Yet another annotated SLEBOK bibliography

Ralf Lämmel

Version 0.003 (2 November 2014)

Abstract

Software Language Engineering (SLE) is a particular view on Software Engineering (SE), which pays specific attention to the many software languages that are used in software development. These are not just programming languages, but also modeling languages, query and transformation languages, schema languages—many of them to be considered domain-specific languages. SLE is concerned with design, implementation, testing, deployment, and evolution of software languages as well as language-based software components.

The purpose of this annotated bibliography is to contribute to the SLE body of knowledge (SLEBOK). The bibliography collects a manageable set of papers that cover many principles and practilities of SLE in an accessible manner. The intension is to favor more fundamental, general papers over specific, highly technical papers. The selection is otherwise not very systematic. The SLE and GTTSE venues were assumed to provide key papers. Yet other venues, such as OOPSLA (SPLASH), ECOOP, and CC as well as special issues on the SLE topic or its vicinity were also considered. Several papers were simply included based on the author's long-term exposition to SLE school of thought. Moreover, several SLE researchers have provided advice on what additional papers to include.

The bibliography could be useful in teaching. In fact, the selection of papers is largely based on what I have covered or wish or could imagine to cover in a relatively advanced SLE course.

Contents

1 Disclaimer

This is unfinished work.

2 Acknowledgment

The following people have made suggestions for inclusion into the bibliography or given more general advice on the project: Anya Helene Bagge, David Lorenz, Richard Paige, Andrei Varanovich, Guido Wachsmuth, Andreas Winter, Vadim Zaytsev.

3 Metamodel of the bibliography

The document is generated from a model. The metamodel is given here informally in terms of how the document looks like. There is one page per entry with data as follows:

Key Descriptor of the paper.

Title Title of the paper.

Citation Bibtex citation for the paper.

Online URL Public access where possible.

Required concepts Assumed background.

Provided concepts Knowledge areas served.

Annotation Description of the paper.

Figure An illustration.

The illustration consists of an annotated figure, which is taken either directly from the paper or assembled. Annotations of papers and their illustrations may also refer to works which are not part of the selection.

4 Papers of the bibliography

4.1 *Koskimies91*

Koskimies91 – Data

Citation

[17]

Title

Object-Orientation in Attribute Grammars

Online URL

http://link.springer.com/chapter/10.1007%2F3-540-54572-7_11

Required concepts

context-free grammar, attribute grammar, object orientation

Provided concepts

object-oriented context-free grammar, object-oriented context-free grammar

Annotation

The attribute grammar formalism is married with the object-oriented paradigm. Arguably, a side effect of this marriage is that the underlying context-free grammar formalism is also married with object orientation, which is interesting in so far that this (early) explanation of the correspondence is exploited nowadays in diverse mapping tools and code generators.

Koskimies91 – Illustration

```
Expression -> Expression AddOp Term | Term
Term -> Term MulOp Factor | Factor
Factor -> number | '(' Expression ')'
AddOp -> '+' | '-'
MulOp -> '*' | '/'
```

From conventional to (strongly) SI-structured context-free grammar

```
Expression -> Sum | Term
Sum -> Expression AddOp Term
Term -> Multiplication | Factor
Multiplication -> Term MulOp Factor
Factor -> Constant | SubExpr
Constant -> number
SubExpr -> '(' Expression ')'
AddOp -> Plus | Minus
MulOp -> Times | Div
Plus -> '+'
Minus -> '-'
Times -> '*'
Div -> '/'
```

The figure shows two grammars for the same expression language taken from the paper. The first grammar is a conventional context-free grammar in terms of style, whereas the second grammar is restructured to be in an explicitly OO-enabled form. That is, an object model with single inheritance could be derived from the second grammar directly.

4.2 *Hughes95*

Hughes95 – Data

Citation
[15]

Title
The Design of a Pretty-printing Library

Online URL
http://link.springer.com/chapter/10.1007%2F3-540-59451-5_3

Required concepts
functional programming

Provided concepts
pretty printing, combinator library

Annotation
Pretty printing is clearly an important form of language processing. This is not the first paper on a declarative and compositional approach to pretty printing; it stands out though with a very accessible presentation explaining the design and implementation of a (Haskell-based) combinator library for pretty printing. This library can be viewed as providing a simple *embedded language* for pretty printing.

Hughes95 – Illustration

Node "foo" (*Node* "baz" *Leaf Leaf*) (*Node* "foobaz" *Leaf Leaf*)

'Ugly' term

```
Node "foo" (Node "baz" Leaf Leaf)
           (Node "foobaz" Leaf Leaf)
```

Pretty term

Pretty printing function

```
pp :: Tree → Doc
pp Leaf = text "Leaf"
pp (Node s l r) = text ("Node " ++ s) <> sep [pp' l, pp' r]
pp' Leaf = pp Leaf
pp' t = text "(" <> pp t <> ")"
```

The figure shows snippets (two Haskell terms and one Haskell function) taken from the paper. The figure illustrates pretty printing for binary trees with a string as info at each fork (i.e., non-leaf) node. The pretty-printed term uses line breaks and indentation for prettiness. The pretty printing function maps trees to documents; see the reference to the *Doc* type. Pretty printer combinators are used; see 'sep' for example.

4.3 *Reynolds98*

***Reynolds98* – Data**

Citation
[28]

Title
Definitional Interpreters for Higher-Order Programming Languages

Online URL
`http://cs.au.dk/~hosc/local/HOSC-11-4-pp363-397.pdf`

Note
This paper originally appeared as [27].

Required concepts
semantics

Provided concepts
interpreter, continuation

Annotation
The paper discusses the use of interpreters as definitions of languages. There are the notions of defining and defined language (similar to what is also called elsewhere meta and object language). The paper analyzes possible differences between the interpreter-based definition and the formal or informal definition. The paper also discusses different styles of interpreter definition, e.g., a less insightful meta-circular interpreter for a higher-order language versus a first-order interpreter for the same defined language. The issue of application-order dependence is analysed and addressed with continuations.

***Reynolds98* – Illustration**

$$
\begin{aligned}
&\mathit{eval} = \lambda(r, e).\\
&\qquad \big(\mathit{const?}(r) \to \mathit{evcon}(r),\\
&\qquad \mathit{var?}(r) \to e(r),\\
&\qquad \mathit{appl?}(r) \to \big(\mathit{eval}(\mathit{opr}(r), e)\big)\big(\mathit{eval}(\mathit{opnd}(r), e)\big),\\
&\qquad \mathit{lambda?}(r) \to \mathit{evlambda}(r, e),\\
&\qquad \mathit{cond?}(r) \to \textbf{if } \mathit{eval}(\mathit{prem}(r), e)\\
&\qquad\qquad \textbf{then } \mathit{eval}(\mathit{conc}(r), e) \textbf{ else } \mathit{eval}(\mathit{altr}(r), e),\\
&\qquad \mathit{letrec?}(r) \to \textbf{letrec } e' =\\
&\qquad\qquad\quad \underline{\lambda x.\ \textbf{if } x = \mathit{dvar}(r) \textbf{ then } \mathit{evlambda}(\mathit{dexp}(r), e') \textbf{ else } e(x)}\\
&\qquad\qquad \textbf{in } \mathit{eval}(\mathit{body}(r), e')\big)\\
&\mathit{evlambda} = \lambda(\ell, e).\ \underline{\lambda a.\ \mathit{eval}\big(\mathit{body}(\ell), \mathit{ext}(\mathit{fp}(\ell), a, e)\big)}\\
&\mathit{ext} = \lambda(z, a, e).\ \underline{\lambda x.\ \textbf{if } x = z \textbf{ then } a \textbf{ else } e(x)}.
\end{aligned}
$$

The figure, taken from the paper, shows a meta-circular interpreter for (in) a simple functional language with lambdas, constants, conditionals, and recursive let.

