23. Languages for Link-Tree Querying, Flat Analysis Transformations and Rewriting

Link-TreeWare for Exchange Formats

Prof. Dr. U. Aßmann

Technische Universität Dresden

Institut für Software- und Multimediatechnik

http://st.inf.tu-dresden.de/teaching/most

Version 21-1.1, 04.12.21

1) DDL for Link Trees
2) Analysis with query languages
3) Transformation languages for Link Trees
    1) Xcerpt
    2) Context-free Transformations
    3) Context-sensitive Transformations
4) Consequences for MDSD applications

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Obligatory Literature

- Sebastian Schaffert, François Bry. Querying the Web Reconsidered: A Practical Introduction to Xcerpt (2004). In Proc. Extreme Markup Languages.
  - http://www.pms.informatik.uni-muenchen.de/publikationen/PMS-FB/PMS-FB-2004-7.pdf
  - http://www.rewerse.net/publications/download/REWERSE-RP-2006-069.pdf
  -
- Tool: https://sourceforge.net/projects/xcerpt/
- A Gentle Introduction to Xcerpt, a Rule-based Query and Transformation Language for XML. Francois Bry and Sebastian Schaffert. Institute for Computer Science, University of Munich.
  - http://www.pms.informatik.uni-muenchen.de/
  - Http://ceur-ws.org/Vol-60/bry_schaffert.pdf
- Informative:
  - Radim Baca, Michal Krátký, Irena Holubová, Martin Necaský, Tomáš Skopal, Martin Svoboda, and Sherif Sakr. 2017. Structural XML Query Processing. ACM Computing Surveys. 50, 5, Article 64 (September 2017), 41 pages. https://doi.org/10.1145/3095798
  - https://cswr.github.io/JsonSchema/

Literature on RAG

Obligatory

- [Hedin11] An Introductory Tutorial on JastAdd Attribute Grammars. In Generative and Transformational Techniques in Software Engineering III, 6491:166-200. Lecture Notes in Computer Science. Springer Berlin / Heidelberg.
  - https://link.springer.com/chapter/10.1007/978-3-642-18023-1_4
  - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.187.5911&rep=rep1&type=pdf
- [Bürger+11] Bürger, Christoff, Sven Karol, Christian Wende, und Uwe Aßmann. 2011. Reference Attribute Grammars for Metamodel Semantics. In Software Language Engineering. Springer Berlin / Heidelberg.
- [Heidenreich+12] Heidenreich, Florian, Jendrik Johannes, Sven Karol, Mirko Seifert, und Christian Wende. 2012. „Model-based Language Engineering with EMFText“. In Generative and Transformational Techniques in Software Engineering, 7680:322ff. Lecture Notes in Computer Science. Springer Berlin / Heidelberg.

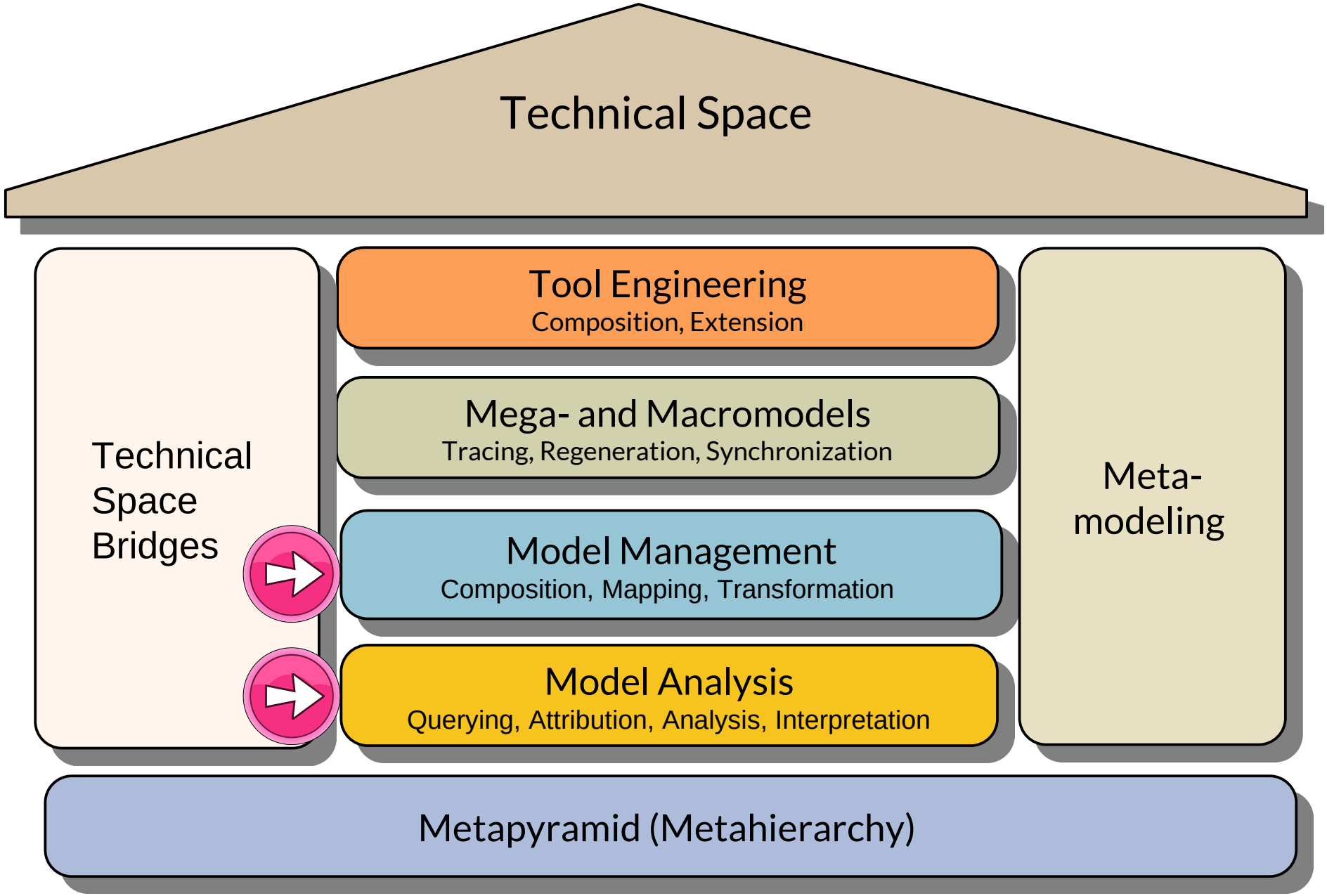
Informative

- [Hedin00] Hedin, Görel. 2000. Reference Attributed Grammars. Informatica (Slovenia) 24, Nr. 3: 301–317.
- [Boyland05] Boyland, John T. 2005. Remote attribute grammars. Journal of the ACM 52, Nr. 4: 627–687.
- [Knuth68] Knuth, D. E. Semantics of context-free languages. Theory of Computing Systems 2, Nr. 2: 127–145.
- [Vogt+89] Vogt, Harald H, Doaitse Swierstra, und Matthijs F Kuiper. 1989. Higher Order Attribute Grammars. In PLDI '89, 131–145. ACM. --- For code generation and template expansion.
- [Ekman06] Ekman, Torbjörn. 2006. Extensible Compiler Construction. University of Lund.
- [Kühnemann+97] Kühnemann, Armin, und Heiko Vogler. Attributgrammatiken -- Eine grundlegende Einführung. Braunschweig/Wiesbaden: Vieweg. 1997.

RAGs, Template Expansion, Invasive Composition

- [Bürger+10] Bürger, Christoff, Sven Karol, und Christian Wende. 2010. Applying attribute grammars for metamodel semantics. In Proceedings of the International Workshop on Formalization of Modeling Languages, 1:1–1:5. FML '10. New York, NY, USA: ACM.
- Sven Karol. Well-Formed and Scalable Invasive Software Composition. PhD thesis, Technische Universität Dresden, May 2015.
  - http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-170162
  - Demonstrator Tool SkAT https://bitbucket.org/svenkarol/skat/wiki/Home.
- [Bürger15] Christoff Bürger. Reference attribute grammar controlled graph rewriting: motivation and overview. In Richard F. Paige, Davide Di Ruscio, and Markus Völter, editors, Software Language Engineering (SLE), pages 89-100. ACM, 2015. http://dl.acm.org/citation.cfm?id=2814251

Q10: The House of a Technical Space

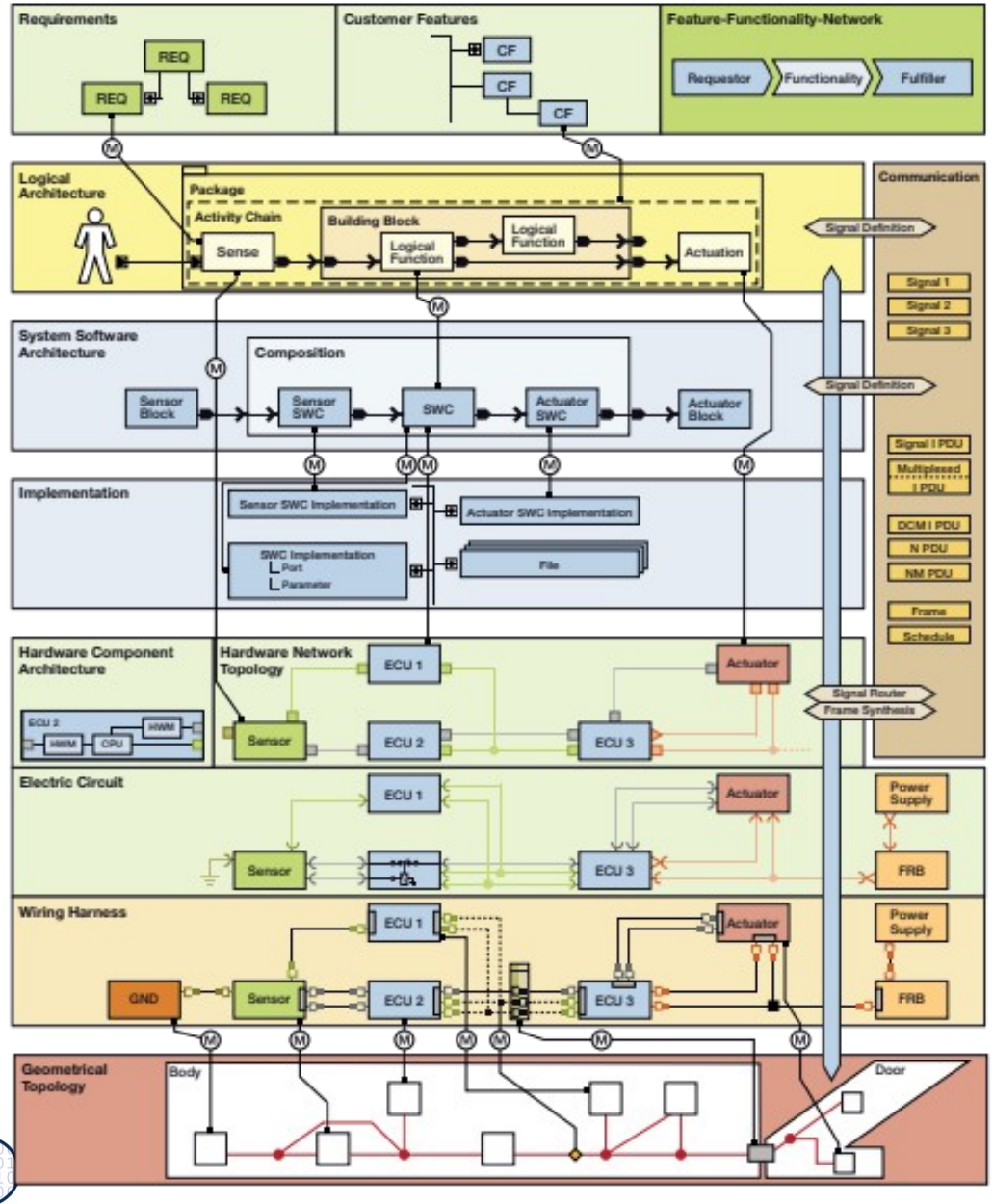


Why Do We Need LinkTreeWare in MDSD Tools?

