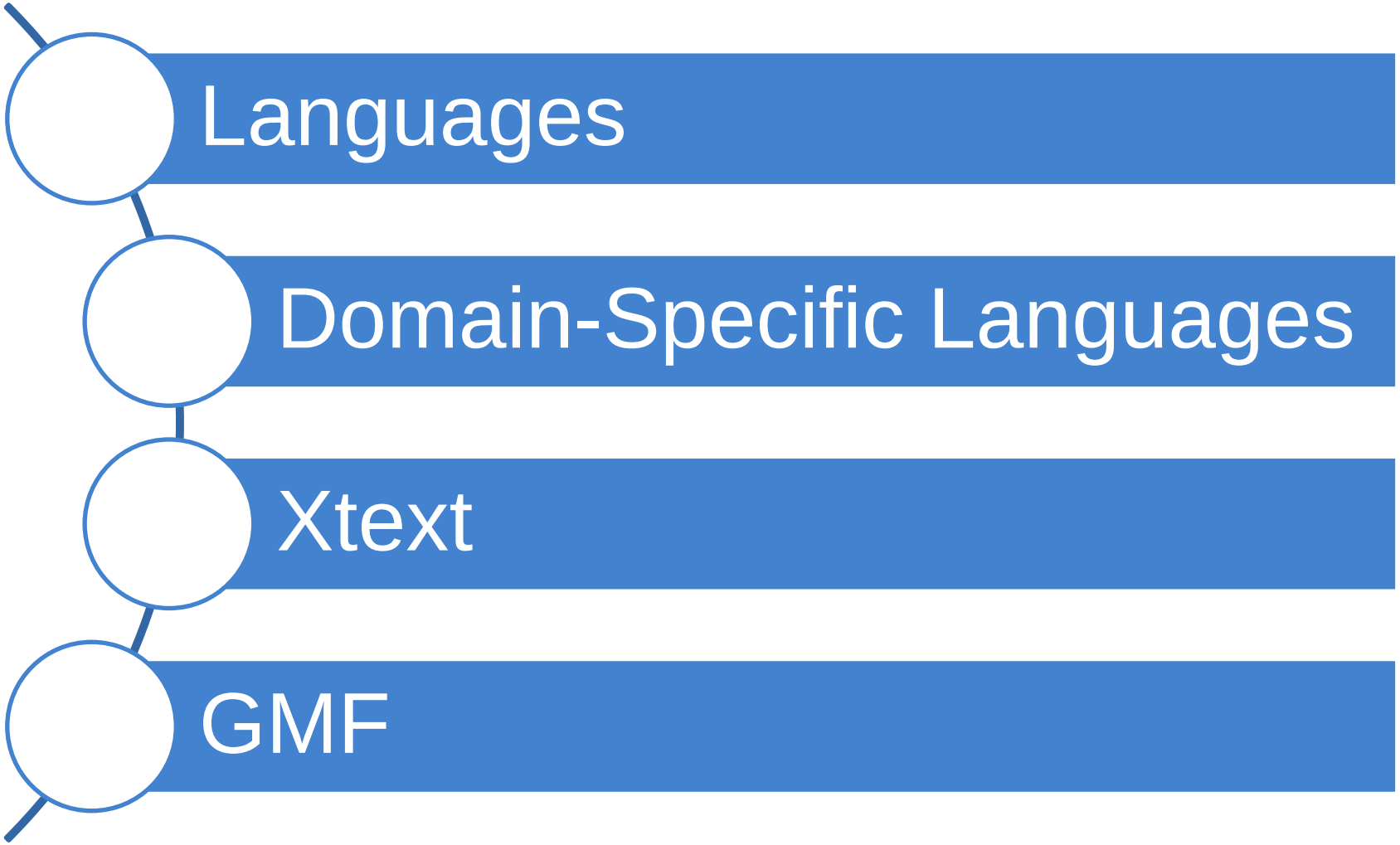
Domain-specific Languages

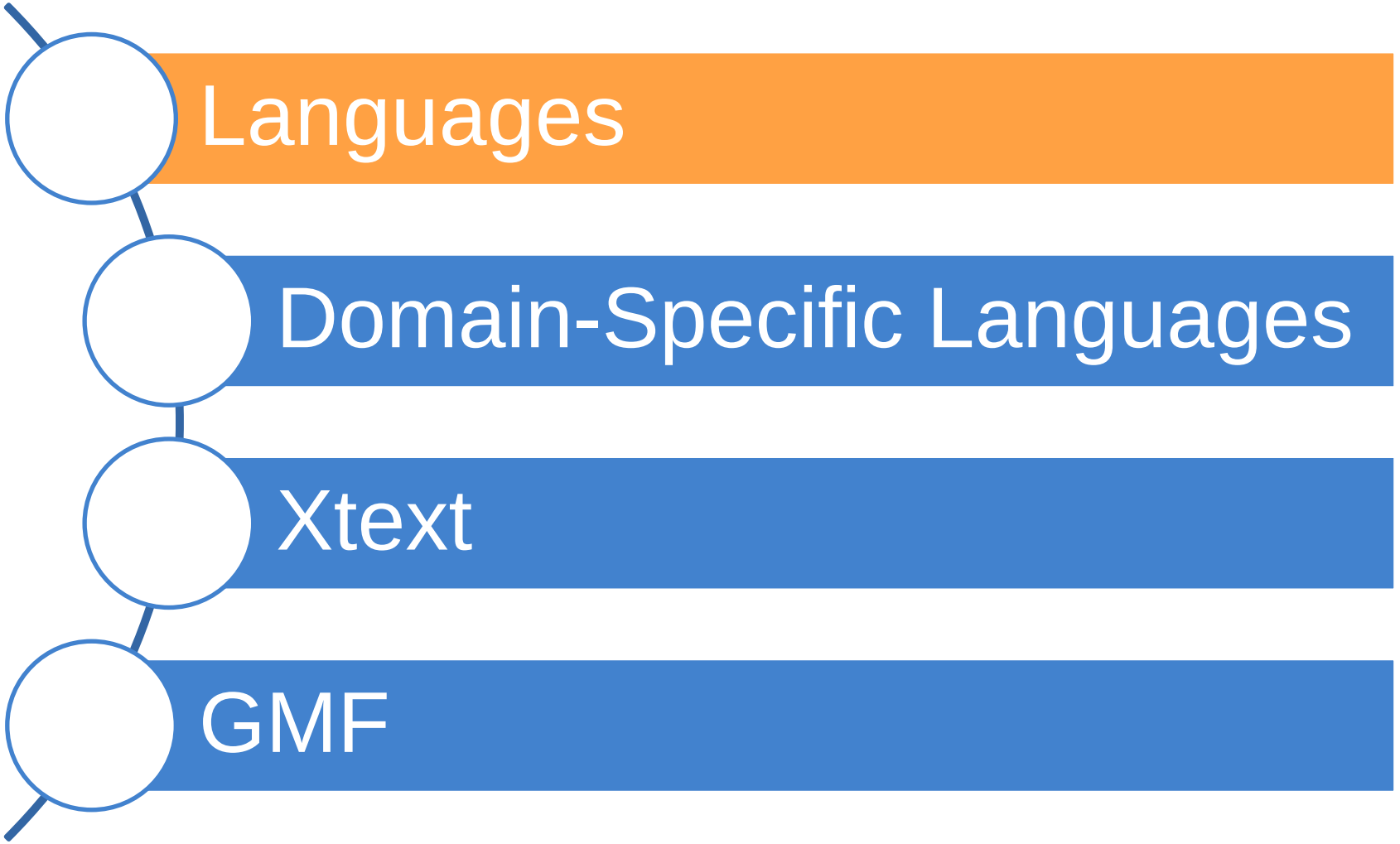
Javier Luis Cánovas Izquierdo
javier.canovas@inria.fr

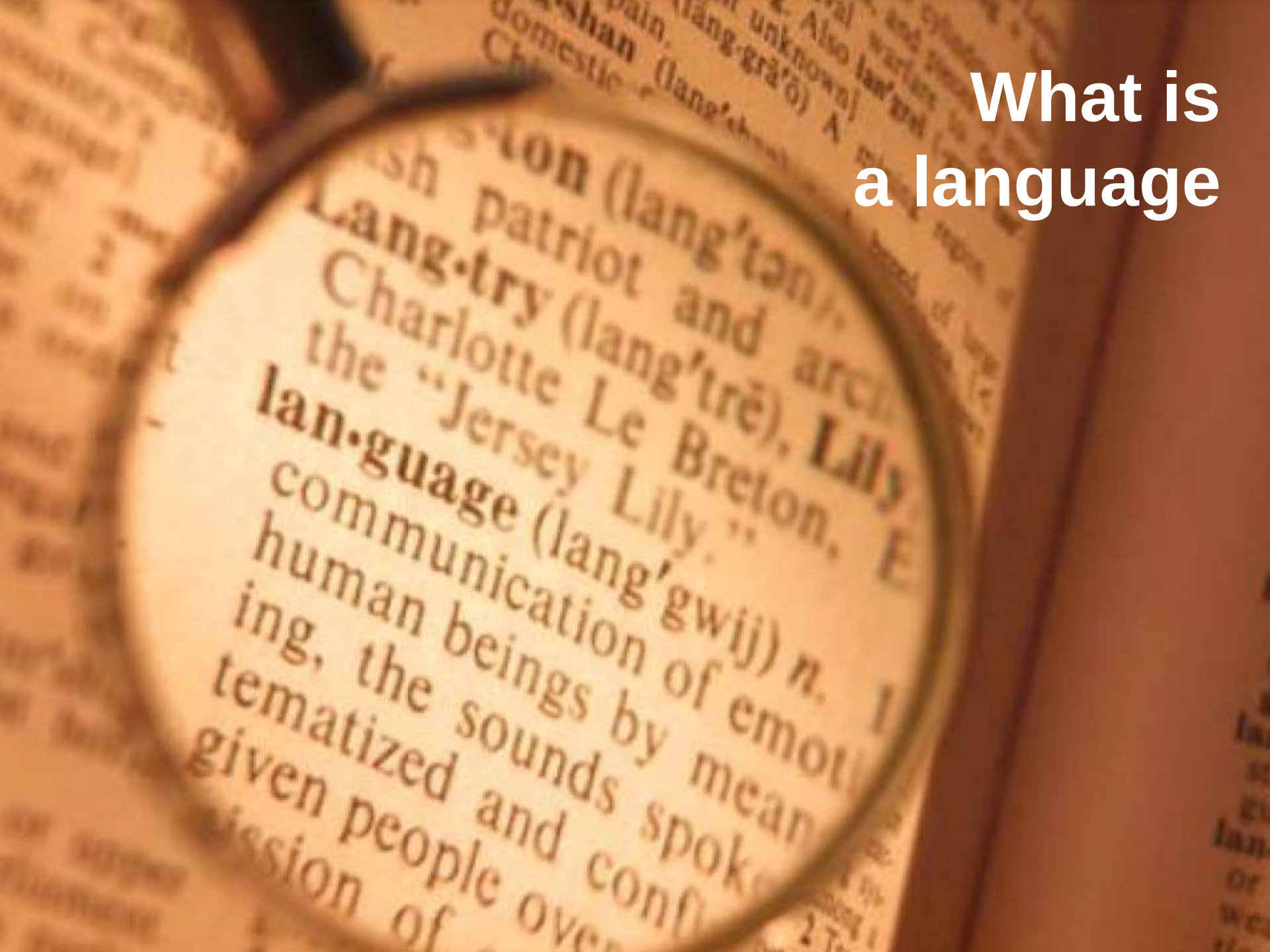
February 2013

Outline



Outline




What is
a language

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?

Table

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.