4.4 *SirerB99*

SirerB99 – Data

Citation

[30]

Title

Using production grammars in software testing

Online URL

http://www.cs.cornell.edu/people/egs/papers/kimera-dsl99.pdf

Required concepts

software engineering

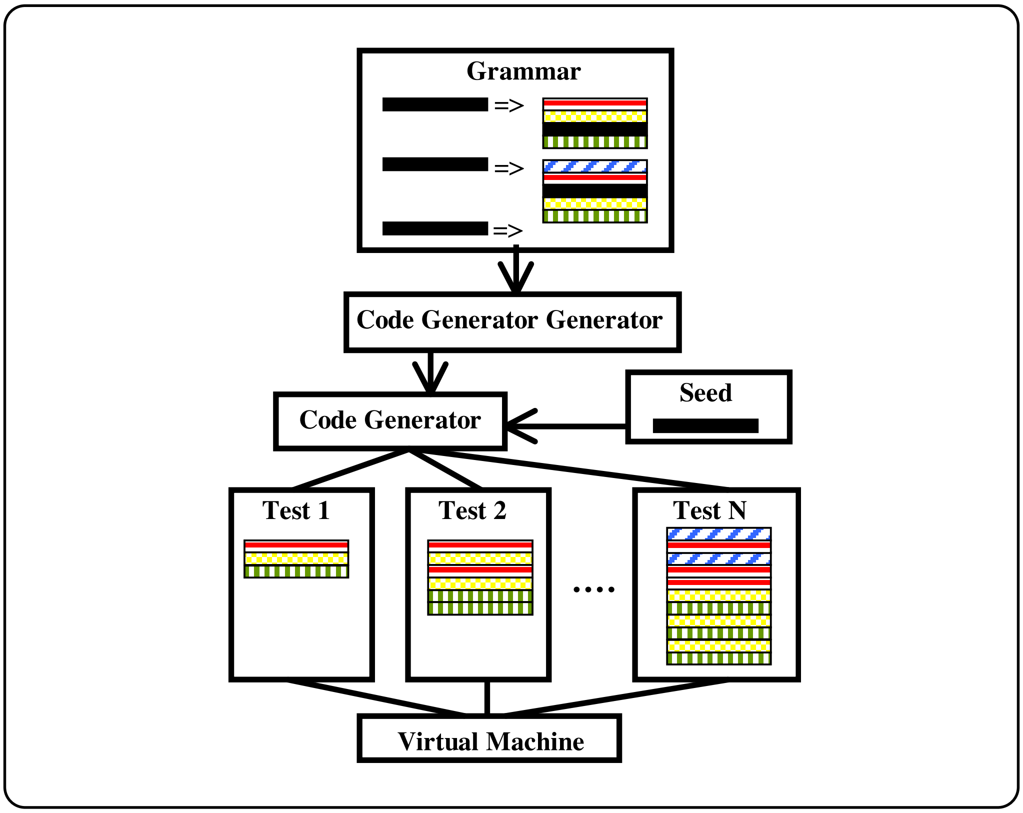
Provided concepts

grammar-based testing

Annotation

The paper shows how grammar-based test-data generation and an accompanying methodology of testing may be highly effective and scalable for testing language-based software, in fact, the Java Virtual Machine. Previous publications on grammar-based testing mainly focused on compiler testing. The paper relies on a domain-specific language *lava* for specifying grammars from which to generate test data – bytecode, in this case. The generated test data is used for stress tesing the JVM verifier and also for comparative tesing of different verifiers.

***SirerB99* – Illustration**



The figure, taken from the paper, carries the following caption (in the paper): The structure of the test generation process. A code-generator-generator parses a production grammar, generates a code-generator, which in turn probabilistically generates test cases based on a seed.

4.5 *Sheard01*

Sheard01 – Data

Citation
[29]

Title
Accomplishments and Research Challenges in Meta-programming

Online URL
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.6848

Required concepts
functional programming, metaprogramming

Provided concepts
taxonomy of metaprogramming, program representation, quasi-quotation, intensional analysis, staged computation, MetaML

Annotation
The paper provides a (possibly outdated) overview over meta-programming with focus on the functional approach towards program representation, code generation, and intensional code analysis. The paper aims to provide a taxonomy of metaprogramming and it discusses problems in metaprogramming in a systematic and illustrative manner. MetaML is shortly introduced as a particular metaprogramming language. The paper brings up research challenges related to, e.g., dependent typing.

Sheard01 – **Illustration**

```
-| fun power_gen m =
     let fun f n x = if n = 0 then <1> else <~x * ~(f (n-1) x)>
     in <let fun power x = ~(f m <x>) in power end> end;
val power_gen = fn  : int -> <int -> int>

-| val power_code = power_gen 3;
```

The figure, taken from the paper, shows the MetaML-based definition of a staged exponentiation function. The `power_gen` function describes the code generation for the n-th power. The `power_code` value holds the code for the 3rd power. The `power_fun` function is the function for said code, which we can ultimately apply.

4.6 *KurtevBA02*

***KurtevBA02* – Data**

Citation
: [18]

Title
: Technological spaces: An initial appraisal

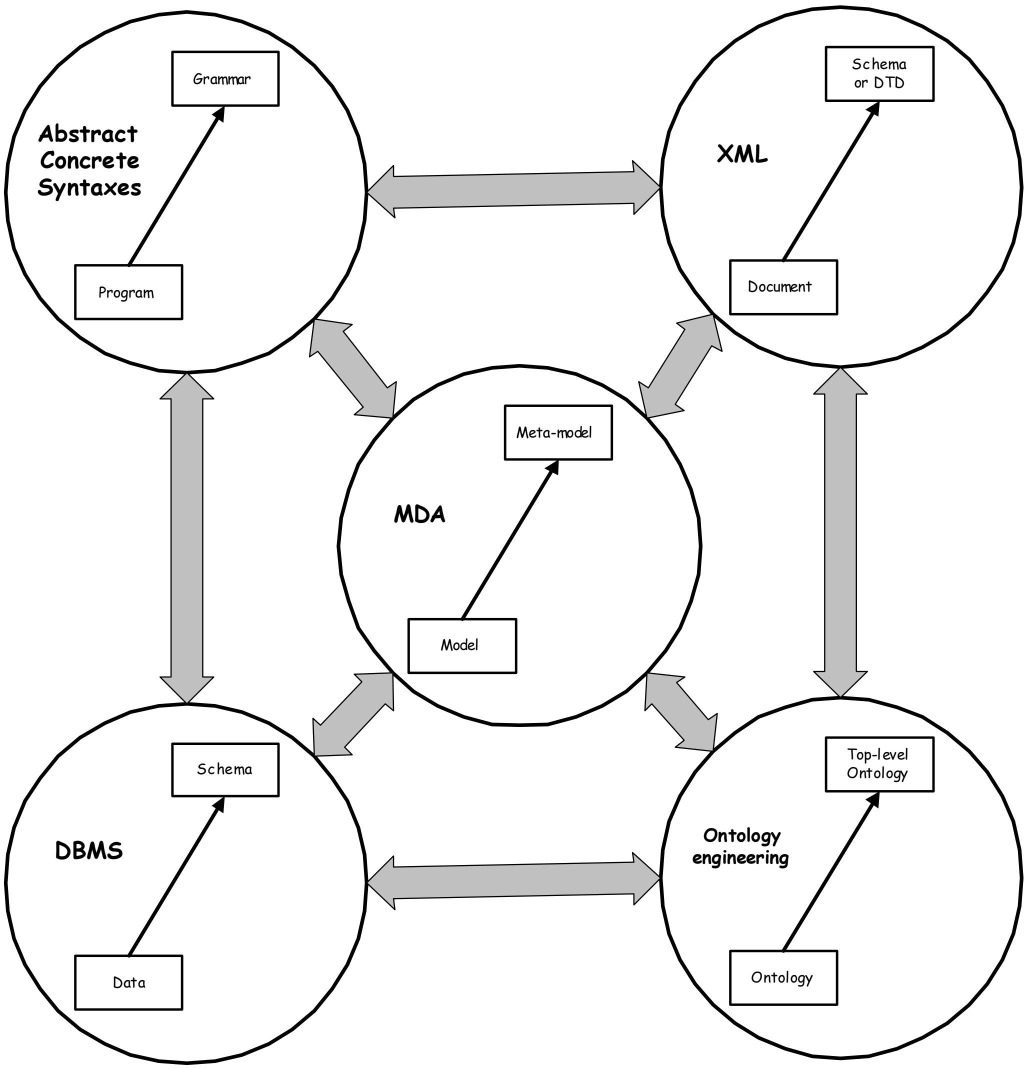
Online URL
: http://eprints.eemcs.utwente.nl/10206/01/0363TechnologicalSpaces.pdf

Required concepts
: model driven engineering

Provided concepts
: technological space

Annotation
: As suggested by the title, this is the record of the introduction of the technological space notion. Several spaces are identified and discussed: abstract/concrete syntaxes, database management systems, XML, ontology engineering, and MDA. The megamodel underlying the spaces is discussed and instantiated for some spaces. The need for and the role of bridges between the spaces is explained. See [3] for another, more recent description of technological spaces.

