



Domain-specific Languages

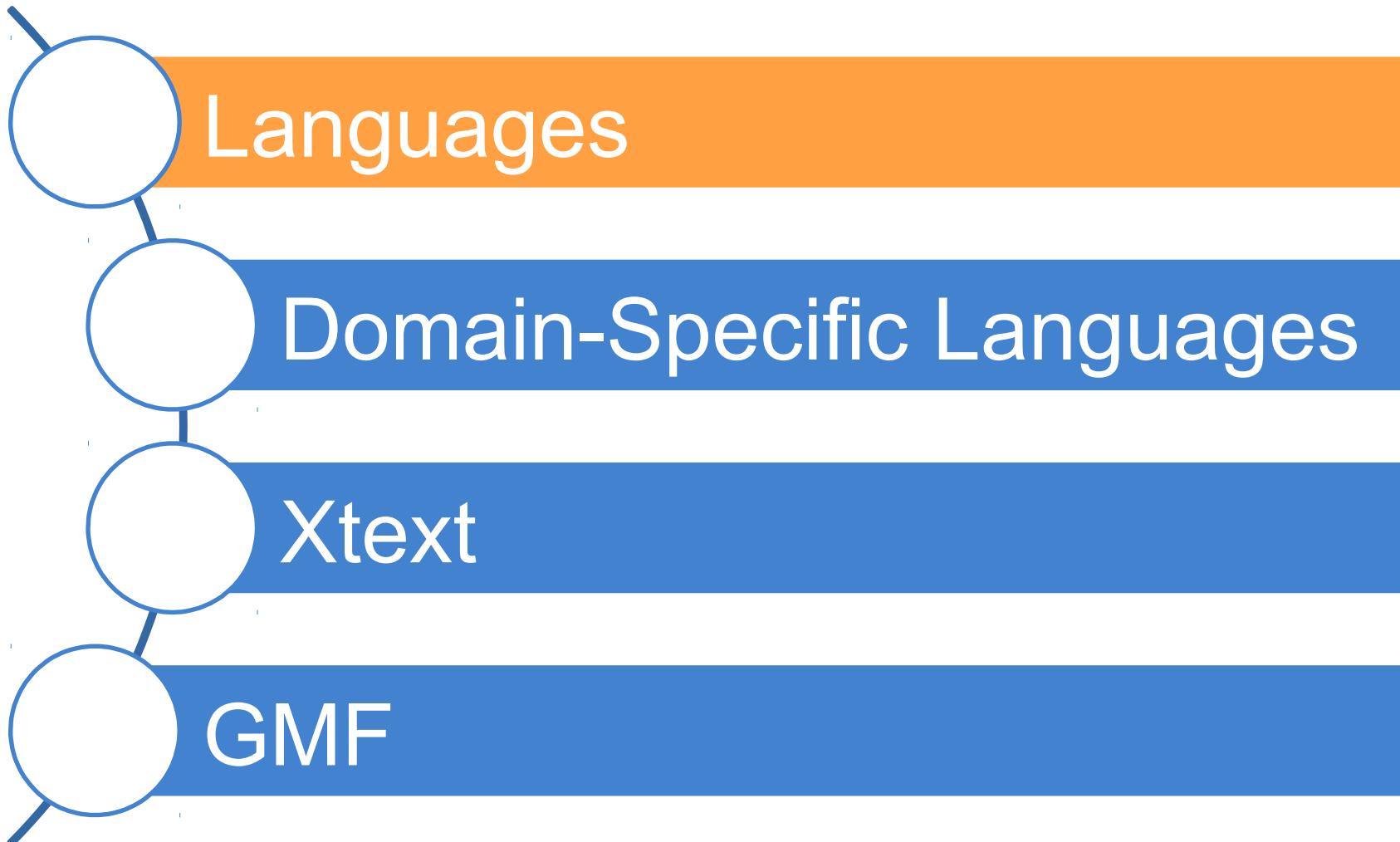
Javier Luis Cánovas Izquierdo
javier.canovas@inria.fr

February 2013

Outline



Outline



What is a language

sh-tion (lang'yun) n.
Langtry (lang'trē), Lily
Charlotte Le Breton, a
language (lang'gwij) n.
communication of emotion
human beings by means of
ing, the sounds spoken or
tematized and con-
given people over
ission of

What is a language?

From Wikipedia:

Language is the human capacity for acquiring and using complex systems of communication, and a language is any specific example of such a system.

What is a language?

From Wikipedia:

Language is the human capacity for acquiring and using complex systems of communication, and a language is any specific example of such a system.

Why?

How?

What is a language?

From Wikipedia:

Language is the human capacity for acquiring and using complex systems of communication, and a language is any specific example of such a system.

Why?

How?

What is a language?

From Wikipedia:

Language is the human capacity for acquiring and using complex systems of communication, and a language is any specific example of such a system.

Why?

How?

What is a language?



What is a language?

From the formal language theory:

A language is a set of linguistic utterances, (i.e., words), also called strings or sentences, which are composed by a set of symbols of an alphabet.

A grammar is the set of structural rules that governs the composition of linguistic utterances

What is a language?

From the formal language theory:

A language is a set of linguistic utterances, (i.e., words), also called strings or sentences, which are composed by a set of symbols of an alphabet.

A grammar is the set of structural rules that governs the composition of linguistic utterances

Why?

How?

What is a language?

From the formal language theory:

A language is a set of linguistic utterances, (i.e., words), also called strings or sentences, which are composed by a set of symbols of an alphabet.

A grammar is the set of structural rules that governs the composition of linguistic utterances

Why?

How?

What is a language?

From the formal language theory:

A language is a set of linguistic utterances, (i.e., words), also called strings or sentences, which are composed by a set of symbols of an alphabet.

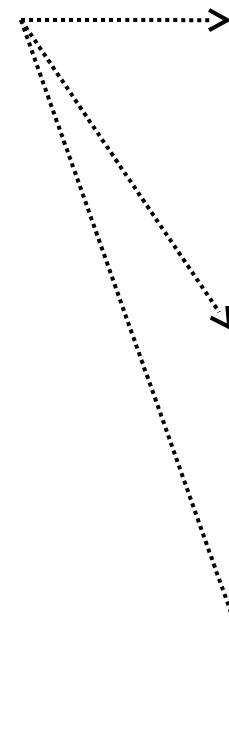
A grammar is the set of structural rules that governs the composition of linguistic utterances

Why?

How?

A grammar?

```
machineDefinition:  
  MACHINE OPEN_SEP stateList  
  transitionList CLOSE_SEP;  
  
stateList:  
  state (COMMA state)*;  
  
state:  
  ID_STATE;  
  
transitionList:  
  transition (COMMA transition)*;  
  
transition:  
  ID_TRANSITION OPEN_SEP  
  state state CLOSE_SEP;  
  
MACHINE: 'machine';  
OPEN_SEP: '{';  
CLOSE_SEP: '}';  
COMMA: ',';  
ID_STATE: 'S' ID;  
ID_TRANSITION: 'T' (0..9)+;  
ID: (a..zA..Z_) (a..zA..Z0..9)*;
```



```
machine {  
  S0ne STwo  
  T1 { S0ne  
  STwo }  
}
```

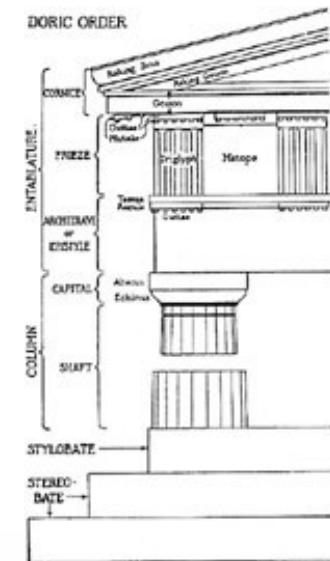
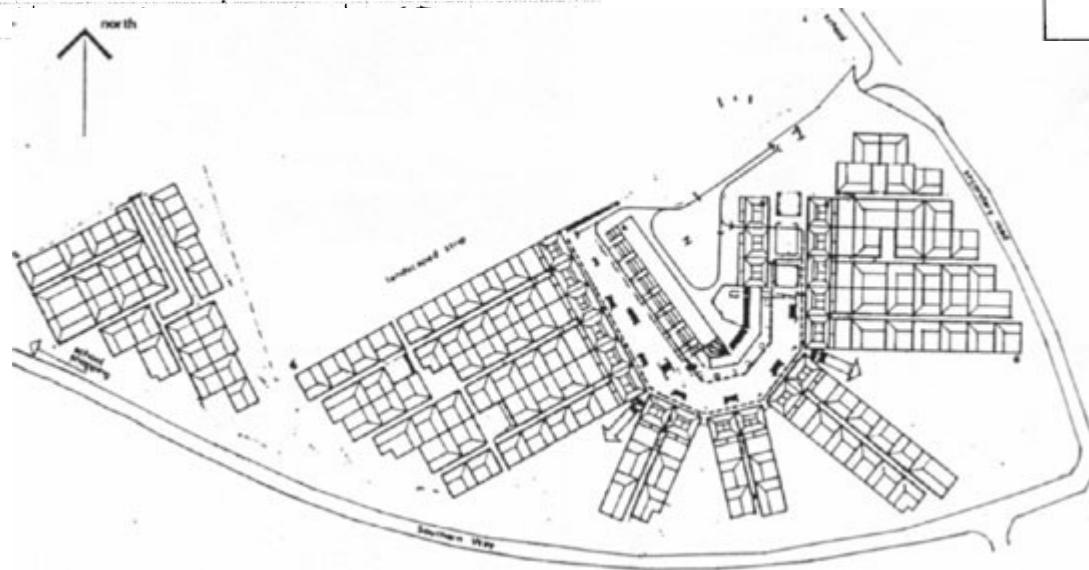
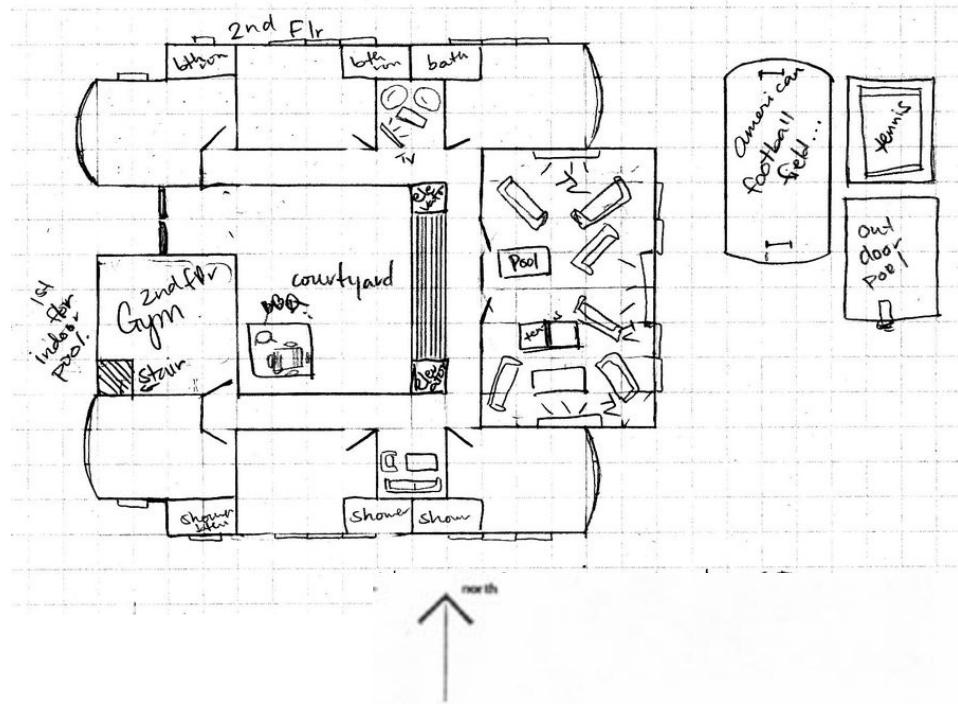
```
machine {  
  S_first S_second  
  T1 { S_first S_second}  
  T1 { S_second S_first }  
}
```

```
machine {  
  SA SB SC SD  
  T1 { SA SD }  
  T2 { SA SB }  
  T3 { SB SC }  
  T4 { SB SA }  
}
```

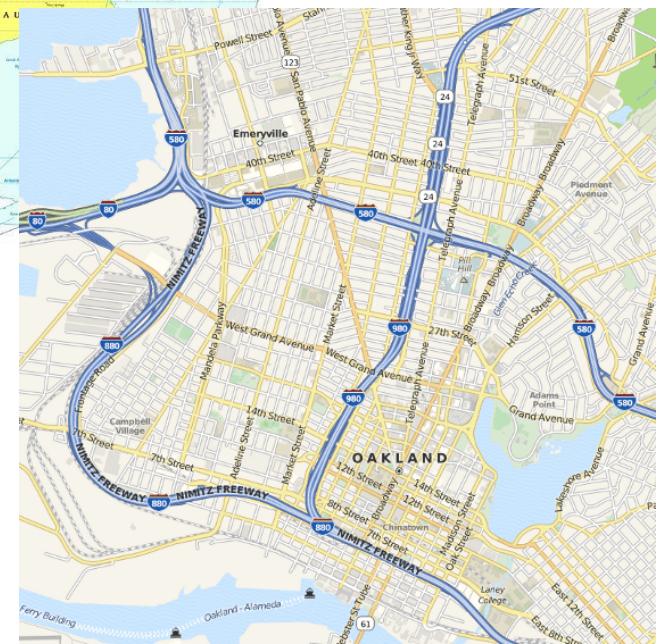
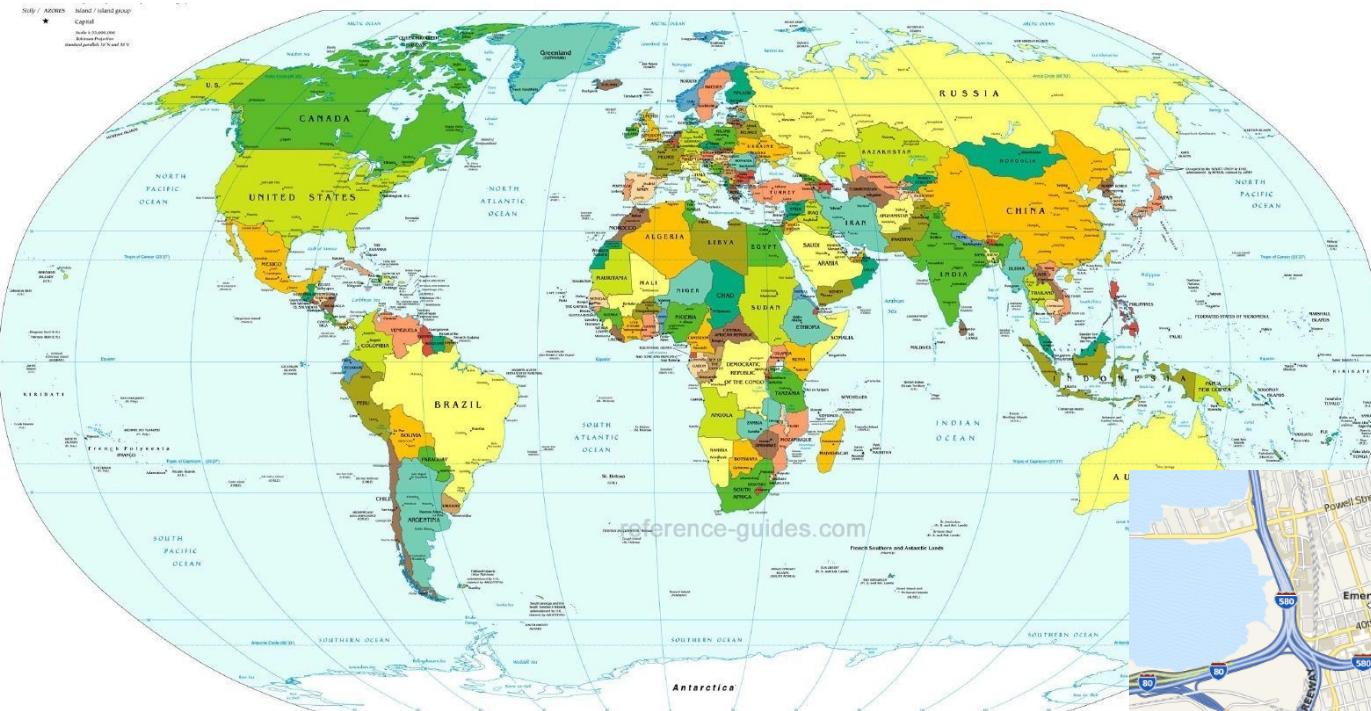
Languages in engineering



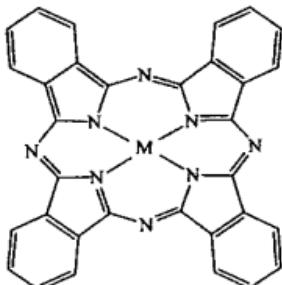
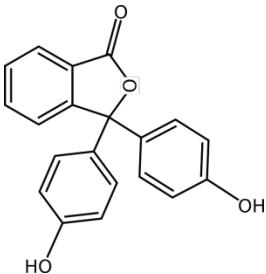
Architecture



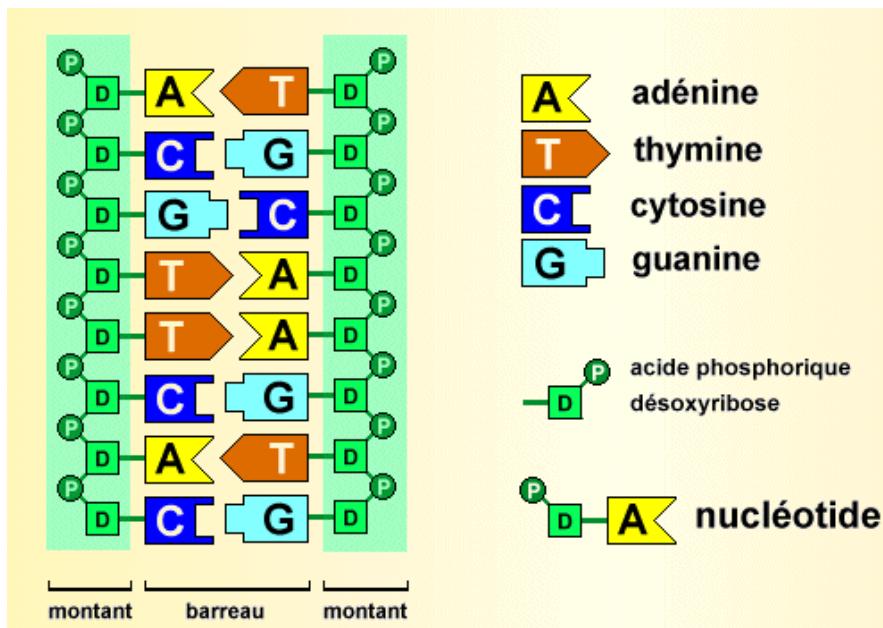
Cartography



Biology



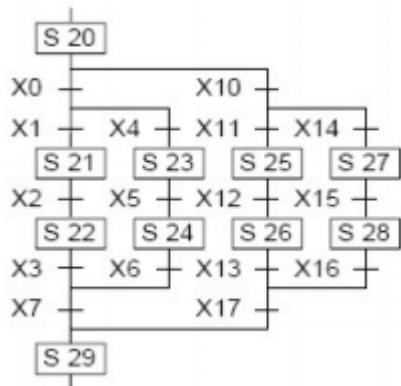
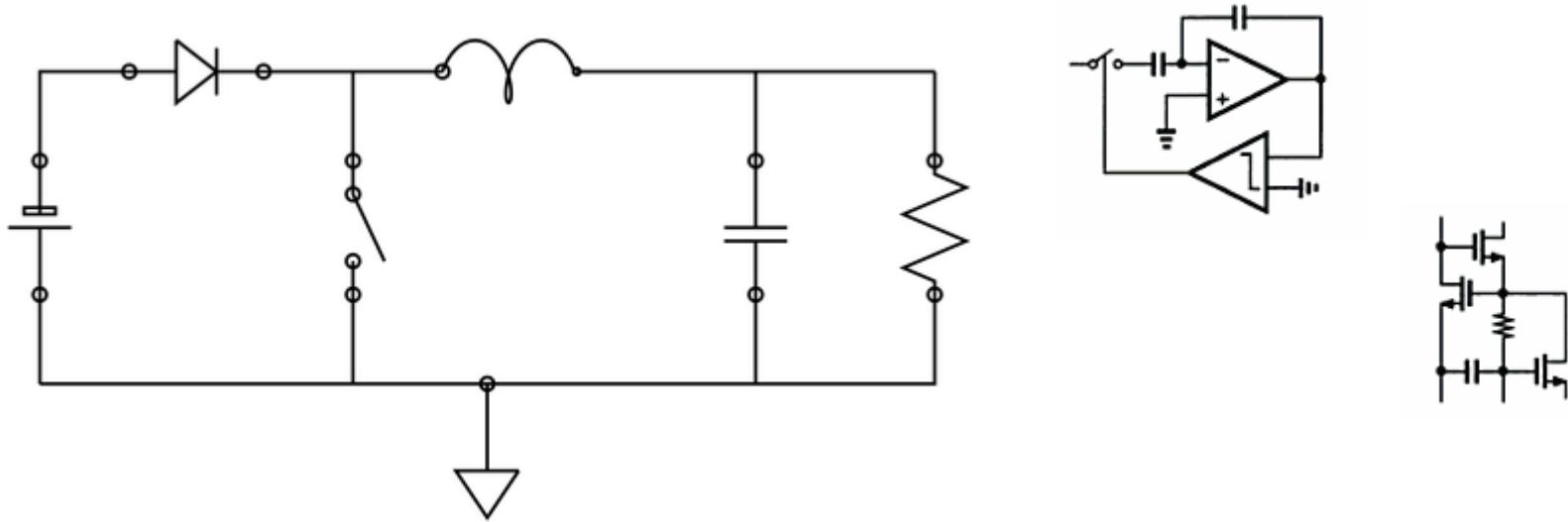
phthalocyanine



60	70	80	90	100
AGACCCCCAG	CAACCCCCGG	GGCCGTGCGG	CCTCGGTCGT	GTCGTGTGAT
160	170	180	190	200
AGACCCCGCG	TACGAATGCC	GGTCCACCAA	CAACCCGTGG	GCTTCGCAGC
260	270	280	290	300
CTGCCGGGCA	TGTACAGTCC	TTGTCGGCAG	TTCTTCCACA	AGGAAGACAT
360	370	380	390	400
GGCTTGCTGG	GGCCCCCGCC	ACCAGCACTA	CAGACCTCCA	GTACGTCGTG
460	470	480	490	500
GGCCTATCCC	ACGCTCGCCG	CCAGCCACAG	AGTTATGCTT	GCCGAGTACA
560	570	580	590	600
GAAGAGGTGG	CGCCGATGAA	GAGACTATT	AAGCTGGAA	ACAAGGTGGT
660	670	680	690	700
ATAGTGGTTA	ACTTCACCTC	CAGACTCTTC	GCTGATGAAC	TGGCCGCCCT
760	770	780	790	800
AAAATATACA	GGCATTTGGC	CTGGGGTGCG	TATGCTCACG	TGAGACATCT
860	870	880	890	900
CCTGGAGGAG	GTTCGCCCCG	ACAGCCTGCG	CCTAACGCGG	ATGGATCCCT
960	970	980	990	1000
AGCAACACCC	AGCTAGCAGT	GCTACCCCCA	TTTTTTAGCC	GAAAGGATTC
1060	1070	Pvu II site	1090	1100
TGCCCGCAGCA	ACTGGGGCAC	GCTATTCTGC	AGCAGCTGTT	GGTGTACCA
1160	1170	1180	1190	1200
ACTTGATCTA	TATACCCACCA	ATGTGTCATT	TATGGGGCCG	ACATATCGTC
1260	1270	1280	1290	1300
CTGTCATGT	ACCTTTGTAT	CCTATCAGCC	TTGGTTCCCA	GGGGGTGTCT
1360	1370	1380	1390	1400
TGTTTGAGGG	GGTGGTGCCTA	GATGAGGTGA	CCAGGATAGA	TCTCGACCAG
1460	1470	1480	1490	1500
TCAGAGTCCTC	AGTTCTATAT	TTAACCTTGG	CCCCAGACTG	CACGTGTATG
1560	1570	1580	1590	1600
CGATTTGAAG	CGGGGGGGGT	ATGGCGTCAT	CTGATATTCT	GTGGGTTGCA
1660	1670	1680	1690	1700
AAAAACTTACC	GTCTACCTGC	CGGACACTGA	ACCCCTGGGTG	GTAGAGACCG
1760	1770	1780	1790	1800
AAGCTTCATC	GTGGTGCCT	GCCCTCAAAT	TCTCACAAAG	GCTTGAGGAT

CTG.