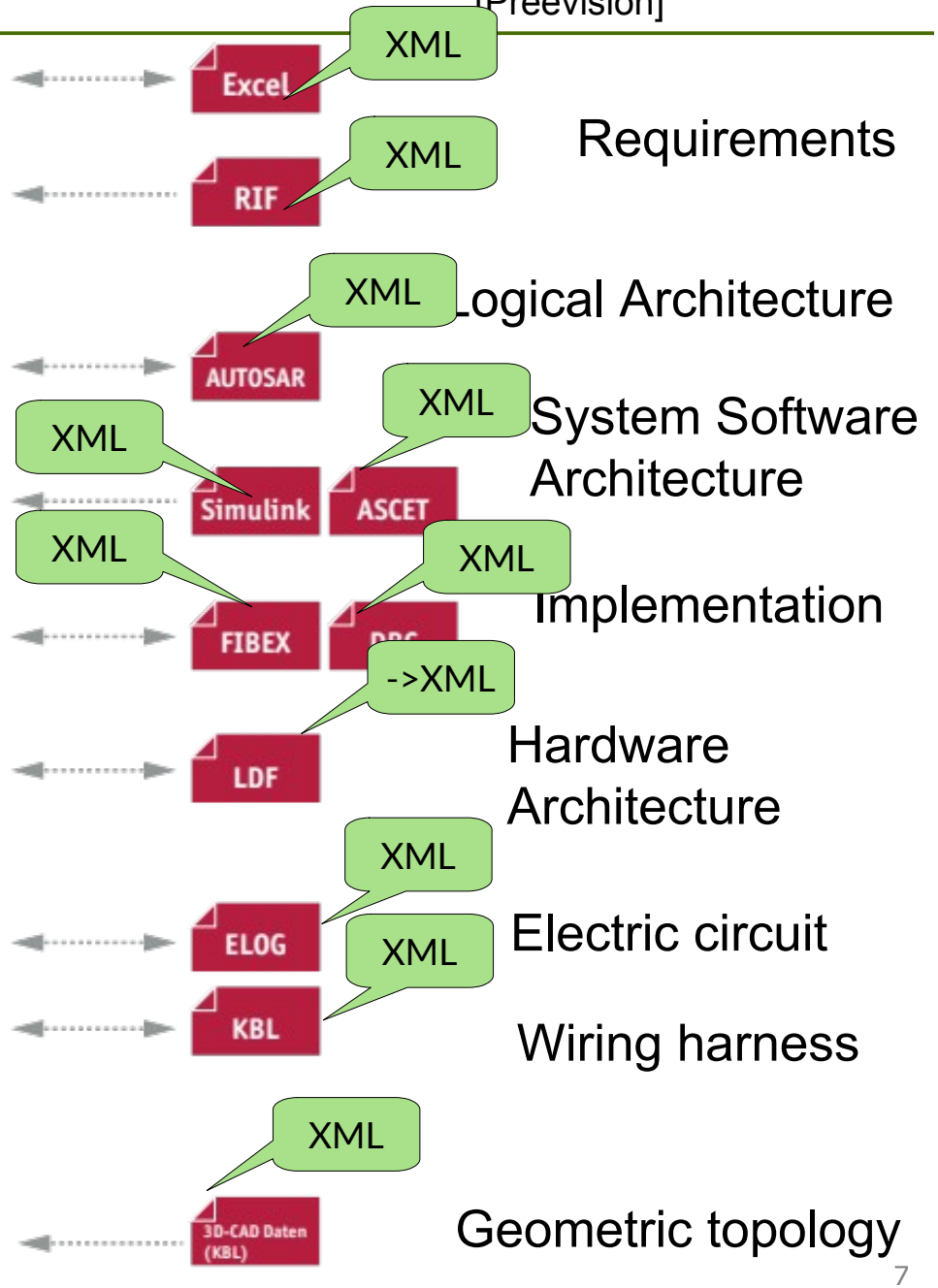
- A ***link tree*** is a tree with secondary cross-links. We call the primary tree the ***skeleton tree***, and the secondary links form the ***overlay graph***.
  - Link trees can be pretty-printed and re-read with linear or quasi-linear speed.
  - Therefore, they form ideal exchange formats for
    - Data exchange between Tools
    - Technical space bridges
- Every MDSD tool uses current Treeware formats, such as JSON, XML, Xcerpt DT, EMF.

  Examples:
- OfBiz of Apache https://ofbiz.apache.org/ is a framework with about 20 XML dialects for business software
  - https://ofbiz.apache.org/business-users.html#UsrModules
    - General Double Ledger
    - Financial Reporting
    - Order Management
  - https://cwiki.apache.org/confluence/display/OFBIZ/Documentation#Documentation-GeneralXMLDefinitions
- PreeVision has many XML formats

Remember the Big Example: Car Design with PREEVision (Vector): Interoperability with XML Link Trees



[Preevision]



23.1 Data Definition Languages (DDL) for Link Trees

The basic layer of LinkTreeWare M2

.. XML, JSON, YAML, ...

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Link TreeWare

- An **attributed link tree (ALT)** is a link tree with attributes, overlay-graph representations
- A ***map tree (hierarchic map, associative tree)*** is a tree of maps, i.e., a tree of associative arrays.
- A ***map link tree*** is a map tree with (secondary) cross-links.
- Examples of map link trees:
  - A JSON document
  - An XML document
  - An EMF tree
- Specification languages (DDL) for link trees *extend RTG* with names (anchors) and links to these names

23.1.1 JSON Associative Map Trees

- http://json-schema.org/examples.html
- Michael Droettboom, et al. Understanding JSON Schema, Release 1.0. Space Telescope Science Institute, October 12, 2015
  - http://spacetelescope.github.io/understanding-json-schema/UnderstandingJSONSchema.pdf

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

JSON

- JSON is a family of link map tree languages, mainly for syntax trees, NOT for markup
- JSON is block-structured with angle brackets; list and set constructors, but no type constructores for nodes
- JSON often used as exchange format
- Metalanguage JSONSchema is a lifted DDL and similar to a RTG with maps
- Example:

```
// JSON Schema of objects with names and ages
{
      "title": "Example Schema",
      "type": "object",
      "properties": {
            "firstName": {  "type": "string"    },
            "lastName": {   "type": "string"    },
            "age": {
                  "description": "Age in years",
                  "type": "integer",
                  "minimum": 0
            }
      },
      "required": ["firstName", "lastName"]
}
```

```
// JSON instance of objects with names and ages
{
      "object": { "firstName": „Uwe“,
                  "lastName“: "Aßmann",
                  "age": { 27 }
       }
      "object": { "firstName": „John“,
                  "lastName“: "Smith"
       }

}
```

```
{  "$schema": "http://json-schema.org/draft-04/schema#",
   "title": "Product set",
   "type": "array",
   "items": {
      "title": "Product",
      "type": "object",
      "properties": {
         "id": { "description": "The unique identifier for
             a product",
            "type": "number"
         },
         "name": {  "type": "string"  },
         "price": { "type": "number",
            "minimum": 0,
            "exclusiveMinimum": true
         },
         "tags": {   "type": "array",
            "items": {  "type": "string"      },
            "minItems": 1,
            "uniqueItems": true
         },
         "dimensions": {  "type": "object",
            "properties": { "length": {"type": "number"},
               "width": {"type": "number"},
               "height": {"type": "number"}
            },
            "required": ["length", "width", "height"]
         },
         "warehouseLocation": {
            "description": "Coordinates of the warehouse
               with the product",
            "$ref": "http://json-schema.org/geo"
         }
      },
      "required": ["id", "name", "price"]
   }
}
```

```
// instance
[
   {
      "id": 2,
      "name": "An ice sculpture",
      "price": 12.50,
      "tags": ["cold", "ice"],
      "dimensions": {
         "length": 7.0,
         "width": 12.0,
         "height": 9.5
      },
      "warehouseLocation": {
         "latitude": -78.75,
         "longitude": 20.4
      }
   },
   {
      "id": 3,
      "name": "A blue mouse",
      "price": 25.50,
      "dimensions": {
         "length": 3.1,
         "width": 1.0,
         "height": 1.0
      },
      "warehouseLocation": {
         "latitude": 54.4,
         "longitude": -32.7
      }
   }
]
```

[IETF-JSON]

EMF as Link Tree DDL

- EMF is link-tree structured because it has a primary aggregation hierarchy, but also allows for general links
- A skeleton tree can be identified in all EMF link trees

23.2. Flat Analysis on Link Trees

“Flat” Analysis with Query Languages (Anfragesprachen)

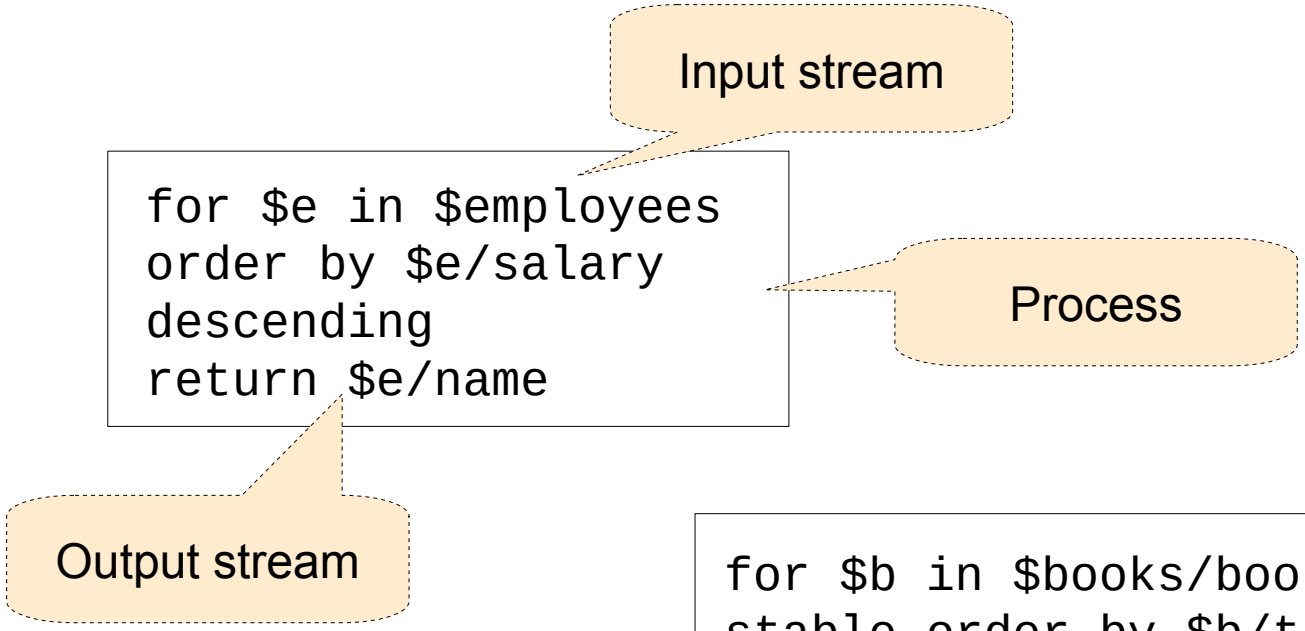
- Query languages (QL) are used for analysis:
  - Match patterns in models or entire repositories
  - Query reachability of model elements
  - Count elements (metrics)
  - Filter streams
- Applications:
  - QL with transformations are useful for writing Filters
  - Consistency Checking in Trees
  - Slicing of trees (view construction)

23.2.1 XQuery

The standard from W3C

Xquery

- http://www.w3.org/XML/Query/ [http://www.w3.org/TR/xquery/]
- Standard of W3C for XML queries
- Patterns and query expressions are embedded in loops over input streams
- Output can be embedded into XML models or HTML pages




```
for $b in $books/book
stable order by $b/title
      collation
"http://www.example.org/collations/fr-ca",
   $b/price descending empty least
return $b
```