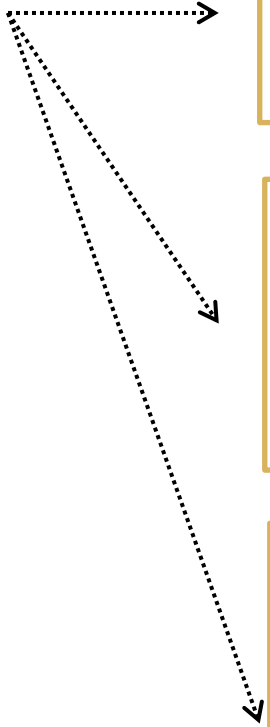
A grammar is the set of structural rules that governs the composition of linguistic utterances

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)*;
```



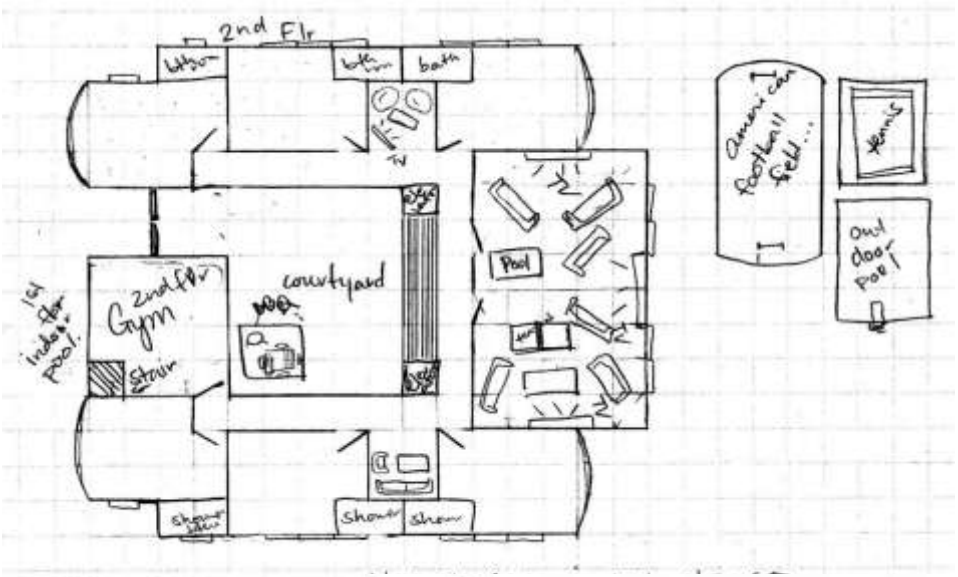
```
machine {
  SOne STwo
  T1 { SOne STwo }
}
```

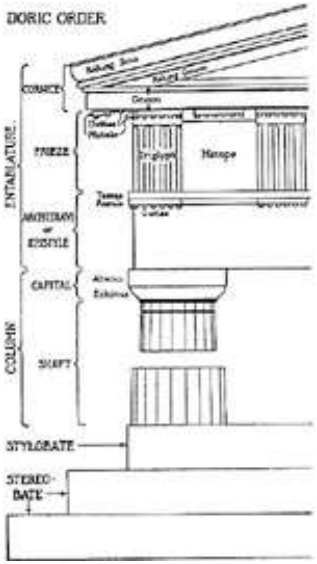
```
machine {
  S_first S_second
  T1 { S_first S_second}
  T1 { S_second S_first }

}
```

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

Languages in engineering

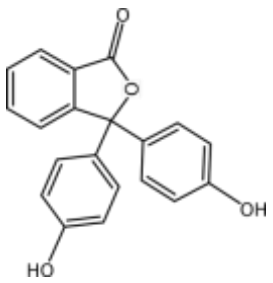
Architecture

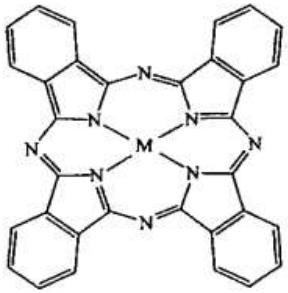




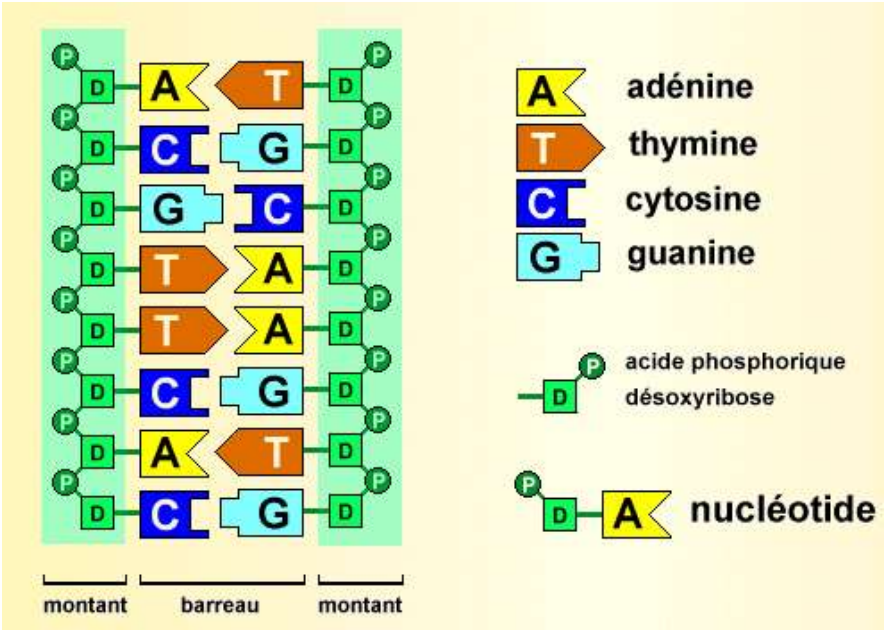
Cartography

Biology



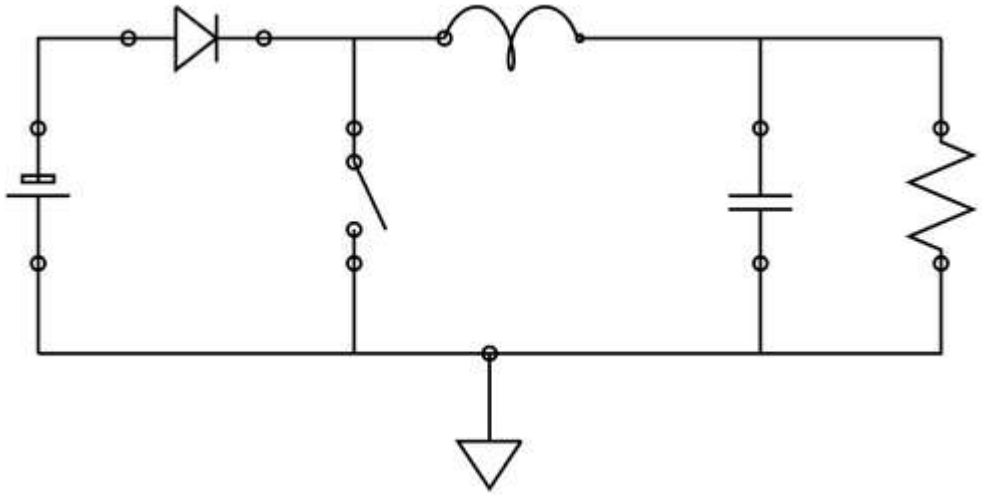


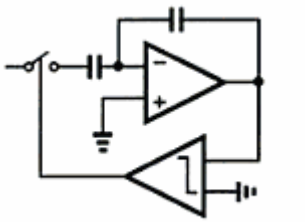
phthalocyanine

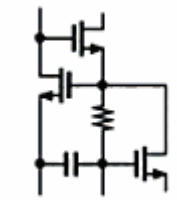


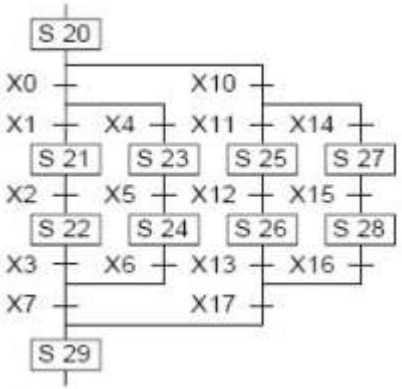

```
        60          70          80          90         100
AGACCCCCAG  CAACCCCCGG  GGGCGTGCGG  CGTCGGTCGT  GTCGTGTGAT
       160         170         180         190         200
AGACCCCGCG  TACGAATGCC  GGTCCACCAA  CAACCCGTGG  GCTTCGCAGC
       260         270         280         290         300
CTGCCGGGCA  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  GAGACTATTA  AAGCTCGGAA  ACAAGGTGGT
       660         670         680         690         700
ATAGTGGTTA  ACTTCACCTC  CAGACTCTTC  GCTGATGAAC  TGGCCGCCCT
       760         770         780         790         800
AAAATATACA  GGCATTGGGC  CTGGGGTGCG  TATGCTCACG  TGAGACATCT
       860         870         880         890         900
CCTGGAGGAG  GTTCGCCCGG  ACAGCCTGCG  CCTAACGCGG  ATGGATCCCT
       960         970         980         990        1000
AGCAACACCC  AGCTAGCAGT  GCTACCCCCA  TTTTTTAGCC  GAAAGGATTC
      1060        1070                Pvu II site 1090    1100
TGCCGCAGCA  ACTGGGGCAC  GCTATTCTGC  AGCAGCTGTT  GGTGTACCAC
      1160        1170        1180        1190        1200
ACTTGATCTA  TATACCACCA  ATGTGTCATT  TATGGGGCGC  ACATATCGTC
      1260        1270        1280        1290        1300
CTGTCCATGT  ACCTTTGTAT  CCTATCAGCC  TTGGTTCCCA  GGGGGTGTCT
      1360        1370        1380        1390        1400
TGTTTGAGGG  GGTGGTGCCA  GATGAGGTGA  CCAGGATAGA  TCTCGACCAG
      1460        1470        1480        1490        1500
TCAGAGTCTC  AGTTCTATAT  TTAATCTTGG  CCCCAGACTG  CACGTGTATG
      1560        1570        1580        1590        1600
CGATTTGAAG  CGGGGGGGGT  ATGGCGTCAT  CTGATATTCT  GTCGGTTGCA
      1660        1670        1680        1690        1700
AAAAACTACC  GTCTACCTGC  CGGACACTGA  ACCCTGGGTG  GTAGAGACCG
      1760        1770        1780        1790        1800
AAGCTTCATC  GTGGTGCCCT  GCCCTCAAAT  TCTCACAACG  GCTTGAGGAT