Electronics



A black and white photograph of a server room. The perspective is looking down a central aisle between two rows of server racks. The racks are filled with hardware and are illuminated from within, with many small lights visible along the front panels. The ceiling above is made of a grid of tiles, with several rectangular recessed light fixtures. The floor is a polished, light-colored material. The overall atmosphere is one of a high-tech, industrial environment.

Languages in Computer Science

GPLs

- Java

```
class HelloWorldApp {  
    public static void main(String[] args) {  
        System.out.println("Hello World!"); }  
}
```

- C

```
#include <stdio.h>  
int main(void)  
{  
    printf("hello, world\n");  
}
```

- Python

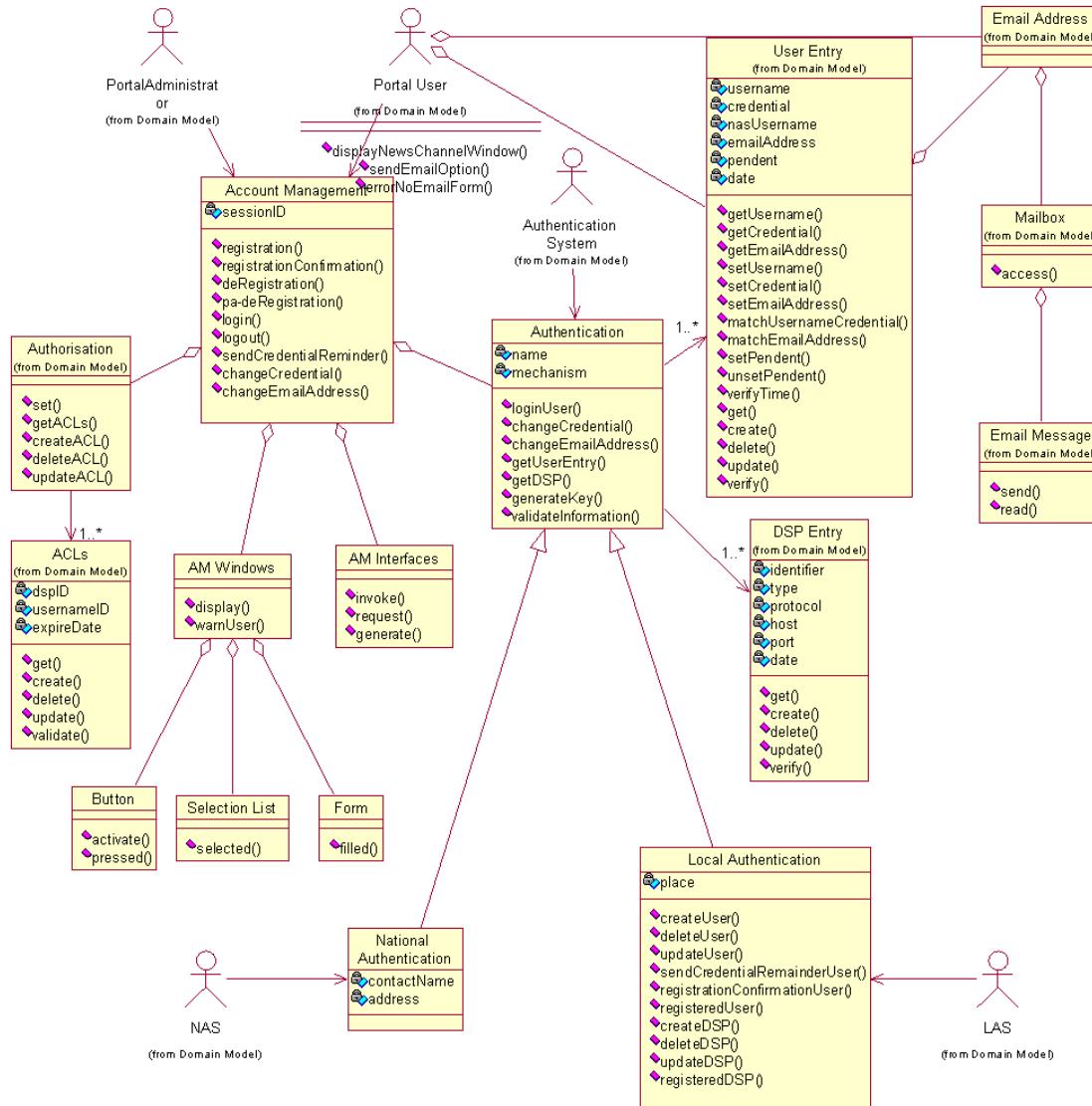
```
print("Hello world")
```

- Delphi

```
program ObjectPascalExample;  
type  
    THelloWorld = object  
        procedure Put;  
    end;  
var  
    HelloWorld: THelloWorld;  
procedure THelloWorld.Put;  
begin  
    WriteLn('Hello, World!');  
end;  
begin  
    New(HelloWorld);  
    HelloWorld.Put;  
    Dispose(HelloWorld);  
end.
```

GPLs

- UML



DSLs

- SQL

```
CREATE TABLE Employee (
    id INT NOT NULL IDENTITY (1,1) PRIMARY KEY,
    name VARCHAR(50),
    surname VARCHAR(50),
    address VARCHAR(255),
    city VARCHAR(60),
    telephone VARCHAR(15),
)
```

- CSS

```
body {
    text-align: left;
    font-family: helvetica, sans-serif;
}
h1 {
    border: 1px solid #b4b9bf;
    font-size: 35px;}
```

- HTML

```
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <p>Example</p>
  </body>
</html>
```

- LaTeX

```
\ifthenelse{\boolean{showcomments}}
{\newcommand{\nb}[2]{%
  \fcolorbox{gray}{yellow}{%
    \bfseries\sffamily\scriptsize#1
  }%
  {\sf\small\textrit{#2}}%
}%
\newcommand{\version}{\scriptsize$-\$working$-$}
}
\newcommand{\nb}[2]{%
\newcommand{\version}{\scriptsize$-\$working$-$}
}
\newcommand{\version}{\scriptsize$-\$working$-$}
```

DSLs

- OCL

```
employees->includes(#dent)
self.questions->size
self.employer->size
self.employee->select (v | v.wages>10000 )->size
Student.allInstances
->forAll( p1, p2 |
    p1 <> p2 implies p1.name <> p2.name )
```

- ATL

```
unique lazy rule customFuncCall {
    from
        src : TupleType(class1 : MM!EClass,
                        class2 : MM!EClass)
    to
        t : FOL!CustomFuncCall(
            name <- src.class1.name,
            declaration <- src.class1,
            arguments <- Sequence{thisModule.variable(src)}
        )
}
```

- MiniUML

```
package Courses {
    class Student {
        attribute studentid : number key
        attribute firstName : string
        attribute lastName : string
    }
}

class Course {
    attribute name : string key
    attribute description : string optional
    attribute ects: number
}
```



GPLs

vs



DSLs

GPLs vs DSLs

A *GPL* provides notations that are used to describe a computation in a human-readable form that can be translated into a machine-readable representation.

A *GPL* is a formal notation that can be used to describe problem solutions in a precise manner.

A *GPL* is a notation that can be used to write programs.

A *GPL* is a notation for expressing computation.

A *GPL* is a standardized communication technique for expressing instructions to a computer. It is a set of syntactic and semantic rules used to define computer programs.

GPLs vs DSLs



The boundary isn't as clear as it could be. Domain-specificity is not black-and-white, but instead gradual: a language is more or less domain specific

GPLs vs DSLs



The boundary isn't as clear as it could be. Domain-specificity is not black-and-white, but instead gradual: a language is more or less domain specific

	GPLs	DSLs
Domain	large and complex	smaller and well-defined
Language size	large	small
Turing completeness	always	often not
User-defined abstractions	sophisticated	limited
Execution	via intermediate GPL	native
Lifespan	years to decades	months to years (driven by context)
Designed by	guru or committee	a few engineers and domain experts
User community	large, anonymous and widespread	small, accessible and local
Evolution	slow, often standardized	fast-paced
Deprecation/incompatible changes	almost impossible	feasible

Outline

DSL

DSL

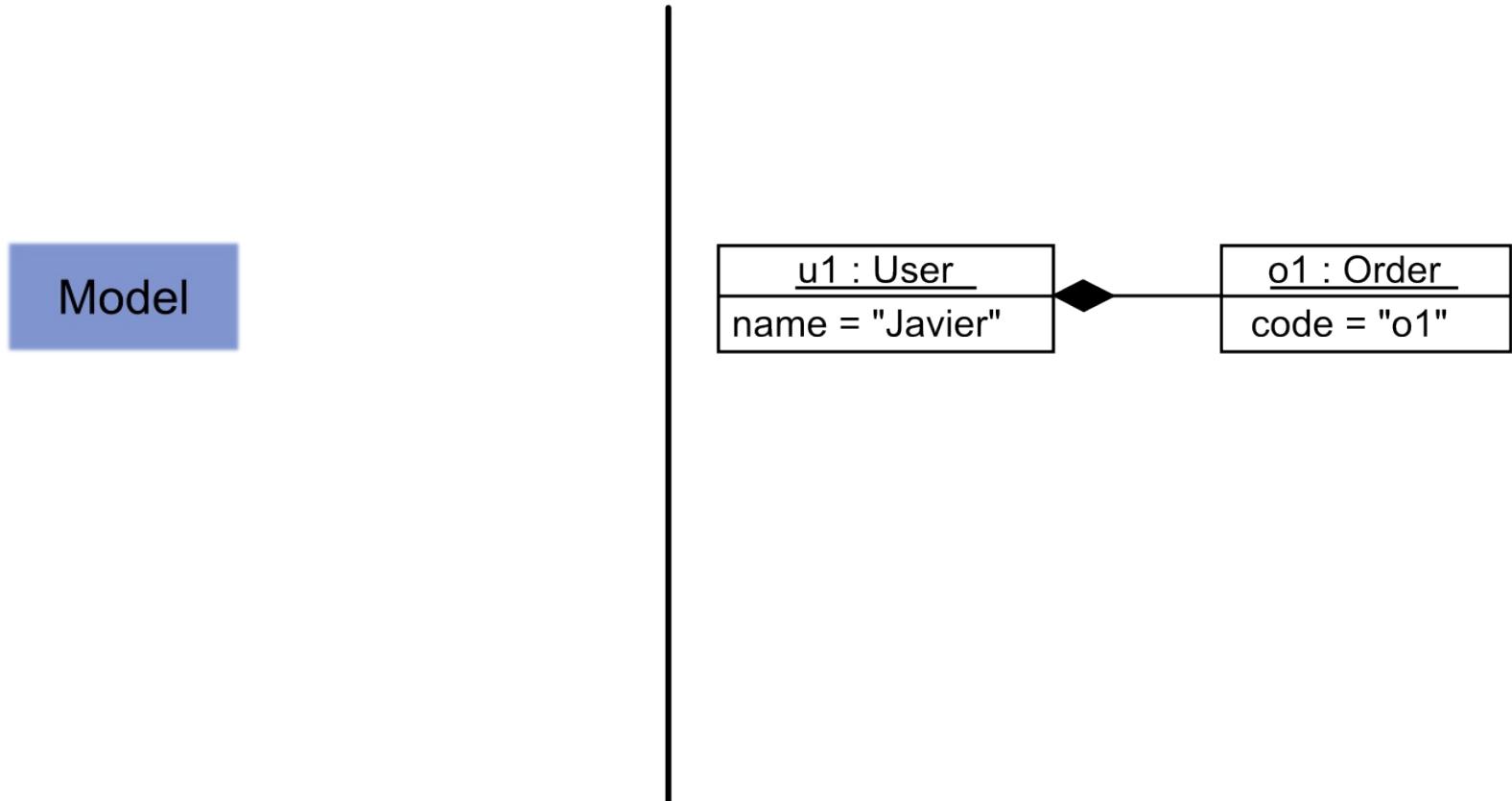
The big picture

A word cloud diagram centered around the acronym **MDE** (Model-Driven Engineering). The words are colored in shades of blue, representing different concepts and tools in the field:

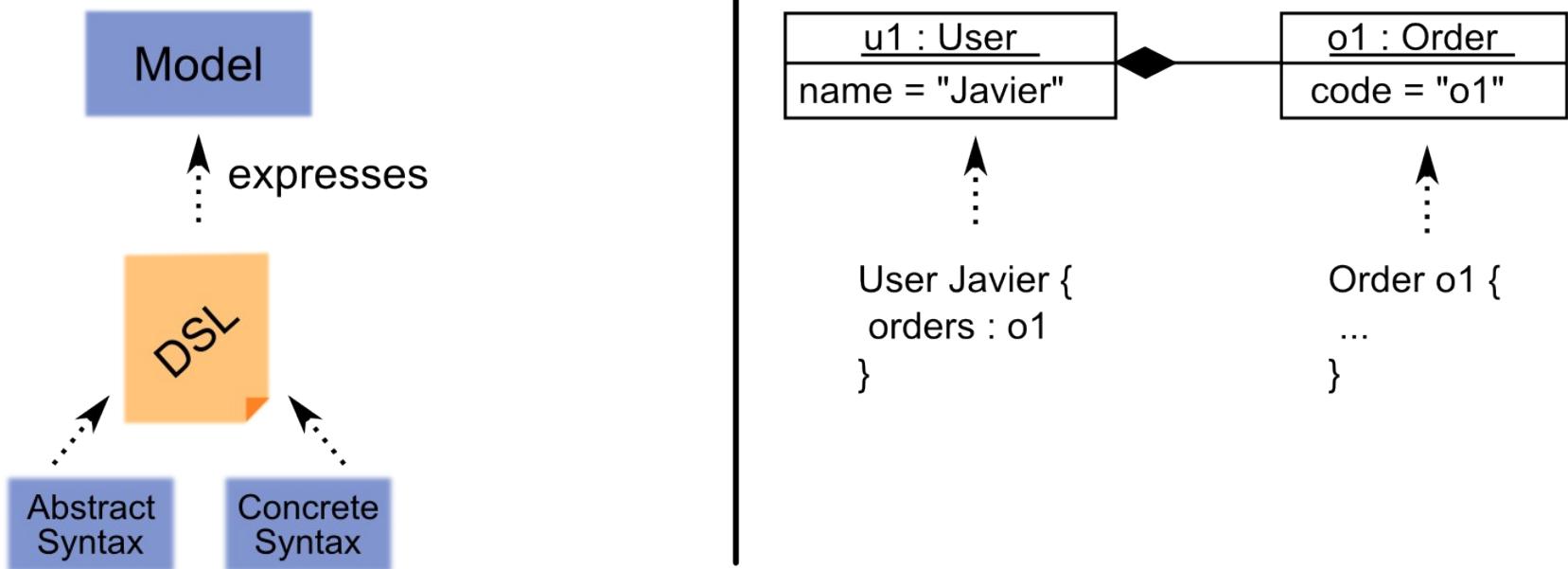
- application domain**: Eclipse best practices, developers foundation, automation CASE basis, UML design effective tools, application modelling tools.
- OMG approach**: Rumbaugh, modeling definitions, Mellor reuse, Booch product managers.
- models**: executable semantics, systems, teams.
- MDA**: process designers, increase productivity.
- MDE**: implementing systems, Jacobson standardized users.
- compatibility**: standardization, problem space.
- standardization**: design patterns, domain.
- problem space**: architecture-focused, simplifying, higher levels of abstraction, extensive communication.
- design patterns**: modeling paradigm, Object Management Group.
- domain**: software development, Unified Modeling Language.
- higher levels of abstraction**: modeling paradigm, Object Management Group.
- extensive communication**: software development, Unified Modeling Language.

Domain-Specific Languages?