Hamlet (Shakespeare) in XML Markup


```
<?xml version="1.0"?>
<!DOCTYPE PLAY SYSTEM "play.dtd">
<PLAY> <TITLE>The Tragedy of Hamlet, Prince of Denmark</TITLE> <FM>
  <P>Text placed in the public domain by Moby Lexical Tools, 1992.</P>
  <P>SGML markup by Jon Bosak, 1992-1994.</P> <P>XML version by Jon Bosak, 1996-1998.</P>
 <P>This work may be freely copied and distributed worldwide.</P>
  </FM>
<PERSONAE>
<TITLE>Dramatis Personae</TITLE>
<PERSONA>CLAUDIUS, king of Denmark. </PERSONA>
<PERSONA>HAMLET, son to the late, and nephew to the present king.</PERSONA>
<PERSONA>POLONIUS, lord chamberlain. </PERSONA>
<PERSONA>HORATIO, friend to Hamlet.</PERSONA>
</PERSONAE>
<ACT><TITLE>ACT I</TITLE>
<SCENE><TITLE>SCENE I.  Elsinore. A platform before the castle.</TITLE>
<STAGEDIR>FRANCISCO at his post. Enter to him BERNARDO</STAGEDIR>
<SPEECH> <SPEAKER>BERNARDO</SPEAKER> <LINE>Who's there?</LINE> </SPEECH>
<SPEECH> <SPEAKER>FRANCISCO</SPEAKER> <LINE>Nay, answer me: stand, and unfold yourself.</LINE> </SPEECH>
<STAGEDIR>Exeunt</STAGEDIR>
</SCENE>
</ACT>
<ACT><TITLE>ACT II</TITLE>
...
</ACT>
</PLAY>
```

[Wikipedia]

Xquery is a Mixed Language: Iterations and XML Patterns

The following script produces a list of speakers of the hamlet plot

```
<html><head/><body>
{
for $act in doc("hamlet.xml")//ACT
let $speakers := distinct-values($act//SPEAKER)
return
  <div>
   <h1>{ string($act/TITLE) }</h1>
   <ul>
   {
    for $speaker in $speakers
    return <li>{ $speaker }</li>
   }
   </ul>
  </div>
}
</body></html>
<?xml version="1.0"?>
```

[Wikipedia]

23.2.2 Flat Analysis on Link Trees with the Query and Term Transformation Language Xcerpt

A modern, declarative query and transformation language for link trees in the JSON/XML technical space

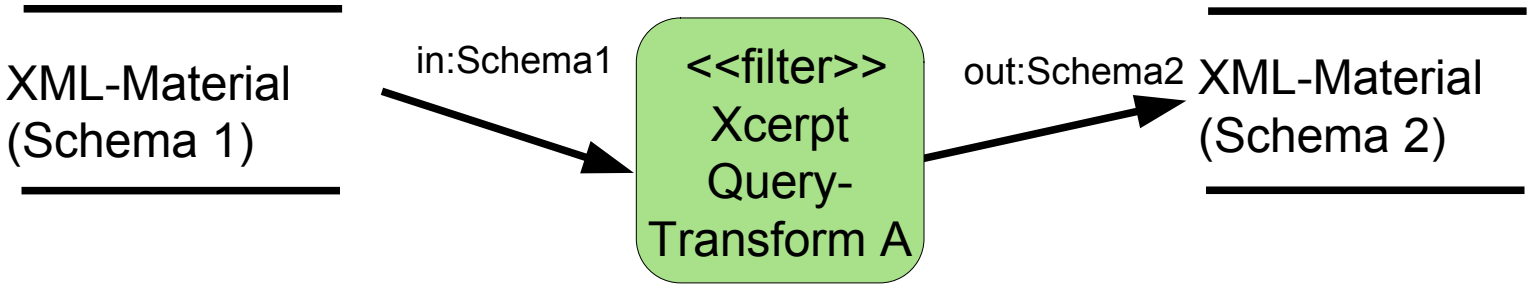
Xcerpt combines a DQL and a DTL for link trees

Literature - Modular Xcerpt

- Xcerpt prototype compiler: http://sourceforge.net/projects/xcerpt
- Sebastian Schaffert. Xcerpt: A Rule-Based Query and Transformation Language for the Web. PhD Thesis,Institute for Informatics, University of Munich, 2004.
- Sebastian Schaffert, François Bry. Querying the Web Reconsidered: A Practical Introduction to Xcerpt (2004) In Proc. Extreme Markup Languages
  - http://www.pms.informatik.uni-muenchen.de/publikationen/PMS-FB/PMS-FB-2004-7.pdf
- U. Aßmann, S. Berger, F. Bry, T. Furche, J. Henriksson, and J. Johannes. Modular web queries from rules to stores. In 3rd International Workshop On Scalable Semantic Web Knowledge Base Systems.
- Uwe Aßmann, Andreas Bartho, Wlodek Drabent, Jakob Henriksson and Artur Wilk. Composition Framework and Typing Technology tutorial In Rewerse I3-d14
  - http://rewerse.net/deliverables/m48/i3-d14.pdf
- Jakob Henriksson and Uwe Aßmann. Component Models for Semantic Web Languages. In Semantic Techniques for the Web. Lecture Notes in Computer Science 5500. Springer Berlin / Heidelberg, ISSN 0302-9743, 2009
  - http://springerlink.metapress.com/content/x8q1m87165873127/?p=edfdbbaec29743d59da1cd6f1ea50826&pi=4
- Artur Wilk. Xcerpt web site with example queries.
  - http://www.ida.liu.se/~artwi/XcerptT

Xcerpt: A Modern Web Query Language for Filter

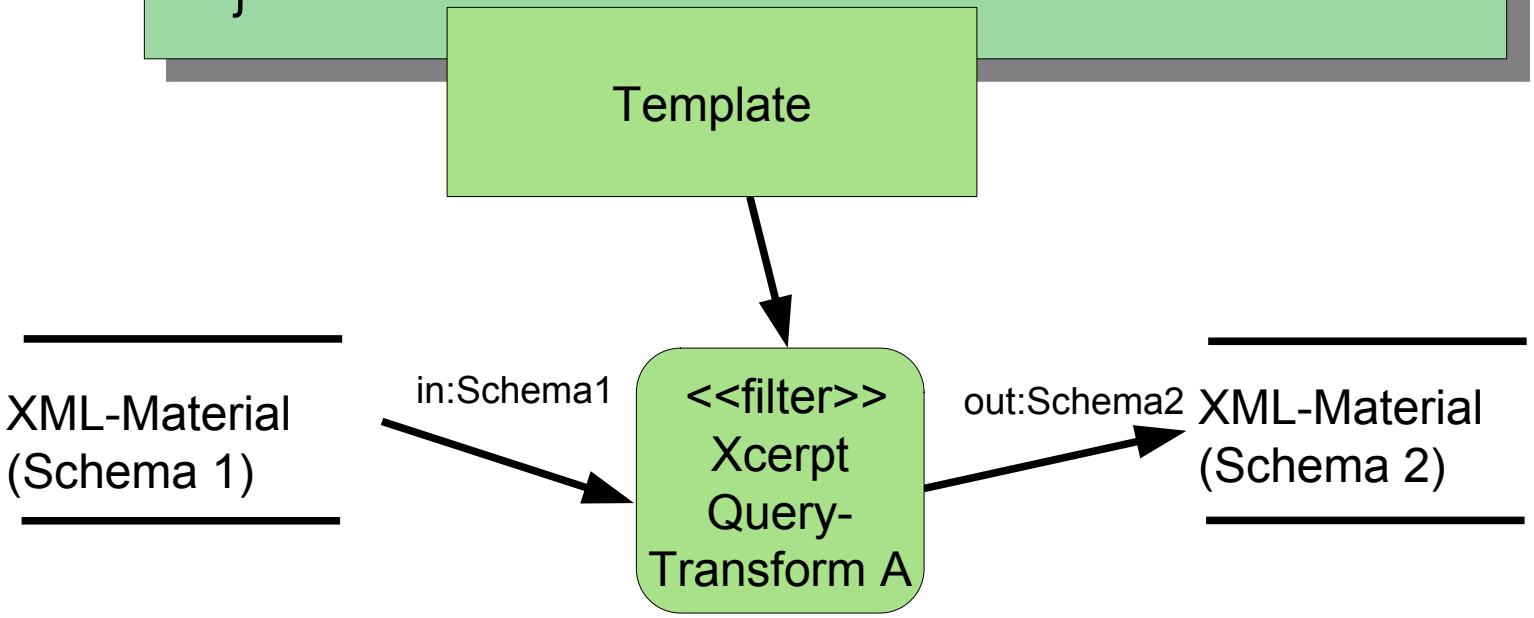
- Xcerpt is a pattern-based query language for XML formatted data
  - Terms, trees, link trees, link terms (JSON and XML terms)
  - Patterns match data w.r.t. tree structure
  - Fully declarative, in contrast to Xquery; rule-based, declarative style of Logic Programming (LP)
  - Much more flexible than XPath, which supports only path-based selection
- Xcerpt is also a transformation language in form of a term rewrite system (Termersetzungssystem):
  - Separate query terms (left-hand side) and construct terms (right-hand side) not like in XQuery)
  - it has “Construct terms” to simplify creation of new documents
- Xcerpt is stream-based **filter**: processes read, filter, and write streams
  - Xcerpt can be used as generator and transformer in DFD



Template Processing with Xcerpt

- Templates may have **for-loops** which can be expanded by Xcerpt transformations

```
[template produceOutput]
[file("output.java")]
Public class Test {
  [for p:Attribute | System.in.readln() separator(,\n')
  private p;
}
```



Xcerpt Data Terms (JSON/XML Link trees)

- Xcerpt data terms represent XML link-trees with nice syntax, simpler than JSON:
- Basic constructors for data terms:
  - **exact description of a collection of children:**
    - ordered list [...], --- ordered (ranked) trees
    - unordered set {...} --- unordered trees
  - **partial description of a collection of children:**
    - ordered partial list [[...]]
    - unordered partial set {{...}}
  - **references/links:**
    - key id@, keyref ^id

```
<book><title>The Last Nizam</title></book>
```

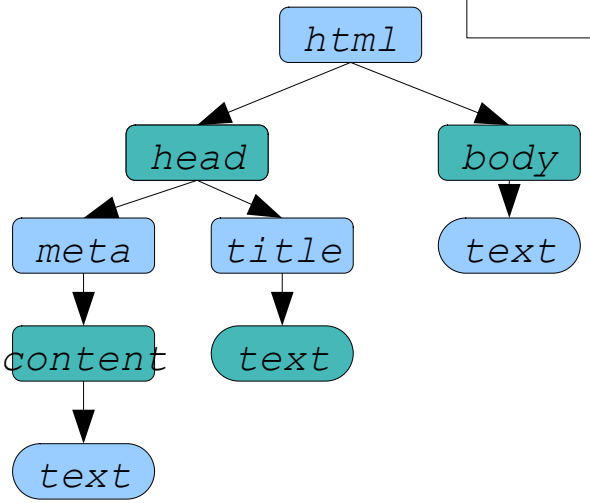
equivalent to:

```
book [ title [ “The Last Nizam” ] ]
```

Xcerpt Data (Link-)Terms


```
html [
    head [
        meta [
          content {“text/html”}
        ]
        title [“Website”]
    ],
    body [“content”]
]
```

```
<html>
  <head>
    <meta content=”text/html”/>
    <title> Website </title>
  </head>
  <body>
       content
  </body>
</html>
```



Xcerpt Query Terms to Match and Filter Link Trees (Data Terms with Variables)

- XML querying is done with Xcerpt query terms. A ***query term*** is a term pattern *containing variables* (noted in uppercase letters) over XML data, underspecified data terms:
  - Ordered matching with List symbol: `data [ term [ … X ] ]`
  - No-order matching with Set symbol: `data { term { … X } }`
  - Ordered partial matching and deep filtering: `[[ X ... ]]`
  - Unordered partial matching and filtering: `{{ … X }}`
  - Queries connect query terms with logical expressions:

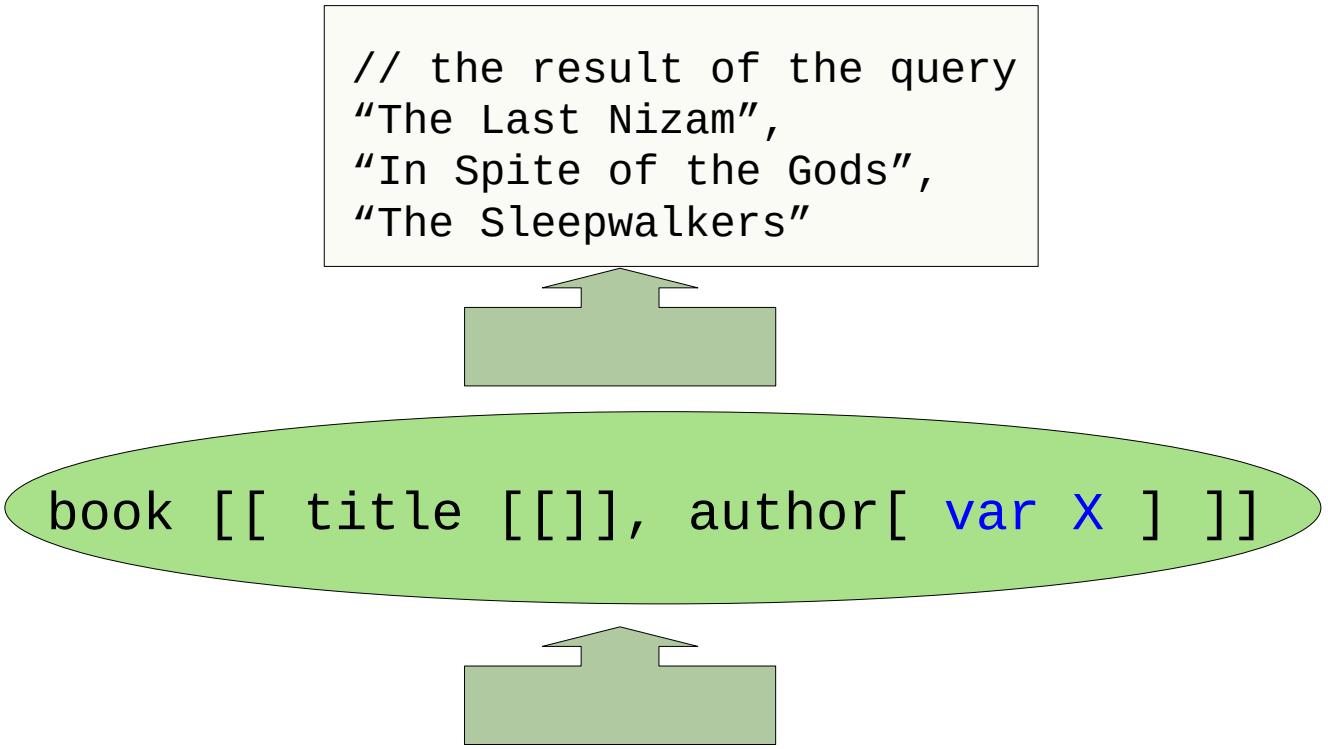
    `and { ... }, or { ... }`
  - Variables can unify to subterms
- Query terms are data terms with variables prefixed by keyword "var"

```
query(book[title[[ ]]], author [var X])
---> "John Zubrzycki", "Edward Luce", "Christopher Clark"
```

```
// the data base of books, to be filtered
bib [
   book [ title [ "The Last Nizam" ],      author [ "John Zubrzycki" ] ],
   book [ title [ "In Spite of the Gods" ],author [ "Edward Luce" ] ]
   book [ title [ "The Sleepwalkers" ],    author [ "Christopher Clark" ] ]
]
```

Xcerpt Query Terms

- **Query terms** are data terms with variables prefixed by keyword "var"
-




```
// the data base
book [ title [ "The Last Nizam" ],  author [ "John Zubrzycki" ] ],
book [ title [ "In Spite of the Gods" ], author [ "Edward Luce" ] ]
book [ title [ "The Sleepwalkers" ],    author [ "Christopher Clark" ] ]
```

Deep Partial Unordered Matching with Xcerpt Query Terms

- A query term can do ***deep partial matching:***

```
library {{ --- somewhere deep in the library
    book {{  --- somewhere deep in the book's data
      // var Author sets a label on a data term
      var Author -> author {{ --- smwh in the author
              surname {"Aßmann"}
      }},
      // var Title matches a subterm
      title [ var Title ]
   }}
}}
```

- Query matches all books with at least one author "Aßmann"
  - assigns the matched authors to variable Author
  - assigns the matched book titles to variable Title
- Produces a stream of a pair of variables (Author, Title)
  -

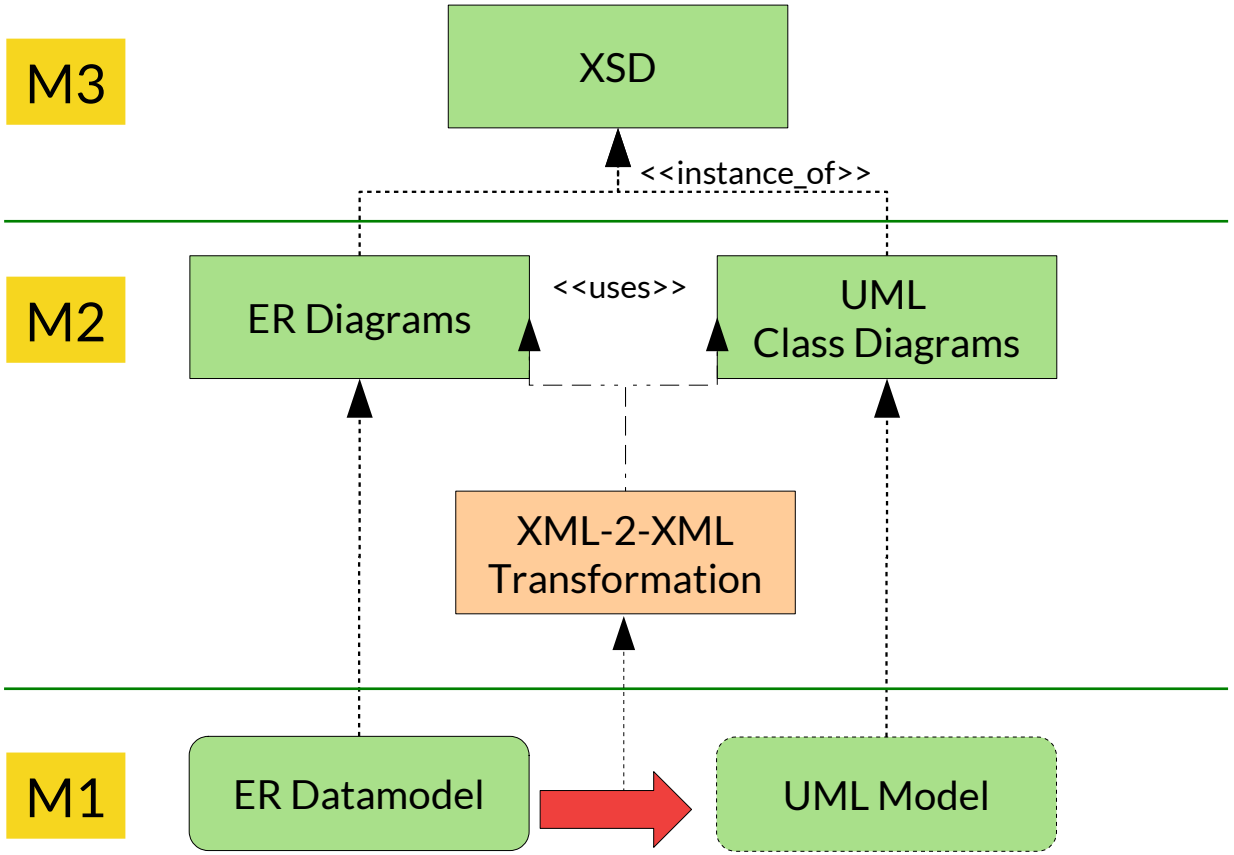
23.3 Writing Link-Tree Filters with Link-Tree- and XML-based Data Transformation Languages (DTL)

Text, XML, Term, and Graph Rewriting

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Tree, Term, Link Tree, and XML transformations

- Model transformations defined in Layer M_{i+1} specify how to transform models on M_i
  - Source and target metamodel
- **Benefit**: Transformation can be reused for all models, which are instances of the source meta-model



DTL and DML

- With a general *data manipulation language (DML, Datenmanipulationssprache)*, data is transformed from one form to another
- **Declarative DML (data transformation languages, Datentransformationssprachen, DTL)** are specific DML using rules to describe transformations.
  - Specifications consist of rules, grouped in *rule sets* or *rule modules*
    - Term rewrite rules (Termersetzungsregeln) transform trees, terms and dags
    - Graph rewrite rules transform graphs and models
  - In *free (chaotic) rewriting*, the control-flow specification is not necessary
  - In *strategic rewriting*, a strategy (higher-order function) controls the rewriting
  - In *programmed rewriting*, a program or workflow controls rewriting
- Examples of declarative DML (DTL) in the Link-Treeware:
  - Xquery
  - Xcerpt als Strom-Manipulationssprache
  - XGRS and Fujaba (on graphs)
- On the other hand, **Imperative DML (allgemeine DML)** know states, side effects, heaps.

Link-Term and Link-Tree Rewrite Systems (Link-Termersetzungssysteme, LTRS)

- **Link-Tree Rewrite Systems (LTRS)** work on trees with links
  - e.g., Xquery or Xcerpt
  - XML-trees **(XML-Rewrite Systems)**
  - JSON trees
- Use:
  - Links are used as abbreviations to remote siblings in the tree (information in the context)
  - Links access context-sensitive information
  - Links abbreviate paths in the tree
  - Links must be controlled on consistency

23.3.1 Transformation and Filtering of Link Trees with Xcerpt

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Xcerpt Provides "Construct Terms"

- **Construct Terms** (transformation expressions) are *XML templates constructing arbitrary structured XML data*
  - access data from variables bound by query terms
  - aggregate/re-group data
  - can only have single brackets (no optional content)
- Example: "Construct one title/author pair in an (unordered) result tag":

```
result {
        var Title, var Author
}
```

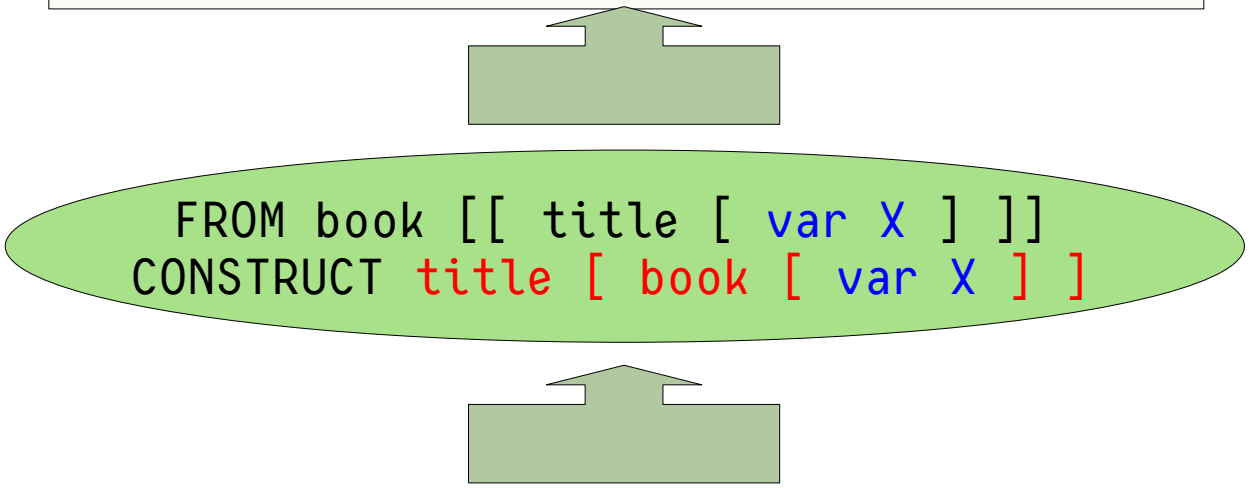
- Example: "Construct a complete book result list grouped by full author name":

```
booklist {
        all books {
                all var Author,
                var Title
        }
}
```

Xcerpt Transformation Rules

- Combine Query and Construct Terms (XML templates) via common variables:
  - Construct terms construct term fragments and embed common variables with the query term

```
// the result
title [ book [ “The Last Nizam” ] ],
title [ book [ “In Spite of the Gods” ] ]
title [ book [ “The Sleepwalkers” ] ]
```




```
// the data base
book [ title [ “The Last Nizam” ],  author [ “John Zubrzycki” ] ],
book [ title [ “In Spite of the Gods” ], author [ “Edward Luce” ] ]
book [ title [ “The Sleepwalkers” ],    author [ “Christopher Clark” ] ]
```

Xcerpt Programs are Rule Sets

- Xcerpt programs consist of a collection of data-terms (database) and rules (with query and construct terms)
  - 0+ data-terms
  - 1+ goal rules
  - 0+ construct-query rules
- *Construct rules (transformation rules)*: produce intermediate resu