CTG.
```

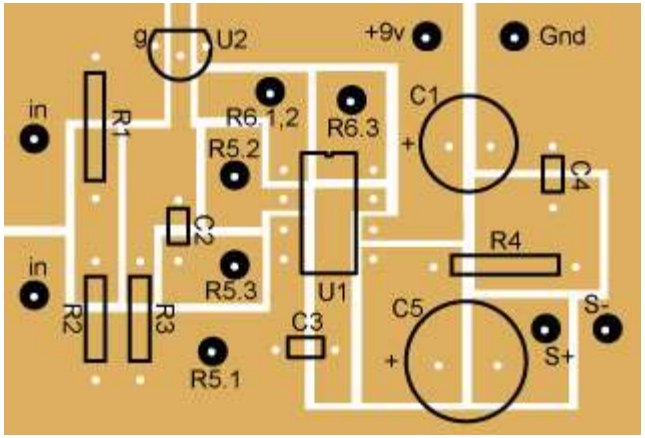
Electronics











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

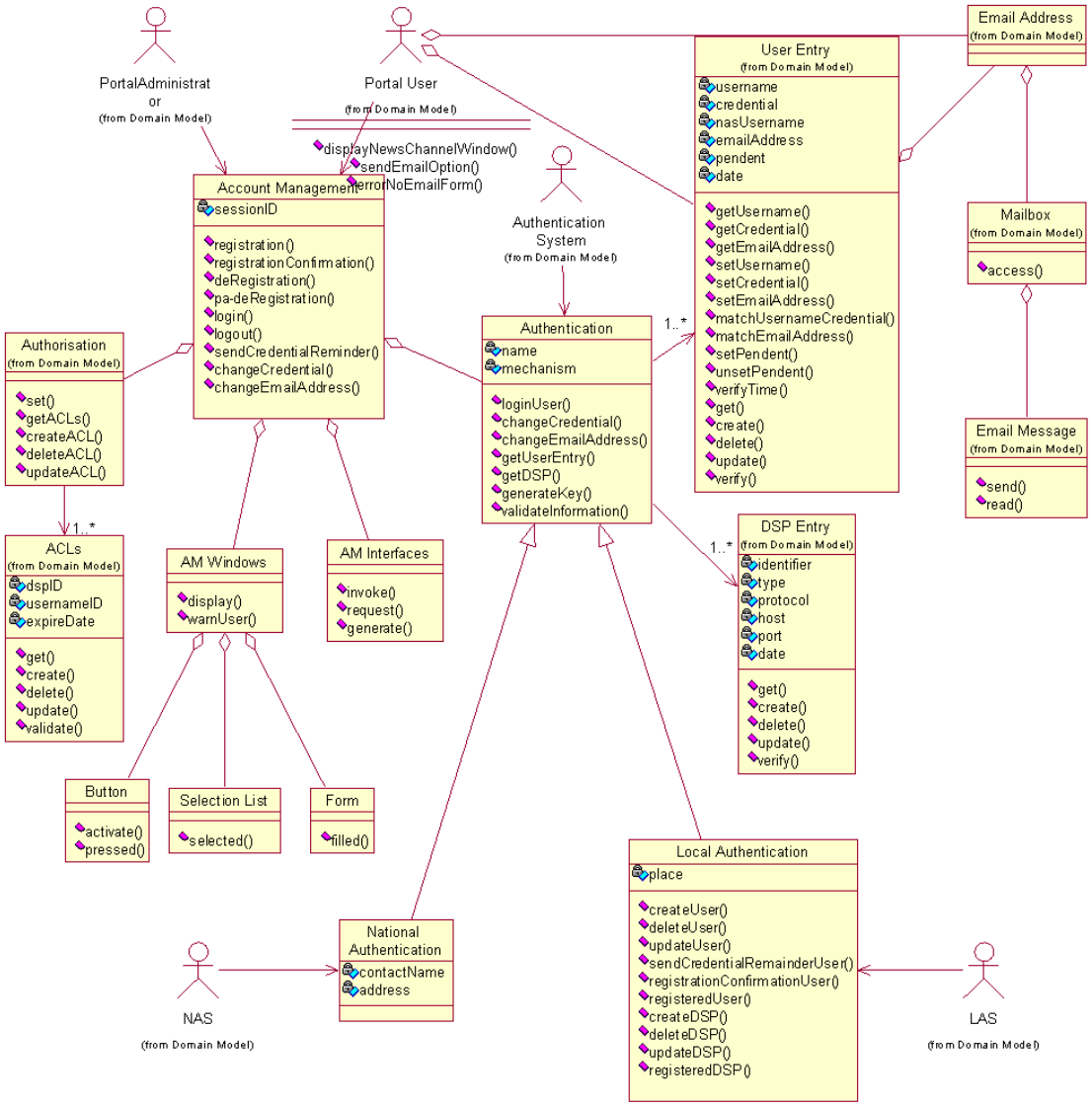
```
print("Hello world")
```

- 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


PortalAdministrator
(from Domain Model)
Portal User
(from Domain Model)
displayNewsChannelWindow()
sendEmailOption()
errorNoEmailForm()
Account Management
sessionID
registration()
registrationConfirmation()
deRegistration()
pa-deRegistration()
login()
logout()
sendCredentialReminder()
changeCredential()
changeEmailAddress()
Authentication System
(from Domain Model)
Email Address
(from Domain Model)
User Entry
(from Domain Model)
username
credential
nasUsername
emailAddress
pendent
date
getUsername()
getCredential()
getEmailAddress()
setUsername()
setCredential()
setEmailAddress()
matchUsernameCredential()
matchEmailAddress()
setPendent()
unsetPendent()
verifyTime()
get()
create()
delete()
update()
verify()
Mailbox
(from Domain Model)
access()
Email Message
(from Domain Model)
send()
read()
Authorisation
(from Domain Model)
set()
getACLs()
createACL()
deleteACL()
updateACL()
Authentication
name
mechanism
loginUser()
changeCredential()
changeEmailAddress()
getUserEntry()
getDSP()
generateKey()
validateInformation()
1..*
ACLs
(from Domain Model)
dspID
usernameID
expireDate
get()
create()
delete()
update()
validate()
AM Windows
display()
warnUser()
AM Interfaces
invoke()
request()
generate()
DSP Entry
(from Domain Model)
identifier
type
protocol
host
port
date
get()
create()
delete()
update()
verify()
Button
activate()
pressed()
Selection List
selected()
Form
filled()
Local Authentication
place
createUser()
deleteUser()
updateUser()
sendCredentialRemainderUser()
registrationConfirmationUser()
registeredUser()
createDSP()
deleteDSP()
updateDSP()
registeredDSP()
National Authentication
contactName
address
NAS
(from Domain Model)
LAS
(from Domain Model)

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\textit{#2}}
    }
    \newcommand{\version}{\scriptsize$-$working$-$}
  }
  {\newcommand{\nb}[2]{}
    \newcommand{\version}{}
}
```

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

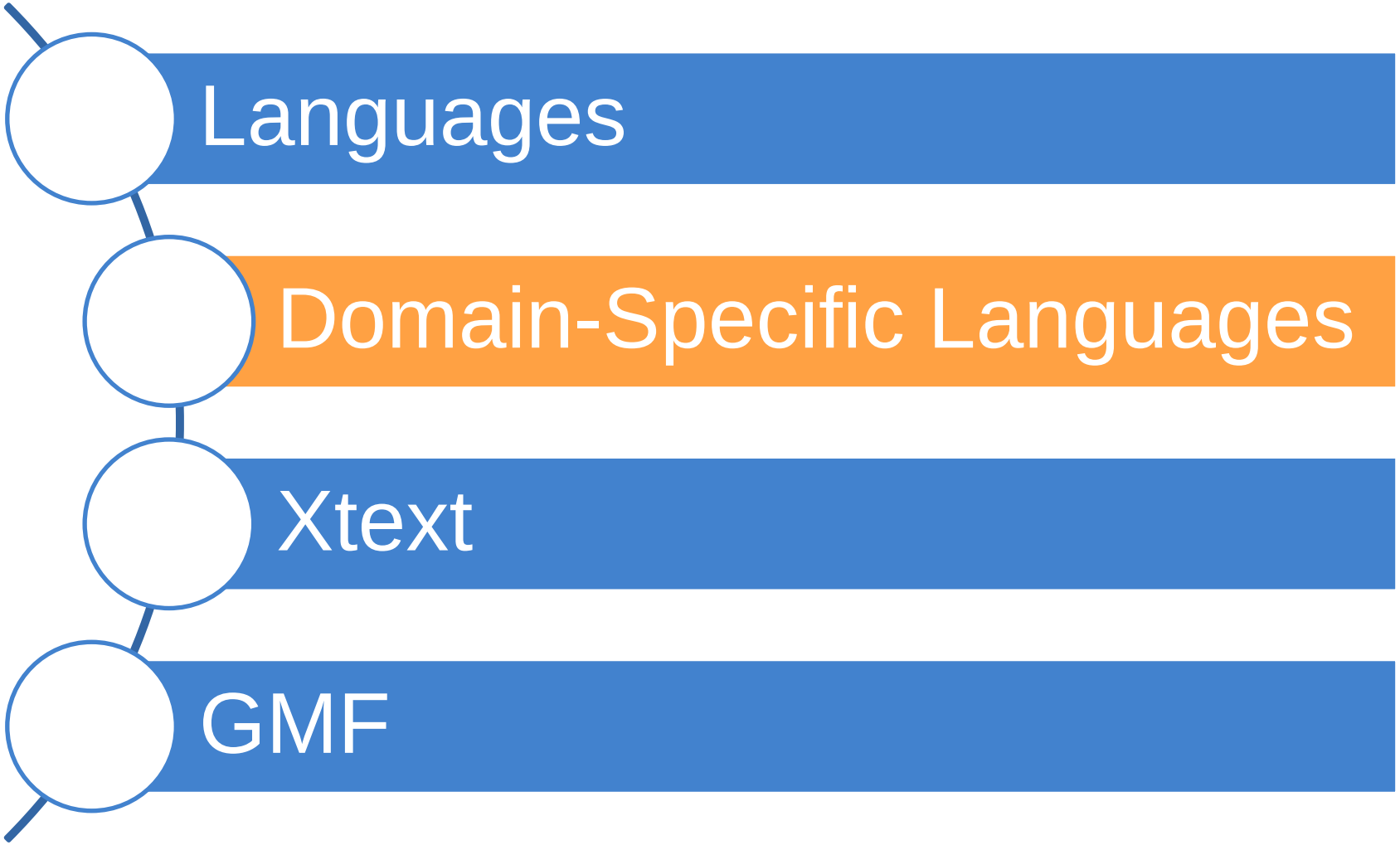
The boundary isn't as clear as it could be. Domain-specificity is not black-and-white, but instead gradual: a language is more or less domain specific

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

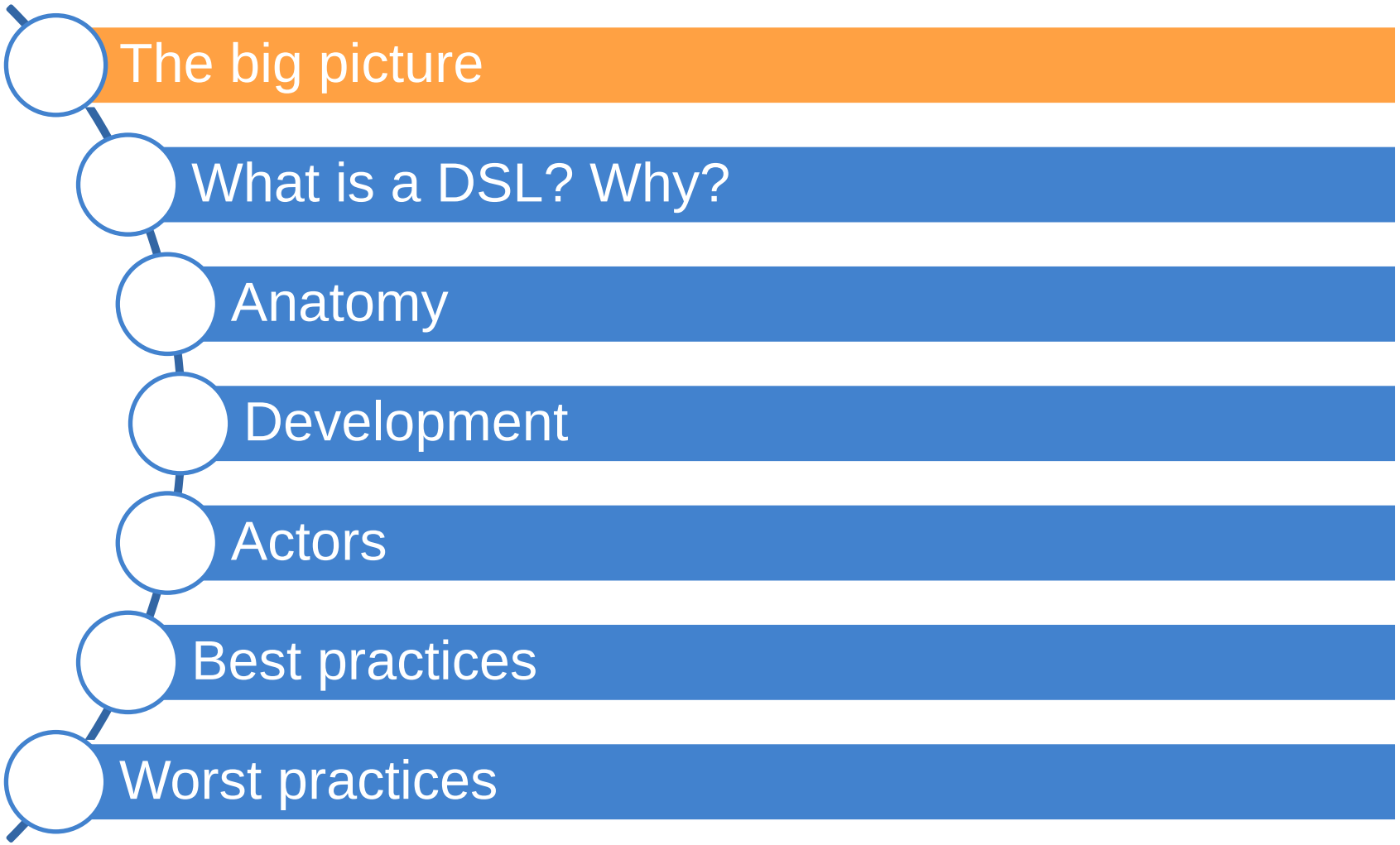
Outline



DSL

- The big picture
- What is a DSL? Why?
- Anatomy
- Development
- Actors
- Best practices
- Worst practices

DSL

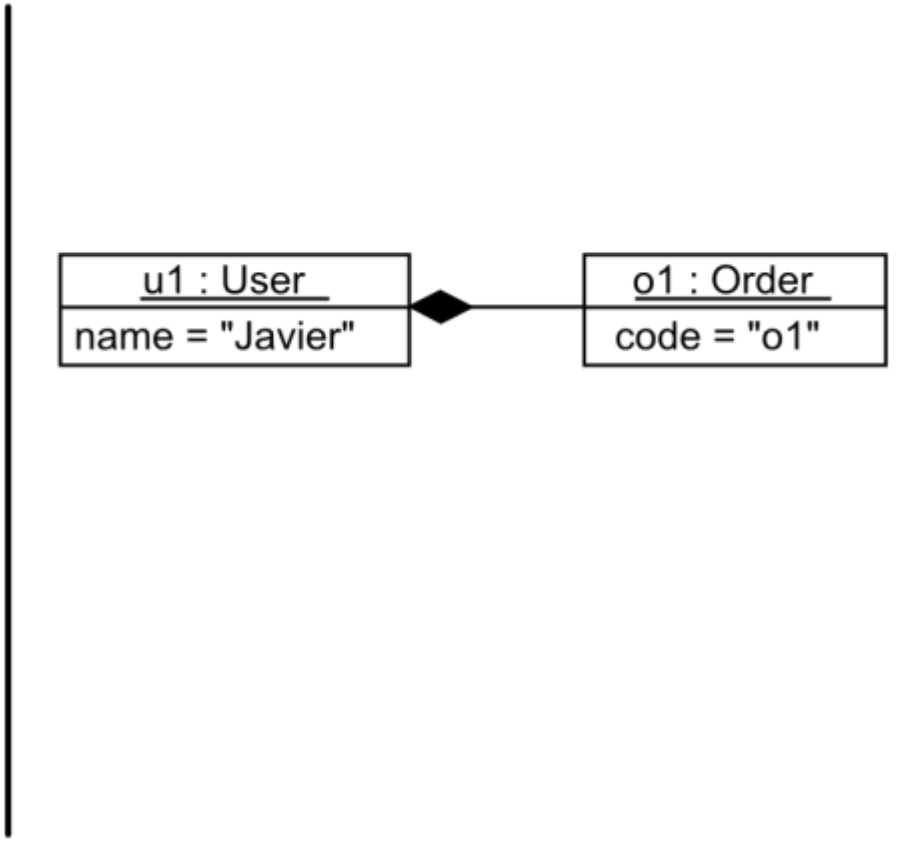

The big picture
What is a DSL? Why?
Anatomy
Development
Actors
Best practices
Worst practices