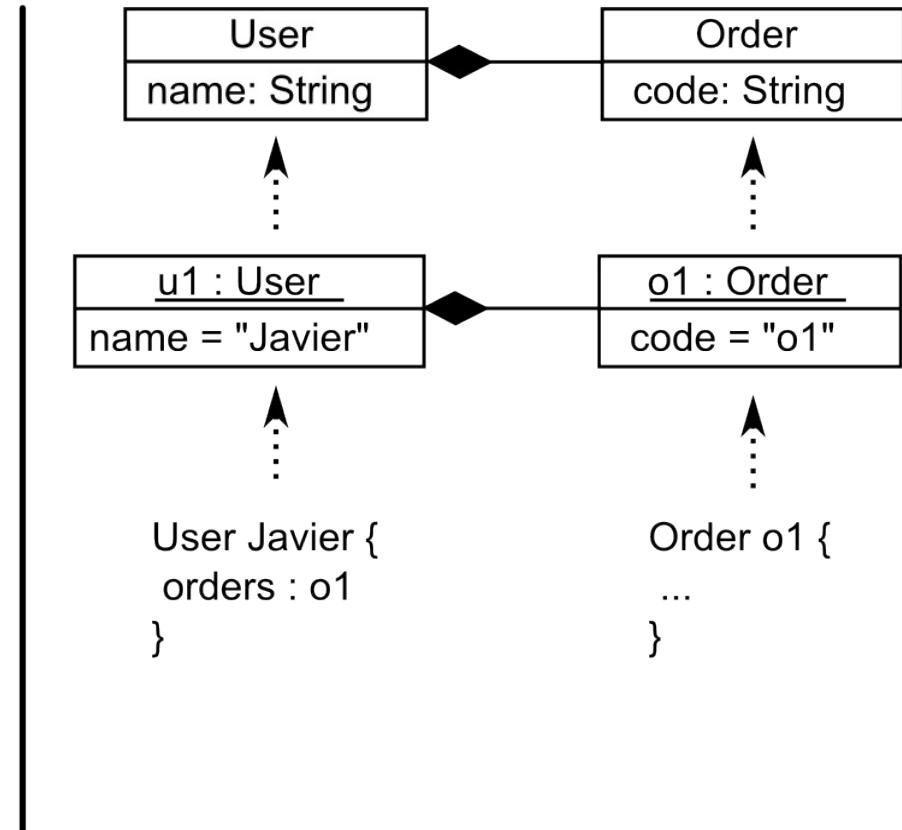
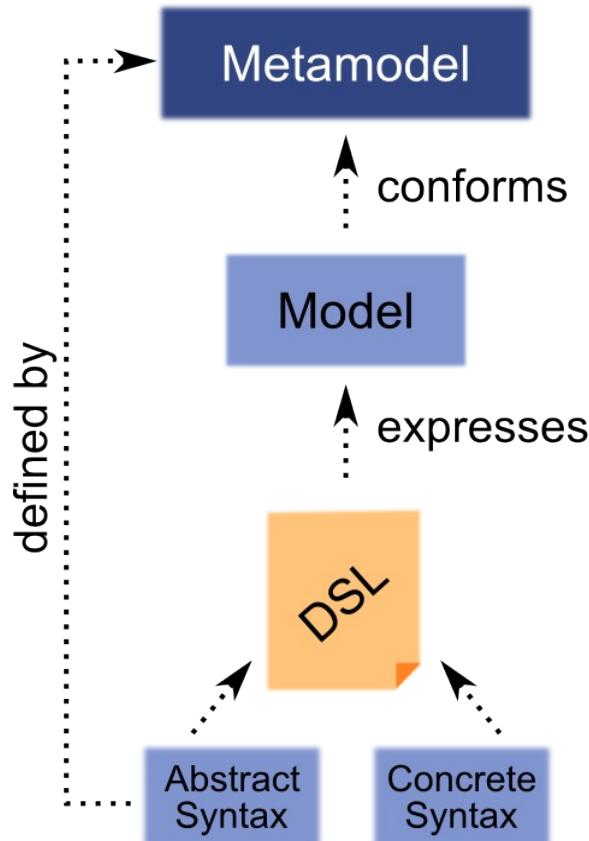
Concepts



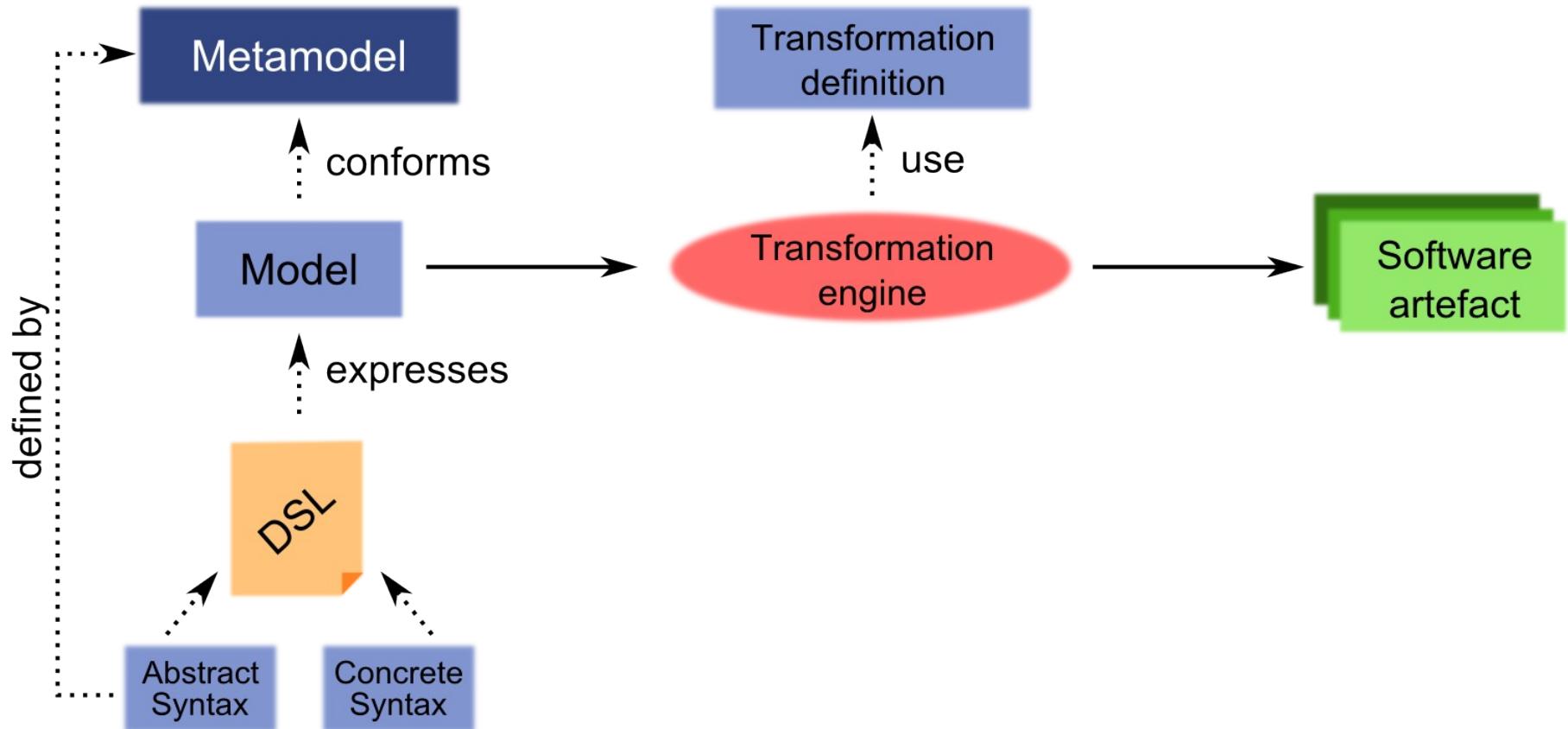
Concepts



Concepts



Concepts



DSL

What is a DSL

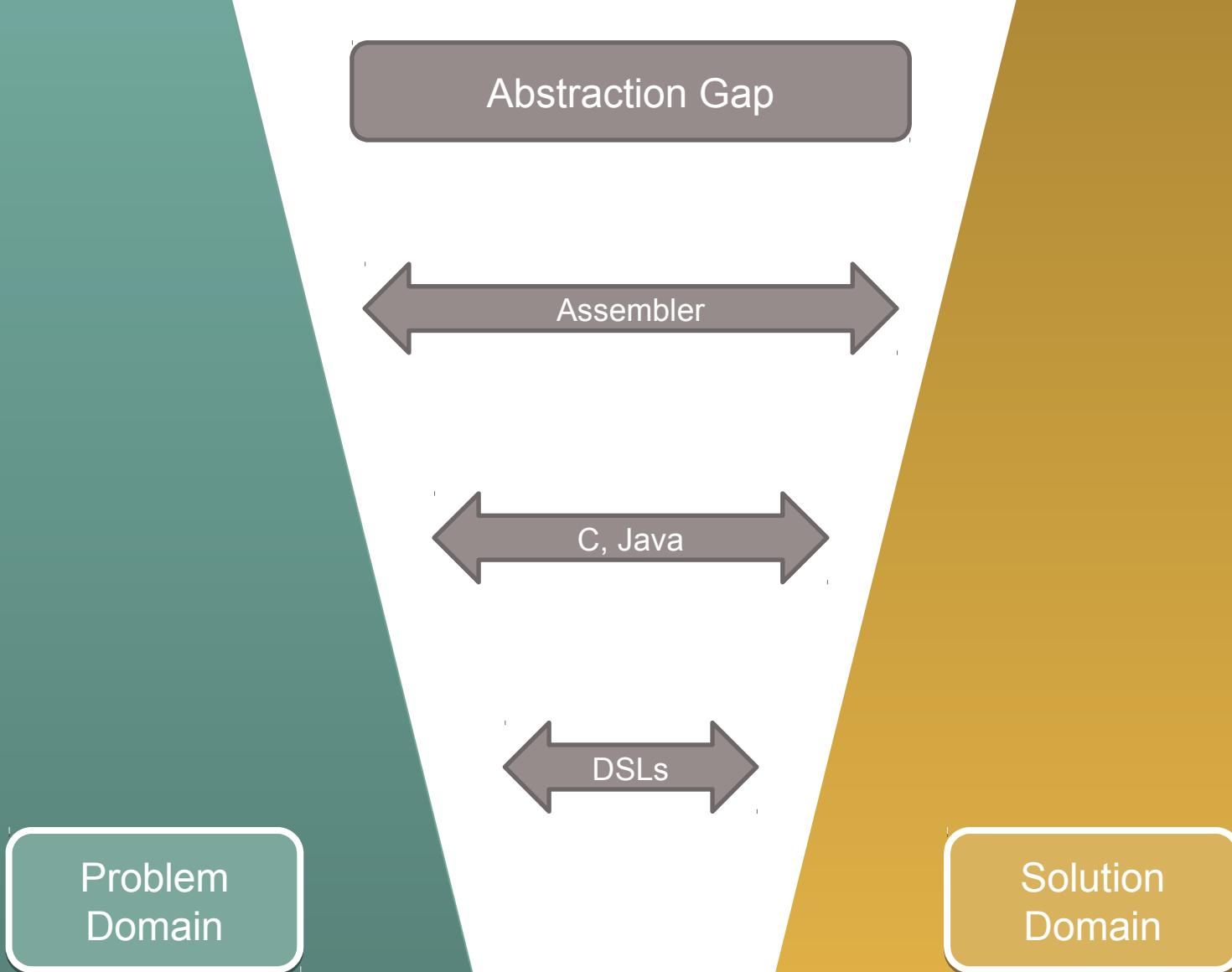
- In general:
Language specially designed to perform a task in a certain domain

What is a DSL

- In general:
Language specially designed to perform a task in a certain domain
- In the context of computer science:
A formal processable language targeting at a specific viewpoint or aspect of a software system.
Its semantics flexibility and notation is designed in order to support working with that viewpoint as good as possible

And the rest of languages?

Why DSLs?



What is offered?

What are the advantages?

What for?

- Generation of interfaces
- Generation of WSDL
- Set up of skeleton projects

What for?

- Architecture-Definition Languages
- AUTOSAR

What for?

- Model transformation languages: ATL, Acceleo
- Constraints language: OCL

What for?

- Describe a cooling algorithm in refrigerators
- Configure hearing aids
- Insurance mathematics

What for?

- Pseudo-structured natural languages

What for?

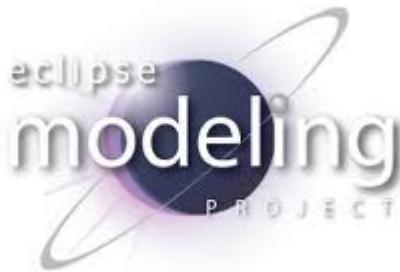
- Mathematical formalisms

What for?

- Composition languages
- Workflow languages

What is a domain?

but... how?



Forward by Sam Guckenheimer



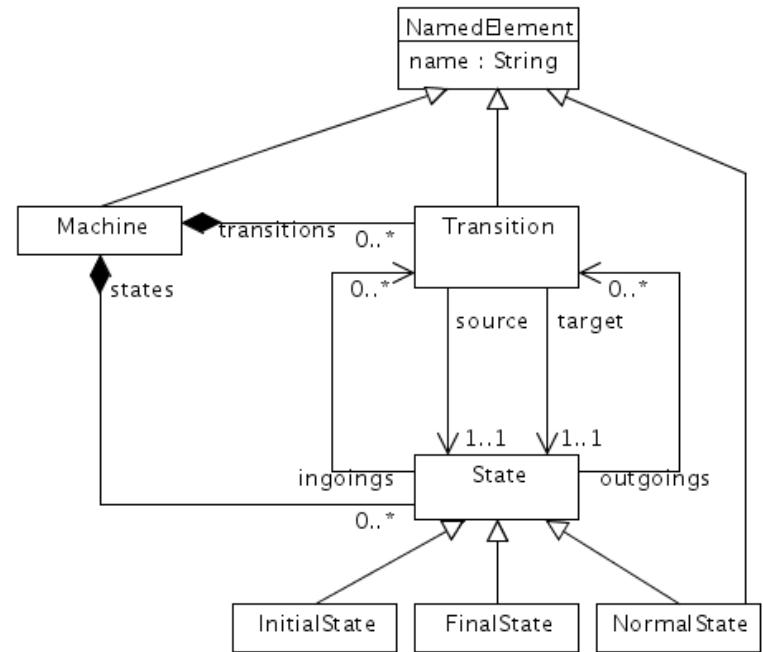
Domain-Specific Development with Visual Studio DSL Tools



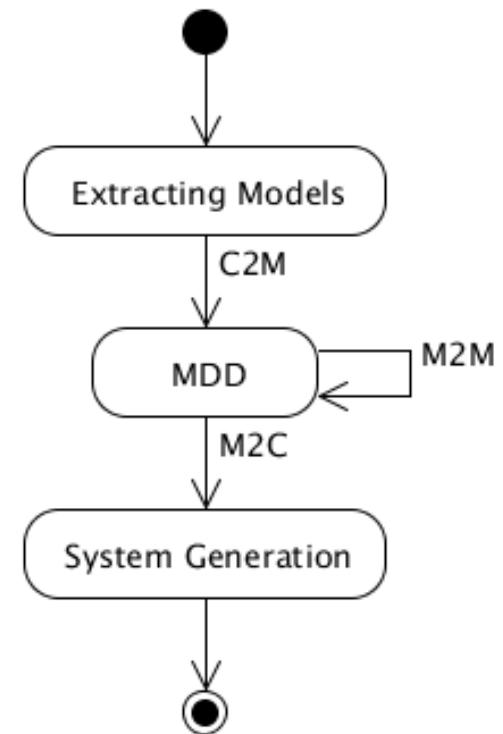
DSL

Anatomy

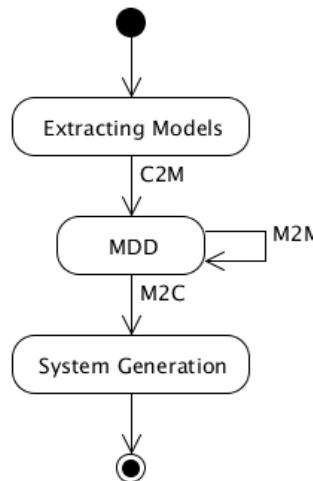
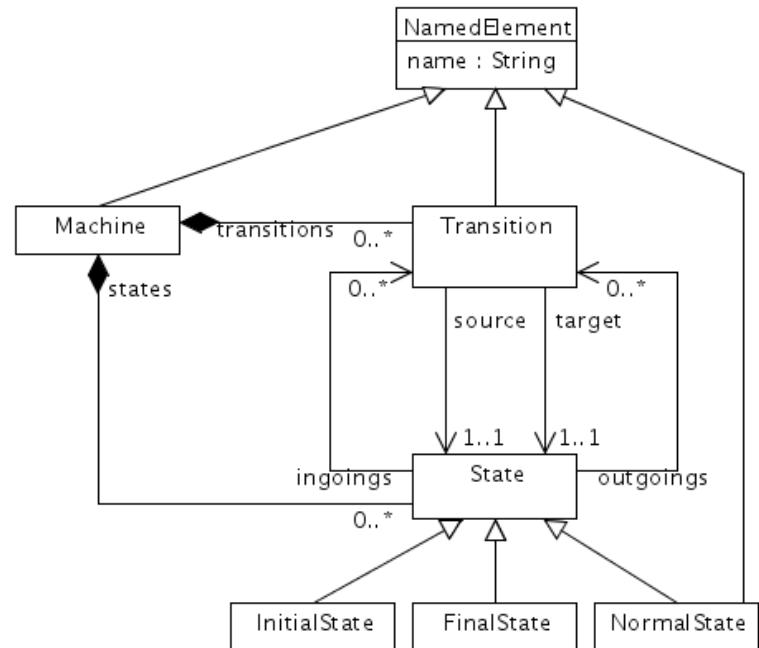
A first look



A first look



A first look



How this is
executed?

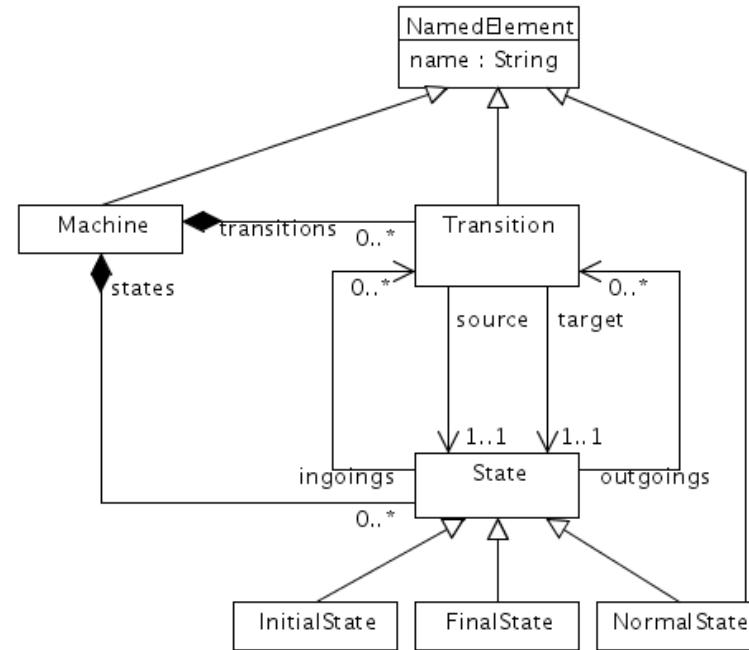
Anatomy

Abstract Syntax

- Metamodelling techniques
 - UML notation
 - Concept => Class
 - Property => Attribute
 - Containment Relationship => Composite Association
 - Relationship => Association
 - Specialization => Inheritance
 - Package => Package
- Well-formed rules
 - OCL expressions

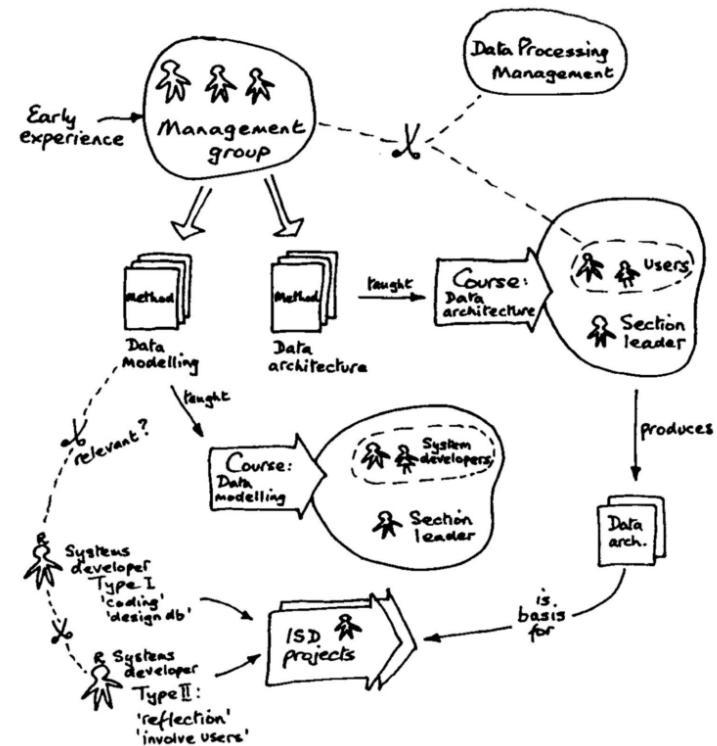
```
context Transition
inv: self.source <> self.target
```

```
context Machine
inv: self.states
    ->exists(s |
s.oclIsKindOf(InitialState))
```



Anatomy

Concrete Syntax



Concrete Syntax

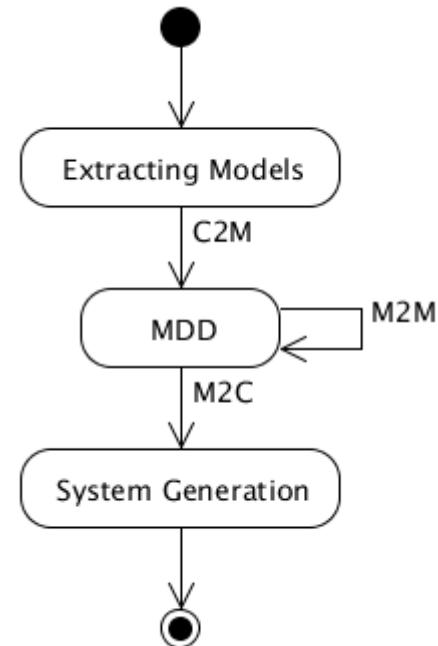


Concrete Syntax

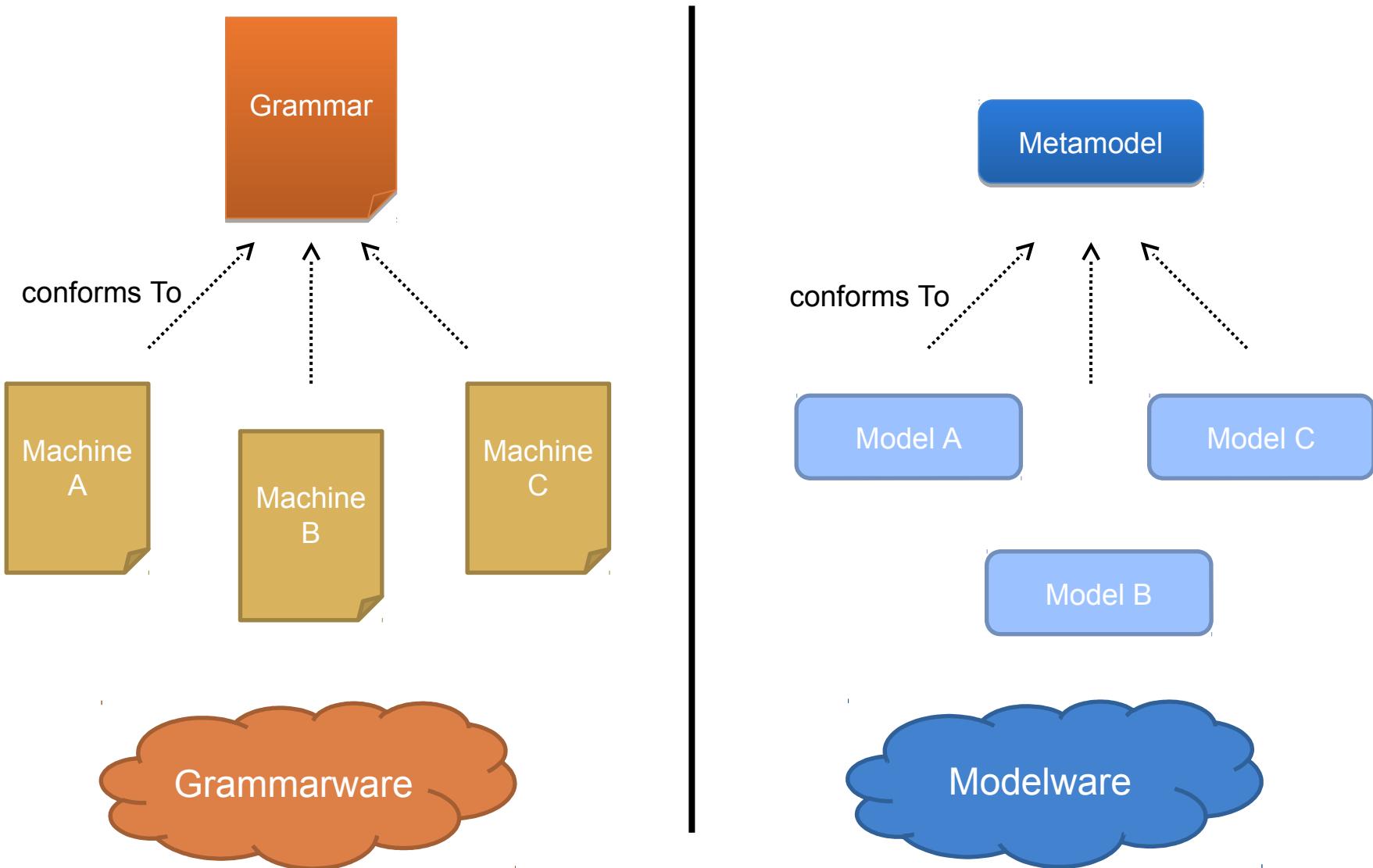
- Textual

```
digraph Modernization {  
    Start -> ExtractingModels;  
    ExtractingModels -> MDD [label="C2M"];  
    MDD -> MDD [label="M2M"];  
    MDD -> SystemGeneration [label="M2C"];  
    SystemGeneration -> End;  
  
    Start  
        [shape=point, width=0.2, label=""];  
    ExtractingModels  
        [label="Extracting Models", shape=ellipse];  
    MDD  
        [shape=ellipse];  
    SystemGeneration  
        [label="System Generation", shape=ellipse];  
    End  
        [shape=doublecircle, label="", width=0.2,  
         fillcolor=black, style=filled];  
}
```