Result schema of a rule

```
CONSTRUCT <head> FROM <body> END
FROM <body> CONSTRUCT <head> END
```

- *Goal rules*: final output

```
GOAL <head> FROM <body> END
Where <head>: construct term;  <body>: query
```

Goal schema

```
CONSTRUCT
  construct term
FROM
  query term
END
```

Simple Xcerpt Program

- Matching query → variable bindings
  → apply bindings to construct term

```
CONSTRUCT   --- template
   titles [
      all title [ var Title ]
   ]
FROM   --- query
   bib {{
      book {{
         title [ var Title ],
   }} }}
END
```

produce

```
titles [
   title [ “The Last Nizam” ],
   title [ “In Spite of the Gods” ]
   title [ „The Sleepwalkers“ ]
]
```

query

```
// the data base
bib [
   book [ title [ “The Last Nizam” ],     author [ “John Zubrzycki” ] ],
   book [ title [ “In Spite of the Gods” ], author [ “Edward Luce” ] ],
   book [ title [ “The Sleepwalkers” ],    author [ “Christopher Clark” ] ]
]
```

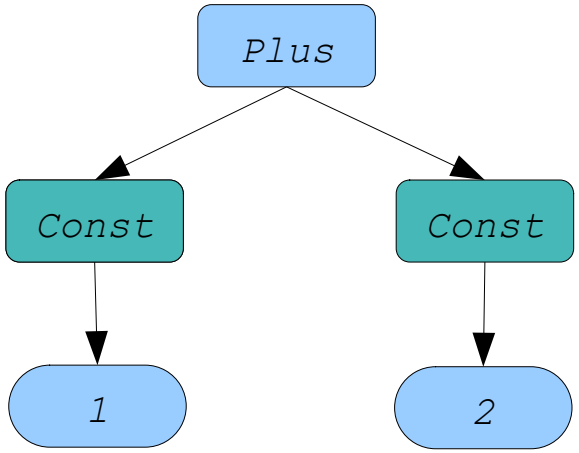
23.3.2 Code Transformations with Xcerpt Term Rewriting and Template Processing

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Xcerpt Data Terms for Representing Expression Code

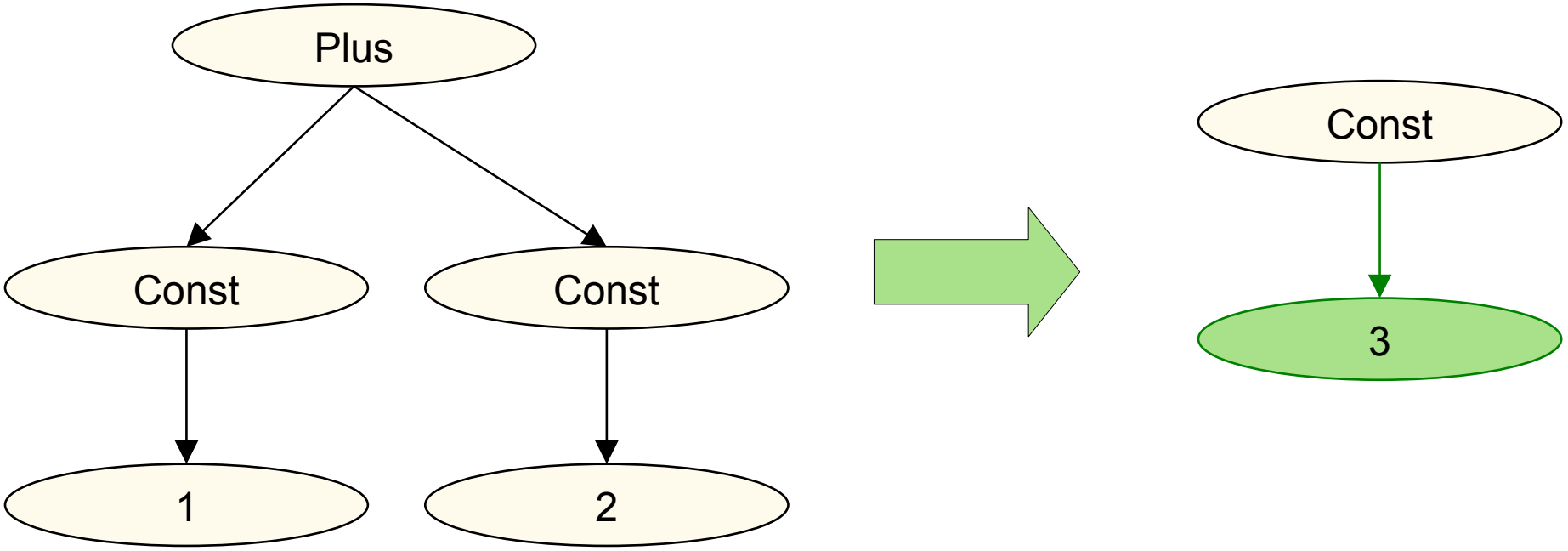

```
Plus [
        Const [ 1 ],
        Const [ 2 ]

]
```

```
<Plus>
        <Const>
                1
        </Const>
        <Const>
                2
        </Const>
</Plus>
```



Constant Folding (Static Evaluation) on Link Trees

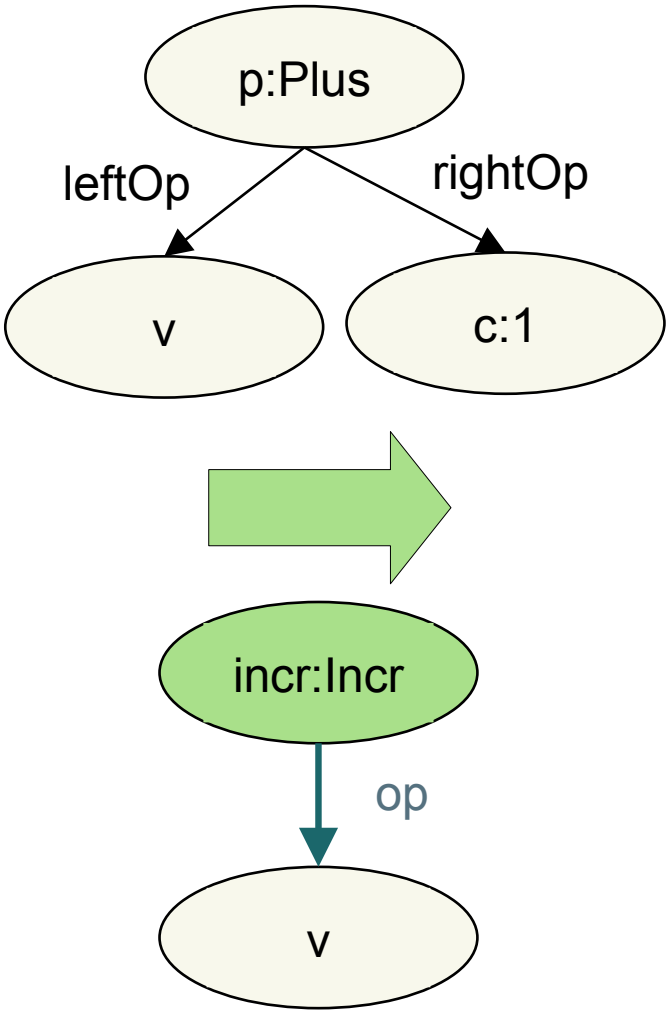
- A **local rewriting (context-free rewriting)** matches a weakly connected left-hand side graph with a redex, independently of a context.
  - Matching of one redex can be done in constant time
- Subtractive because redexes are destroyed




```
FROM Plus [ Const [ 1 ], Const [ 2 ] ] CONSTRUCT Const [3]
```

Context-Free Local Rewritings: Operator Strength Reduction on Link Trees

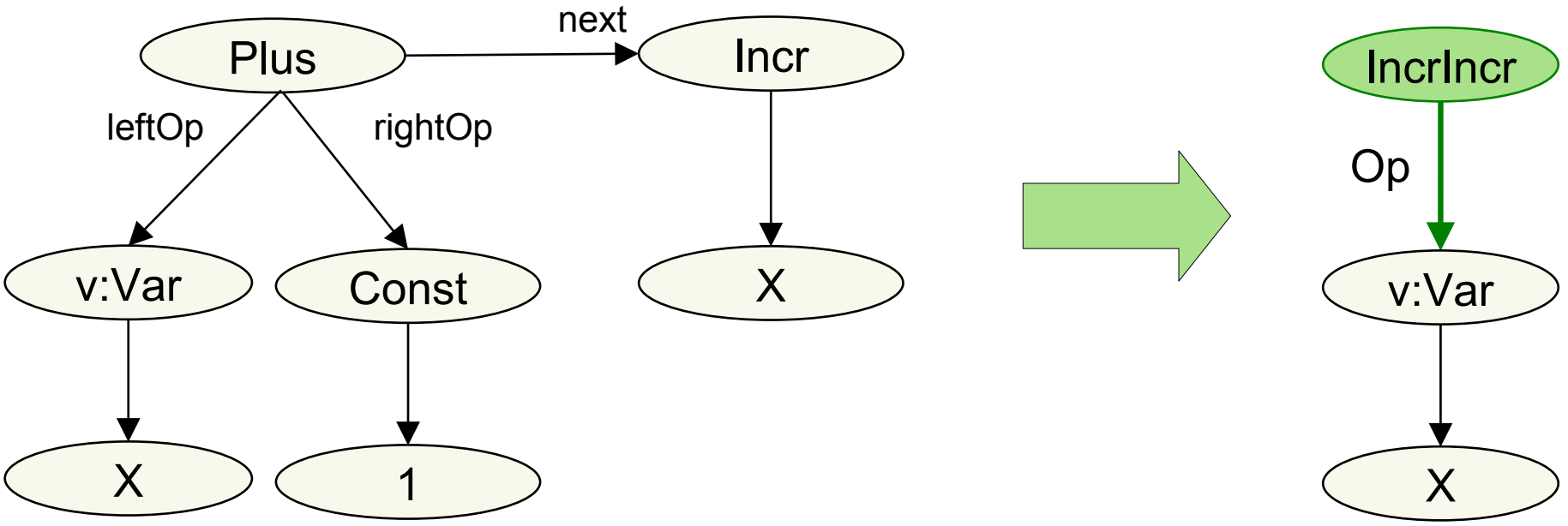

```
// if-then rules as in GrGen
   Graph Rewriting System:
if  leftOp(p:Plus,v),
    rightOp(p,c:1),
then
   Delete p,
   Delete c,
   Add incr:Incr,
   op(incr,v);
```




```
FROM Plus [ leftOp [ var v ], rightOp [ c:1 ] ]
CONSTRUCT incr:Incr [ var v]
```

Peephole Optimization on Link Trees

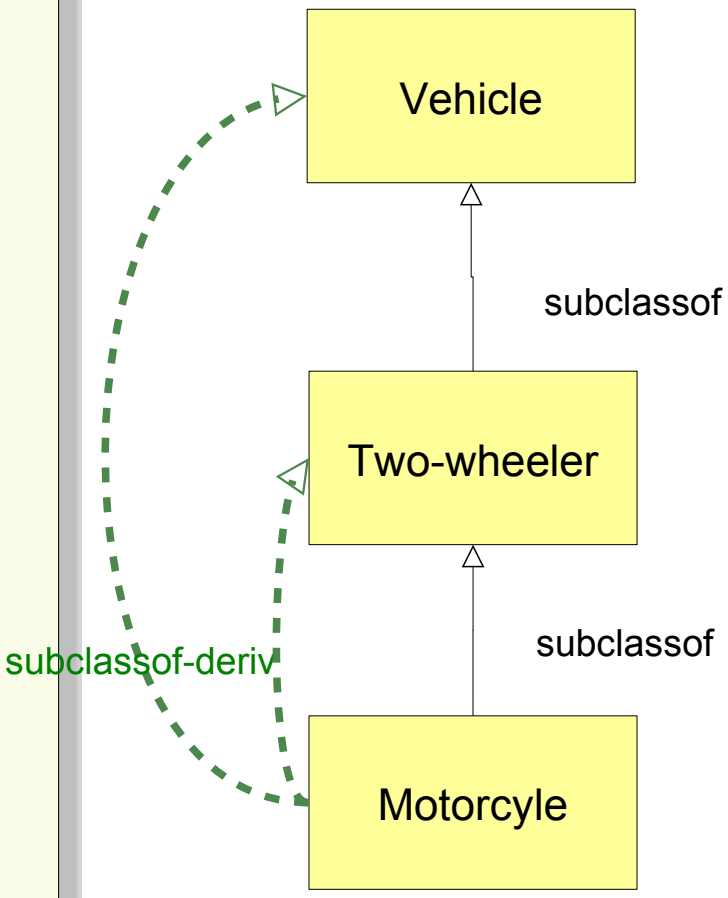
- Peephole optimization is done on statement lists or trees
- Subtractive problem, because redexes are destroyed




```
FROM Plus[ leftOp[ var v->Var[ var X] ],
     rightOp[ Const[1] ], next[Incr[ var X]]
CONSTRUCT IncrIncr[ Op[ var v] ]
```