The big picture

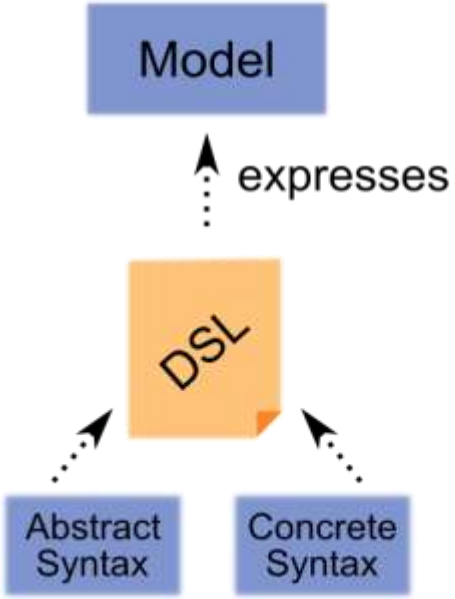
Domain-Specific Languages?

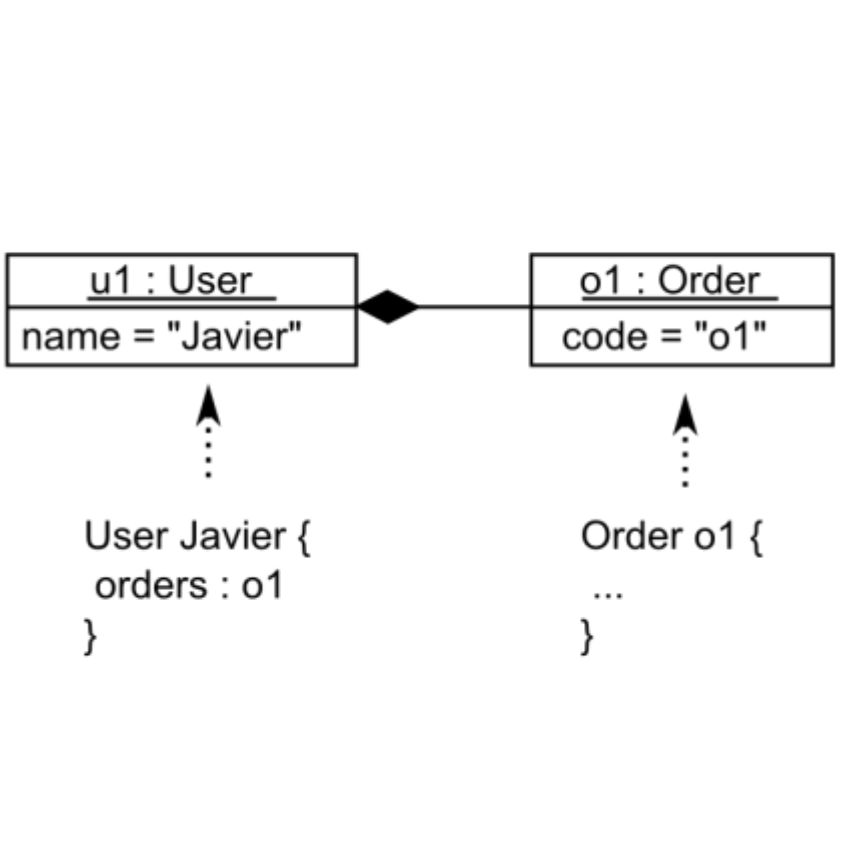
Concepts

Model

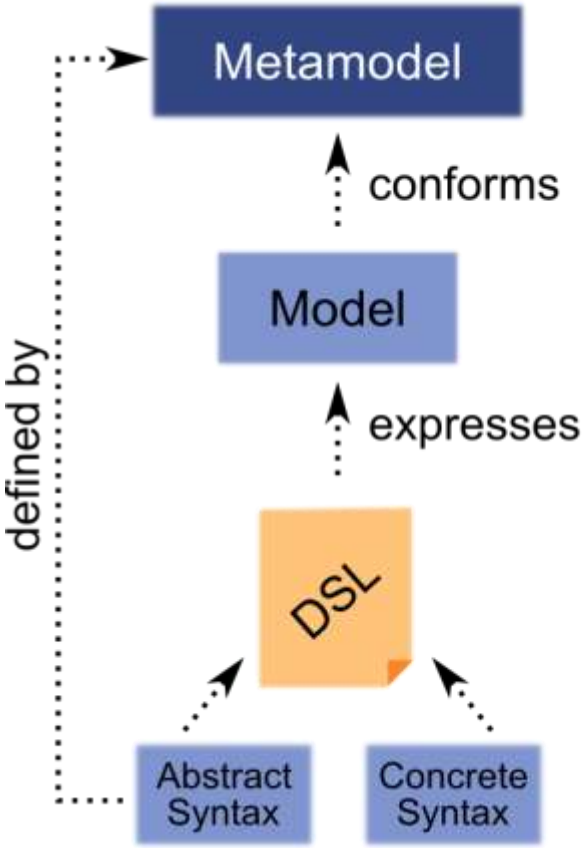


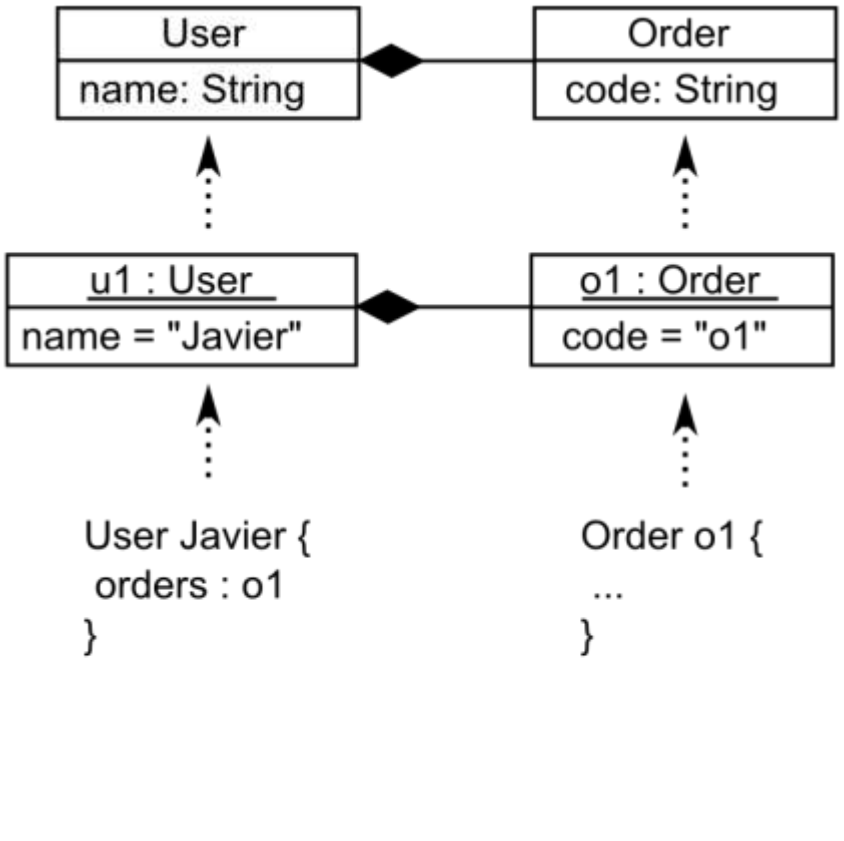
Concepts


Model
expresses
DSL
Abstract Syntax
Concrete Syntax

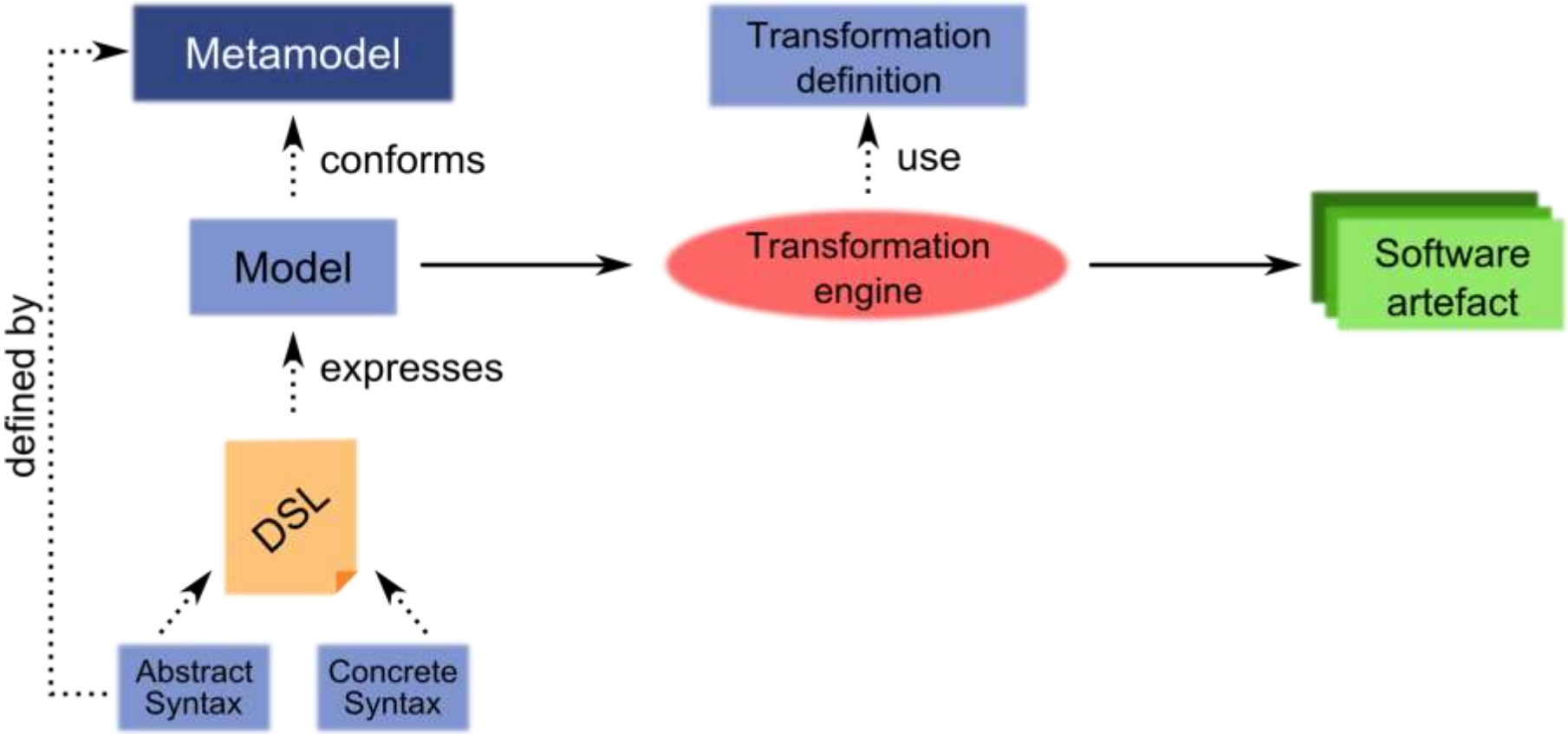

u1 : User
name = "Javier"
o1 : Order
code = "o1"
User Javier {
orders : o1
}
Order o1 {
...
}