KurtevBA02 – Illustration



The figure, taken from the paper, shows five technological spaces and bridges between them.

4.7 *Thomas03*

Thomas03 – Data

Citation

[31]

Title

The Impedance Imperative, Tuples + Objects + Infosets = Too Much Stuff!

Online URL

http://www.jot.fm/issues/issue_2003_09/column1.pdf

Required concepts

data programming

Provided concepts

impedance mismatch

Annotation

The paper (a column, in fact) takes a critical look at data programming—specifically in the sense of CRUD (Create, Read, Update, Delete). The discussion covers indexed files, SQL and database access APIs, object-oriented databases, modern wrapping/mapping-based approaches (e.g., object/relational mapping). The column identifies various problems with data programming: diversity of data modeling and CRUD programming options and the practical need to mix them, difficulties of integrating different type systems and data query/transformation languages, proprietary developments, performance issues, and complexity of support technologies. The discussion also briefly touches some contenders that may address some of the problems. The paper may be a good starting point to look for technical publications on the topic.

***Thomas03* – Illustration**

http://en.wikipedia.org/wiki/File:Bermuda_Triangle.png

The figure, taken from Wikipedia, obviously shows the Bermuda triangle. While working with Erik Meijer on [20, 21], I picked up his intuition that data programming (because of the impedance mismatch) is essentially like operating in the Bermuda triangle. That is, data may disappear, if we allow this exaggeration. Just replace Bermuda, Florida, and Puerto Rico by XML, relational databases, and objects. (The idea of a triangle is an understatement because there are, of course, more competitors, e.g., Cobol and ontologies.)

4.8 *Hainaut06*

Hainaut06 – Data

Citation

[11]

Title

The Transformational Approach to Database Engineering

Online URL

http://link.springer.com/chapter/10.1007%2F11877028_4

Required concepts

entity-relationship model, relational database

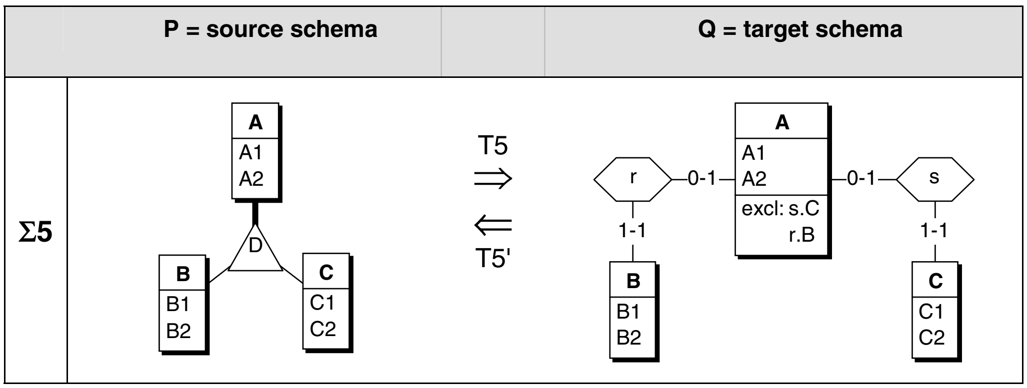
Provided concepts

schema normalization, logical design, schema integration, view derivation, schema equivalence, data conversion, data reverse engineering, schema optimization, data access wapper generation

Annotation

The paper describes fundamental and practical aspects of database transformation techniques. In particular, the notion of transformation is developed in combination with the correctness and reversibility properties.

Hainaut06 **– Illustration**



The figure, taken from the paper, shows a particular transformation rule. Quoting from the paper “Transforming an is-a hierarchy into one-to-one relationship types and conversely. The exclusion constraint (excl:s.C,r.B) states that an A entity cannot be simultaneously linked to a B entity and a C entity. It derives from the disjoint property (D) of the subtypes”

4.9 *HappelS06*

***HappelS06* – Data**

Citation

[12]

Title

Applications of Ontologies in Software Engineering

Online URL

https://km.aifb.kit.edu/ws/swese2006/final/happel_full.pdf

Required concepts

software engineering, ontology

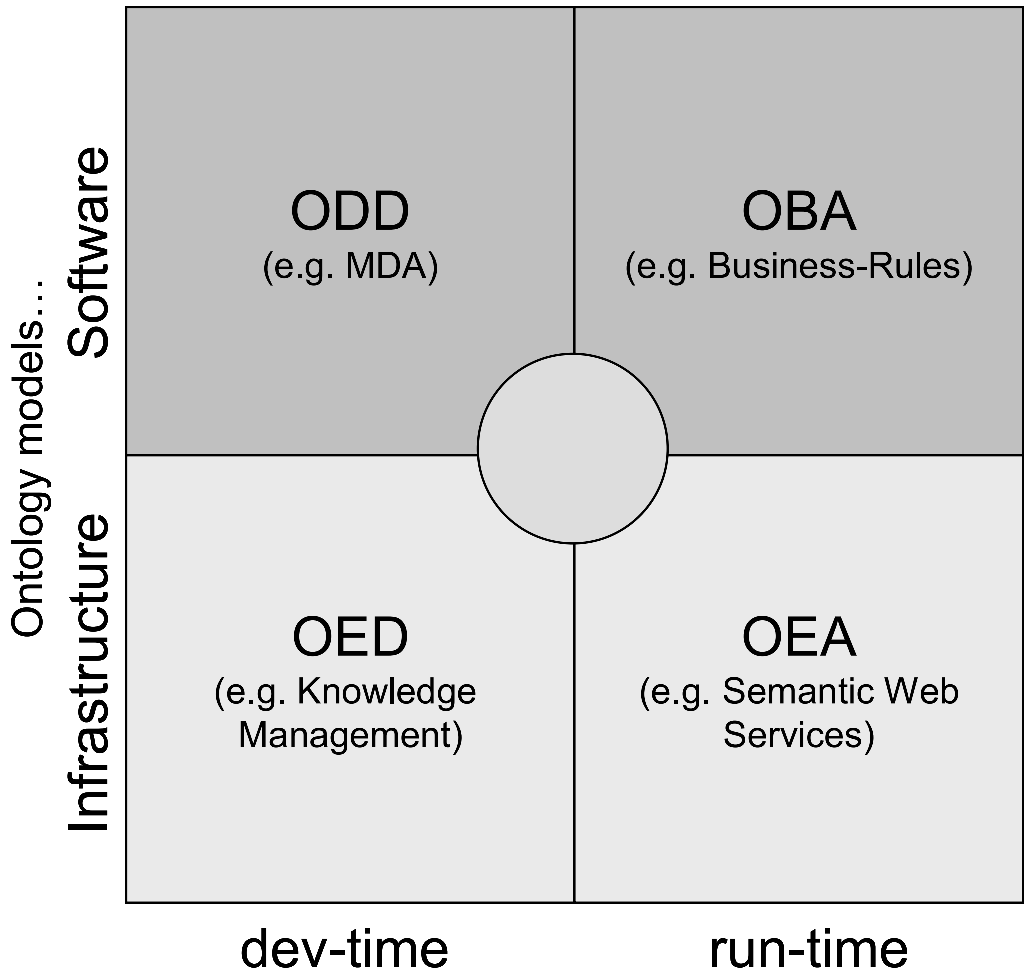
Provided concepts

analysis, design, requirements engineering, component reuse, implementation, modeling, documentation, semantic middleware, semantic web service, maintenance, testing

Annotation

This paper takes an inventory of applications (usage categories) of ontologies in software engineering. It is rich in pointing out the relevance and potential of ontologies in various contexts (e.g., lifecyle phases) in software engineering.

***HappelS06* – Illustration**



The figure is taken from the paper. Different roles of ontologies in the context of software engineering are identified along two axes. Legend of acronyms used: Ontology-driven development (ODD), Ontology-enabled development (OED), Ontology-based architectures (OBA), Ontology-enabled architectures (OEA).