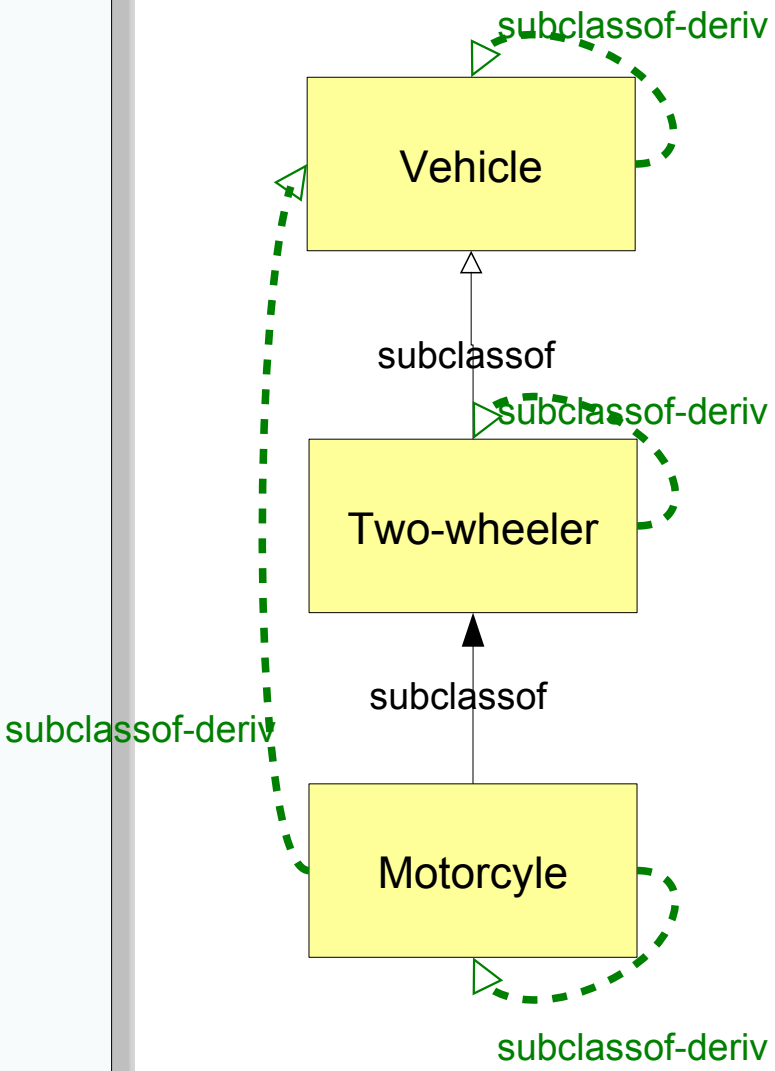
Rule Dependencies in a Set of Rules (here: All Superclasses as Transitive Closure in Inheritance Hierarchy): Simple Variant


```
CONSTRUCT // Base case of transitive closure
   subclassof-deriv [ var Sub, var Super ]
FROM
   subclassof [ var Sub, var Super ]
END

CONSTRUCT // Alternative, recursive rule
   subclassof-deriv [ var Sub, var Sup ]
FROM
   subclassof [ var Sub, var Z ],
   subclassof-deriv [ var Z, var Sup ]
END

// Basic relation of direct subclassof
CONSTRUCT subclassof [ var Sub, var Sup ]
FROM
   in { resource { "file:...", "xml" } }
END
```



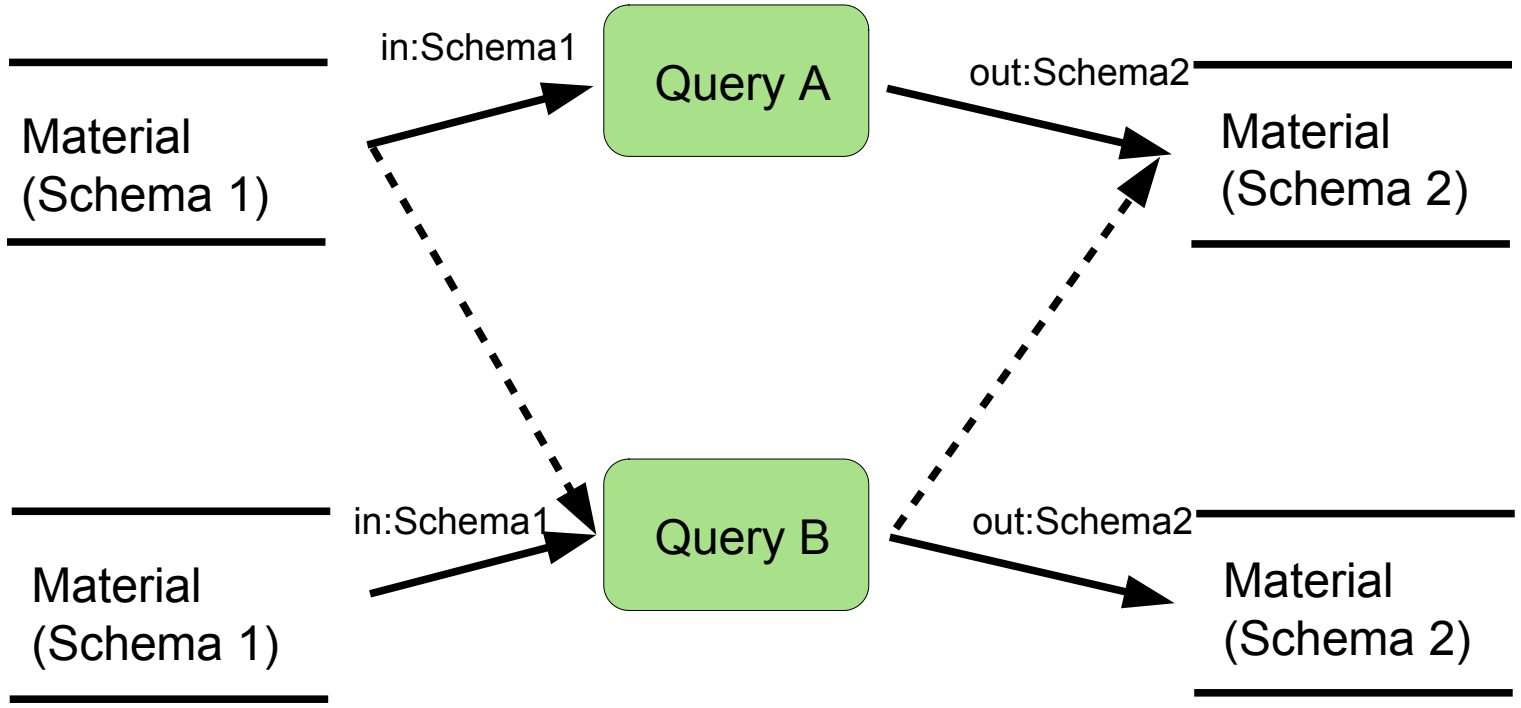
Rule Dependencies in a Set of Rules (here: All Superclasses as Transitive Closure in Inheritance Hierarchy) Variant with Self-Superclasses


```
CONSTRUCT // Base case of transitive closure
   subclassof-deriv [ var Cls, var Cls ]
FROM
   or { subclassof [ var Z, var Cls ],
        subclassof [ var Cls, var Z ] }
END

CONSTRUCT // Alternative, recursive rule
   subclassof-deriv [ var Sub, var Sup ]
FROM
   or { // direct subclass
        subclassof [ var Sub, var Sup ],
        and {
         subclassof [ var Sub, var Z ],
         subclassof-deriv [ var Z, var Sup ]
   } }
END
// Basic relation of direct subclassof
CONSTRUCT subclassof [ var Sub, var Sup ]
FROM
   in { resource { "file:...", "xml" },
      <query> }
END
```



Use of DQL and DTL in Werkzeugen

- Stream-processing QL and TL are useful for the composition of tools on material streams

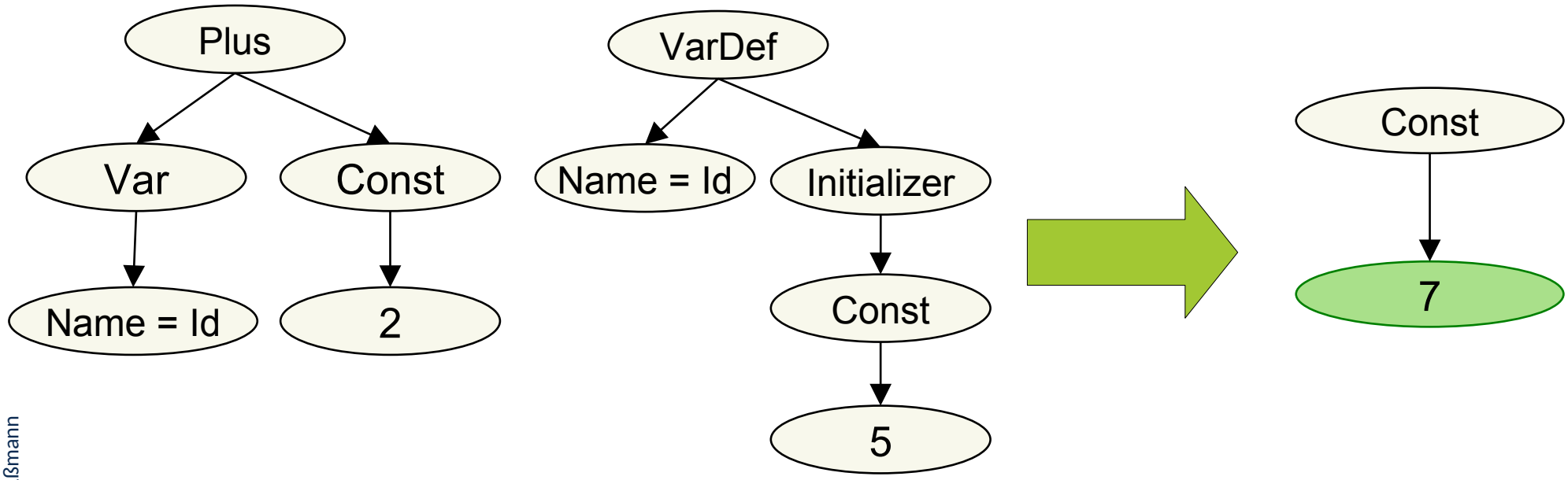


Tool composition theorem 1: Two stream-based tools can be composed if
Their input and output types are compatible and
the output order in the output streams is commutative

23.3.3 Context-Sensitive Term Rewritings

Extended Constant Folding as Subtractive TRS

- A term rewrite rule usually works context-free, i.e., matches and rewrites only one term.
- A **context-sensitive term rewriting** matches a set of non-connected left-hand side terms with a redex.
  - Matching of one redex can be done in quadratic time, because non-connected nodes have to be pairwise compared




```
FROM Plus[ Var[ Name [var Id] ], Const[2],
  AND  VarDef[ Name[var Id], Initializer[Const[5]] ]
CONSTRUCT Const[7]
```