Concepts

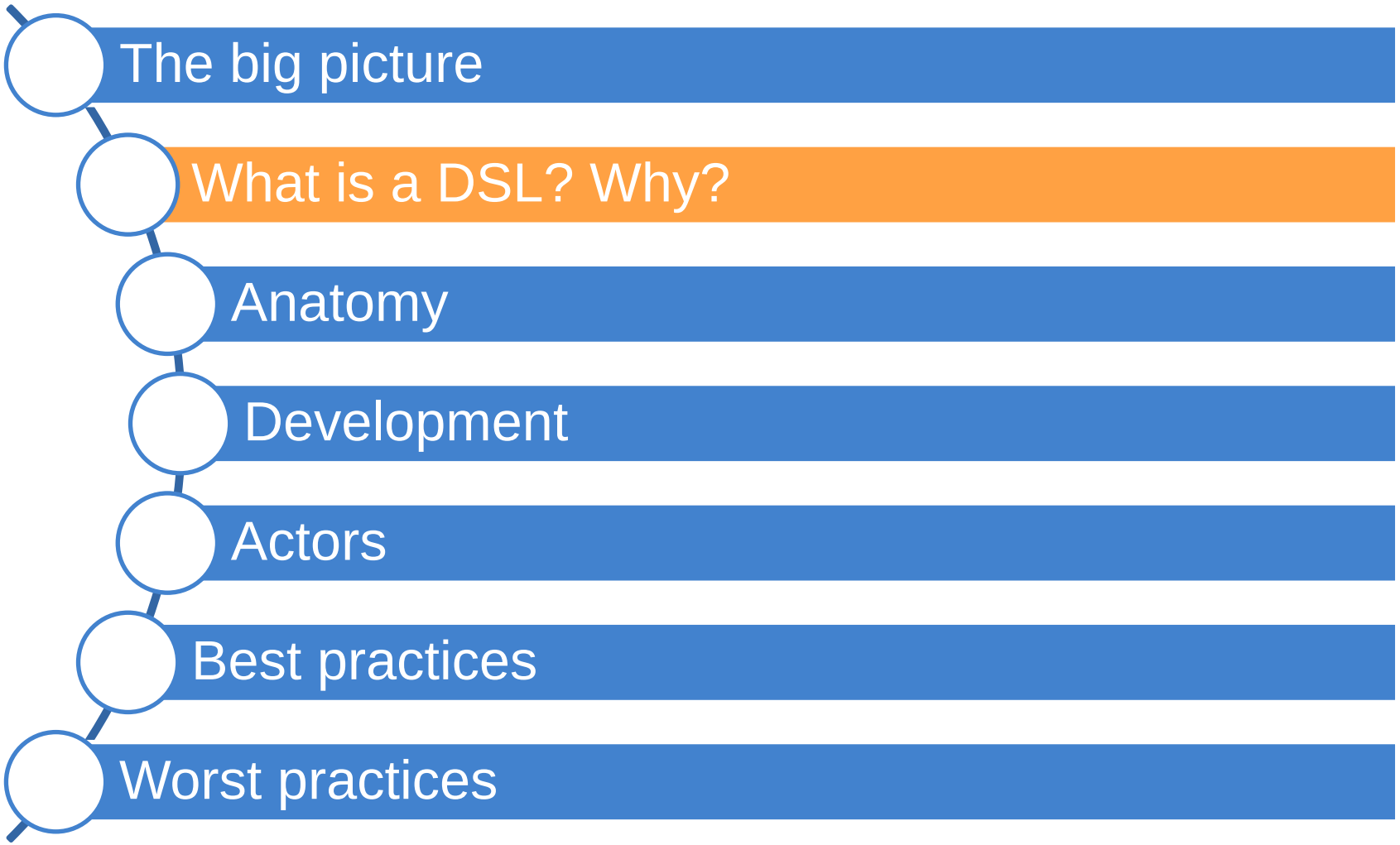

Metamodel
conforms
Model
expresses
DSL
defined by
Abstract Syntax
Concrete Syntax


User
name: String
Order
code: String
u1 : User
name = "Javier"
o1 : Order
code = "o1"
User Javier {
orders : o1
}
Order o1 {
...
}

Concepts


Metamodel
conforms
Model
expresses
DSL
Abstract Syntax
Concrete Syntax
defined by
Transformation definition
use
Transformation engine
Software artefact

DSL


The big picture
What is a DSL? Why?
Anatomy
Development
Actors
Best practices
Worst practices

What is a DSL

- In general:

  Language specially designed to perform a task in a certain domain

What is a DSL

- In general:

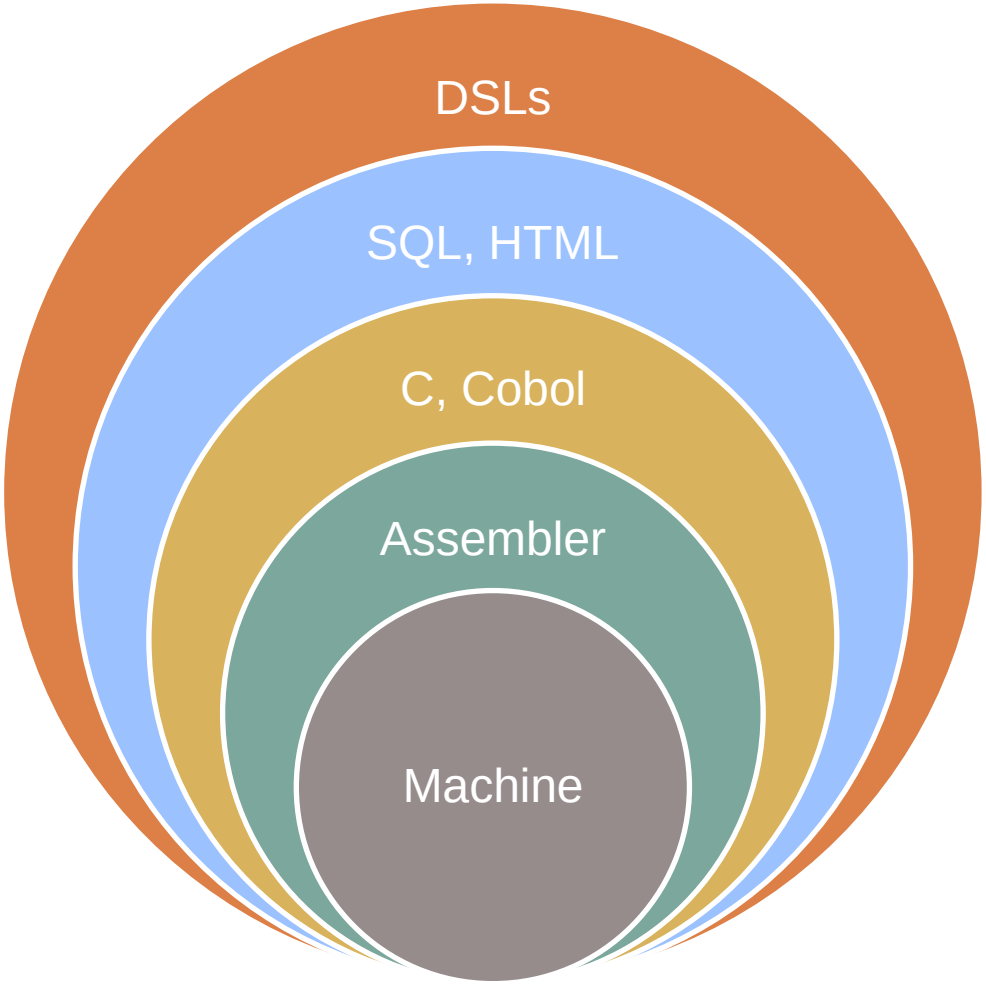
  Language specially designed to perform a task in a certain domain

- In the context of computer science:

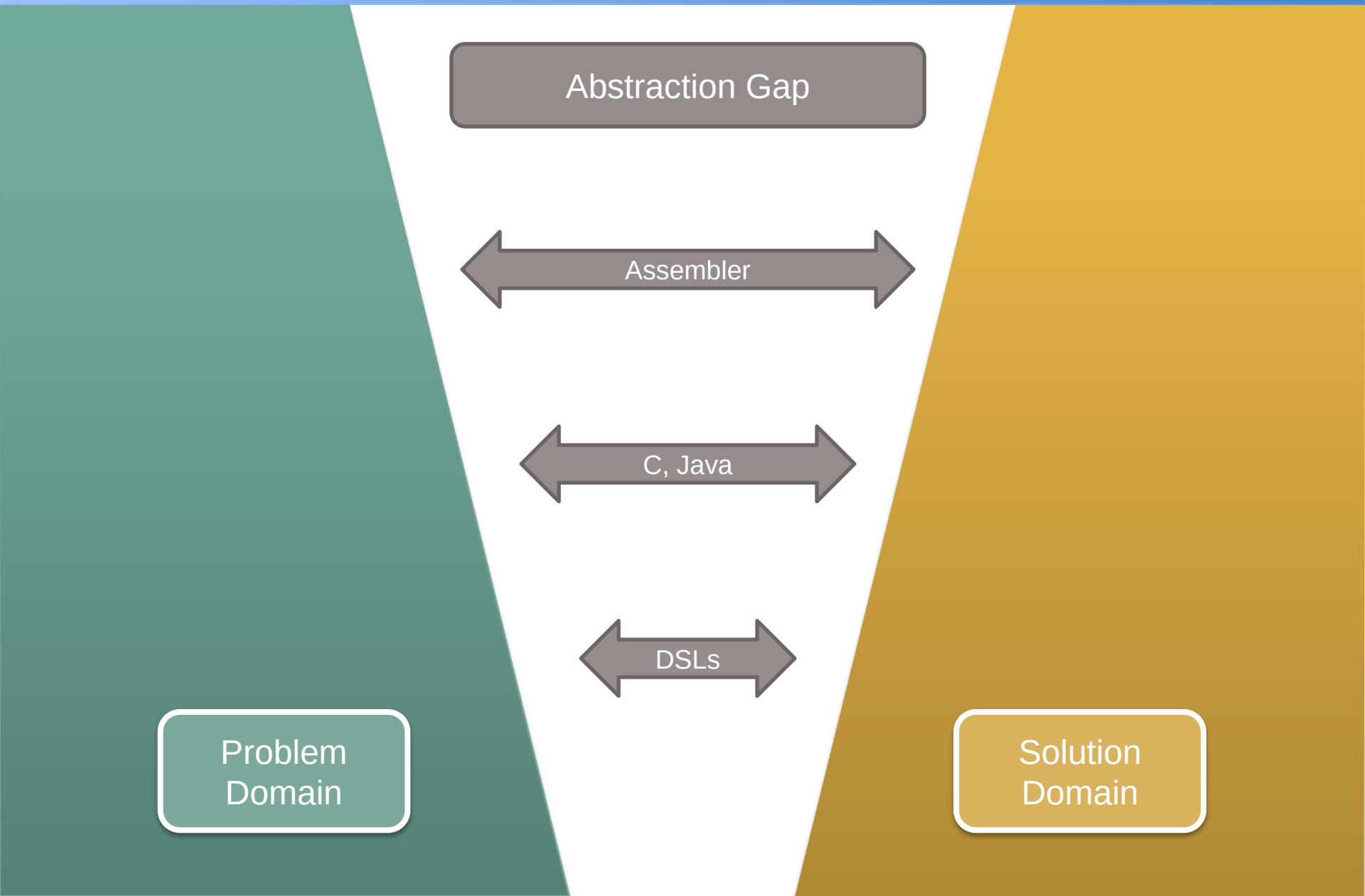
  A formal processable language targeting at a specific viewpoint or aspect of a software system.

  It's 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?


Abstraction Gap
Assembler
C, Java
DSLs
Problem Domain
Solution Domain

What is offered?