4.10 *Bezivin06*

Bezivin06 – Data

Citation
[3]

Title
Model Driven Engineering: An Emerging Technical Space

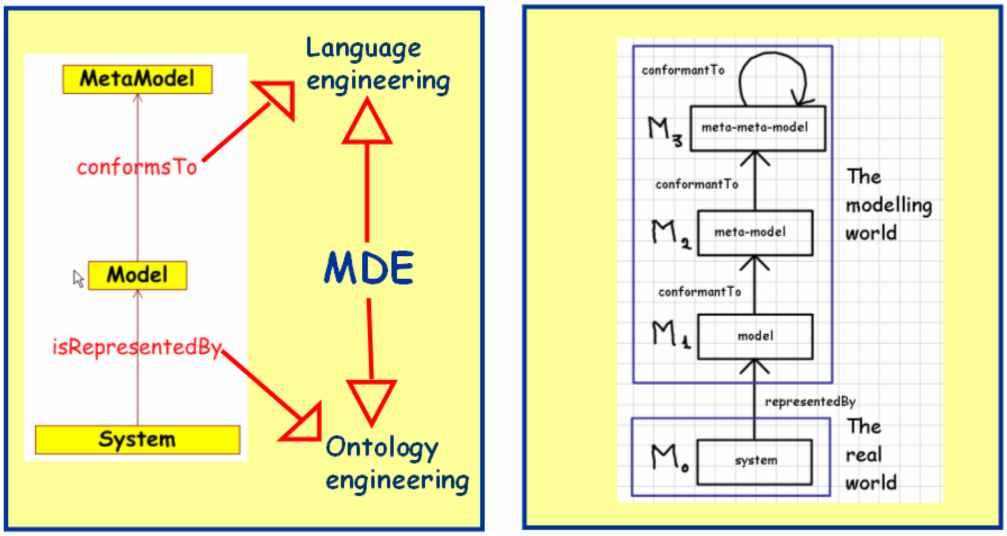
Online URL
`http://link.springer.com/chapter/10.1007%2F11877028_2`

Required concepts
software development

Provided concepts
technological space, model driven engineering, model transformation, metamodeling

Annotation
The paper describes the basic principles and practical characteristics of model driven engineering (MDE). The technological space notion (see also [18]) is used to organize much of the description. In particular, MDE is also compared to other technological spaces. The key notions of metamodeling and model transformation are illustrated. Various technologies and standards are placed in context, e.g., EMF and ATL.

***Bezivin06* – Illustration**



The figure, taken from the conclusion of the paper, on the left, highlights two important relations involved in MDE—the 'isRepresentedBy' relation that some thing (perhaps a model) is represented by a model and the 'conformsTo' relation related to metamodeling. On the right, the progression from real-world entities, through models and metamodels, up to metametamodels is megamodeled.

4.11 *BezivinBFGJKKP06*

BezivinBFGJKKP06 – Data

Citation
: [4]

Title
: A Canonical Scheme for Model Composition

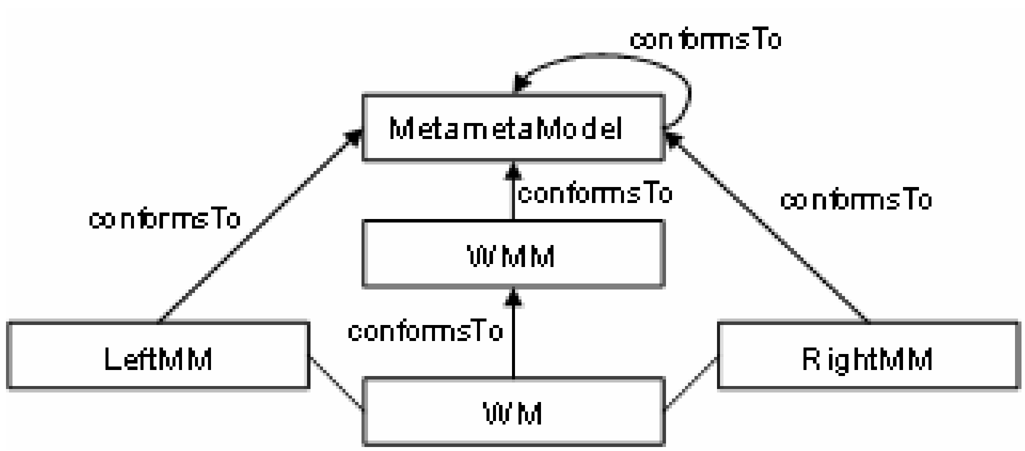
Online URL
: http://ssei.pbworks.com/f/Bezivin.A+Canonical+Scheme+for+Model+Composition.pdf

Required concepts
: metamodeling

Provided concepts
: model composition, model weaving, Glue Generator Tool, Epsilon Merging Language, Atlas Model Weaver

Annotation
: The paper surveys different methods and tools for model composition. It also analyzes composition scenarios and assesses them, for example, in terms of degree of feasible automation. Further, general requirements for model composition tools are postulated and the degree of tool support is considered for existing technologies (at the time of writing).

***BezivinBFGJKKP06* – Illustration**



The figure, taken from the paper' section on the Atlas Model Weaver, shows the schema (say, megamodel) of models and metamodels involved in a weaving (i.e., composition) situation together with the required conformance relationships.

4.12 *BravenboerTV06*

BravenboerTV06 – Data

Citation
[5]

Title
Declarative, formal, and extensible syntax definition for AspectJ

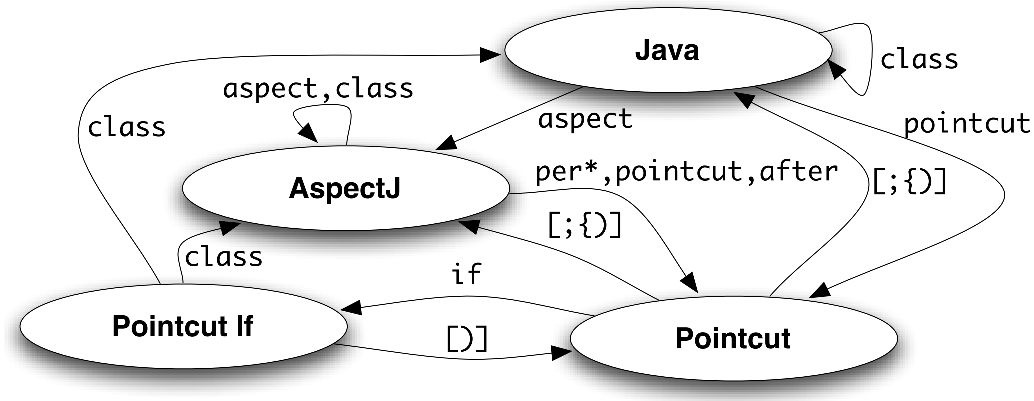
Online URL
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.96.7867

Required concepts
parsing, AspectJ

Provided concepts
scannerless parsing, Generalized LR parsing

Annotation
The paper is a showcase of using generalized LR parsing in the implementation of frontends. While the underlying theory has been published elsewhere, the paper is nevertheless suitable for studying the basics of the method. AspectJ is picked as the target of a case study because parsing AspectJ challenges more convential parsing techniques. We mention, in passing, another recent paper on generalized LR parsing [8].

***BravenboerTV06* – Illustration**



In the figure, taken from the paper, the non-trivial issue of state maintenance in a scanner-based frontend for AspectJ is described at a higher level of abstraction. Depending on the context of parsing (Java, AspectJ, Pointcut), the scanner needs to work differently. In scannerless implementation, such extra effort is not needed.