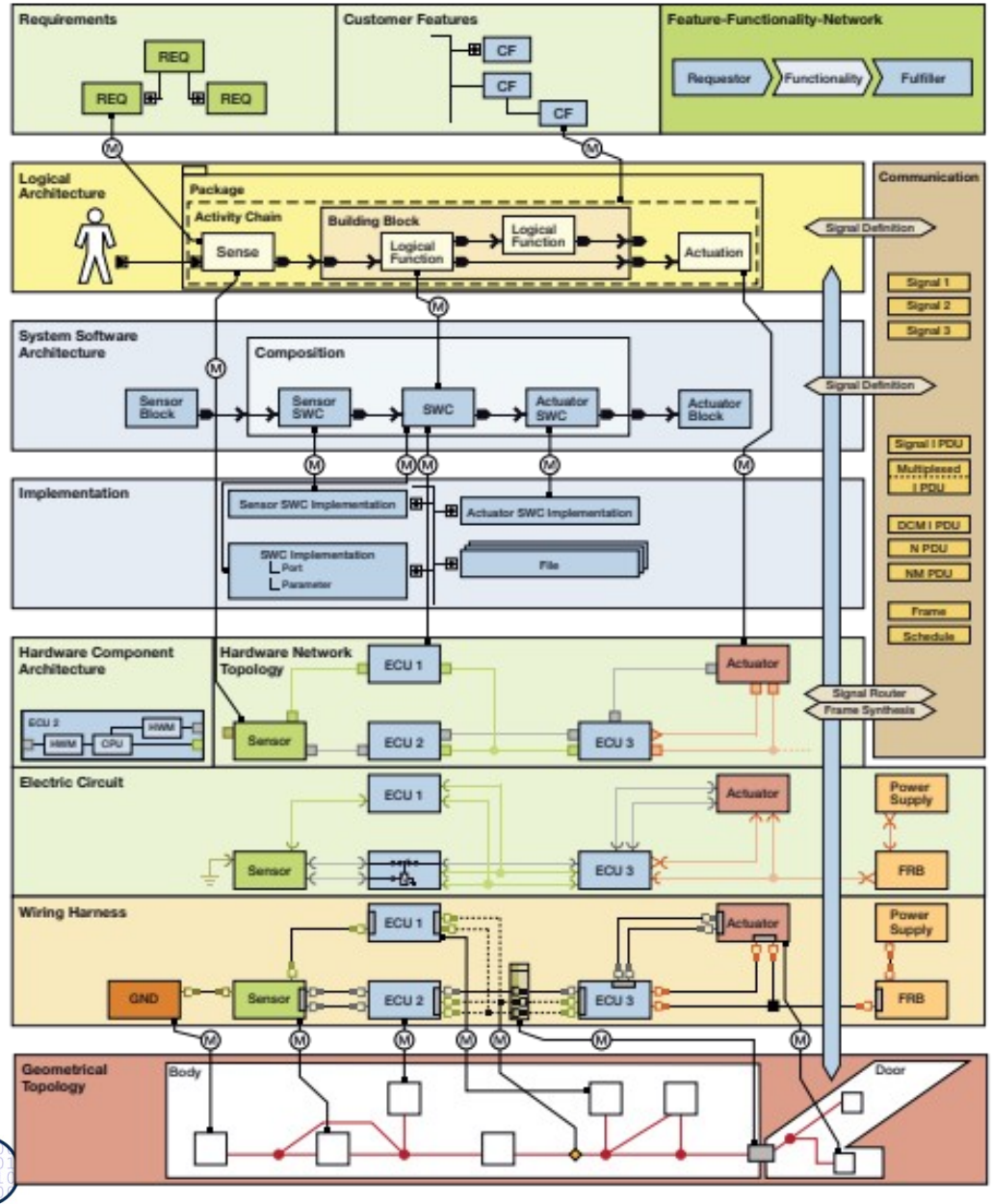
Covered Code Optimizations

- Global transformations
  - Common Subexpression Elimination
    - introducing links that share a subexpression
  - Refactorings:
    - **Rename** all uses of a variable and its definition
    - **Move** a method into another class
    - **Split** a class
  - Code motion
    - Mode an expression out of a loop
    - **Clone removal:** Outline a method from a set of clones and transform all clones to calls of the method

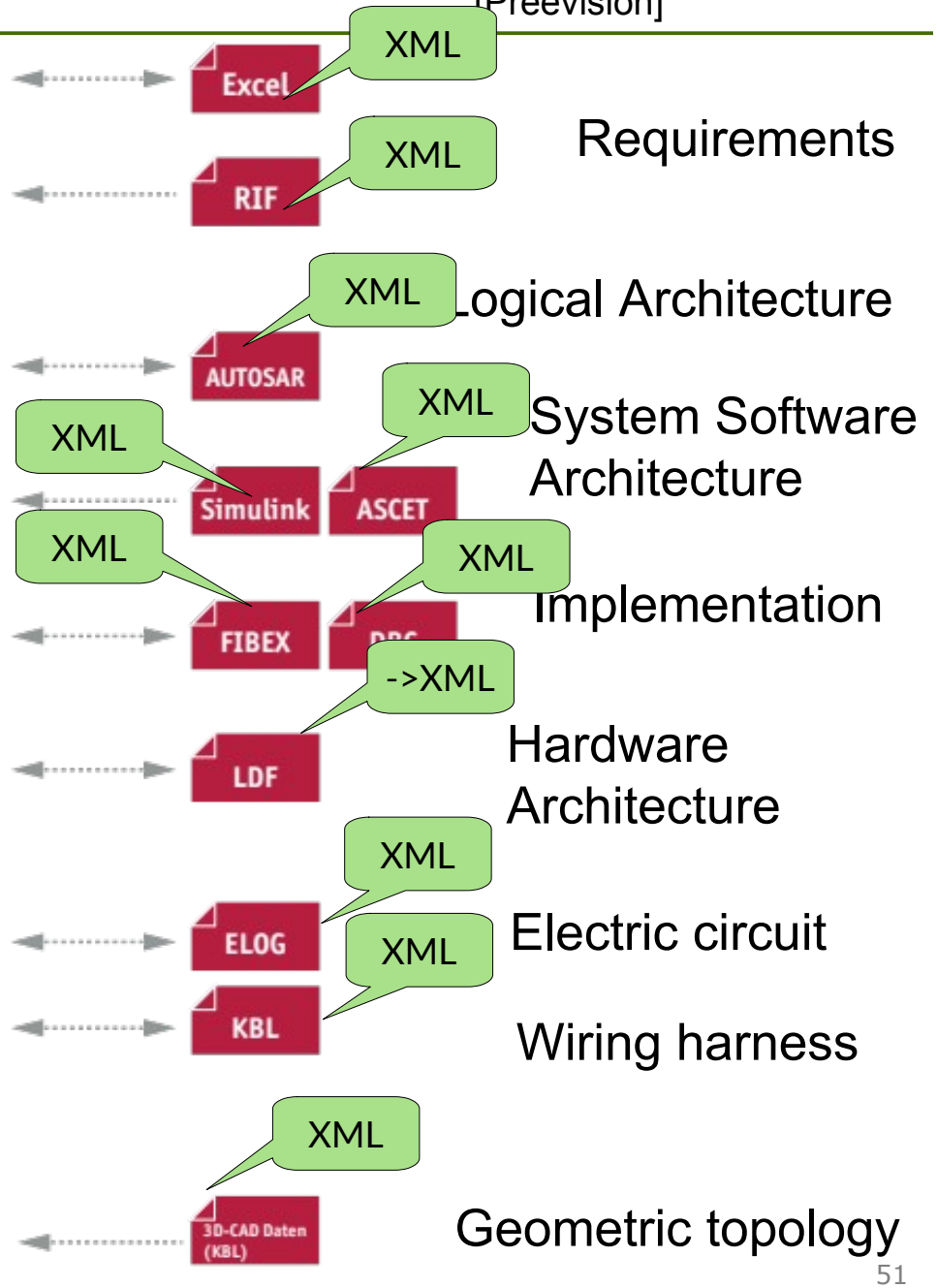
23.4 The Big Picture (I): The Importance of Link Trees for MDSD Applications

- Link trees, their querying, attribution, and rewriting is very important for an MDSD IDE

Remember the Big Example: Car Design with PREEVision (Vector): Interoperability with XML Link Trees



[Preevision]



Links on the XML Formats of PreeVision

- Excel:
  https://support.office.com/de-de/article/%C3%9Cberblick-%C3%BCber-XML-in-Excel-f11faa7e-63ae-4166-b3ac-c9e9752a7d80
- RIF: https://en.wikipedia.org/wiki/Requirements_Interchange_Format
- Simulink:
  http://de.mathworks.com/help/rptgenext/ug/how-to-compare-xml-files-exported-from-simulink-models.html?requestedDomain=www.mathworks.com
- AutoSAR and FIBEX https://vector.com/vi_autosar_de.html
  - https://de.wikipedia.org/wiki/AUTOSAR
  - http://xn--brrkens-b1a.de/publications/pagel_broerkens_ECMDA2006.pdf
  - http://www.elektronikpraxis.vogel.de/embedded-computing/articles/226651/index3.html
  - http://www.autosar.org/fileadmin/files/releases/4-2/methodology-and-templates/tools/auxiliary/AUTOSAR_TR_InteroperabilityOfAutosarTools.pdf
  - http://www.sse-tubs.de/publications/Hoe_ASE07.pdf
- LDF http://www.fullconvert.com/XML-to-LDF/

Links on the XML Formats of PreeVision

- ELOG http://www.ecad-if.de/elog.html
- KBL (Kabelbaumliste) http://www.ecad-if.de/kbl.html
- ASCET (von ETAS) http://www.etas.com/data/RealTimes_2010/rt_2010_2_32_de.pdf
  - http://www.etas.com/de/products/ascet_software_products-details.php
  - http://www.file-extensions.org/amd-file-extension-ascet-xml-model-description-file
  - http://www.etas.com/download-center-files/products_ASCET_Software_Products/ETAS_ASCET_6.1_flyer_DE.pdf

How To Develop an MDSD Application with Link Trees

- Read in XML with XML parser
- Query XML link trees with languages like Xcerpt
- Semantic analysis of the trees with RAG, with languages like JastAdd
- Transform with languages like
  - Xcerpt
  - Stratego (rewriting)
  - RAG tree generation and template expansion
- Problematic: Tool maturity

The End

- Why are XML documents link trees? Is such a document a link term or link tree?
- How does Xcerpt do deep match?
- Explain how Xcerpt transformation expressions filter an input stream and produce an output stream
- Why can RAG work on link trees?
- How would you analyse the link structure of an XML document?
- How do references in a link tree abbreviate the way from uses to definitions of variables?
- What does name analysis do with regard to the links of a link tree?
- What does type analysis do with regard to the links of a link tree?
- Does a downward query disturb the rest of the attribution in the subtree? (hint: it depends...)

- Many slides are courtesy to Jakob Henriksson, Sven Karol and Christoff Bürger. Thanks.

23.A.1 Technical Space Link-Treeware with Metalanguages for XML

Data in tree format with overlaying links (references)

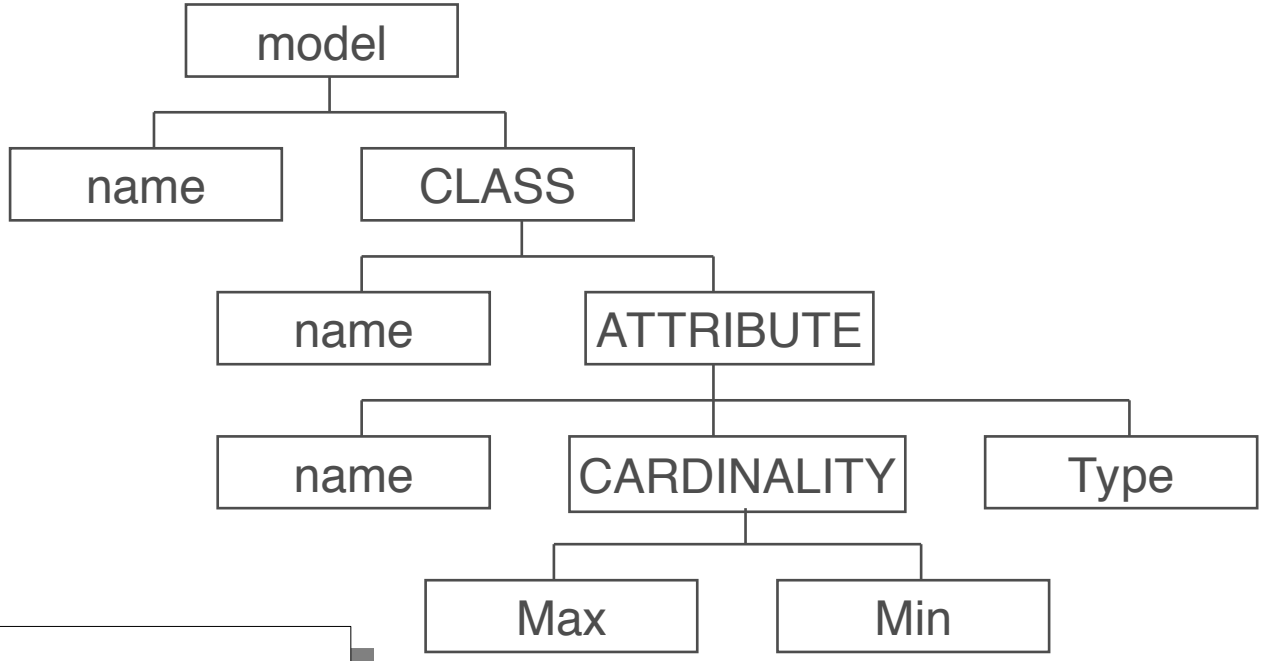
DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