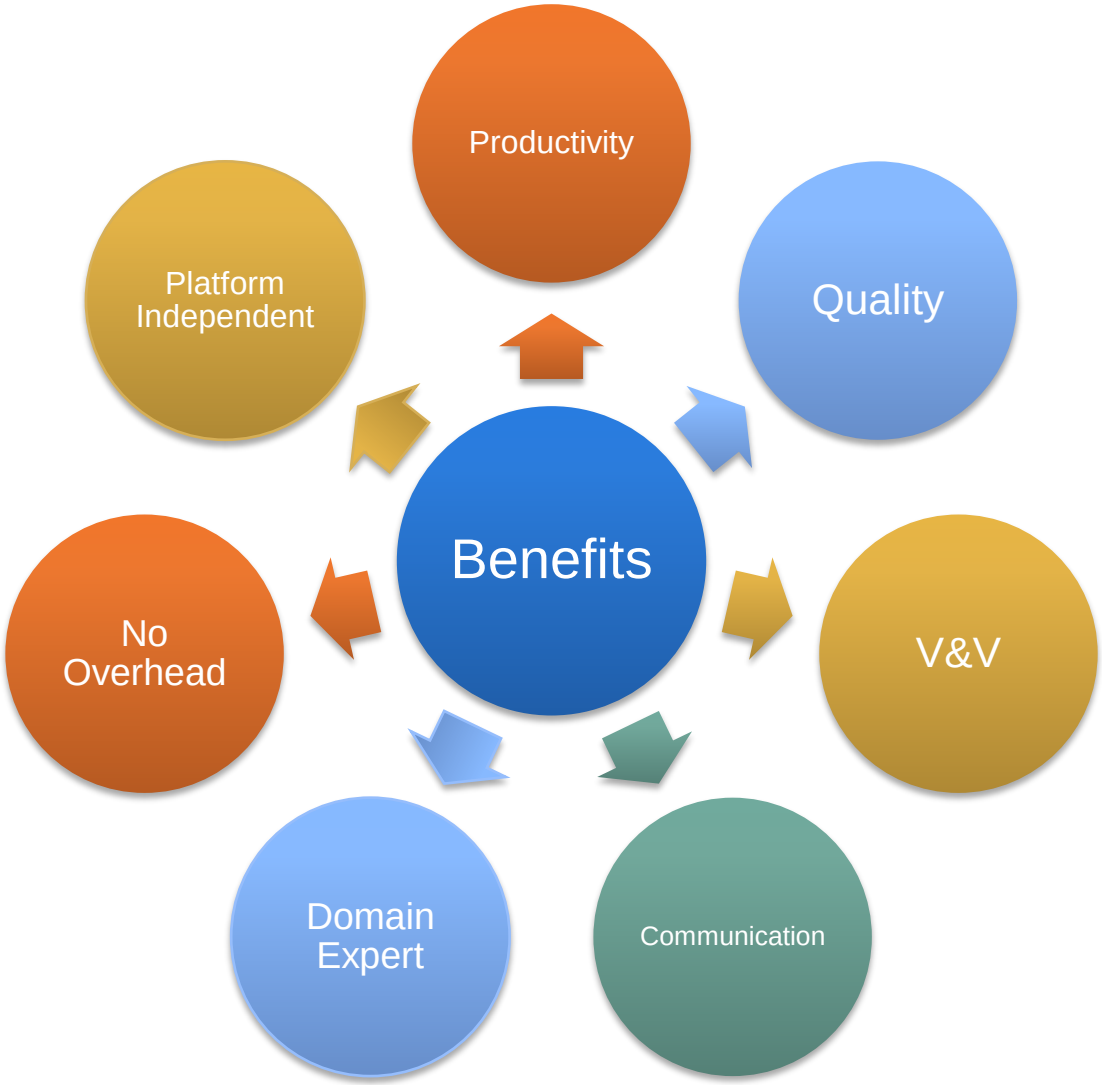
Higher abstractions

Avoid redundancy

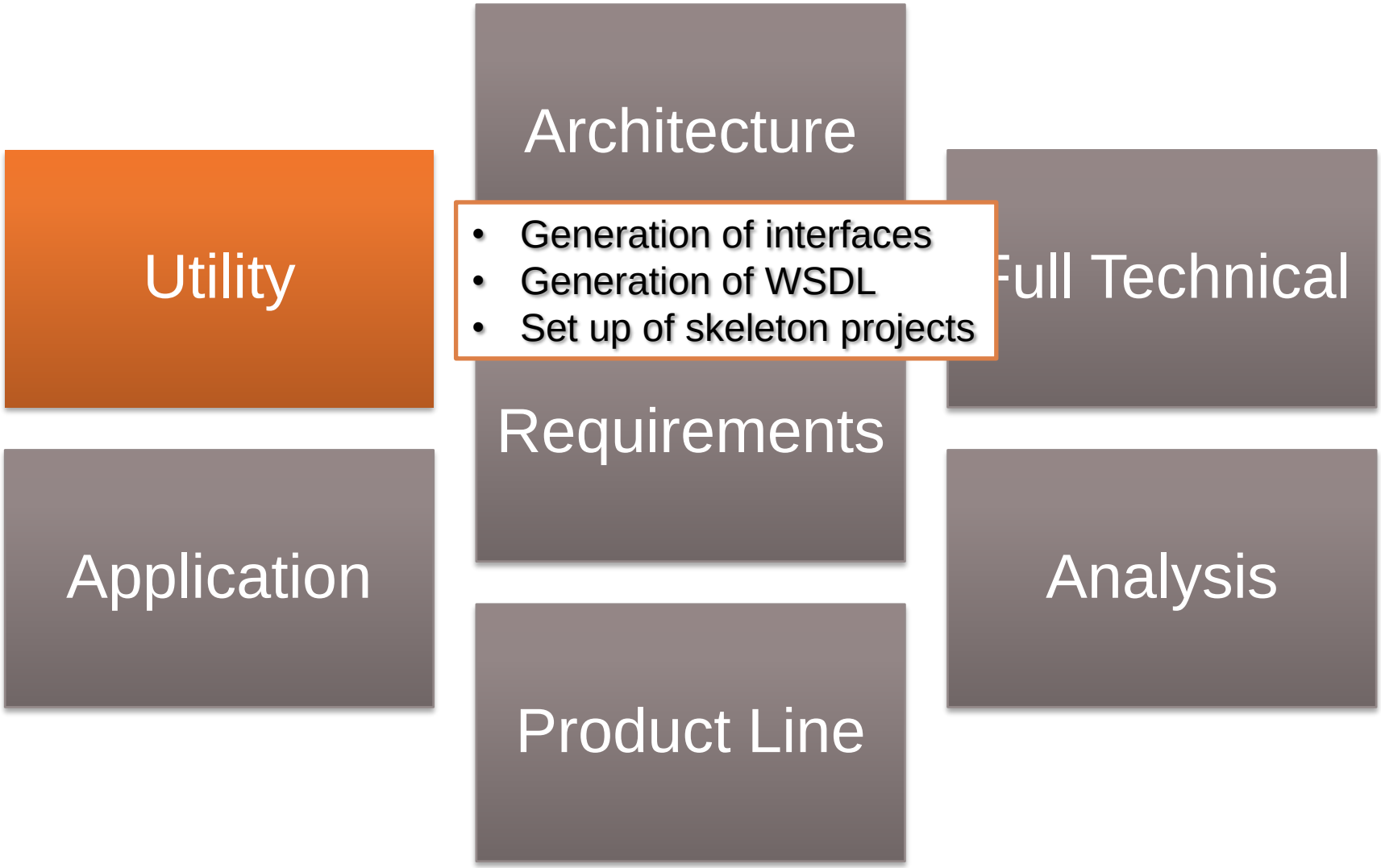
Separation of concerns

Use domain concepts

What are the advantages?

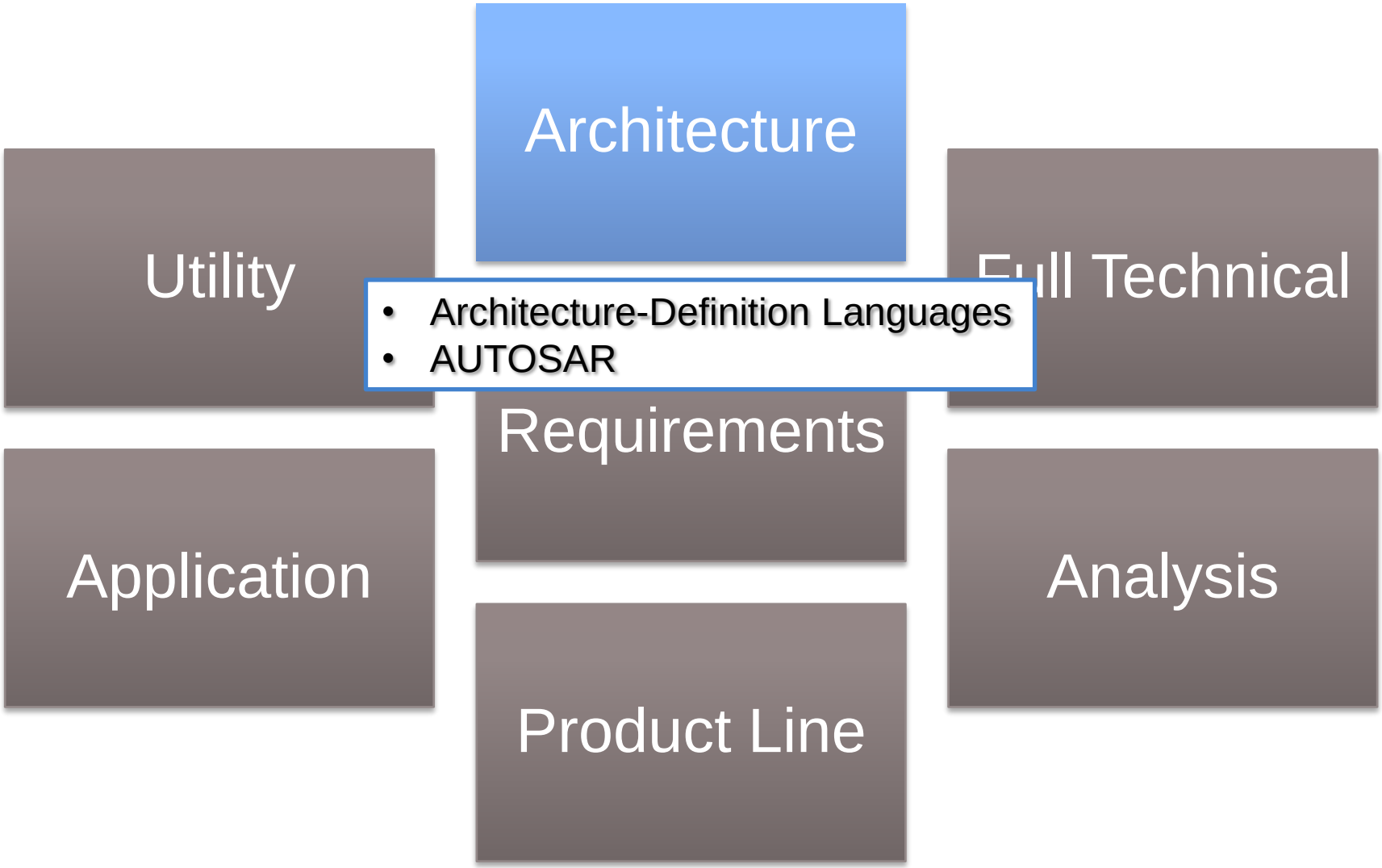


What for?

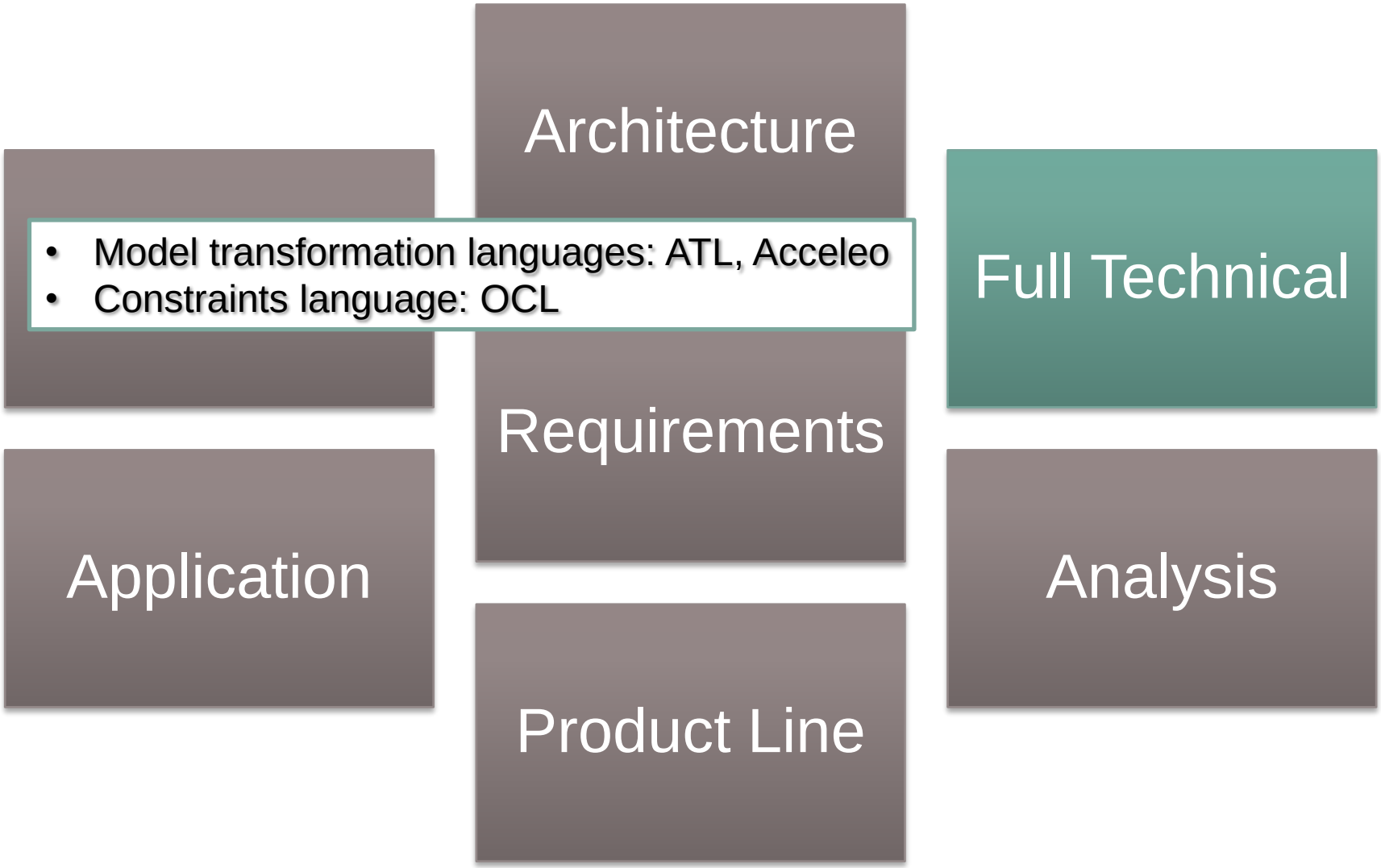


- Generation of interfaces
- Generation of WSDL
- Set up of skeleton projects

What for?