4.13 *AlvesV09*

AlvesV09 – Data

Citation
[1]

Title
A Case Study in Grammar Engineering

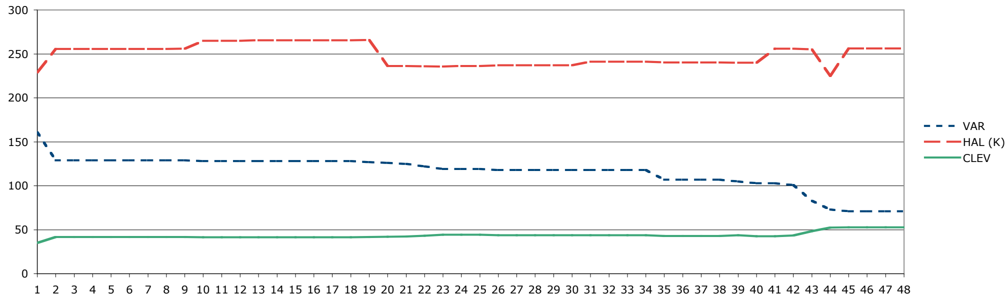
Online URL
http://wiki.di.uminho.pt/twiki/pub/Personal/Tiago/Publications/grammar-eng.pdf

Required concepts
software engineering, parsing, metrics

Provided concepts
grammar engineering, grammar recovery, grammar metrics, grammar testing, grammar versioning

Annotation
The 'Grammarware Agenda' [25] properly established the terms grammar engineering (and grammarware engineering). The present paper presents a study that involves several areas of grammarware engineering. The study is concerned with the development of a VDM-SL grammar for actual parsing from its ISO standard language reference. The study involves grammar transformation (recovery), testing, metrics, and version management.

AlvesV09 – Illustration



The figure, taken from the paper, shows the development of different grammar metrics over time. The timeline is defined by the commits of the grammar, as it was changed over time to complete the recovery process and to otherwise develop the parser. Test coverage also drives this process.

4.14 *Wachsmuth09*

Wachsmuth09 – Data

Citation
[35]

Title
A Formal Way from Text to Code Templates

Online URL
http://link.springer.com/chapter/10.1007%2F978-3-642-00593-0_8

Required concepts
metamodeling, formal semantics

Provided concepts
template instantiation

Annotation
The paper addresses the problem of unsafe template instantiation; see [13] for a description of the problem. Both papers share the overall line of attack: adaptation of a language's metamodel (syntax) so that template-instantiation concepts are made available in a systematic way. The present paper stands out by deploying techniques of programming language theory (operational semantics and type systems) as well as grammar adaptation based on appropriate transformation operators in the tradition of [19].

***Wachsmuth09* – Illustration**

$$
\begin{array}{l}
\textbf{introduce}\ \textit{texpr ttype tvn tn}\\
\textbf{introduce}\ \textit{targ}\ =\ \textit{ttype tvn}\\
\textbf{foreach}\ nt : N\\
\quad \textbf{include}\ nt = \ll \textbf{expand}\ \textit{tn texpr}^{*} \gg\\
\quad \textbf{include}\ nt = \ll \textbf{if}\ \textit{texpr}^{*} \gg\ nt\ \ll \textbf{else} \gg\ nt\ \ll \textbf{endif} \gg\\
\quad \textbf{include}\ \textit{tmpl} = \ll \textbf{define}\ \lfloor nt \rfloor\ \textit{tn targ}^{*} \gg nt\ \ll \textbf{enddef} \gg\\
\quad \textbf{include}\ \textit{tstmt} = nt\\
\quad \textbf{include}\ dn = \lfloor nt \rfloor\\
\textbf{endfor}\\
\textbf{foreach}\ nt : M\\
\quad \textbf{fold}\ nt_M = nt\\
\quad \textbf{include}\ nt_M = \ll \textit{texpr} \gg\\
\quad \textbf{include}\ \textit{tstmt} = nt_M\\
\quad \textbf{include}\ dn = \lfloor nt \rfloor\\
\textbf{endfor}\\
\textbf{foreach}\ nt : L\\
\quad \textbf{fold}\ nt_L = nt\\
\quad \textbf{include}\ nt_L = \ll \textbf{for}\ \textit{tvn}\ \textbf{in}\ \textit{texpr} \gg\ nt\ \ll \textbf{endfor} \gg\\
\quad \textbf{include}\ \textit{tstmt} = nt_L\\
\textbf{endfor}\\
\textbf{introduce}\ \textit{tcoll}\ =\ \textit{tmpl}^{*}
\end{array}
$$

The figure, taken from the paper, shows a transformation script which loops over symbols of the underlying language grammar and includes additional productions in a systematic manner so that concepts for template instantiation are made available.

4.15 *Moody09*

Moody09 – Data

Citation

[23]

Title

The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering

Online URL

http://www.dama.org.au/wp-content/uploads/2013/02/IEEE-TSE-35-5-November-December-2009-The
L.Moody_.pdf

Required concepts

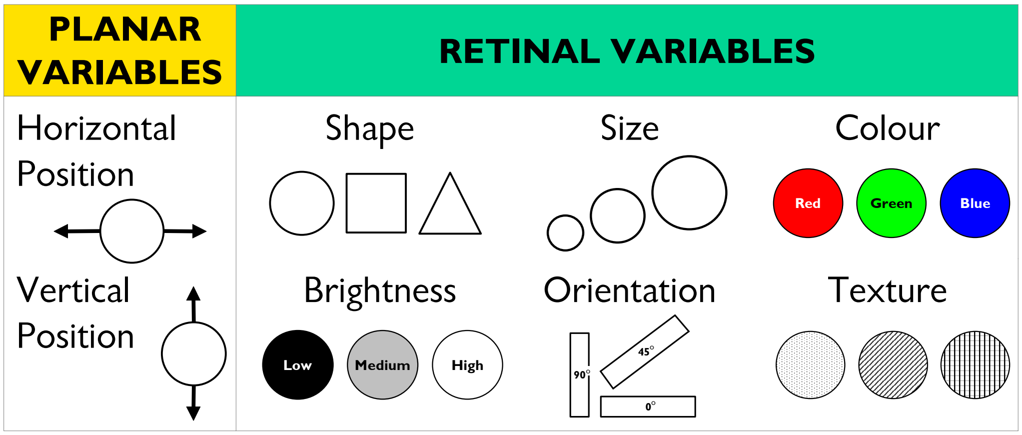
visual language

Provided concepts

Annotation

We quote from the paper: "This paper defines a set of principles for designing cognitively effective visual notations: ones that are optimised for human understanding and problem solving. Together these form a design theory, called the Physics of Notations as it focuses on the physical (perceptual) properties of notations rather than their logical (semantic) properties. The principles were synthesised from theory and empirical evidence from a wide range of fields and rest on an explicit theory of how visual notations communicate. They can be used to evaluate, compare, and improve existing visual notations as well as to construct new ones. The paper identifies serious design flaws in some of the leading SE notations together with practical suggestions for improving them. It also showcases some examples of visual notation design excellence from SE and other fields."

***Moody09* – Illustration**



In the figure, taken from the paper and based on seminal work [2], different visual variables are listed. These variables can thought of as defining a set of primitives, we quote from the paper: "a visual alphabet–for constructing visual notations: graphical symbols can be constructed by specifying particular values for visual variables (e.g., shape = rectangle, colour = green). Notation designers can create an unlimited number of graphical symbols by combining the variables together in different ways."

4.16 *RenggliGN10*

RenggliGN10 – Data

Citation

[26]

Title

Embedding Languages without Breaking Tools

Online URL

http://scg.unibe.ch/archive/papers/Reng10aEmbeddingLanguages.pdf

Required concepts

Smalltalk

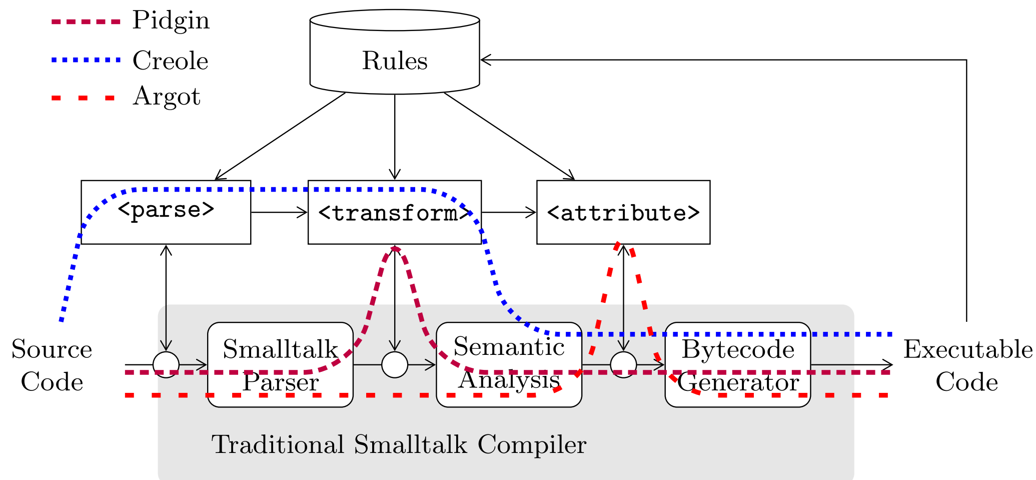
Provided concepts

embedded language

Annotation

The paper describes an embedding approach for the implementation of domain-specific languages (DSLs). Specifically, DSLs are modeled as language extensions of the underlying host language. The approach addresses the challenge of providing the language extensions in a manner that they integrate well with the development tools of the host language. The paper presents the extensible system *Helvetia* which intercepts the compilation pipeline of the Smalltalk host language to seamlessly integrate language extensions. See [32] for another extensive discussion of language embedding.