XML

- XML is a family of link-tree languages, mainly for
  - Markup of data
  - Syntax trees
- http://www.w3.org/XML
- XML is block-structured with “tag” brackets; only list, no set constructor
- XML often used as exchange format
- XML metamodels on M2 can be defined with several metalanguages on M3
  - Document Type Definitions (DTD): special DDL
  - XML Schema Definition (XSD) is a lifted DDL and similar to a RTG with maps
  - RelaxNG: special DDL

Document Type Definition (DTD) for XML

- A **DTD** is a simple metalanguage for XML
- Based on regular tree grammars (RTG)




```
<! ELEMENT model (name, CLASS)>
<! ELEMENT CLASS  (name, ATTRIBUTE*)>
<! ELEMENT name #PCDATA REQUIRED>
<! ELEMENT ATTRIBUTE  (name, CARDINALITY?, Type?)>
<! ELEMENT CARDINALITY  (Min, Max)>
<! ELEMENT Min  (#PCDATA) REQUIRED>
<! ELEMENT Max  (#PCDATA)>
<! ELEMENT Type  (#PCDATA)>
```

[Tolksdorf, R.: XML und darauf basierende Standards: Die neuen Aufzeichnungssprachen des Web; Informatik-Spektrum 22(1999) H. 6, S. 407 - 421]

Example for an XML Document Instance

- Uses all ELEMENTs of a XSD or DTD as “tags” in an XML model

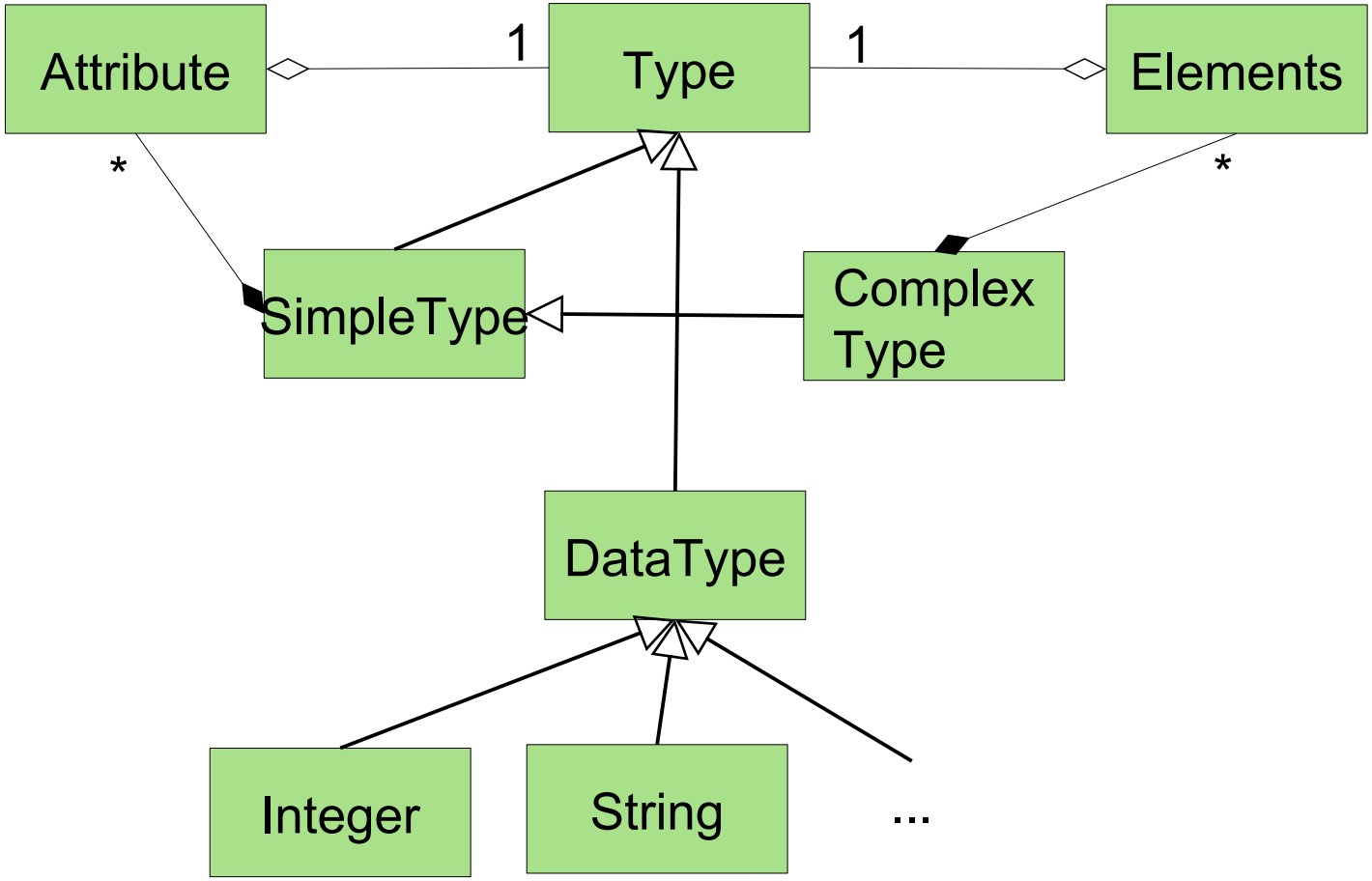
```
<? xml version = „1.0“?>
<! DOCTYPE oomodel SYSTEM „oom.dtd“>
<model>
  <name> „Model 1“ </name>
  <CLASS>
    <name> „Class 1“ </name>
    <ATTRIBUTE>
       <name> „attribute 1“ </name>
       <CARDINALITY>
            <Min> „1“ </Min>
            <Max> „1“ </Max>
       </CARDINALITY>
       <Type> „Class 1“ </Type>
    </ATTRIBUTE>
  </CLASS>
</model>
```

Ex.: DTD for Property Lists in XML


```
<!ENTITY % plistObject "(array | data | date | dict | real | integer | string |
     true | false )" >
<!ELEMENT plist %plistObject;>
<!ATTLIST plist version CDATA "1.0" >
<!-- Collections -->
<!ELEMENT array (%plistObject;)*>
<!ELEMENT dict (key, %plistObject;)*>
<!ELEMENT key (#PCDATA)>
<!--- Primitive types -->
<!ELEMENT string (#PCDATA)>
<!ELEMENT data (#PCDATA)> <!-- Contents interpreted as Base-64 encoded -->
<!ELEMENT date (#PCDATA)> <!-- Contents should conform to a subset of ISO 8601
    (in particular, YYYY '-' MM '-' DD 'T' HH ':' MM ':' SS 'Z'.  Smaller units
     may be omitted with a loss of precision) -->
<!-- Numerical primitives -->
<!ELEMENT true EMPTY>  <!-- Boolean constant true -->
<!ELEMENT false EMPTY> <!-- Boolean constant false -->
<!ELEMENT real (#PCDATA)> <!-- Contents should represent a floating point
    number matching ("+" | "-")? d+ ("."d*)? ("E" ("+" | "-") d+)?
    where d is a digit 0-9.  -->
<!ELEMENT integer (#PCDATA)> <!-- Contents should represent a (possibly
    signed) integer number in base 10 -->
```

XML Schema Definition (XSD)

- XSD is a metalanguage (lifted metamodel) for the definition of XML-dialects
- For definition of the valid “tags” and their structure
  - XML-Syntax
- MOF-metamodel von XSD:



XML Example


```
<treatment>
  <patient insurer="1577500"nr='0503760072'/>
  <doctor  city   ="HD"     nr='4321'/>

  <service>
    <mkey>1234-A</mkey>
    <date>2001-01-30</date>
    <diagnosis>No complications.
    </diagnosis>
  </service>

</treatment>
```

simple

complex

[W. Löwe, Linnaeus Universitet Växjö]

Example: Definition of Simple and Complex Tag Types with XML Schema (XSD)


```
<simpletype name='mkey' base='string'>
  <pattern value='[0-9]+(-[A-Z]+)?'/>
</simpletype>

<simpletype name='insurer' base='integer'>
  <precision value='7'/>
</simpletype>

<simpletype name='myDate' base='date'>
  <minInclusive value='2001-01-01'/>
  <maxExclusive value='2001-04-01'/>
</simpletype>

<complextype name='treatment'>
  <element name='patient' type='patient'/>
  <choice>
    <element ref='doctor'/>
    <element ref='hospital'/>
  </choice>
  <element ref='service' maxOccurs='unbounded'/>
</complextype>
```

Example: XML Schema Attributes


```
<complextype name='patient' content='empty'>
  <attribute ref ='insurer' use='required'/>

  <attribute name='nr' use='required'>
    <simpletype base='integer'>
      <precision value='10'/>
    </simpletype>
  </attribute>

  <attribute name='since' type='myDate'/>
</complextype>
```

23.A.2 Larger Xcerpt Programs

- Optional
- With modular Xcerpt

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Modular Xcerpt

- *Modular Xcerpt* = Xcerpt + Module support
- http://www.reuseware.org/index.php/Screencast_LoadMXcerptProject_0.5.1
- Declaring a module in Modular Xcerpt:

```
MODULE module-id
 module-imports
 xcerpt-rules
```

- Declaring a module's interface
  - Modular Xcerpt programs importing a module can reuse public construct terms

```
public construct term
```

  - Modular Xcerpt programs can provide data to an imported module's public query terms

```
public query term
```

Modular Xcerpt

- Modular Xcerpt is an Extension of Xcerpt for larger programs

- A query can be reused via a module's interface

```
IMPORT module AS name
```

  - reuses public construct terms as a data provider for the given query term

```
in module ( query term )
```

  - provides the given construct term to public query terms of an imported module

```
to module ( construct term )
```

Xcerpt Query Modules

- A *module* is a set of rules with interfaces
  - Interfaces define reusable query services

- Define a module:

```
MODULE <name> <rule>*
```

- Define *input* and *output* interfaces:

```
public <query>; public <cterm>
```

- Import a module:

```
IMPORT <location> AS <name>
```

- Query module:

```
IN <module> ( <query> )
```

- Provision module:

```
TO <module> ( <cterm> )
```

```
IMPORT file:subclassof.mx AS mod

GOAL   vehicles [ all var Sub ]
FROM
    IN mod (
        output [[
            subclassof [ var Sub,
                         "Vehicle" ]
    ]] )
END

CONSTRUCT
    TO mod (
        input [
            subclassof [
               var Sub, var Sup ]
    ] )
FROM
    in { resource {
            "file:...", "xml" },
            <query> }
END
```

Reusable module, in file:subclassof.mx

```
MODULE subclassof-reasoner

CONSTRUCT
    public output [
        all subclassof [ var Sub, var Sup ] ]
FROM  subclassof-deriv [ var Sub, var Sup ]
END

CONSTRUCT
    subclassof-deriv [ var Cls, var Cls ]
FROM
    or { subclassof [ var Z, var Cls ],
         Subclassof [ var Cls, var Z ] }
END

CONSTRUCT
    subclassof-deriv [ var Sub, var Sup ]
FROM
  or { subclassof [ var Sub, var Sup ],
    and { subclassof [ var Sub, var Z ],
          subclassof-deriv [ var Z, var Sup ]
  } }
END

CONSTRUCT subclassof [ var Sub, var Sub ]
FROM
    public input [[
       subclassof [ var Sub, var Sup ] ]]
END
```

Result:

```
vehicles [
"Vehicle", "Two-wheeler", "Motorcycle"
]
```

→ = data flow