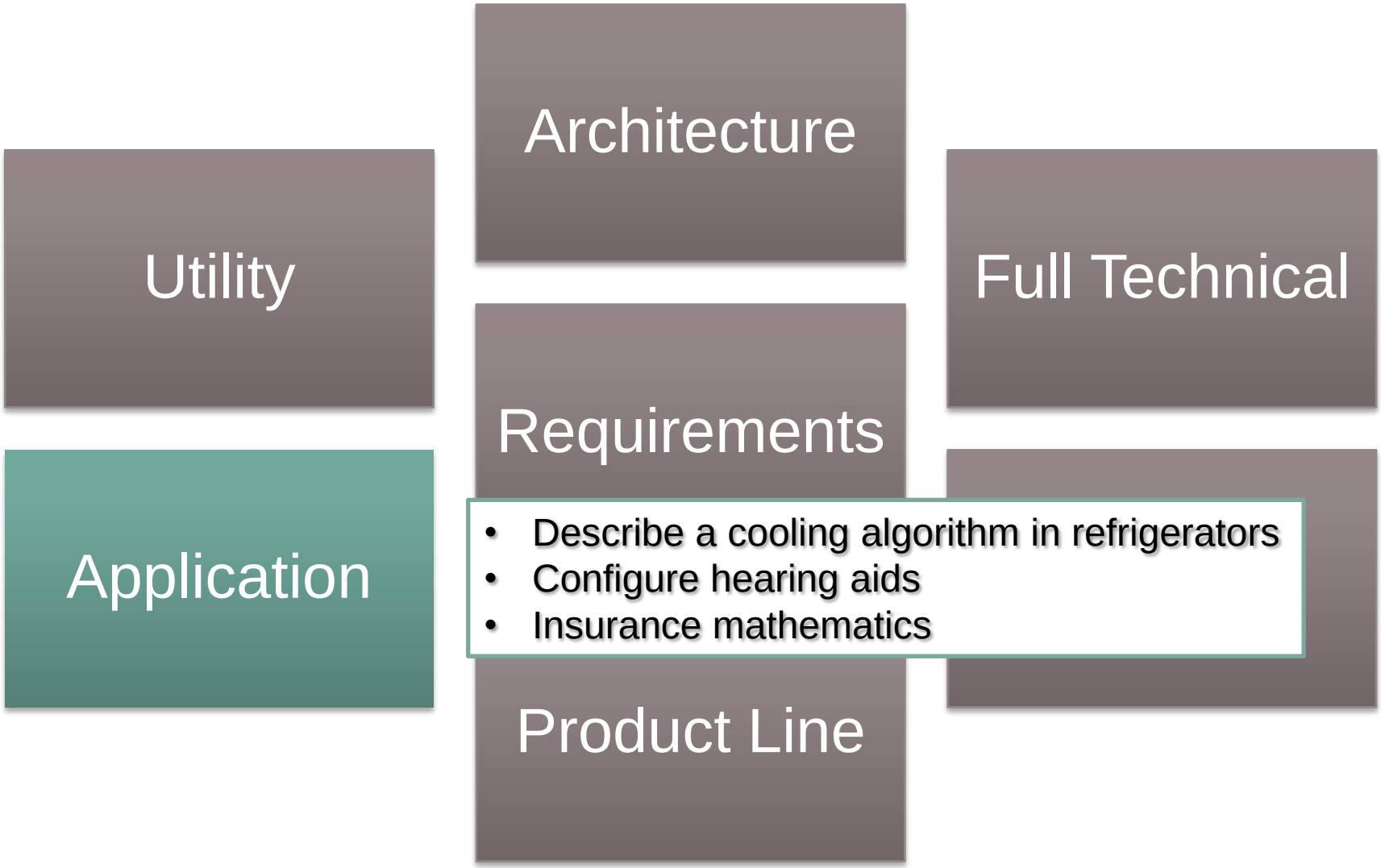

Architecture
Utility
Full Technical
• Architecture-Definition Languages
• AUTOSAR
Requirements
Application
Analysis
Product Line

What for?

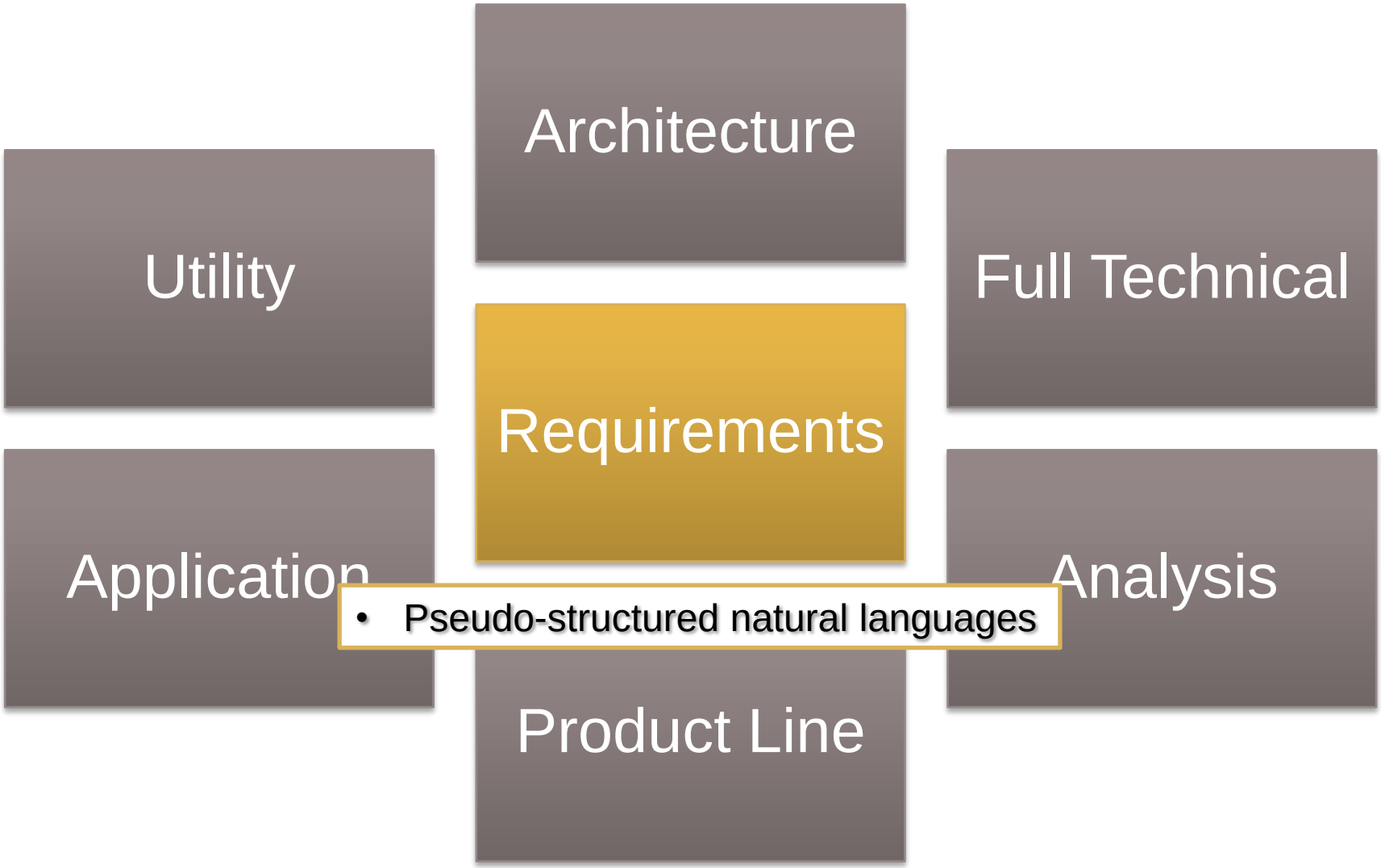


- Model transformation languages: ATL, Acceleo
- Constraints language: OCL

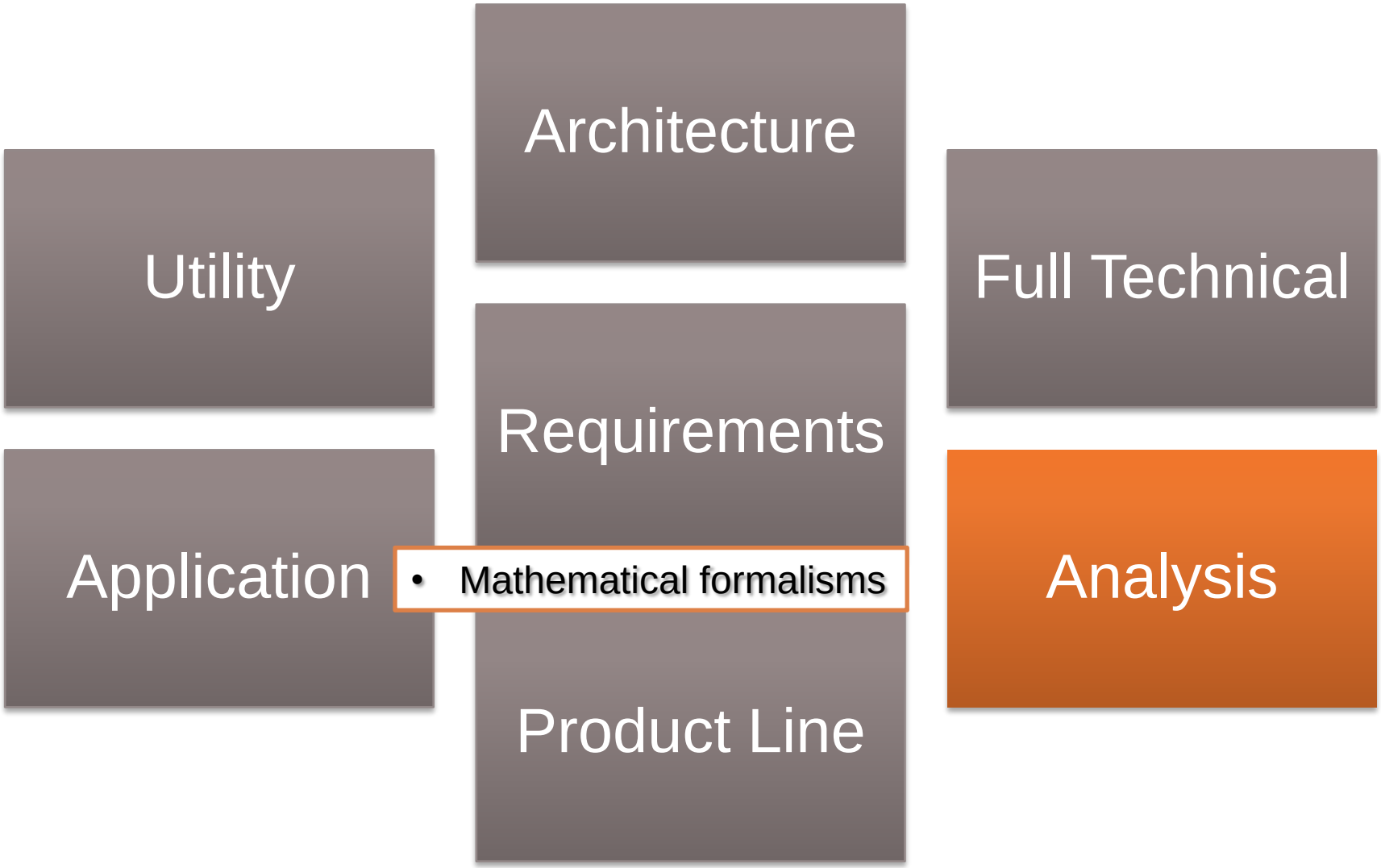
What for?