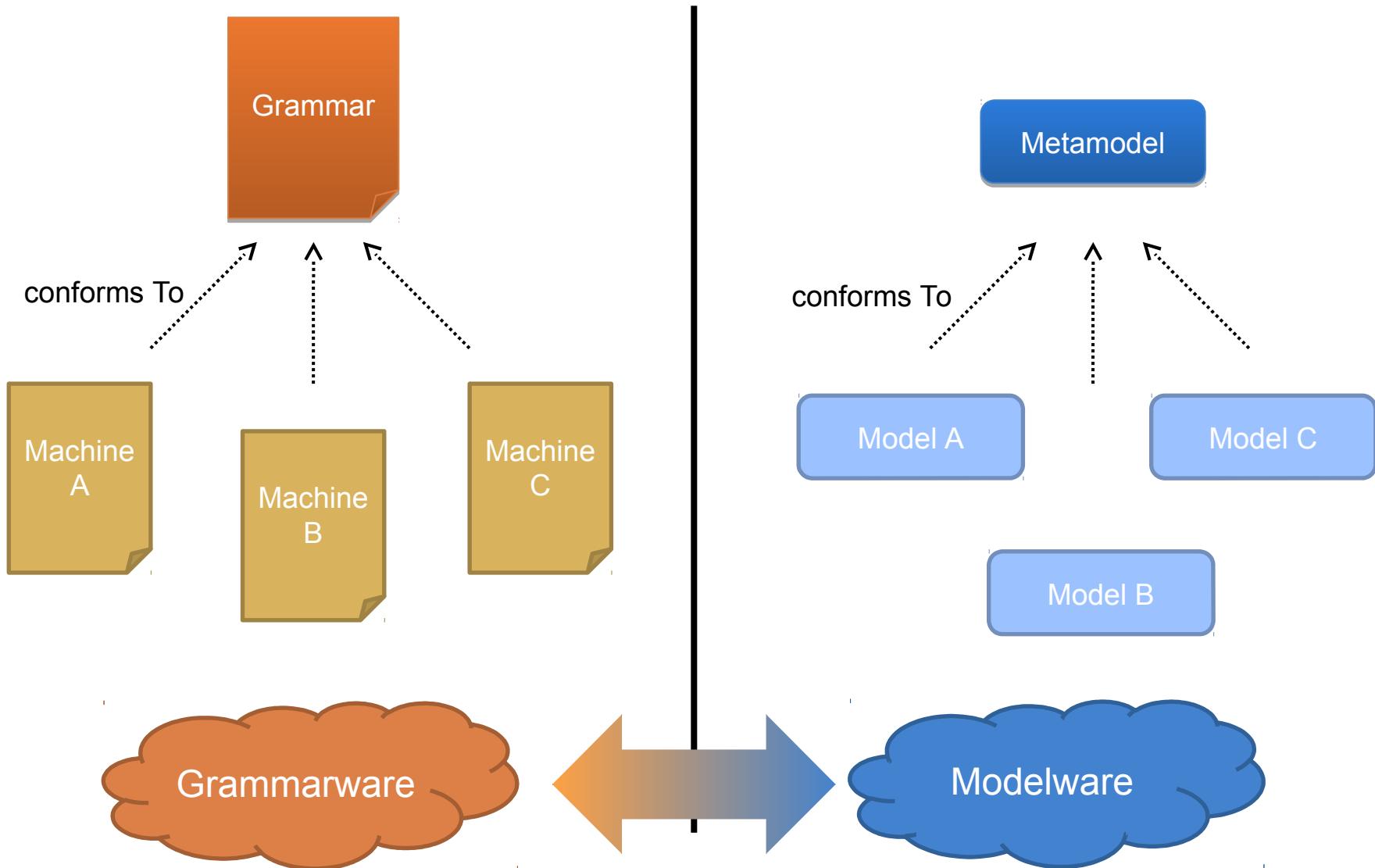
- Graphical



Textual Syntax



Textual Syntax



Textual Syntax

```
machineDefinition:  
  MACHINE OPEN_SEP stateList  
  transitionList CLOSE_SEP;  
  
stateList:  
  state (COMMA state)*;  
  
state:  
  ID_STATE;  
  
transitionList:  
  transition (COMMA transition)*;  
  
transition:  
  ID_TRANSITION OPEN_SEP  
  state state CLOSE_SEP;  
  
MACHINE: 'machine';  
OPEN_SEP: '{';  
CLOSE_SEP: '}';  
COMMA: ',';  
ID_STATE: 'S' ID;  
ID_TRANSITION: 'T' (0..9)+;  
ID: (a..zA..Z_) (a..zA..Z0..9)*;
```

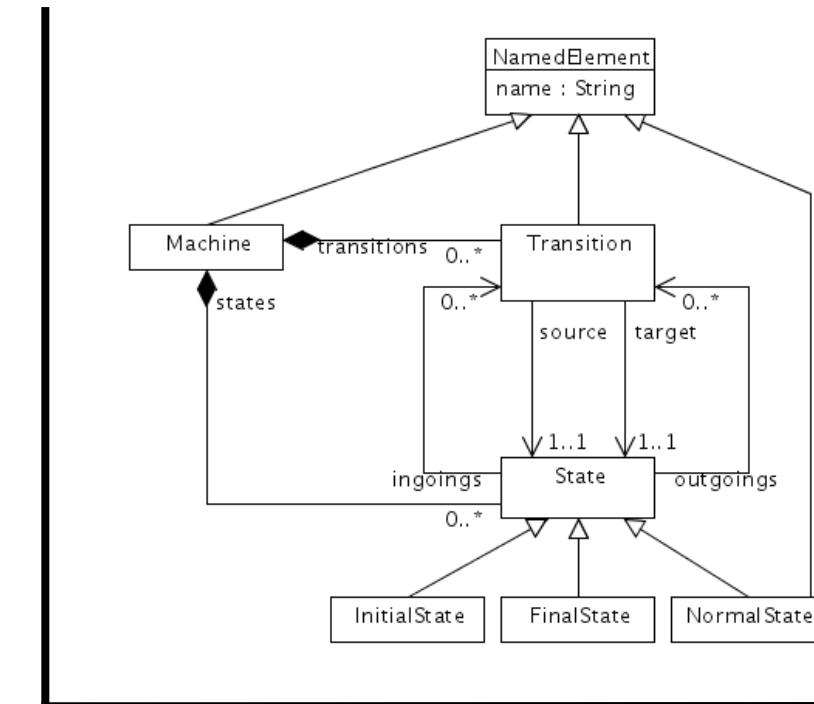
conforms To



```
machine {  
  SOne STwo  
  T1 { SOne  
  STwo }  
}
```

Textual Syntax

```
machineDefinition:  
  MACHINE OPEN_SEP stateList  
  transitionList CLOSE_SEP;  
  
stateList:  
  state (COMMA state)*;  
  
state:  
  ID_STATE;  
  
transitionList:  
  transition (COMMA transition)*;  
  
transition:  
  ID_TRANSITION OPEN_SEP  
  state state CLOSE_SEP;  
  
MACHINE: 'machine';  
OPEN_SEP: '{';  
CLOSE_SEP: '}';  
COMMA: ',';  
ID_STATE: 'S' ID;  
ID_TRANSITION: 'T' (0..9)+;  
ID: (a..zA..Z_) (a..zA..Z0..9)*;
```



conforms To

<-----

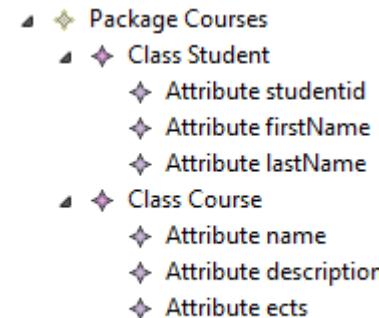
```
machine {  
  SOne STwo  
  T1 { SOne  
  STwo }  
}
```

You have already work with one

```
package Courses {  
    class Student {  
        attribute studentid : number key  
        attribute firstName : string  
        attribute lastName : string  
    }  
  
    class Course {  
        attribute name : string key  
        attribute description : string optional  
        attribute ects: number  
    }  
}
```

```
package Courses {  
    class Student {  
        attribute studentid : number key  
        attribute firstName : string  
        attribute lastName : string  
    }  
  
    class Course {  
        attribute name : string key  
        attribute description : string optional  
        attribute ects: number  
    }  
  
    association Enrolment {  
        end student : Student [1..*]  
        end course : Course [0..*]  
    }  
}
```

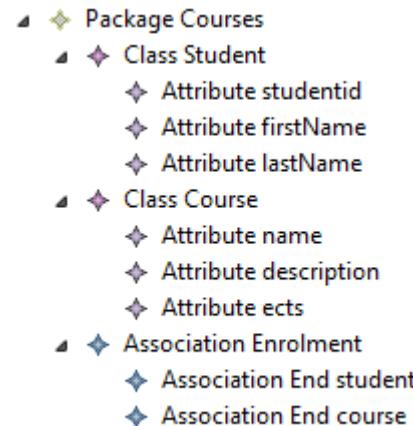
syntax of



conforms To

Ecore MM

syntax of

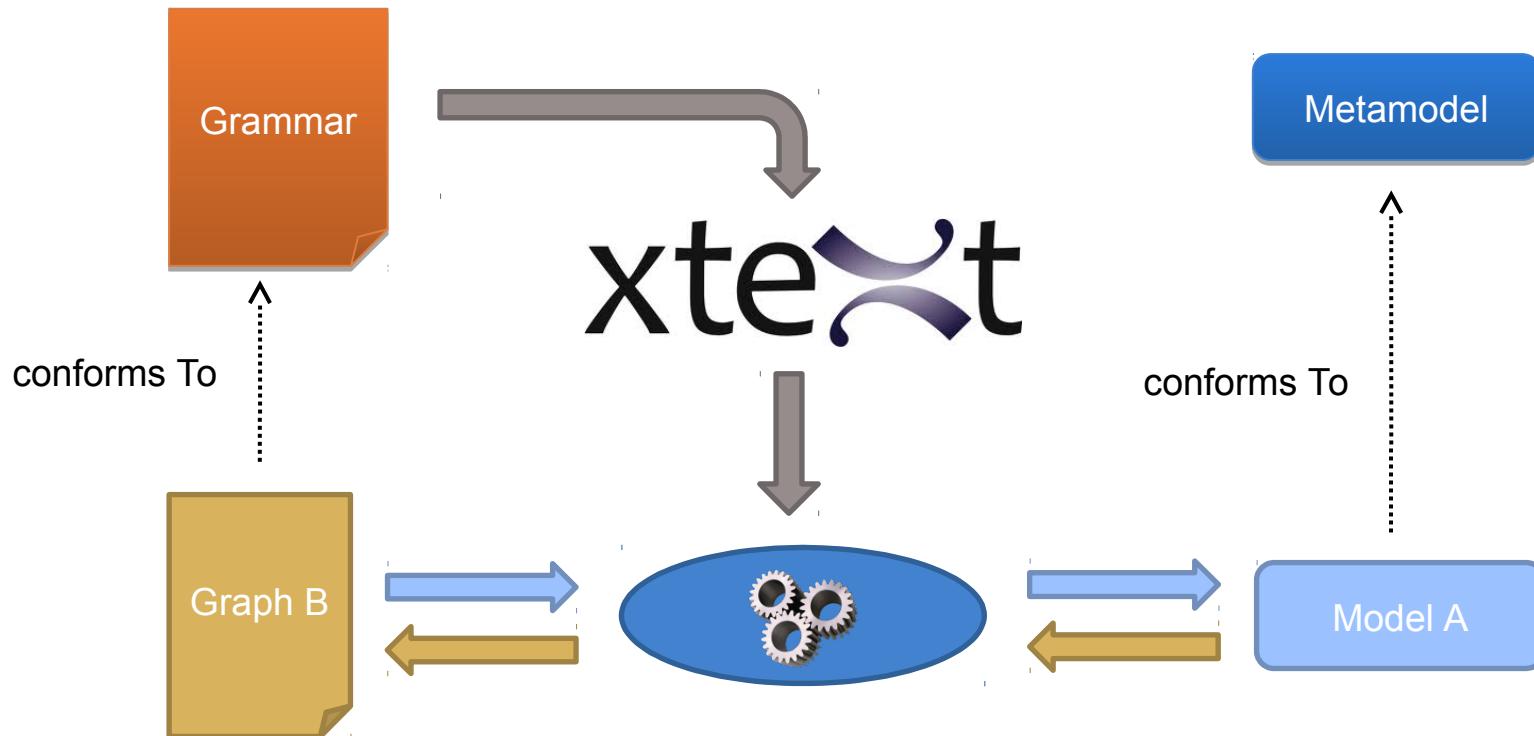


conforms To

Ecore MM

Textual Syntax

- Grammar-oriented (Xtext)

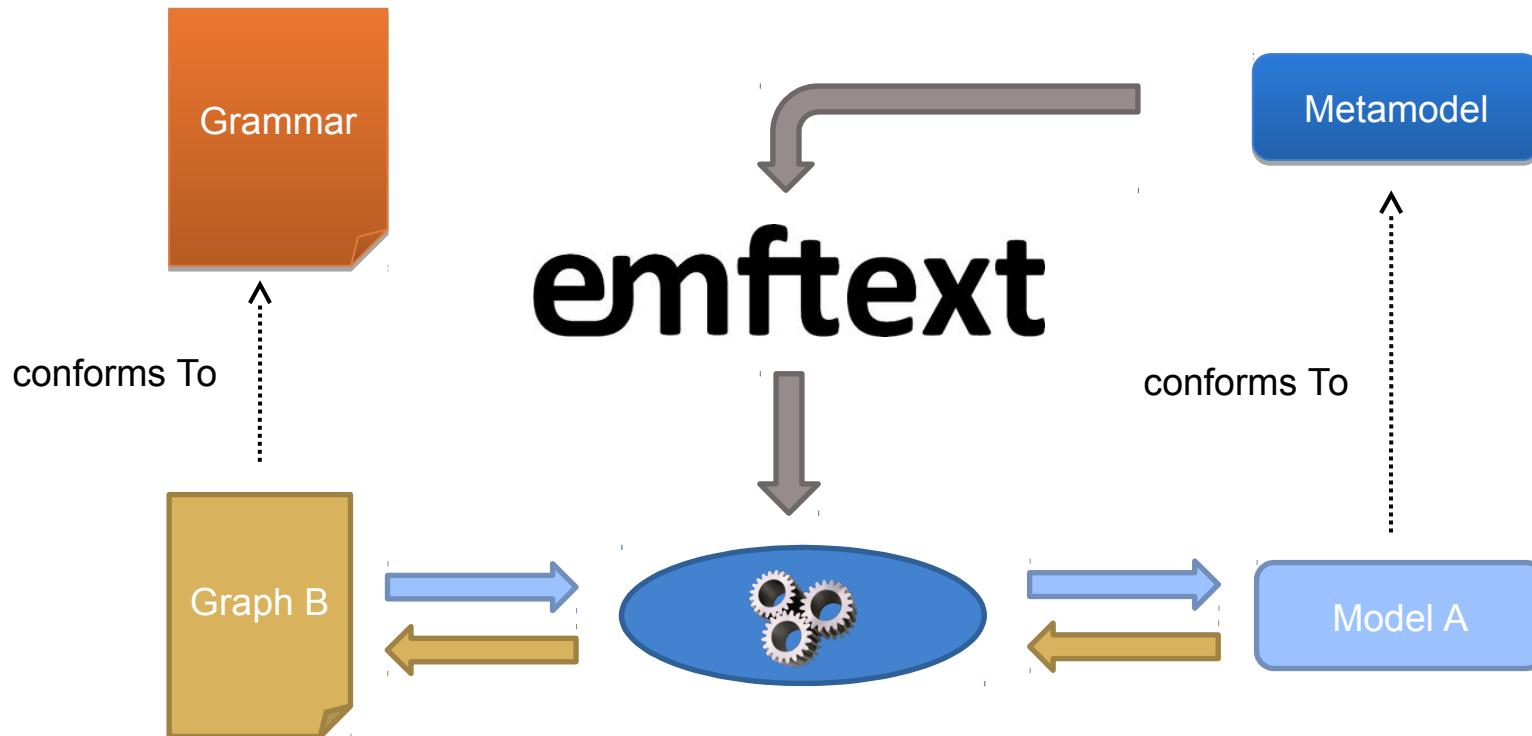


Xtext

- Model-Driven framework to develop textual DSLs
- Is part of Eclipse Modeling Project
- Is part of a DSL tooling initially called Open Architecture Ware
- Main features:
 - Syntax Coloring
 - Content Assist
 - Validation and Quick Fixes
 - Advanced Java Integration
 - Integration with other Eclipse tools

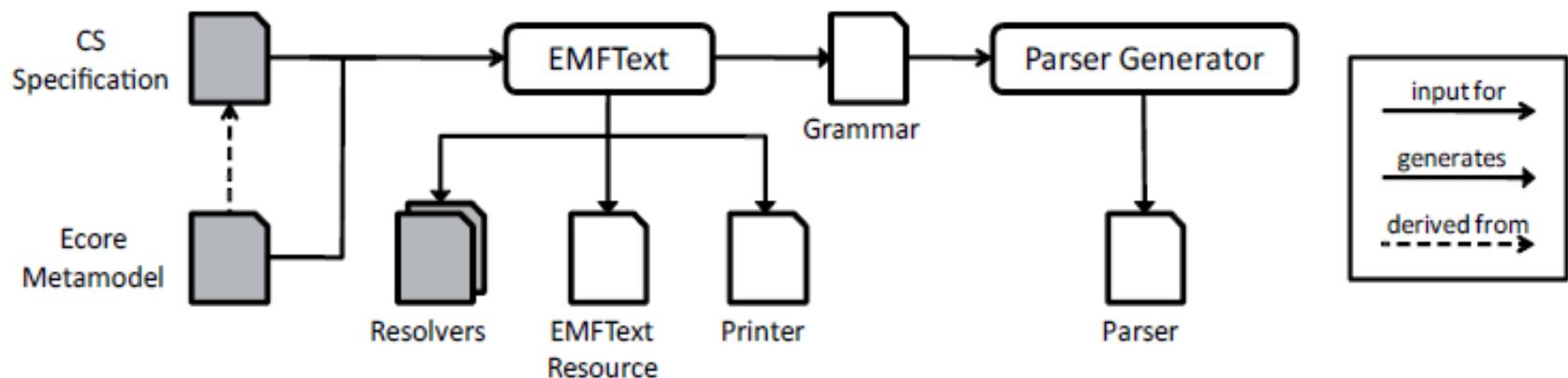
Textual Syntax

- Metamodel-oriented (EMFText)



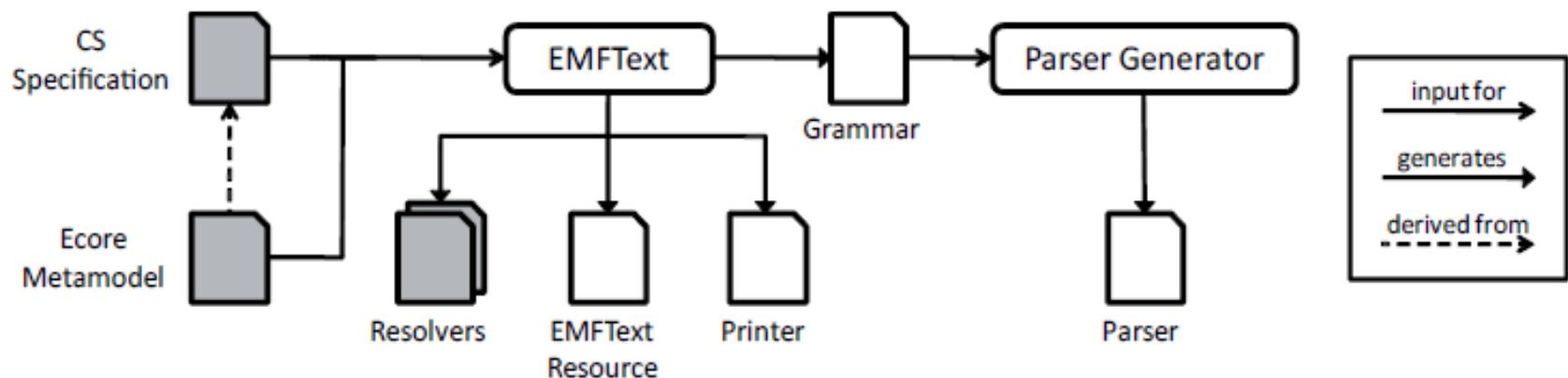
EMFText

- Model-Driven framework to develop textual DSLs
- Developed as a Eclipse plugin
- Main features:
 - Syntax Coloring
 - Content Assist
 - Validation and Quick Fixes
 - Reference resolving mechanism