RenggliGN10 – Illustration



The figure, taken from the paper, shows different interception options for realizing embedded languages in the Smalltalk-based *Helvetia* framework. A pidgin does not require a new parser, but the code needs to be transformed before the semantic analysis. A creole also requires a designated parser. An argot only affects the backend.

4.17 *HeidenreichJSWB09*

HeidenreichJSWB09 – Data

Citation
[13]

Title
Generating Safe Template Languages

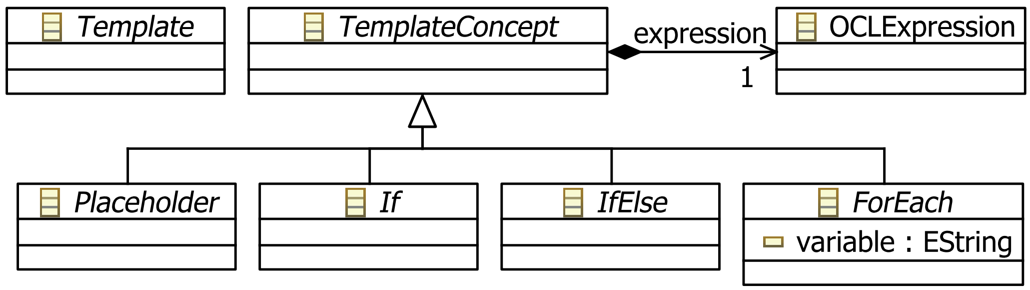
Online URL
https://www.st.cs.uni-saarland.de/~boehme/paper/GPCE09.pdf

Required concepts
metamodeling

Provided concepts
template instantiation

Annotation
The paper addresses the problem of unsafe template instantiation. Such instantiation can be unsafe in the sense that string-level operations are performed at run-time and thus it is not obvious or known at design time whether the described instantiation will actually lead to syntactically correct output eventually. The proposed approach involves the addition of generic template-instantiation concepts to existing language definitions in a generic manner. Thus, template instantiation would only be described in terms of metamodel instantiation, thereby implying syntactically correctness. Similar problems are present in programming languages with a macro system.

HeidenreichJSWB09 – Illustration



The figure, taken from the paper, shows the metamodel for template-instantiation concepts, e.g., a conditional and a loop form. The idea is that these concepts can be specialized for the syntactic categories of the language that is to be extended with template concepts. Whether or not a conditional or a loop is allowed in a certain position also depends on the cardinalities of the model element in the position. There is a generic algorithm to weave the template support in a given metamodel for a language.

4.18 *Cordy11*

Cordy11 – Data

Citation
[7]

Title
Excerpts from the TXL Cookbook

Online URL
http://cs.queensu.ca/~cordy/Papers/JC_TXLCookbook_LNCS.pdf

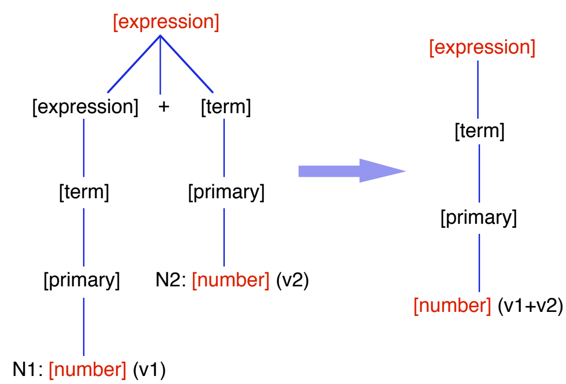
Required concepts
software engineering

Provided concepts
source-code analysis, source-code transformation

Annotation
The paper captures some reusable knowledge of implementating software components for source-code analysis and transformation. While the paper is focused on TXL as the underlying transformation system, the overall approach to knowledge representation would also make sense for other systems. The following classes of problems are considered: parsing, restructuring, optimization, static analysis, and interpretation. The solutions to the problems are described in terms of 'paradigms' such as 'Use sequences, not recursions', 'Preserve comments in output', 'Generate unique identifiers'.

Cordy11 – Illustration

```
rule resolveAdditions
   replace [expression]
      N1[number] + N2[number]
   by
      N1 [+ N2]
end rule
```



The figure, taken from the paper, shows a simple TXL rule and its effect on a parse tree. In fact, a binary addition on constants is evaluated, thereby contributing to expression simplification.

4.19 *ErwigW12a*

ErwigW12a – Data

Citation
: [10]

Title
: Semantics First! - Rethinking the Language Design Process

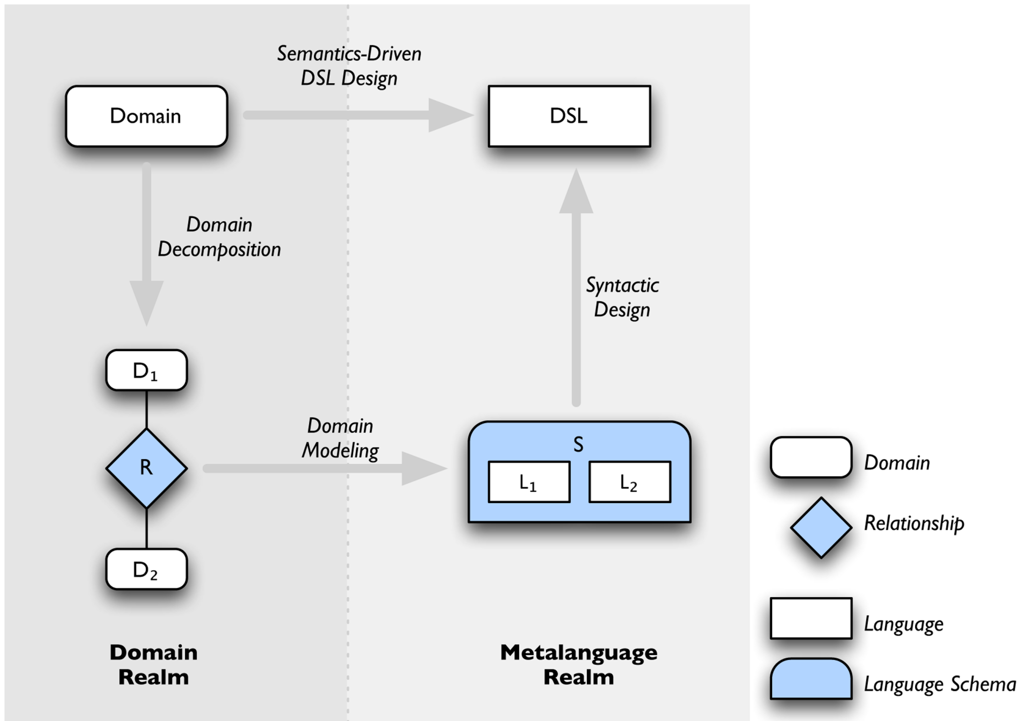
Online URL
: `http://web.engr.oregonstate.edu/~erwig/papers/SemanticsFirst_SLE11.pdf`

Required concepts
: functional programming

Provided concepts
: language design

Annotation
: The paper suggests a semantics-centric approach to language design as opposed to a more syntax-based one. Haskell is used as a metalanguage. General language operators are employed to adapt and grow sophisticated languages out of simple semantics concepts.

ErwigW12a – Illustration



The figure is taken from a book chapter [9] that was derived from the conference paper at hand. The semantics-driven DSL design process is summarized. The idea is that one performs domain decomposition on the semantic side; one associates small languages with domains through domain modeling, and one also performs syntactic design to build a full language from the small languages.

