.mdsd.info] has more resources to generative approaches. Softmetaware [http://www.softmetaware.com/links.html] has a nice collection with MDA/MDSD tools e.g. OpenArchitectureWare [http://sourceforge.net/projects/architecturware], an open source framework for MDA/MDSD generation purposes.

1. **Jorn Bettin, Model-Driven Software Development [http://www.softmetaware.com/mdsd-and-isad.pdf] covers MDSD quite extensively and also mentions valuable MDSD desing patterns - the best I've read so far.**
2. **Markus Voelter, Parsen und Verarbeiten Textueller Spezifikationen, explains the differences between concrete textual syntax and the representation in an AST.**
3. **Markus Voelter/Jorn Betting, Patterns for model-driven software development. [http://www.voelter.de/data/pub/MDDPatterns.pdf] How to generate a meta-object protocol layer. How to exploit the model. How to integrate different DSLs etc. Excellent.**

And some more on Model-Driven Software Development

4. Jorn Bettin, Model-driven Software Development Activities.. The process view of an MDSD project [http://www.softmetaware.com/mdsd-process.pdf]
5. Jorn Bettin, Process Implications of Model Driven Software Development Objectspectrum Article [http://www.softmetaware.com/process-implications-of-mdsd.pdf]. This article is also available in German [http://www.sigs.de/publications/os/2004/MDD/bettin_MDD_2004.pdf]
6. b+m AG, Generative Development Process [http://www.architectureware.de/download/b+m_Generative_Development_Process.pdf]. The b+m generator toolkit is now used quite frequently in the software industry.

Books

I always try to have all recommended books available in our library. Also take a look at my special section there where I collect books which should be present at all times.

1. Stephen J.Mellor et.al., Executable UML. Tries to turn UML2.0 into a programming language as well. Adds executable expressions to UML diagrams. Decide for yourself if this is the way to go.
2. Harrington, Code Generation in action, also available as e-book. Covers a lot of different technologies. Uses Ruby and generates Java. http://www.codegeneration.net/cgia/ [http://www.codegeneration.net/cgia/] Provides source code as well. Chapter one (introduction to code generation) and chapter four (building simple generators) are free and required literature for this course.
3. Eisenecker/Cernecki, Generative Programming. The bible of GP. see also http://www.generativeprogramming.com [http://www.generativeprogramming.com]
4. Cleaveland, Program Generators with XML and Java. Not so deep as Eisenecker et.al. but with a nice example of domain analysis.
5. Uwe Schöning, Ideen der Informatik, Grundlegende Modelle und Konzepte. Finally a book for all those which are mentally challanged by theoretical computer science (;-). Uwe Schöning wrote this downscaled version for people who need to understand the concepts but not the mathematical proofs etc. behind them. According to the book more and more computer science faculties also recognize that they have been

Books (Continued)

teaching those advanced concepts for many years completely over the heads of the students by putting them too early in the study plan. In Ulm where Uwe Schöning teaches theoretical computer science they have introduced an introductory course at the undergraduate level and teach the full version now much later. Read Chapter 3: Grammatiken und Automaten for this course.

6. Paul Clements, Linda Northrop, Software Product Lines, Practices and Patterns. The bible of product line engineering.

Thesis work (only my students. I know of several others

Please note that some of those works may not be released for public viewing by participating companies.

1. Markus Wichmann, Untersuchung zum Einsatz metamodell-basierter generativer Programmierung am Beispiel einer Produktdatenbank.
2. Peter Laube, Produktiver Einsatz eines metamodell-basierten Generators für eine Produktdatenbank. Erweiterungen und Korrekturen zur obigen Arbeit.
3. Michael Taege, Automatische Generierung von Business Componenten auf EJB Platformen
4. Maurice Tauscher, Generative Methoden und Erweiterungen im Trend Framework
5. Harry Krämer, Domain Analyse für ein large-scale Software Project.
6. Christoph Birkhold, Towards an improved architecture for Game Mechanis.