EMFText

- Model-Driven framework to develop textual DSLs
- Developed as a Eclipse plugin
- Main features:
 - Syntax Coloring
 - Content Assist
 - Validation and Quick Fixes
 - Reference resolving mechanism

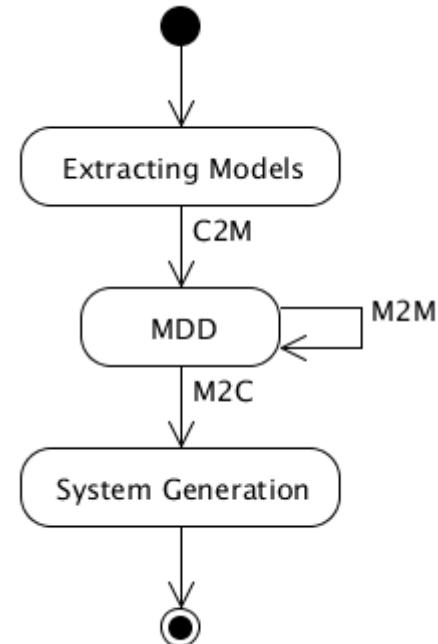


Graphical Syntax

A draw is better than a long explanation

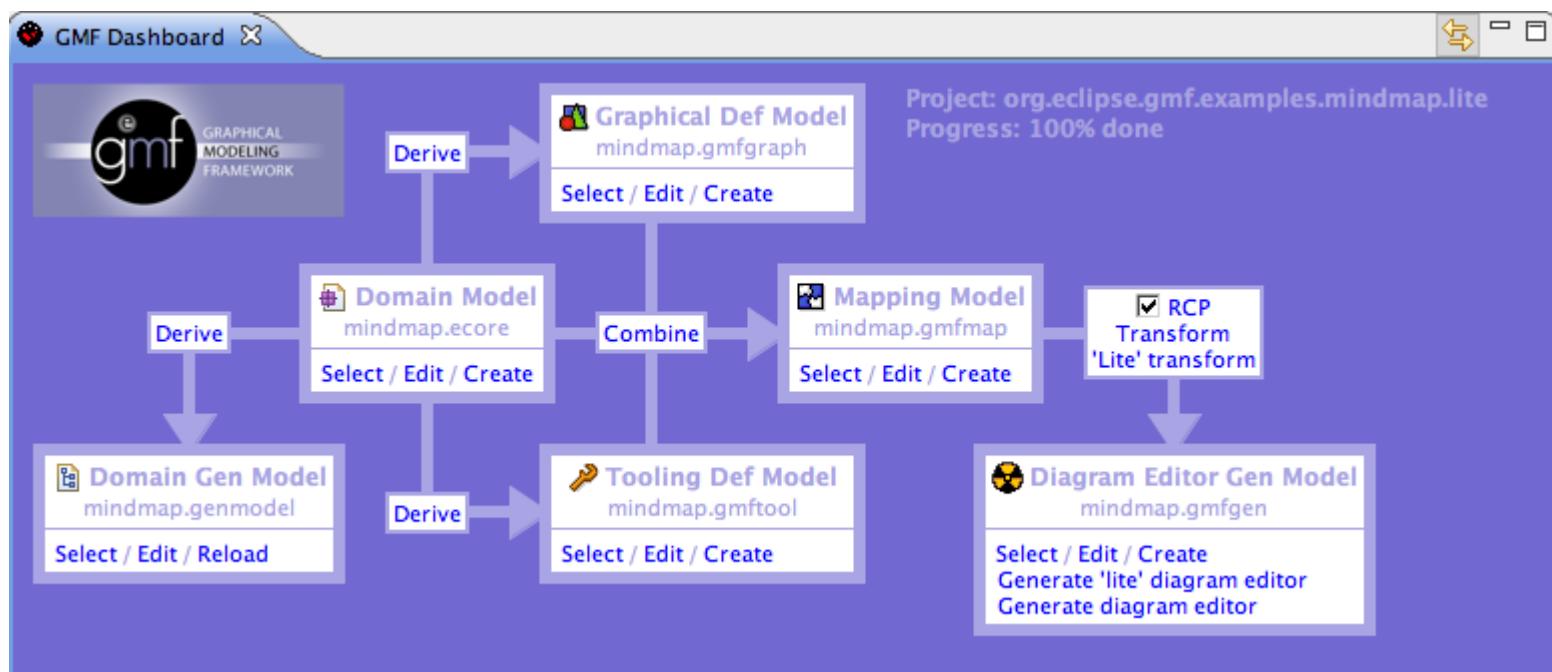
```
digraph Modernization {
    Start -> ExtractingModels;
    ExtractingModels -> MDD [label="C2M"];
    MDD -> MDD [label="M2M"];
    MDD -> SystemGeneration [label="M2C"];
    SystemGeneration -> End;

    Start
        [shape=point, width=0.2, label=""];
    ExtractingModels
        [label="Extracting Models", shape=ellipse];
    MDD
        [shape=ellipse];
    SystemGeneration
        [label="System Generation", shape=ellipse];
    End
        [shape=doublecircle, label="", width=0.2,
         fillcolor=black, style=filled];
}
```



GMF

- Graphical Modeling Framework
 - Model-Driven Framework to develop graphical editors based on EMF and GEF
 - GMF is part of Eclipse Modeling Project
 - Provides a generative component to create the DSL tooling
 - Provides a runtime infrastructure to facilitate the development of graphical DSLs

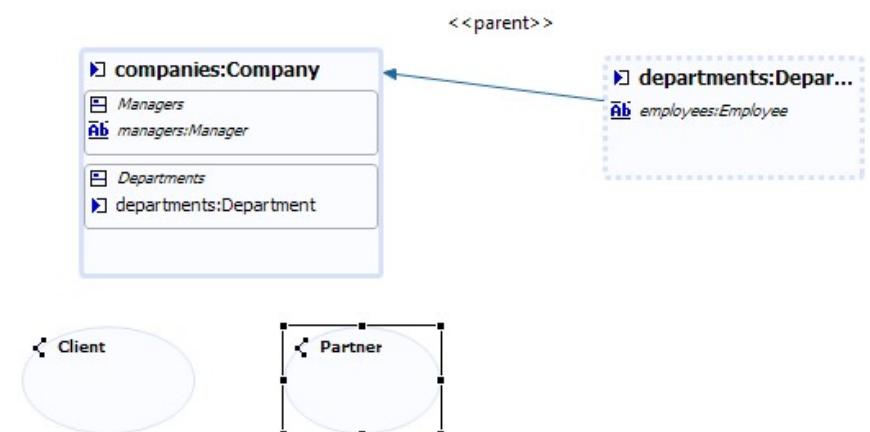


Alternatives

- GMF can be too complex
- EuGENia
 - Automatically generated GMF infrastructure
 - Lowers the entrance barrier for creating GMF editors
 - Provides textual DSLs and wizards to define the graphical DSL
- GMF simple mapping editor
 - Graphical DSL to configure GMF
 - Generators for creating the required GMF infrastructure

```
@namespace(uri="filesystem", prefix="filesystem")  
package filesystem;
```

```
@gmf.diagram  
class Filesystem {  
    val Drive[*] drives;  
    val Sync[*] syncs;  
}
```

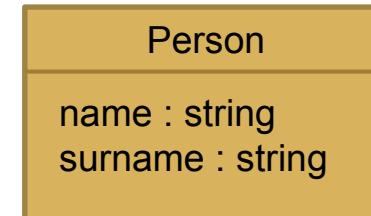


Graphical vs. Textual

- Success depends on how the notation fits the domain

```
class Person {  
    private String name;  
    private String name;  
}
```

```
Person has (name, surname)
```

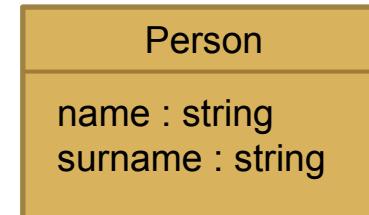


Graphical vs. Textual

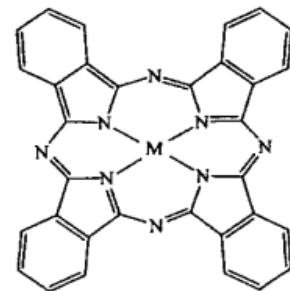
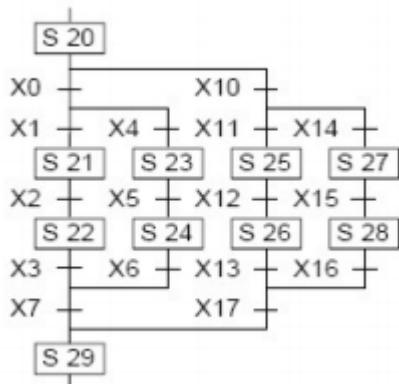
- Success depends on how the notation fits the domain

```
class Person {  
    private String name;  
    private String name;  
}
```

Person has (name, surname)



- Graphical DSLs are not always easier to understand



Graphical vs. Textual

- Expression of common concerns simple and concise

```
class Person {  
    private String name;  
    private String name;  
}
```

Person has (name, surname)

Graphical vs. Textual

- Expression of common concerns simple and concise

```
class Person {  
    private String name;  
    private String name;  
}
```

Person has (name, surname)

- Providing sensible defaults

```
public class MyClass {  
    public void readStream(InputStream stream, boolean  
closeStreamAferReading) {  
        ...  
    }  
}
```

```
public class MyClass {  
    public void readStream(InputStream stream) {  
        readStream(stream, true);  
    }  
    public void readStream(InputStream stream, boolean  
closeStreamAferReading) {  
        ...  
    }  
}
```

Graphical vs. Textual

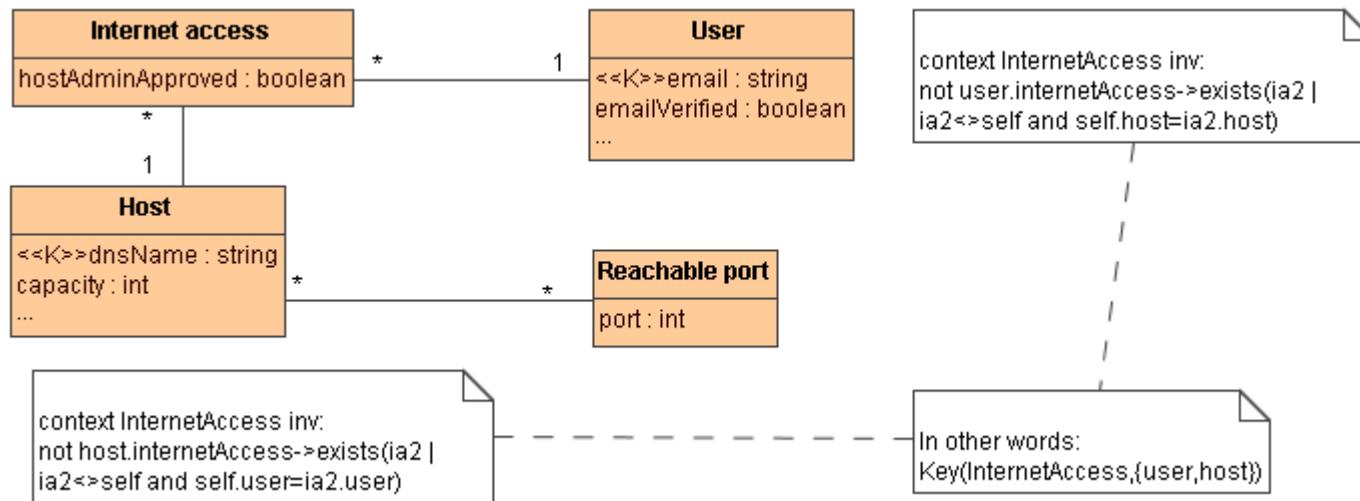
- Verbosity for less common concerns

```
Person has (name, surname)
```

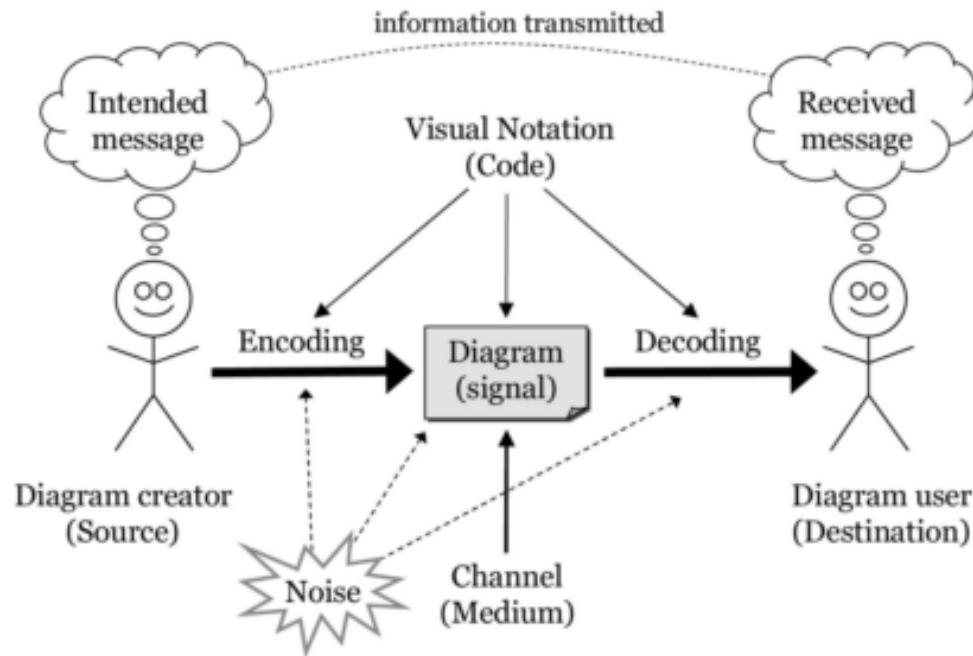
```
Person has (name, surname)
extension
  name is String
  surname is String
```

- Tool support and limitations
 - Syntax highlighting
 - Outline
 - Auto-completion
 - Validation
 - Layout
 - ...

Graphical AND Textual



Recommendations

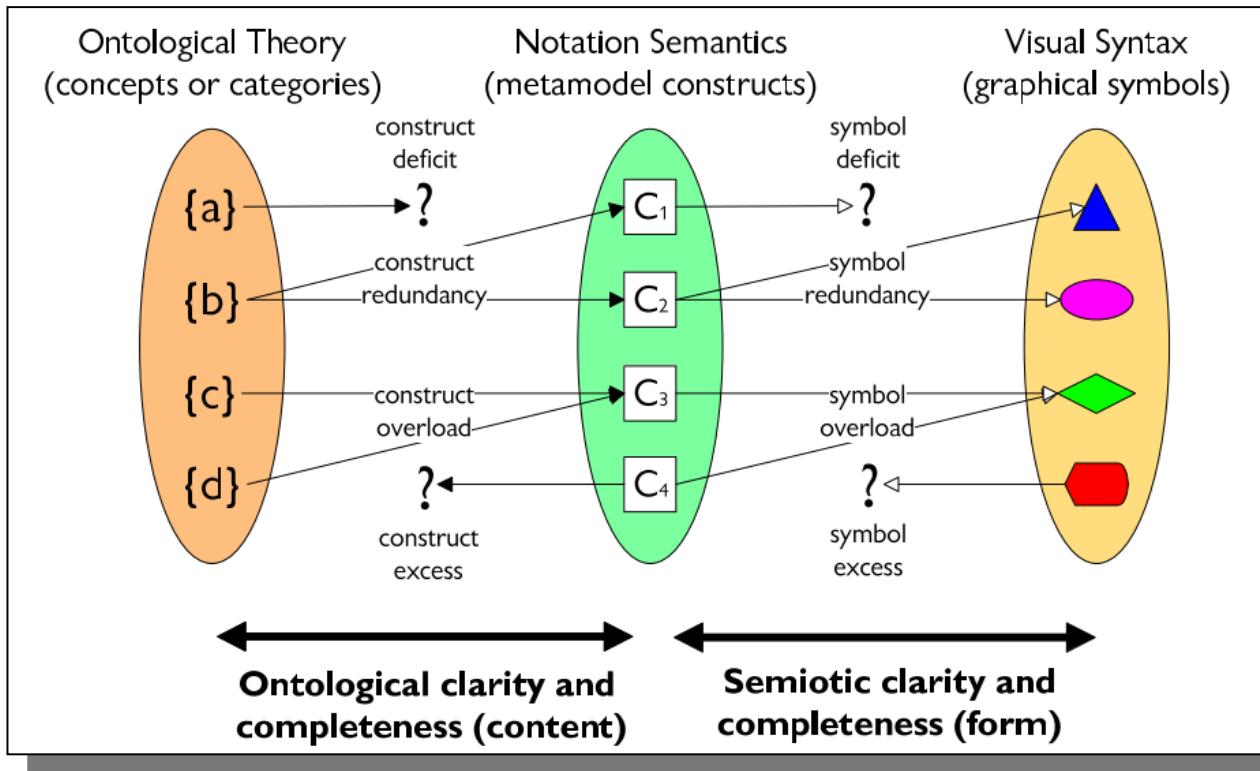


Recommendations

Recommendations

There should be a 1:1 correspondence
between concepts and graphical symbols

Recommendations



Recommendations

Different symbols should be clearly distinguishable from each other

Recommendations

Aggregation	Association (navigable)	Association (non-navigable)	Association class relationship	Composition
Constraint	Dependency	Generalisation	Generalisation set	Interface (provided)
Interface (required)	N-ary association	Note reference	Package containment	Package import (public)
Package import (private)	Package merge	Realisation	Substitution	Usage

Recommendations

Use visual representations
whose appearance suggests their
meaning

Recommendations

Include mechanisms for dealing
with complexity

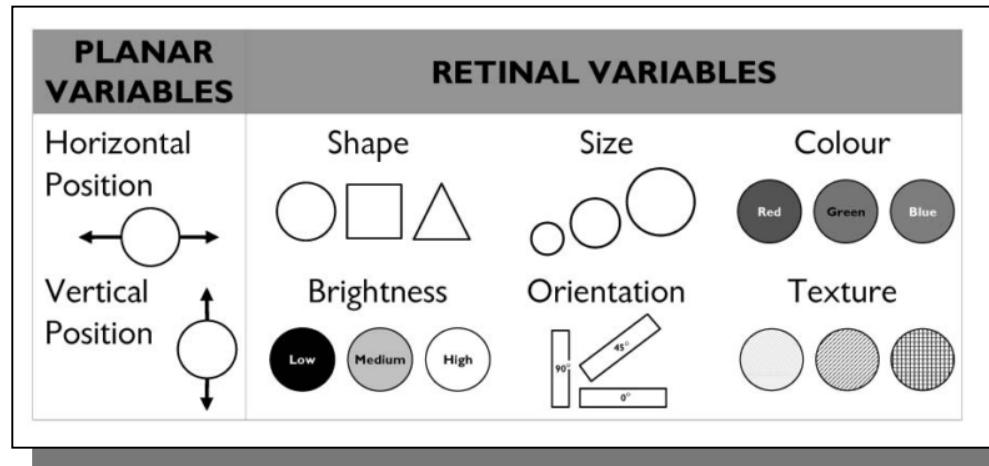
Recommendations

Include explicit mechanisms to support integration of information from different diagrams

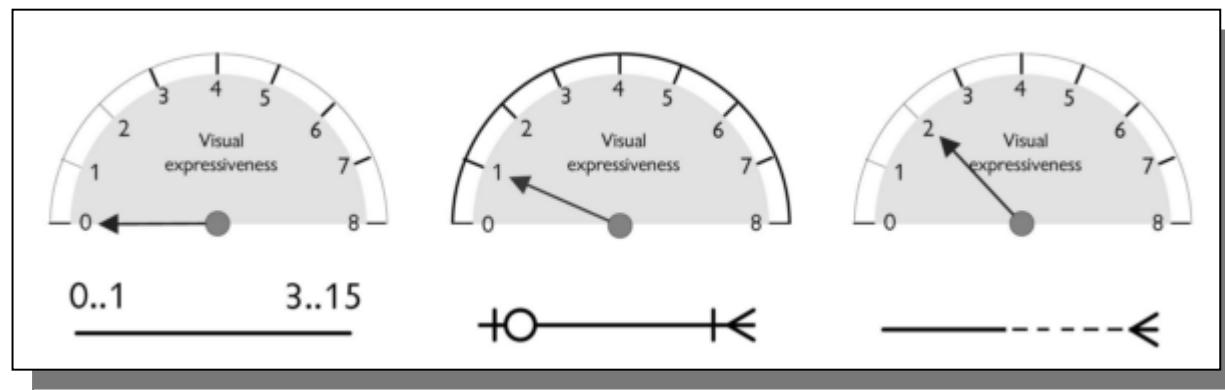
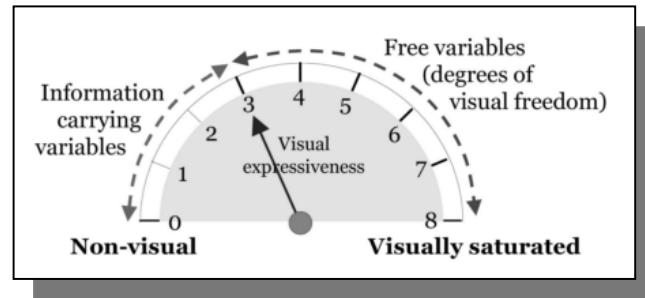
Recommendations