4.20 *CookL11*

CookL11 – Data

Citation
[6]

Title
Tutorial on Online Partial Evaluation

Online URL
http://arxiv.org/abs/1109.0781v1

Required concepts
functional programming

Provided concepts
partial evaluation

Annotation
We quote from the abstract of the paper: "This paper is a short tutorial introduction to online partial evaluation. We show how to write a simple online partial evaluator for a simple, pure, first-order, functional programming language. In particular, we show that the partial evaluator can be derived as a variation on a compositionally defined interpreter. We demonstrate the use of the resulting partial evaluator for program optimization in the context of model-driven development."

CookL11 **– Illustration**

-- Haskell interpreter for state machines

```
run :: State -> Accept -> Transitions -> [Label] -> Bool
run current accept transitions [] = current ‘elem‘ accept
run current accept transitions (l:ls) =
  case lookup l (fromJust (lookup current transitions)) of
    Nothing -> False
    Just next -> run next accept transitions ls
```

-- Desired output from partial evaluation

```
run1 ls = if null ls
            then False
            else if head ls == ’a’
                    then run2 (tail ls)
                    else False

run2 ls = if null ls
            then True
            else if head ls == ’b’
                    then run1 (tail ls)
                    else False
```

The figure, taken from the paper, shows a general, sufficiently interpreter function for the simulation of state machines. When provided with the actual description of the state machine, partial evaluation can speziale the function to specific (efficient) dispatch code that essentially represents the state machine as code.

4.21 *HerrmannsdoerferVW11*

HerrmannsdoerferVW11 – Data

Citation
[14]

Title
An Extensive Catalog of Operators for the Coupled Evolution of Metamodels and Models

Online URL
https://www4.in.tum.de/~herrmama/publications/SLE2010_herrmannsdoerfer_catalog_coupled_operators.pdf

Required concepts
metamodeling

Provided concepts
co-evolution

Annotation
The evolution of a language implies that its metamodel has to evolve. Further, in most cases, existing instances may also need to co-evolve. Operation-nased transformation has matured as an automated method of carrying out metamodel/model coevolution. The present paper collects a catalogue of operations on the grounds of a literature survey and case studies; it also organizes the operations along several dimensions.

***HerrmannsdoerferVW11* – Illustration**

Adaptation	**Semantics-preservation**	**Inverse**
Refactoring		
rename element	preserving modulo variation	rename element
move property	preserving modulo variation	move property
extract class	preserving modulo variation	inline class
inline class	preserving modulo variation	extract class
association to class	preserving modulo variation	class to association
class to association	preserving modulo variation	association to class
Construction		
introduce class	introducing	eliminate class
introduce property	increasing modulo variation	eliminate property
generalise property	increasing	restrict property
pull property	increasing modulo variation	push property
extract superclass	introducing	flatten hierarchy
Destruction		
eliminate class	eliminating	introduce class
eliminate property	decreasing modulo variation	introduce property
restrict property	decreasing	generalise property
push property	decreasing modulo variation	pull property
flatten hierarchy	eliminating	extract superclass

The figure, taken from an earlier paper [34] by one of the authors of the paper at hand, lists some adaptation operators and classifies them in terms of their purpose (refactoring, construction, descruction) and their semantics preservation properties. The paper at hand compiles a much more extensive catalogue and engages in a richer classification.

4.22 *MullerFBC12*

MullerFBC12 – Data

Citation

[24]

Title

Modeling modeling modeling

Online URL

http://people.rennes.inria.fr/Benoit.Baudry/wp-publications/muller2010/

Required concepts

modeling, model driven engineering

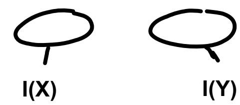
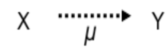
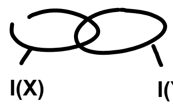
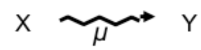
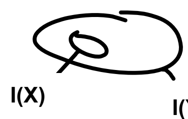
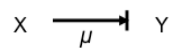
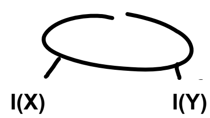
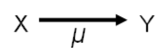
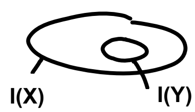
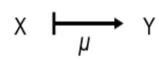
Provided concepts

representation, theory of modeling

Annotation

The paper works towards a theory of modeling. There is a focus on the representation relation that is so central to modeling (in the sense that one thing *represents* another thing). In fact, different (canonical) kinds of representation relations are identified and organized in a corresponding metamodel. This foundational work is well positioned in the context of previous work on the foundations of modeling (and metamodeling).

***MullerFBC12* – Illustration**

Kind	Intention	Description	Notation
`different`		X and Y have totally different intentions. This usually denotes a shift in viewpoints.	
`share`		X and Y share some intention. X and Y can be partially represented by each other. The representation is both partial and extended.	
`sub`		The intention of X is a part of Y's intention. Everything which holds for X makes sense in the context of Y. Y can be partially represented by X.	
`same`		X and Y share the same intention. They can represent each other. This usually denotes a shift in linguistic conformance.	
`super`		X covers the intention of Y; X can represent Y, but X has additional properties. It is an extended representation.	

The figure, taken from the paper, shows variations on the μ relation. These variations are essentially based on differences with regard to the *intention* of things. Quoting from the paper: "The intention of a thing thus represents the reason why someone would be using that thing, in which context, and what are the expectations vs. that thing. It should be seen as a mixture of requirements, behavior, properties, and constraints, either satisfied or maintained by the thing."

4.23 *JezequelCDGR12*

JezequelCDGR12 – Data

Citation

[16]

Title

Bridging the chasm between MDE and the world of compilation

Online URL

http://link.springer.com/article/10.1007%2Fs10270-012-0266-8

Required concepts

modelware, grammarware, compilation, MDE

Provided concepts

cross-fertilization

Annotation

The paper attempts a deeper comparison of the technological spaces [18] of modelware (MDE) and grammarware (specifically compiler construction). We quote: "To address the growing complexity of soft- ware systems, Model-Driven Engineering (MDE) leverages Domain Specific Languages (DSL) to define abstract models of systems and automated methods to process them. Mean- while, compiler technology mostly concentrates on advanced techniques and tools for program transformation. For this, it has developed complex analyses and transformations (from lexical and syntaxic to semantic analyses, down to platform- specific optimizations). These two communities appear today quite complementary and are starting to meet again in the Software Language Engineering (SLE) field."

JezequelCDGR12 – Illustration

MDE shortcomings	Compilation solutions
Increasing need for parsing tools due to increase in number of DSLs	Efficient parsing and parser generators
Platform Description Model	Capture of platform specific knowledge through dedicated descriptions
Tool efficiency and scalability	Sophisticated algorithms and heuristics
Increasingly complex model transformations	Know-how in sophisticated algorithms development and program transformation paradigms

Compilation shortcomings	MDE solutions
IRs contain more and more complex information, more and more complex IR processings,	Complex data representation and Separation of Concerns
Maintainability	Homogeneization of software through generative approaches
Documentation	Metamodels as documentation
Error-prone and time consuming development tasks	Automation through metatools and metatooling
Ordering of the compilation pass	Design-by-Contract to limit possible choices to meaningful choices

The figure, taken from the paper, shows how the two spaces may mutally benefit from each other: shortcomings of one space may be addressed by adopting solutions known in the other space.

4.24 *VolterSBK14*

VolterSBK14 – Data

Citation
: [33]

Title
: Towards User-Friendly Projectional Editors

Online URL
: http://mbeddr.com/files/projectionalEditing-sle2014.pdf

Required concepts
: parsing, IDE

Provided concepts
: projectional editing

Annotation
: The paper analyzes usability issues with projectional editing, but it actually may also serve a good reference for a definition and characterization of projectional editing as such. The discussion demonstrates key characteristics of projectional editing, e.g., the combination of notional styles and the use of composition techniques. We mention, in passing, to another recent paper on projectional editing [22]

***VolterSBK14* – Illustration**

The figure, taken from the paper, illustrates the difference between parser-based editors (ParEs) and projectional editors (ProjEs). We quote from the paper: "In ParEs (left), users see and modify the concrete syntax. A parser constructs the AST. In ProjEs, users see and interact with the concrete syntax, but changes directly affect the AST. The concrete syntax is projected from the changing AST."