Use the full range and capacities
of visual variables

Recommendations



Recommendations



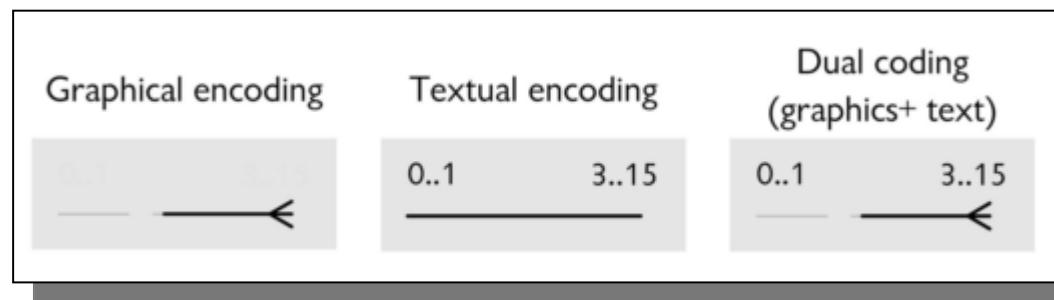
Recommendations

Diagram Type	X	Y	Size	Brightness	Colour	Shape	Texture	Orientation
Activity	●	●		●		●		
Class				●		●		
Communication				●		●		
Component				●		●		
Composite structure				●		●		
Deployment				●		●		
Interaction overview				●		●		
Object				●		●		
Package				●		●		
Sequence	●					●		
State machine				●		●		
Timing	●	●				●		
Use case	●					●		

Recommendations

Use text to complement graphics

Recommendations



Recommendations

The number of different graphical symbols should be cognitively manageable

Recommendations

Use different visual dialects for
different tasks and audiences

Anatomy

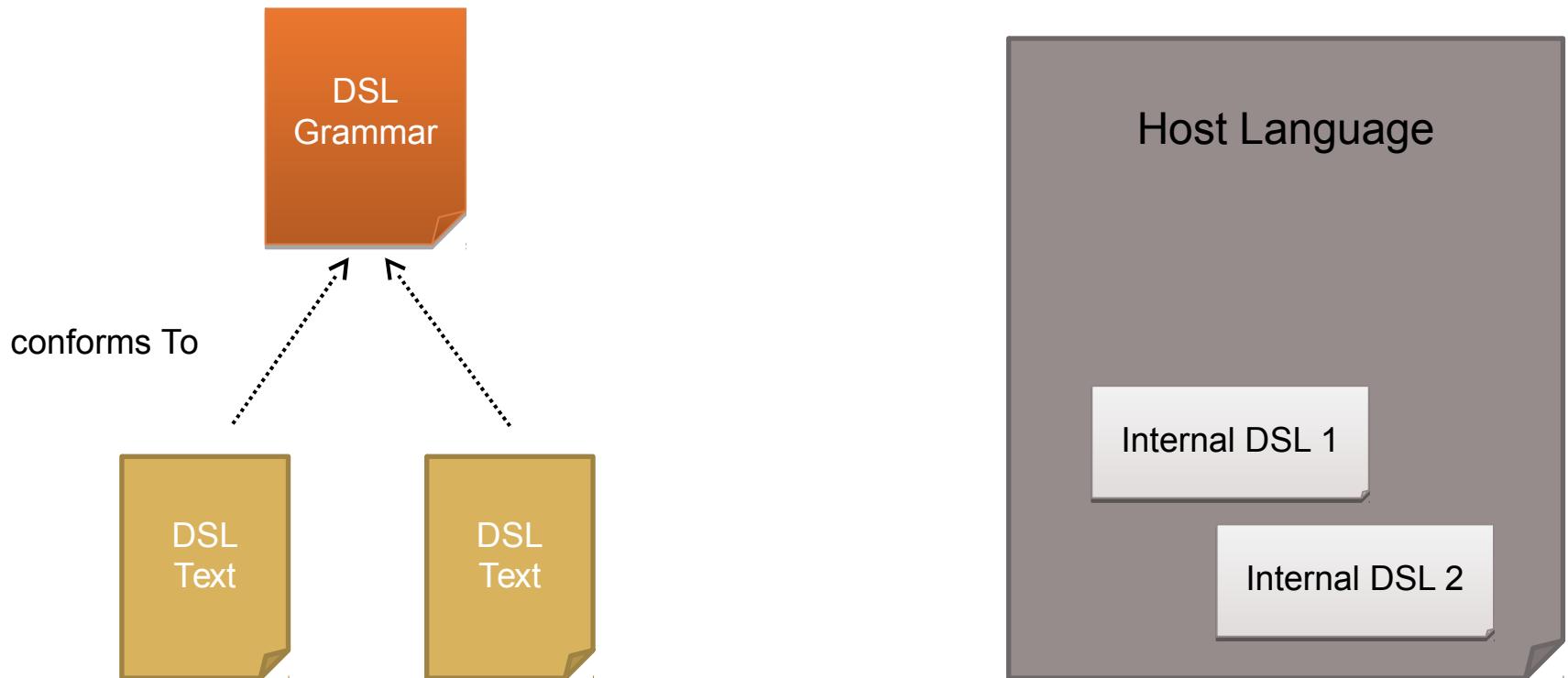
Semantics

DSL

Development process

Regarding the implementation

External vs Internal



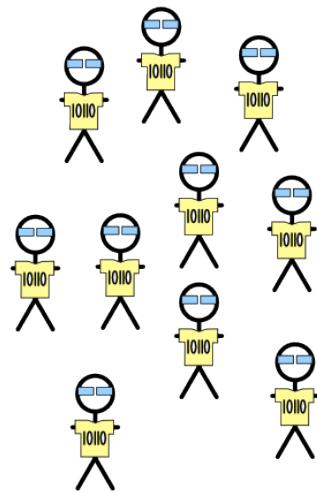
Internal DSL

```
Mailer.mail()  
  .to("you@gmail.com")  
  .from("me@gmail.com")  
  .subject("Writing DSLs in Java")  
  .body("...")  
  .send();
```

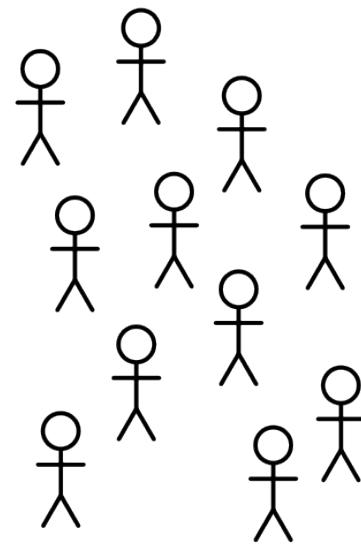
External vs Internal

DSL

Actors

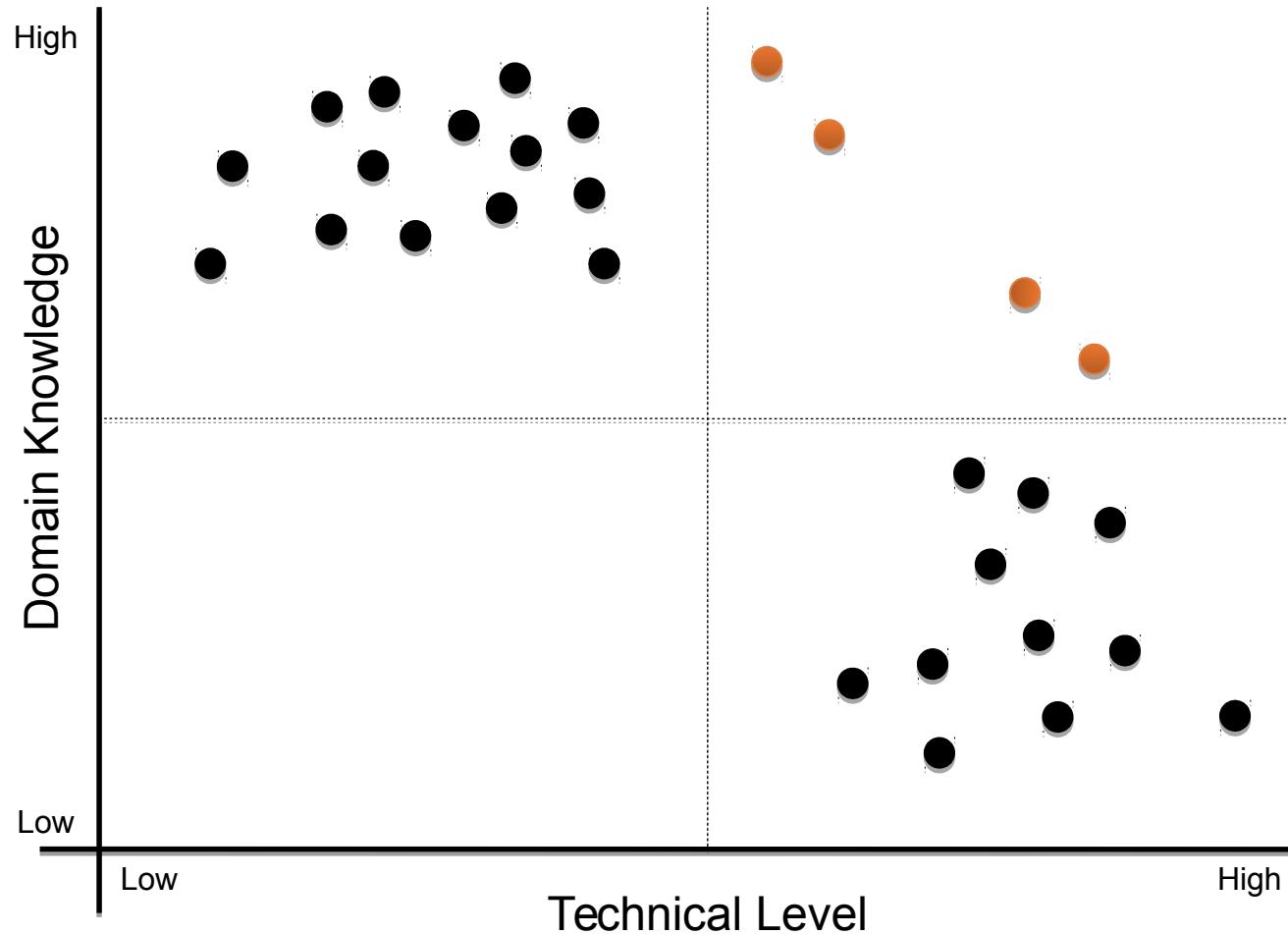


Developers



End-Users

Actors



DSL

Best practices

DSL

Worst practices

- Initial conditions
 - Only Gurus allowed
 - Believe that only gurus can build languages or that “I’m smart and don’t need help”
 - Lack of Domain Understanding
 - Insufficiently understanding the problem domain or the solution domain
 - Analysis paralysis
 - Wanting the language to be theoretically complete, with its implementation assured

Worst practices

- The source for Language Concepts
 - UML: New Wine in Old Wineskins
 - Extending a large, general-purpose modeling language
 - 3GL Visual Programming
 - Duplicating the concepts and semantics of traditional programming languages
 - Code: The Library is the Language
 - Focusing the language on the current code's technical details
 - Tool: if you have a hammer
 - Letting the tool's technical limitations dictate language development

Worst practices

- The resulting language
 - Too Generic / Too Specific
 - Creating a language with a few generic concepts or too many specific concepts, or a language that can create only a few models
 - Misplaced Emphasis
 - Too strongly emphasizing a particular domain feature
 - Sacred at Birth
 - Viewing the initial language version as unalterable

Worst practices

- Language Notation
 - Predetermined Paradigm
 - Choosing the wrong representational paradigm or the basis of a blinkered view
 - Simplistic Symbols
 - Using symbols that are too simple or similar or downright ugly

Worst practices

- Language Use
 - Ignoring the use process
 - Failing to consider the language's real-life usage
 - No training
 - Assuming everyone understands the language like its creator
 - Pre-adoption Stagnation
 - Letting the language stagnate after successful adoption

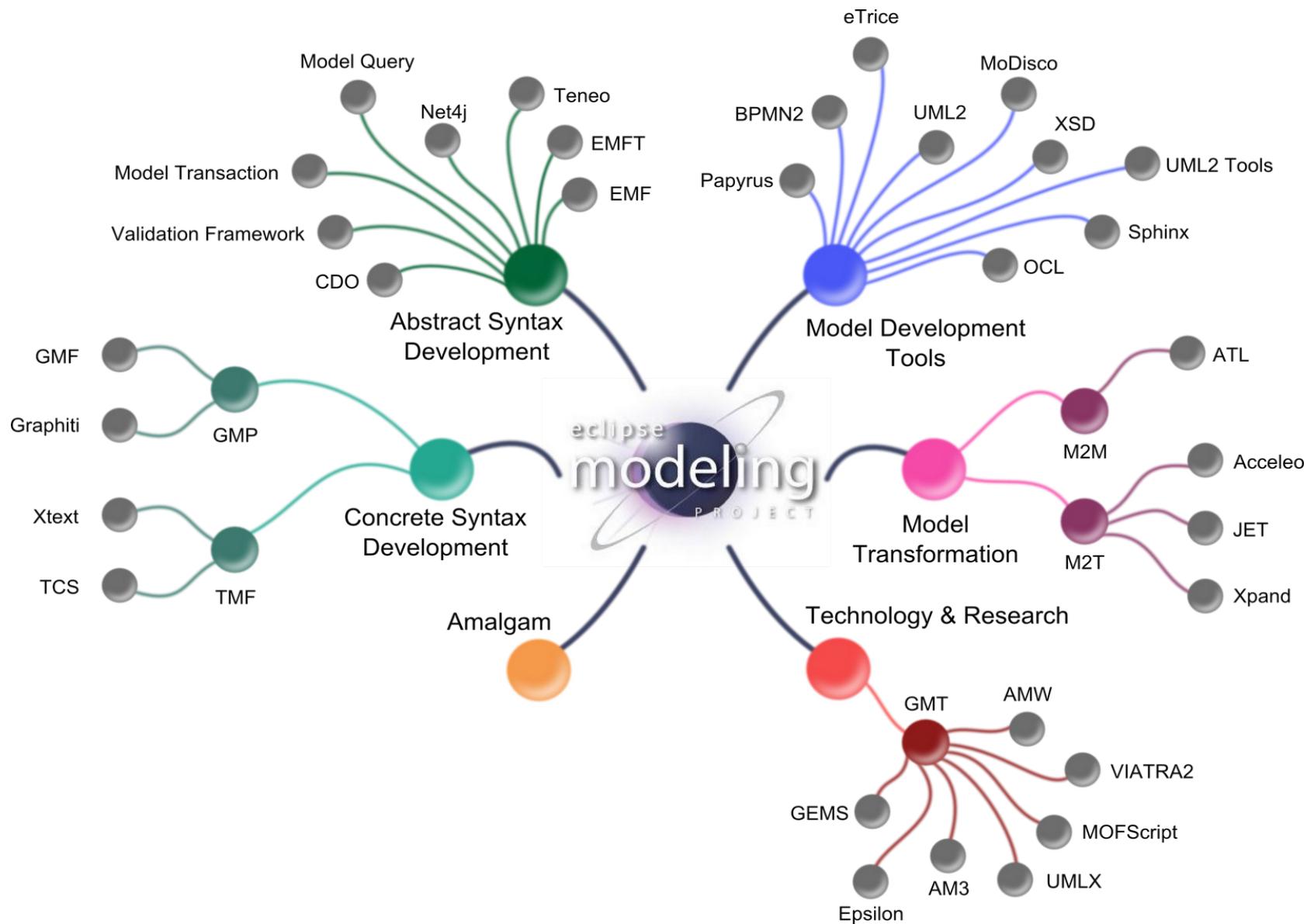
Outline

Xtext

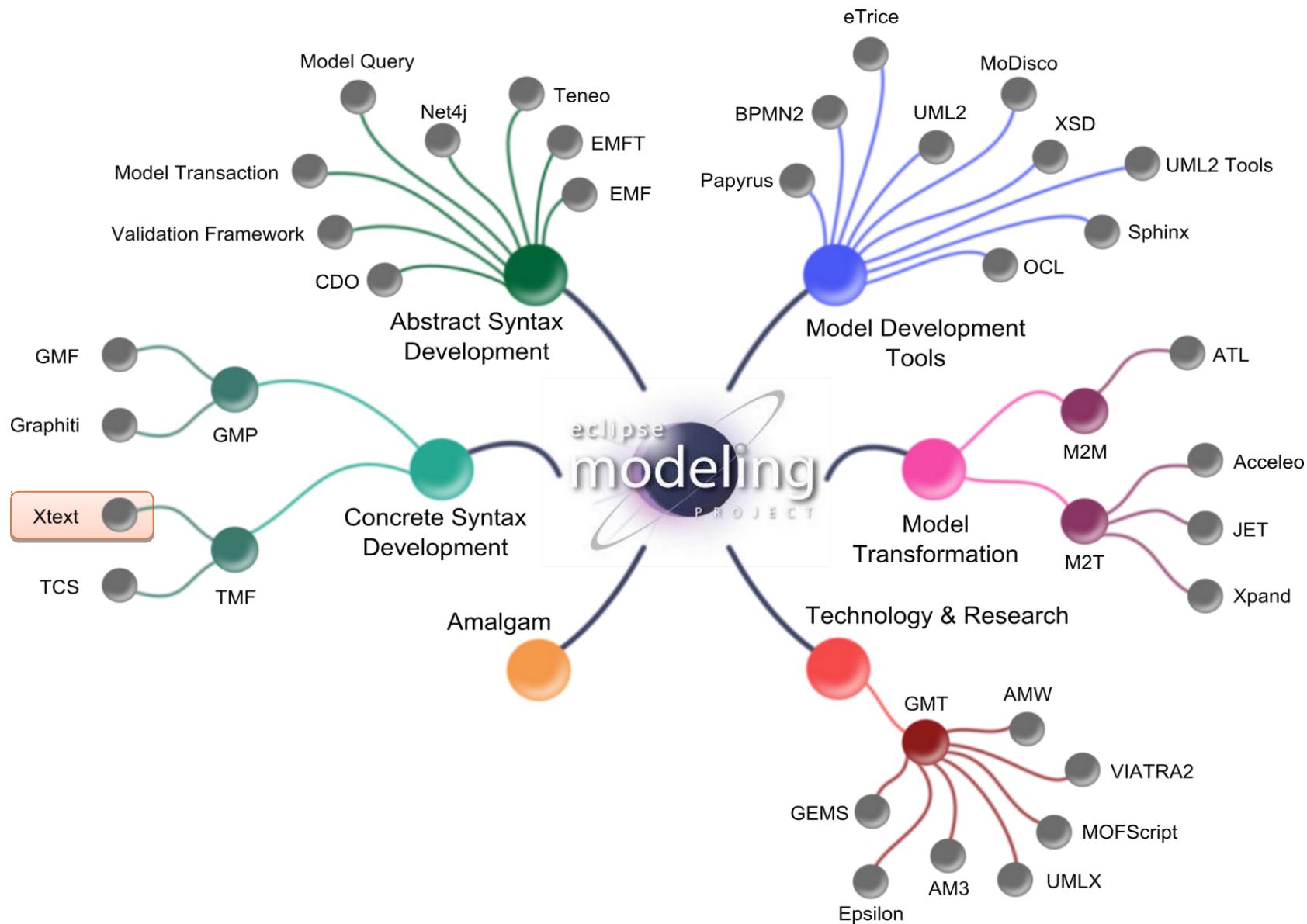
- Eclipse Project
 - Part of Eclipse Modeling
 - Part of Open Architecture Ware
- Model-driven development of Textual DSLs
- Part of a family of languages
 - **Xtext**
 - Xtend
 - Xbase
 - Xpand
 - Xcore



Eclipse Modeling Project



Eclipse Modeling Project



The grammar language

- Corner-stone of Xtext
- DSL to define textual languages
 - Describe the concrete syntax
 - Specify the mapping between concrete syntax and domain model
- From the grammar, it is generated:
 - The domain model
 - The parser
 - The tooling

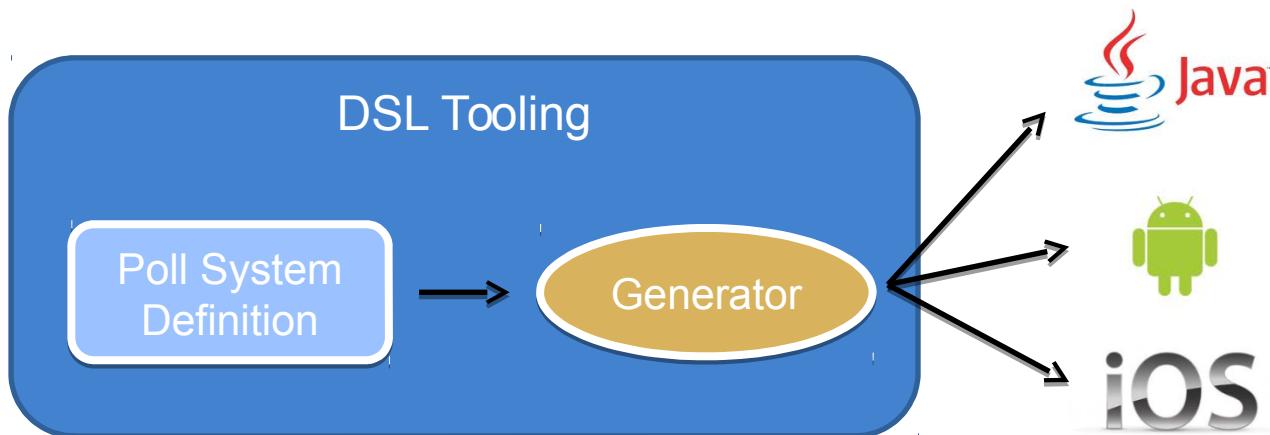
Main advantages

- Consistent look and feel
- Textual DSLs are a resource in Eclipse
- Open editors can be extended
- Complete framework to develop DSLs
- Easy to connect to any Java-based language

Proposed development process

Example DSL

- Poll System application
 - Define a Poll with the corresponding questions
 - Each question has a text and a set of options
 - Each option has a text
- Generate the application in different platforms



Example DSL

DSL Tooling

```
PollSystem {  
    Poll Quality {  
        Question q1 {  
            "Value the user experience"  
            options {  
                A : "Bad"  
                B : "Fair"  
                C : "Good"  
            }  
        }  
        Question q2 {  
            "Value the layout"  
            options {  
                A : "It was not easy to locate elements"  
                B : "I didn't realize"  
                C : "It was easy to locate elements"  
            }  
        }  
    }  
    Poll Performance {  
        Question q1 {  
            "Value the time response"  
            options {  
                A : "Bad"  
                B : "Fair"  
                C : "Good"  
            }  
        }  
    }  
}
```

Generator



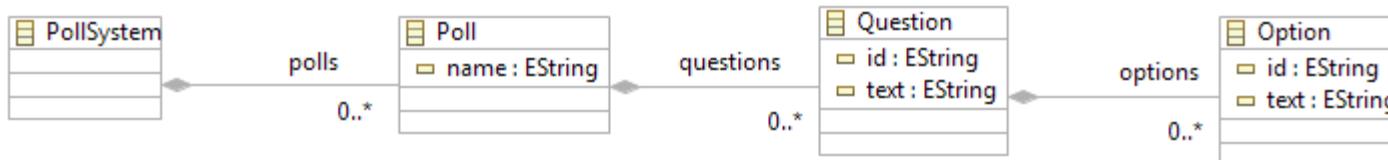
Grammar Definition

Grammar definition



```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals
generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}';
    
Poll:
    'Poll' name=ID '{' questions+=Question+ '}';
    
Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}'} '}';
    
Option:
    id=ID ':' text=STRING;
```



Grammar Definition

Grammar
reuse

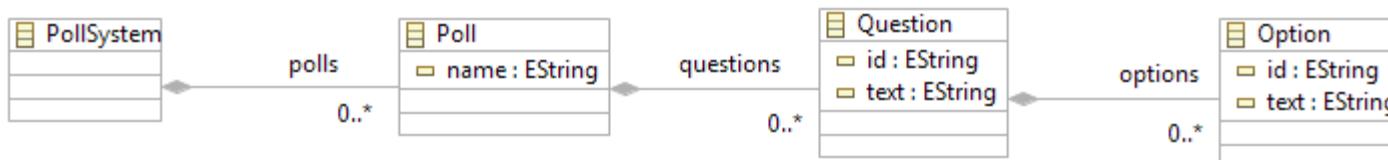
```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}';
    
Poll:
    'Poll' name=ID '{' questions+=Question+ '}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}'}';

Option:
    id=ID ':' text=STRING;
```

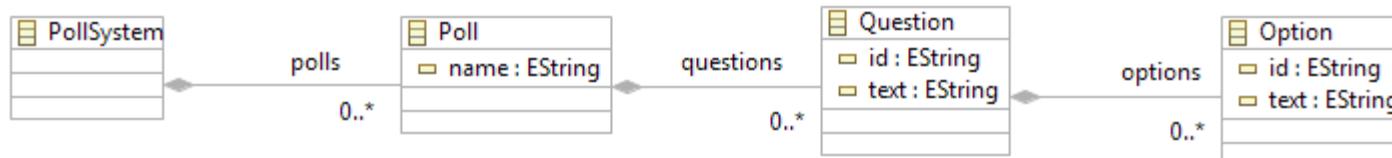


Grammar Definition

Derived
metamodel

```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals
generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}';
    
Poll:
    'Poll' name=ID '{' questions+=Question+ '}';
    
Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}'} '}';
    
Option:
    id=ID ':' text=STRING;
```



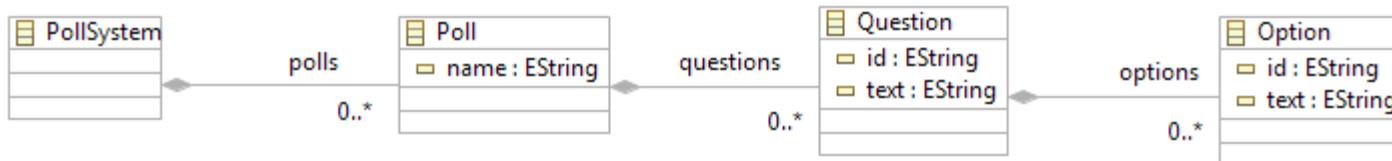
Grammar Definition

Parser Rules

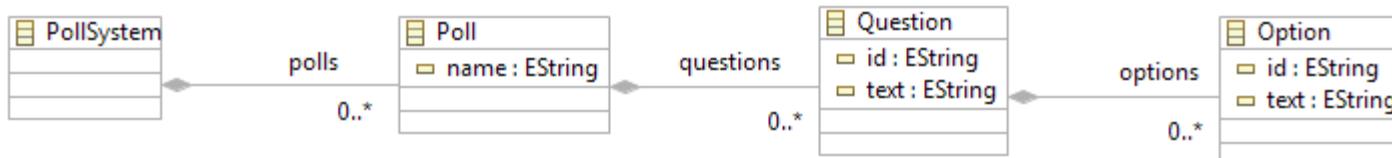
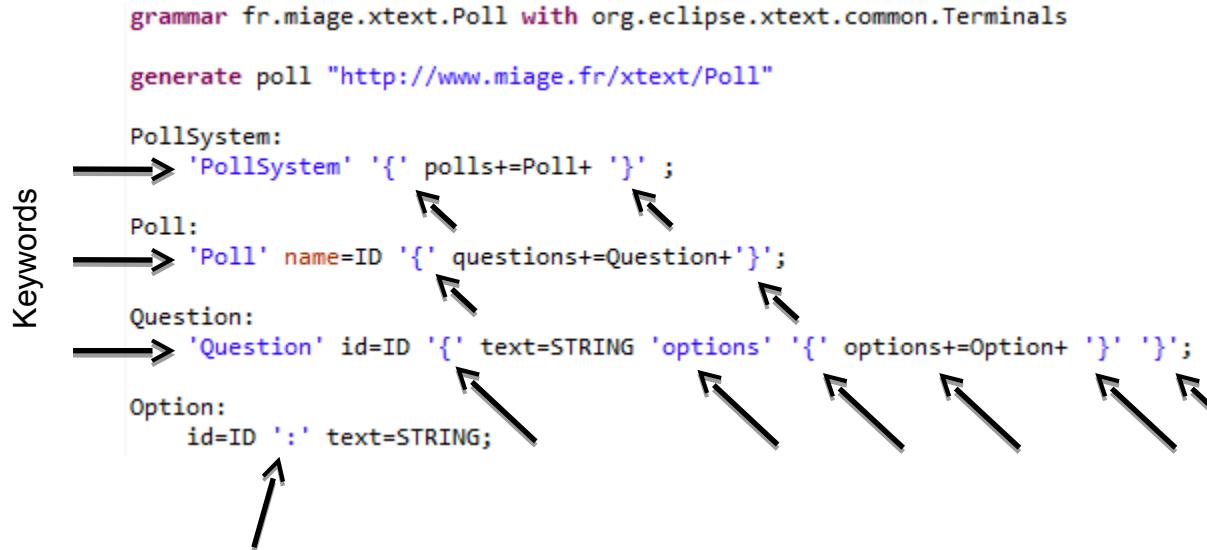
```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

generate poll "http://www.miage.fr/xtext/Poll"

→ PollSystem:
   'PollSystem' '{' polls+=Poll+ '}';
→ Poll:
   'Poll' name=ID '{' questions+=Question+ '}';
→ Question:
   'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}';
→ Option:
   id=ID ':' text=STRING;
```



Grammar Definition



Grammar Definition

```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

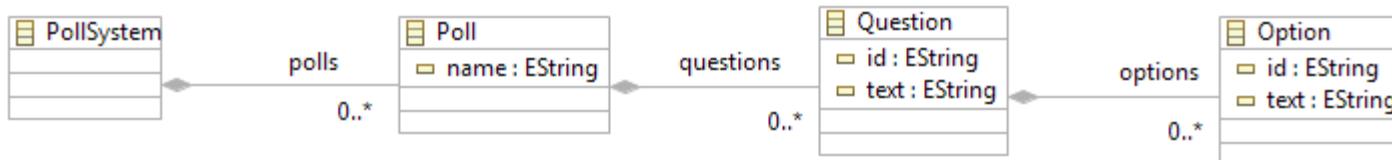
generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}';
    ^ Multivalue assignment

Poll:
    'Poll' name=ID '{' questions+=Question+ '}';
    ^ Simple assignment

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}'} ';
    ^ Boolean assignment

Option:
    id=ID ':' text=STRING;
```



Grammar Definition

```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

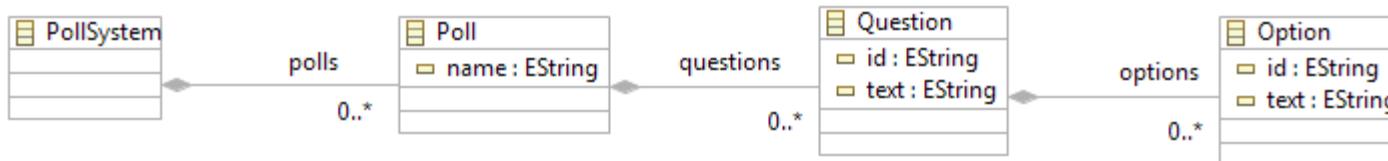
generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}';
    ^                                     Cardinality (others: * ?)

Poll:
    'Poll' name=ID '{' questions+=Question+ '}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}'}';

Option:
    id=ID ':' text=STRING;
```



Grammar Definition

```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

generate poll "http://www.miage.fr/xtext/Poll"

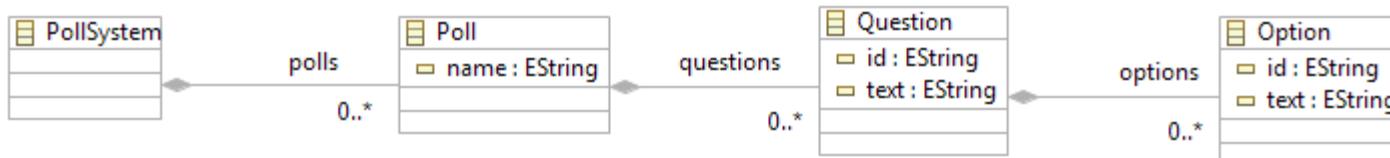
PollSystem:
    'PollSystem' '{' polls+=Poll+ '}';
    ^

Poll:
    'Poll' name=ID '{' questions+=Question+ '}';
    ^

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}';
    ^

Option:
    id=ID ':' text=STRING;
```

Containment



Grammar Definition

```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

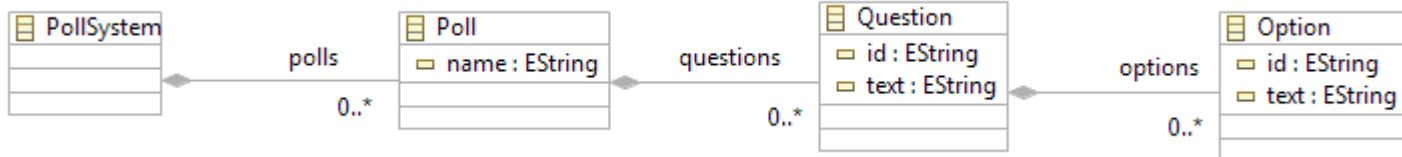
generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}';
    
Poll:
    'Poll' name=ID '{' questions+=Question+'}'';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}'}'';

Option:
    id=ID ':' text=STRING;
```

```
PollSystem {
    Poll Quality {
        Question q1 {
            "Value the user experience"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
        Question q2 {
            "Value the layout"
            options {
                A : "It was not easy to locate elements"
                B : "I didn't realize"
                C : "It was easy to locate elements"
            }
        }
    }
    Poll Performance {
        Question q1 {
            "Value the time response"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
    }
}
```



Grammar Definition

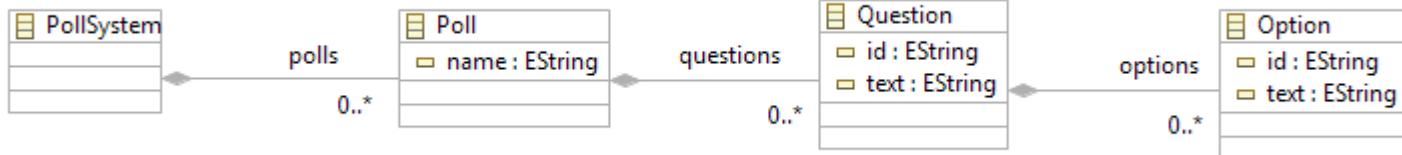
```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}';
    
Poll:
    'Poll' name=ID '{' questions+=Question+'}'';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}';
    
Option:
    id=ID ':' text=STRING;
```

```
PollSystem {
    Poll Quality {
        Question q1 {
            "Value the user experience"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
        Question q2 {
            "Value the layout"
            options {
                A : "It was not easy to locate elements"
                B : "I didn't realize"
                C : "It was easy to locate elements"
            }
        }
    }
    Poll Performance {
        Question q1 {
            "Value the time response"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
    }
}
```



Grammar Definition

```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

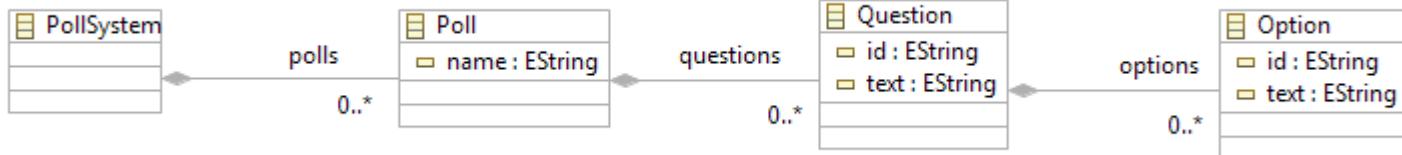
generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}';
    
Poll:
    'Poll' name=ID '{' questions+=Question+'}'';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+'}' '}';

Option:
    id=ID ':' text=STRING;
```

```
PollSystem {
    Poll Quality {
        Question q1 {
            "Value the user experience"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
        Question q2 {
            "Value the layout"
            options {
                A : "It was not easy to locate elements"
                B : "I didn't realize"
                C : "It was easy to locate elements"
            }
        }
    }
    Poll Performance {
        Question q1 {
            "Value the time response"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
    }
}
```



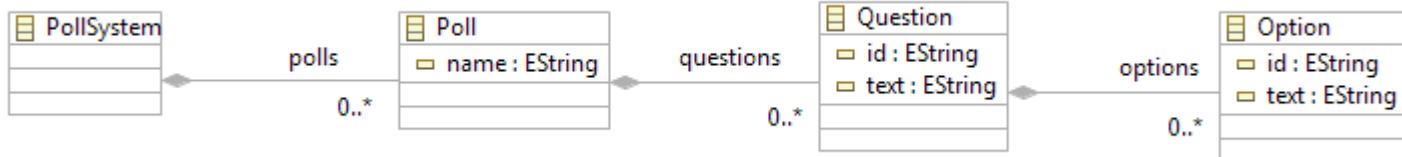
Grammar Definition

```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}';
    
Poll:
    'Poll' name=ID '{' questions+=Question+'}' ;
    
Question:
    'Question' id=ID '{' text=STRING options='{' options+=Option+ '}'} '}';
    
Option:
    id=ID ':' text=STRING;
```

```
PollSystem {
    Poll Quality {
        Question q1 {
            "Value the user experience"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
        Question q2 {
            "Value the layout"
            options {
                A : "It was not easy to locate elements"
                B : "I didn't realize"
                C : "It was easy to locate elements"
            }
        }
    }
    Poll Performance {
        Question q1 {
            "Value the time response"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
    }
}
```



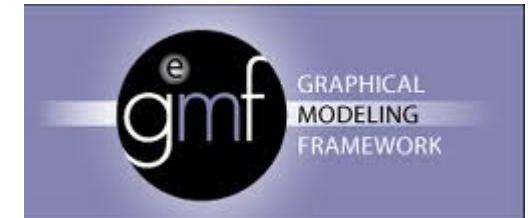


Xtext Demo

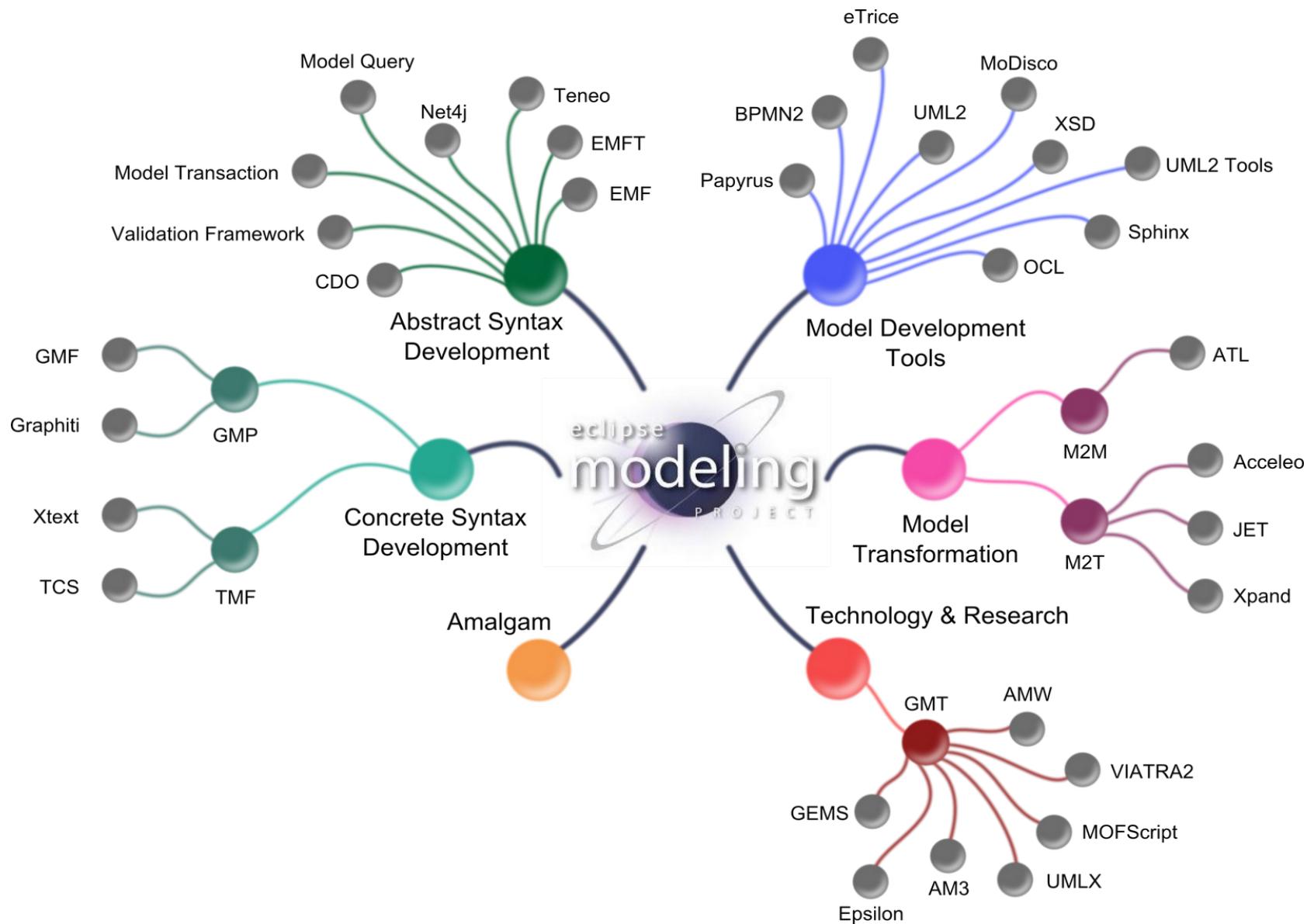
Outline

GMF

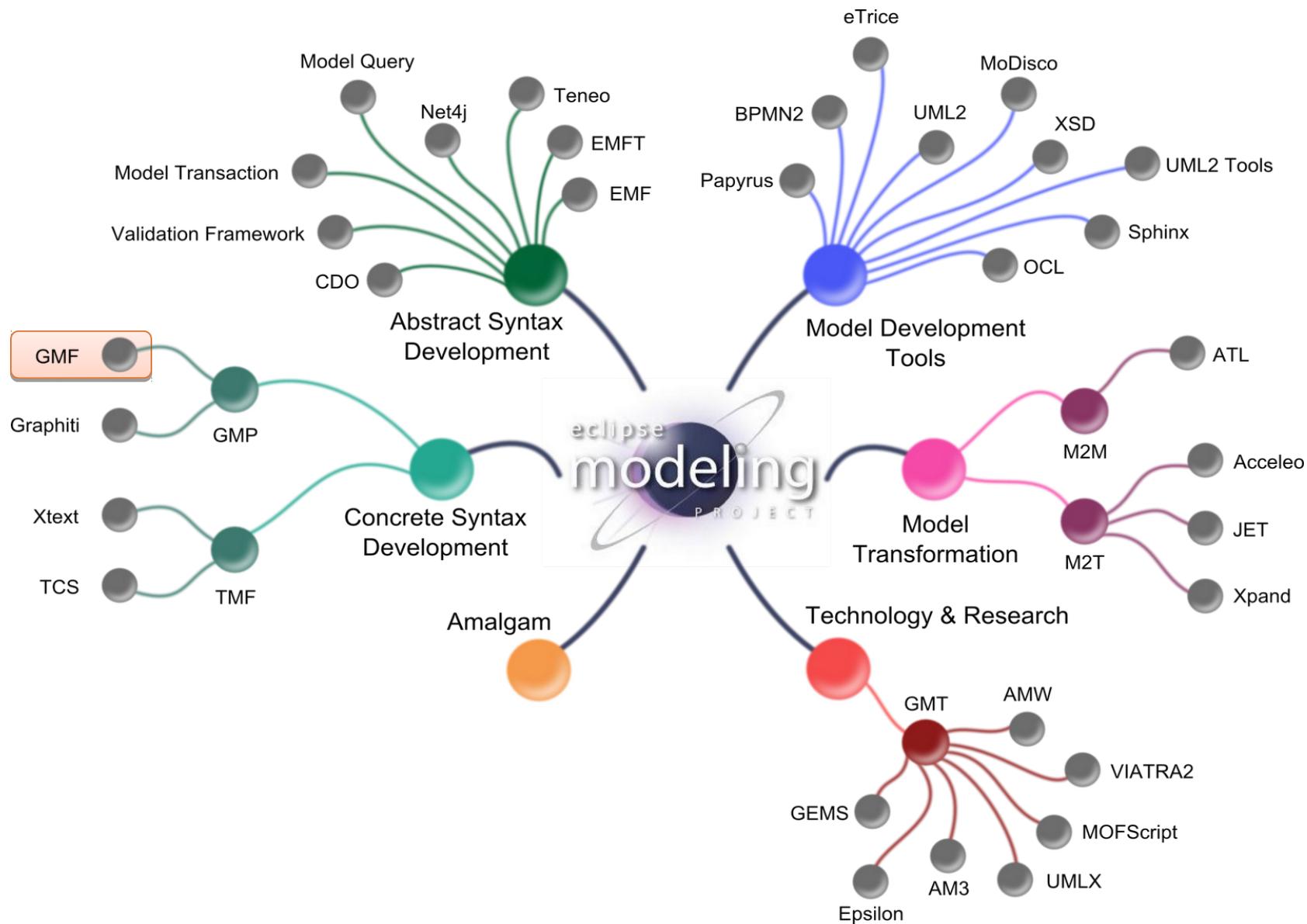
- Eclipse project
 - Eclipse Modelling components
 - Uses
 - EMF (Eclipse Modeling Framework)
 - GEF (Graphical Editing Framework)
- Model-driven framework for Graphical DSLs
 - Everything is a model
- Still under development (perpetual beta)
- The only alternative for now
- DSL definition easy, tweaking hard