References

[1] Tiago L. Alves and Joost Visser. A Case Study in Grammar Engineering. In *Software Language Engineering, First International Conference, SLE 2008, Revised Selected Papers*, volume 5452 of *LNCS*, pages 285–304. Springer, 2009.

[2] J. Bertin. *Semiology of Graphics: Diagrams, Networks, Maps*. Madison, Wisconsin, USA: University of Wisconsin Press, 1983.

[3] Jean Bézivin. Model Driven Engineering: An Emerging Technical Space. In *Generative and Transformational Techniques in Software Engineering, International Summer School, GTTSE 2005, Revised Papers*, volume 4143 of *LNCS*, pages 36–64. Springer, 2006.

[4] Jean Bézivin, Salim Bouzitouna, Marcos Didonet Del Fabro, Marie-Pierre Gervais, Frédéric Jouault, Dimitrios S. Kolovos, Ivan Kurtev, and Richard F. Paige. A Canonical Scheme for Model Composition. In *Model Driven Architecture - Foundations and Applications, Second European Conference, ECMDA-FA 2006, Proceedings*, volume 4066 of *LNCS*, pages 346–360. Springer, 2006.

[5] Martin Bravenboer, Éric Tanter, and Eelco Visser. Declarative, formal, and extensible syntax definition for AspectJ. In *Proceedings of the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006*, pages 209–228. ACM, 2006.

[6] William R. Cook and Ralf Lämmel. Tutorial on Online Partial Evaluation. In *Proceedings IFIP Working Conference on Domain-Specific Languages, DSL 2011*, volume 66 of *EPTCS*, pages 168–180, 2011.

[7] James R. Cordy. Excerpts from the TXL Cookbook. In *Generative and Transformational Techniques in Software Engineering III - International Summer School, GTTSE 2009, Revised Papers*, volume 6491 of *LNCS*, pages 27–91. Springer, 2011.

[8] Giorgios Economopoulos, Paul Klint, and Jurgen J. Vinju. Faster Scannerless GLR Parsing. In *Compiler Construction, 18th International Conference, CC 2009, Proceedings*, volume 5501 of *LNCS*, pages 126–141. Springer, 2009.

[9] Martin Erwig and Eric Walkingshaw. Semantics-Driven DSL Design. In Marjan Mernik, editor, *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*, pages 56–80. IGI Global, 2012.

[10] Martin Erwig and Eric Walkingshaw. Semantics First! - Rethinking the Language Design Process. In *Software Language Engineering - 4th International Conference, SLE 2011, Revised Selected Papers*, volume 6940 of *LNCS*, pages 243–262. Springer, 2012.

[11] Jean-Luc Hainaut. The Transformational Approach to Database Engineering. In *Generative and Transformational Techniques in Software Engineering, International Summer School, GTTSE 2005, Revised Papers*, volume 4143 of *LNCS*, pages 95–143. Springer, 2006.

[12] Hans-Jörg Happel and Stefan Seedorf. Applications of Ontologies in Software Engineering. In *Proceedings of International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006)*, 2006. 14 pages.

[13] Florian Heidenreich, Jendrik Johannes, Mirko Seifert, Christian Wende, and Marcel Böhme. Generating safe template languages. In *Generative Programming and Component Engineering, 8th International Conference, GPCE 2009, Proceedings*, pages 99–108. ACM, 2009.

[14] Markus Herrmannsdoerfer, Sander Vermolen, and Guido Wachsmuth. An Extensive Catalog of Operators for the Coupled Evolution of Metamodels and Models. In *Software Language Engineering - Third International Conference, SLE 2010, Revised Selected Papers*, volume 6563 of *LNCS*, pages 163–182. Springer, 2011.

[15] John Hughes. The Design of a Pretty-printing Library. In *Advanced Functional Programming, First International Spring School on Advanced Functional Programming Techniques, AFP 1995, Tutorial Text*, volume 925 of *LNCS*, pages 53–96. Springer, 1995.

[16] Jean-Marc Jézéquel, Benoît Combemale, Steven Derrien, Clement Guy, and Sanjay V. Rajopadhye. Bridging the chasm between MDE and the world of compilation. *Software and System Modeling*, 11(4):581–597, 2012.

[17] Kai Koskimies. Object-Orientation in Attribute Grammars. In *Attribute Grammars, Applications and Systems, International Summer School SAGA 1991, Proceedings*, volume 545 of *LNCS*, pages 297–329. Springer, 1991.

[18] Ivan Kurtev, Jean Bézivin, and Mehmet Aksit. Technological spaces: An initial appraisal. In *Proceedings of CoopIS, DOA'2002 Federated Conferences, Industrial track*, 2002. 6 pages.

[19] Ralf Lämmel. Grammar Adaptation. In *Proceedings of FME 2001 (International Symposium of Formal Methods Europe)*, volume 2021 of *LNCS*, pages 550–570. Springer, 2001.

[20] Ralf Lämmel and Erik Meijer. Mappings Make Data Processing Go 'Round. In *Generative and Transformational Techniques in Software Engineering, International Summer School, GTTSE 2005, Revised Papers*, volume 4143 of *LNCS*, pages 169–218. Springer, 2006.

[21] Ralf Lämmel and Erik Meijer. Revealing the X/O Impedance Mismatch - (Changing Lead into Gold). In *Datatype-Generic Programming - International Spring School, SSDGP 2006, Revised Lectures*, volume 4719 of *LNCS*, pages 285–367. Springer, 2007.

[22] David H. Lorenz and Boaz Rosenan. Cedalion: a language for language oriented programming. In *Proceedings of the 26th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2011, part of SPLASH 2011*, pages 733–752. ACM, 2011.

[23] Daniel L. Moody. The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Trans. Software Eng.*, 35(6):756–779, 2009.

[24] Pierre-Alain Muller, Frédéric Fondement, Benoit Baudry, and Benoît Combemale. Modeling modeling modeling. *Software and System Modeling*, 11(3):347–359, 2012.

[25] Paul Klint and Ralf Lämmel and Chris Verhoef. Toward an engineering discipline for grammarware. *ACM Trans. Softw. Eng. Methodol.*, 14(3):331–380, 2005.

[26] Lukas Renggli, Tudor Gîrba, and Oscar Nierstrasz. Embedding Languages without Breaking Tools. In *ECOOP 2010 - Object-Oriented Programming, 24th European Conference, Proceedings*, volume 6183 of *LNCS*, pages 380–404. Springer, 2010.

[27] John C. Reynolds. Definitional interpreters for higher-order programming languages. In *Proceedings of the ACM annual conference - Volume 2*, ACM '72, pages 717–740. ACM, 1972.

[28] John C. Reynolds. Definitional Interpreters for Higher-Order Programming Languages. *Higher-Order and Symbolic Computation*, 11(4):363–397, 1998.

[29] Tim Sheard. Accomplishments and Research Challenges in Metaprogramming. In *Semantics, Applications, and Implementation of Program Generation, Second International Workshop, SAIG 2001, Proceedings*, volume 2196 of *LNCS*, pages 2–44. Springer, 2001.

[30] Emin Gün Sirer and Brian N. Bershad. Using production grammars in software testing. In *Proceedings of the Second Conference on Domain-Specific Languages (DSL 1999)*, pages 1–13. ACM, 1999.

[31] Dave A. Thomas. The Impedance Imperative - Tuples + Objects + Infosets = Too Much Stuff! *Journal of Object Technology*, 2(5):7–12, 2003.

[32] Laurence Tratt. Domain specific language implementation via compile-time meta-programming. *ACM Trans. Program. Lang. Syst.*, 30(6), 2008.

[33] Markus Völter, Janet Siegmund, Thorsten Berger, and Bernd Kolb. Towards User-Friendly Projectional Editors. In *Software Language Engineering - 7th International Conference, SLE 2014, Proceedings*, volume 8706 of *LNCS*, pages 41–61. Springer, 2014.

[34] Guido Wachsmuth. Metamodel Adaptation and Model Co-adaptation. In *ECOOP 2007 - Object-Oriented Programming, 21st European Conference, Proceedings*, volume 4609 of *LNCS*, pages 600–624. Springer, 2007.

[35] Guido Wachsmuth. A Formal Way from Text to Code Templates. In *Fundamental Approaches to Software Engineering, 12th International Conference, FASE 2009, Proceedings*, volume 5503 of *LNCS*, pages 109–123. Springer, 2009.