Eclipse Modeling Project



Eclipse Modeling Project



Parts of GMF

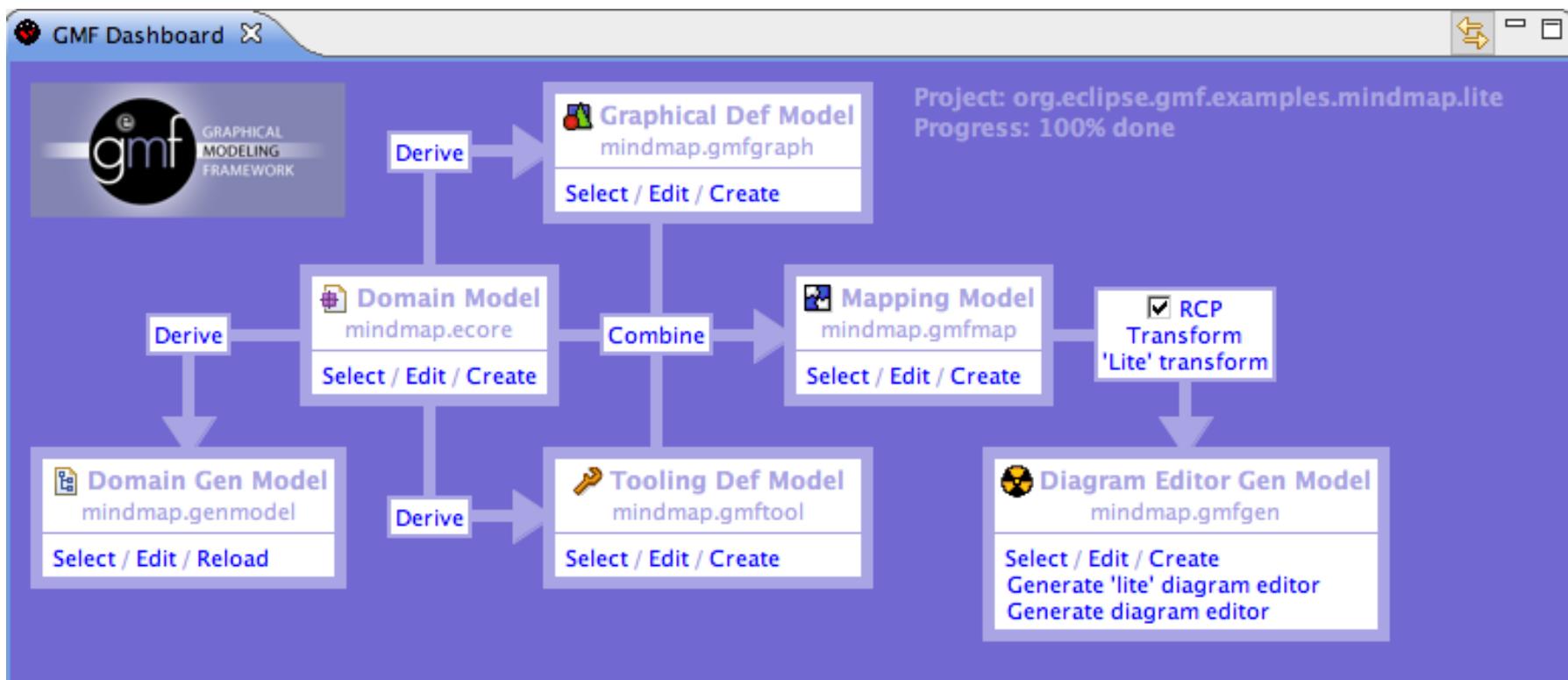
- Tooling
 - Editors for notation, semantic and tooling
 - GMF Dashboard
 - Generator to produce the DSL implementation
- Runtime
 - Generated DSLs depend on the GMF Runtime to produce an extensible graphical editor

Main advantages

- Consistent look and feel
- Diagram persistence
- Open editors can be extended by third-parties
- Already integrated with various Eclipse components
- Extensible notation metamodel to enable the isolation of notation from semantic concerns
- Future community enhancements will easily be integrated

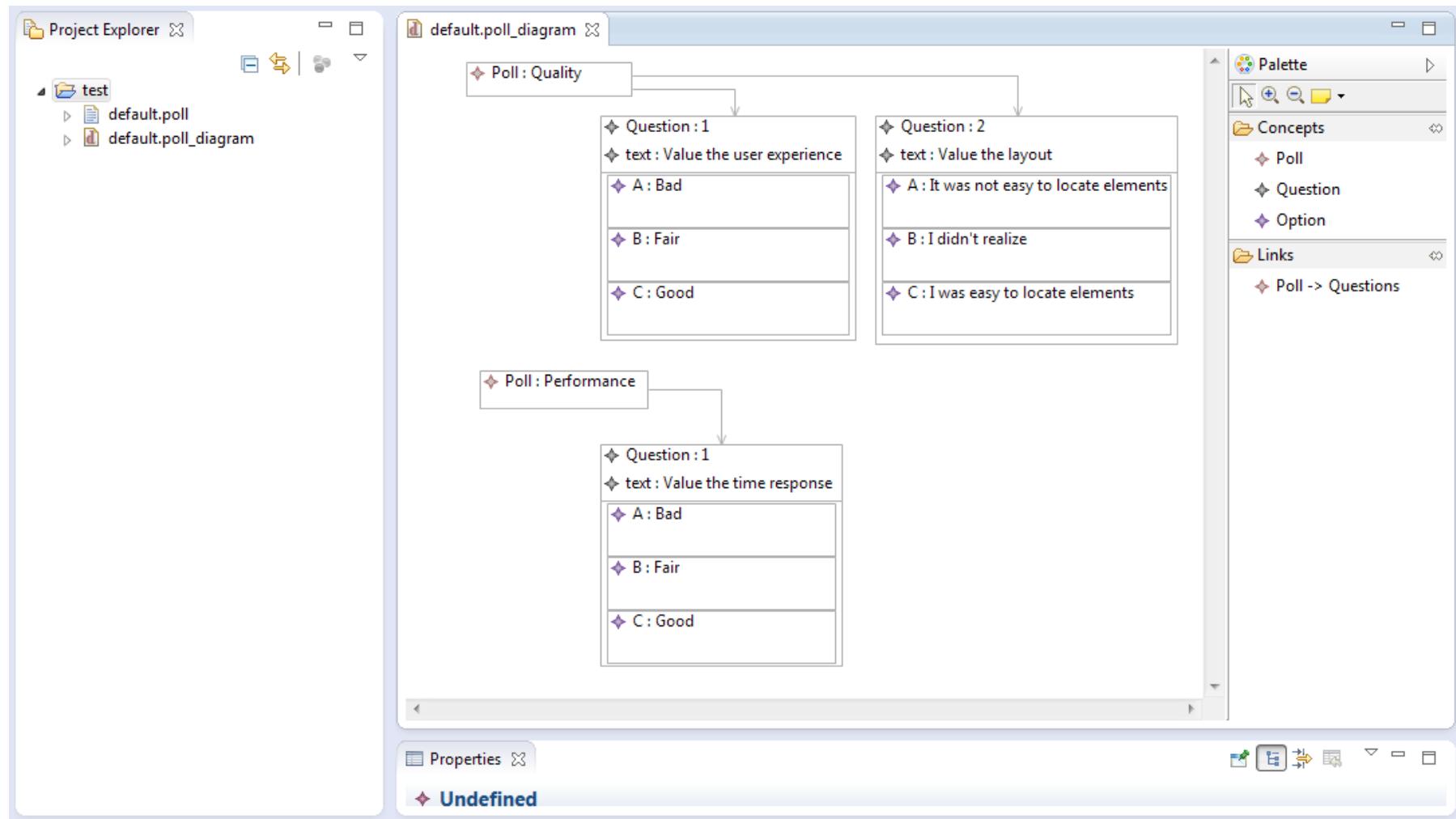
Proposed development process

Proposed development process



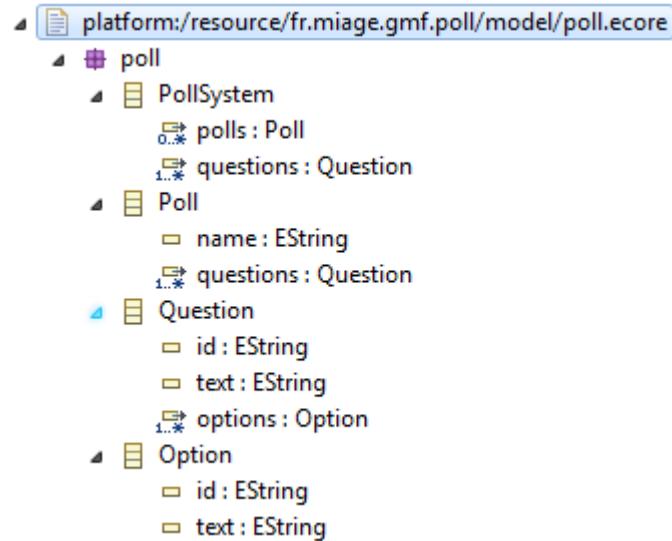
Defining a graphical DSL

Example DSL



Domain Model

- Concepts
 - PollSystem
 - Poll
 - Question
 - Option
- Attributes
 - A Poll has a name
 - A Question has an identifier and a descriptive text
 - An Option has an identifier and a descriptive text
- Relationships
 - PollSystem is composed of polls and questions
 - Question has a set of options

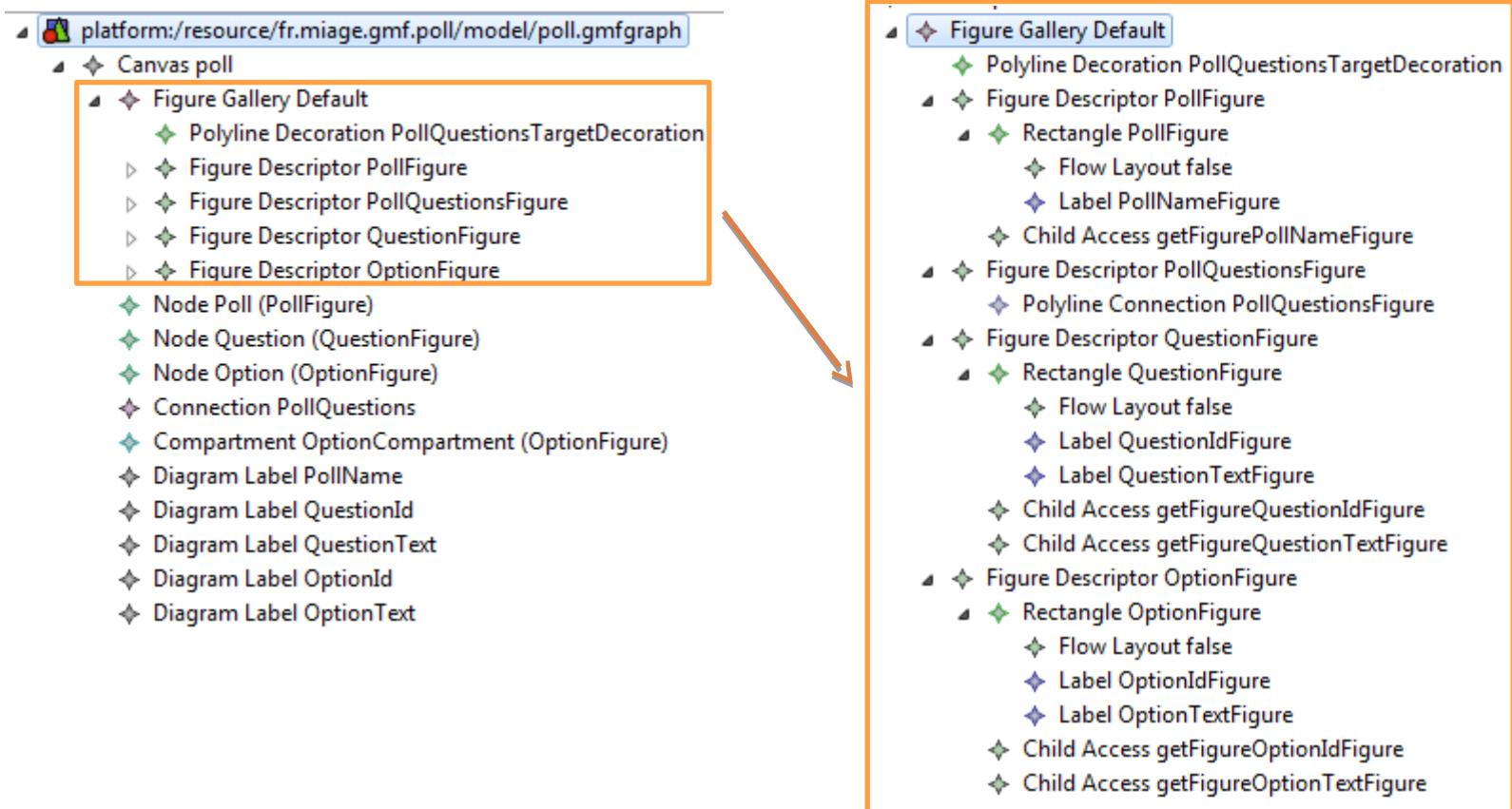


Graphical Definition

- Defines the graphical representation of the DSL elements
- Tree-based editor
- Main elements
 - Node = DSL node
 - Connector = DSL link
 - Compartment = Node compartment (for complex nodes)
 - Label = The label of a Node, Connector or compartment
 - Figure descriptor = Graphical representation of a Node
 - Rectangle, ellipse, polygon,...
 - Polyline,...
 - Decorators, icons, ...
 - Custom figures,...
- Provides simple figures that they can be customized

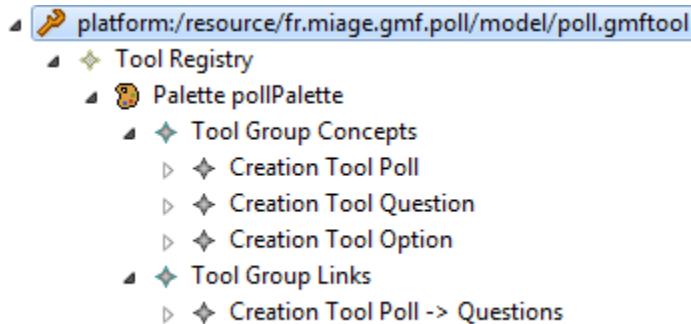
Graphical Definition

- A model will represent a PollSystem
- A Poll will be a node
- A Question will be a rectangular node
- An Option will be a rectangular node included in the Question node



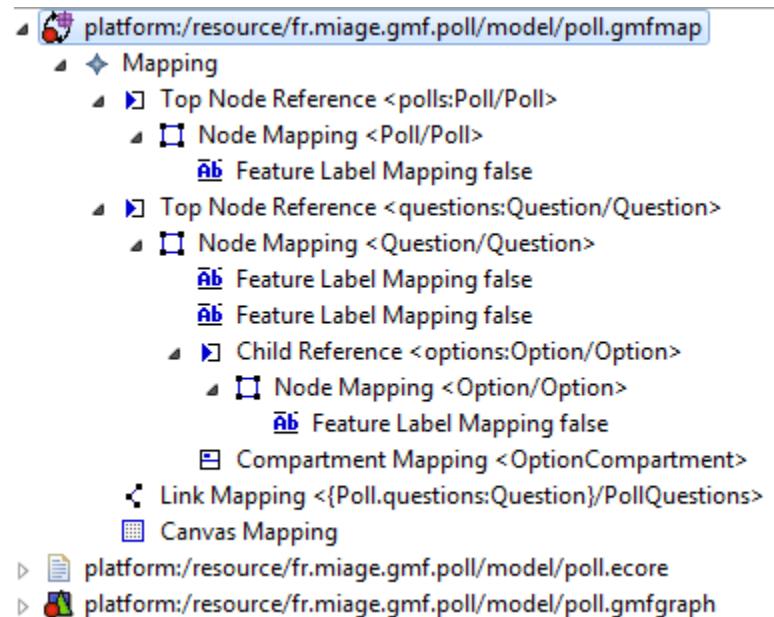
Tooling Definition

- Defines the tooling to create the DSL elements in the editor
- Tree-based editor
- Main elements
 - Tool Group □ Set of tools
 - Creation Tool □ Tool for creating a DSL element (node, connector, etc,...)
- Each tool can have an icon and a contextual help



Mapping Model

- Defines the correspondences between model elements, graphical elements and tooling elements
- Automatically generated (at least a kick-off version)
- Tree-based editor
- Main elements
 - Top Node Reference ↗ Maps a concept of the root model element
 - Node Mapping ↗ Maps a concept with the graphical node and tooling
 - Feature Label Mapping ↗ Maps a concept attribute with the graphical label
 - Child Reference ↗ Maps a containment reference with the compartment
 - Compartment Mapping ↗ Maps a concept with the graphical node
 - Link Mapping ↗ Maps a reference of the root model elements





GMF Demo

References

1. Kleppe, A.: Software Language Engineering
2. J. Bézivin, “Model Driven Engineering: An Emerging Technical Space,” in *GTTSE conf.*, 2006, vol. 4143, pp. 36–64
3. M. Brambilla, J. Cabot, M. Wimmer. “Model-Driven Software Engineering in Practice”. Morgan & Claypool
4. M. Mernik, J. Heering, and A. M. Sloane, “When and how to develop domain-specific languages,” *ACM Computing Surveys*, vol. 37, no. 4, pp. 316–344, Dec. 2005.
5. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice
6. Y. Sun and Z. Demirezen, “Is My DSL a Modeling or Programming Language?,” in *DSPD conf.*, 2008.
7. M. Strembeck and U. Zdun, “An approach for the systematic development of domain specific languages,” *Software: Practice and Experience*, vol. 39, pp. 1253–1292, 2009.
8. J.-M. Favre, “Foundations of Meta-Pyramids : Languages vs . Metamodels,” pp. 1–28.
9. J. De Lara and E. Guerra, “Languages for Model Driven Engineering,” in *ECMDA conf.*, 2012, vol. 7349, pp. 259–274.
10. D. L. Moody, “The ‘physics’ of notations: Toward a scientific basis for constructing visual notations in software engineering,” *Software Engineering, IEEE Transactions on*, vol. 35, no. 6, pp. 756–779, Nov. 2009.
11. D. L. Moody and J. Van Hillegersberg, “Evaluating the visual syntax of UML: An analysis of the cognitive effectiveness of the UML family of diagrams,” in *SLE conf.*, 2009, vol. 5452, no. Figure 1, pp. 16–34.
12. M. Völter, “MD */ DSL Best Practices Update March 2011,” 2011.
13. S. Kelly and R. Pohjonen, “Worst practices for domain-specific modeling,” *IEEE Software*, vol. 26, no. 4, pp. 22–29, 2009.
14. M. Völter, “DSL Engineering. Designing, Implementing and Using Domain-Specific Languages”.
15. I. Kurtev, J. Bézivin, and M. Aksit, “Technological Spaces : an Initial Appraisal,” in *DOA*, 2002, pp. 1–6.
16. Xtext Reference Guide
17. GMF documentation