Architecture
Utility
Full Technical
Requirements
Application
Product Line
• Describe a cooling algorithm in refrigerators
• Configure hearing aids
• Insurance mathematics

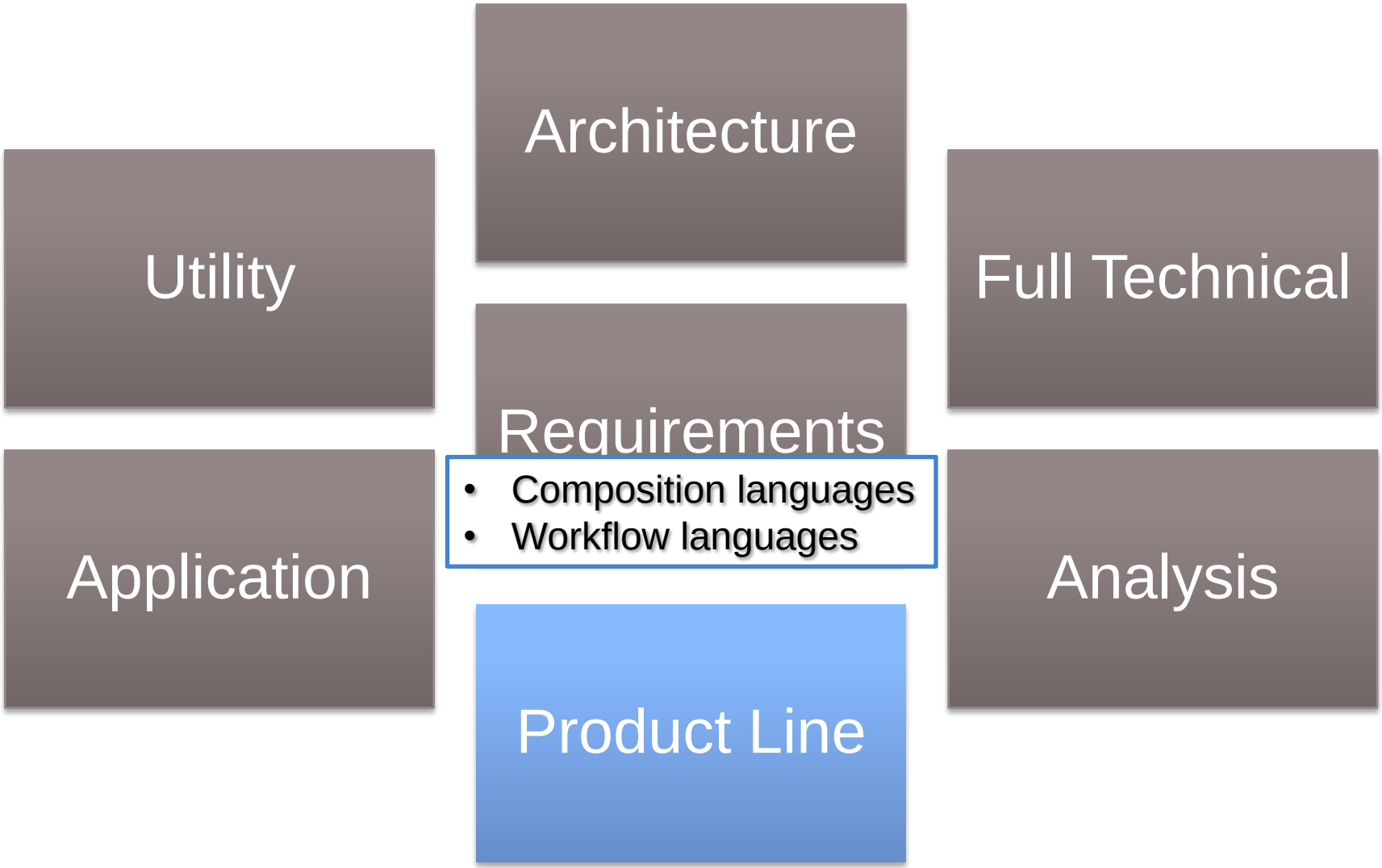
What for?


Architecture
Utility
Full Technical
Requirements
Application
Analysis
Product Line
• Pseudo-structured natural languages

What for?


Architecture
Utility
Full Technical
Requirements
Application
• Mathematical formalisms
Analysis
Product Line

What for?



- Composition languages
- Workflow languages

What is a domain?

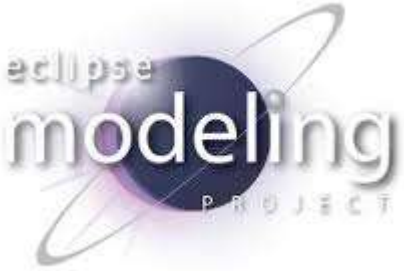
Inductive / Bottom-up

- The domain is defined in terms of existing software used to address a particular class of problems or products
- A domain D is identified as a set of programs with common characteristics or similar purpose
- Often such domains do not exists outside the realm of software
- Special case when we define a domain as a subset of programs written in a specific language (idiom)

Deductive / top-down

- The domain is defined as a body of knowledge about the real world, i.e., outside the realm of software
- A domain D is a body of knowledge for which we want to provide some form of software support
- Much harder to address using DSLs because of the need to understand precisely the nature of the domain and identify interesting programs in such domain

but… how?


eclipse
modeling
PROJECT


MPS
Meta Programming System


Domain-Specific
Development
with Visual Studio DSL Tools
Microsoft .net Development Series
Steve Cook
Gareth Jones
Stuart Kent
Alan Cameron Wills

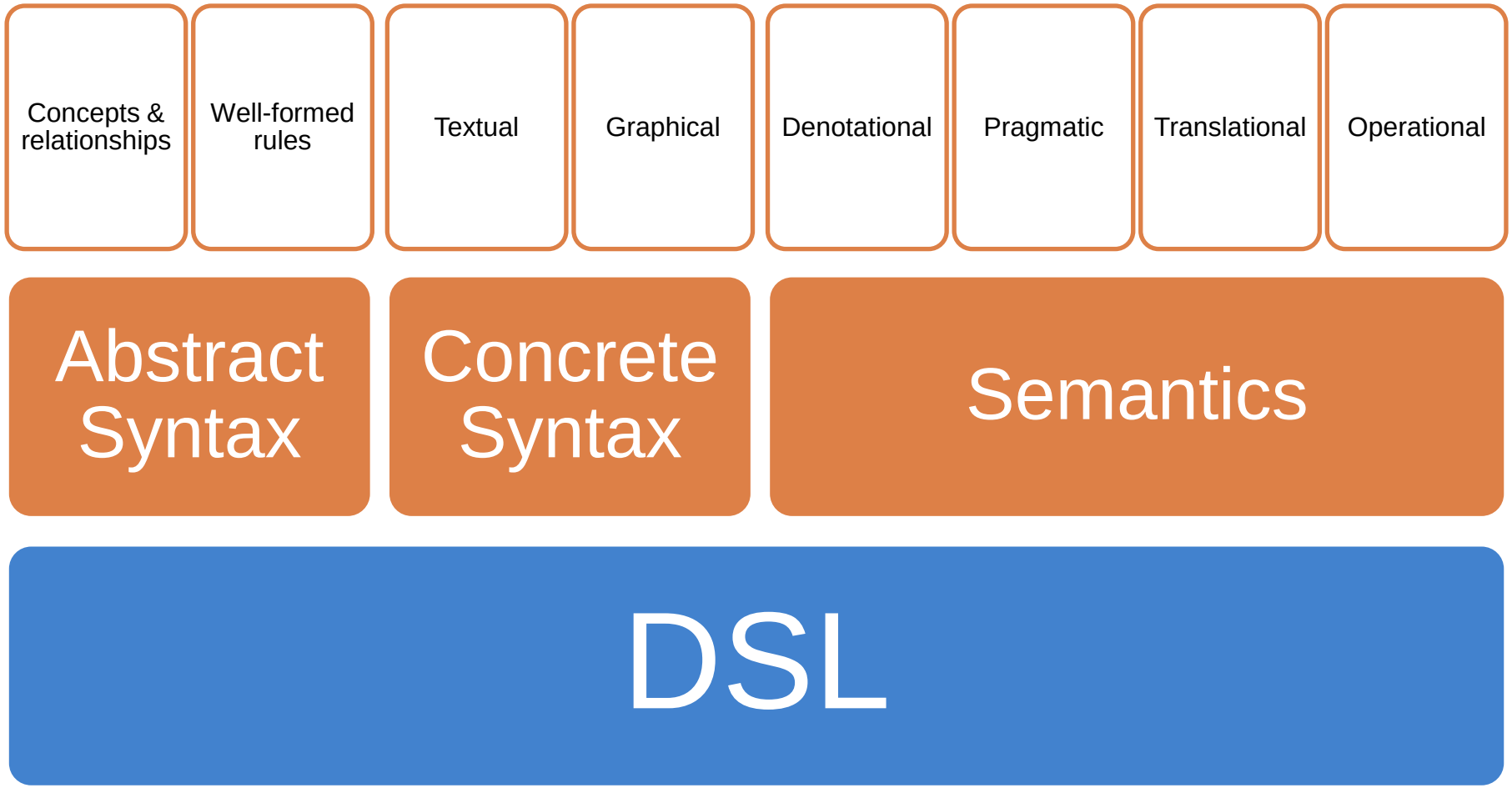

MetaEdit+


GME 6

DSL

- The big picture
- What is a DSL? Why?
- Anatomy
- Development
- Actors
- Best practices
- Worst practices

Anatomy


Concepts & relationships
Well-formed rules
Textual
Graphical
Denotational
Pragmatic
Translational
Operational
Abstract Syntax
Concrete Syntax
Semantics
DSL

A first look

Abstract Syntax

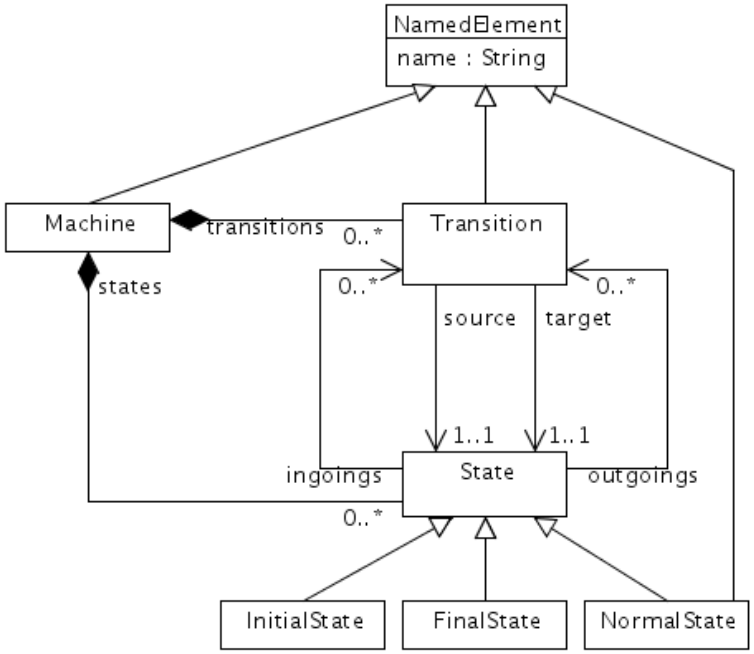
- Describes the structure of the language and the way the different primitives can be combined together, independently of any particular representation or encoding.

Concrete Syntax

- Describes specific representations of the modeling language, covering encoding and/or visual appearance

Semantics

- Describing the meaning of the elements defined in the language and the meaning of the different ways of combining them.



A first look

Abstract Syntax

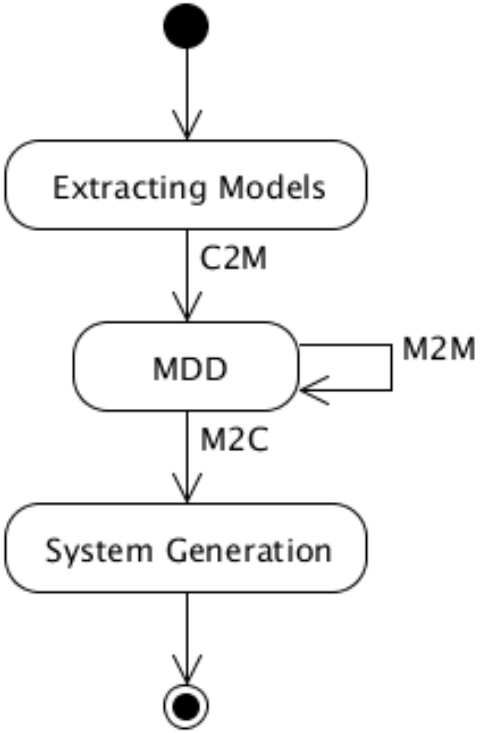
- Describes the structure of the language and the way the different primitives can be combined together, independently of any particular representation or encoding.

Concrete Syntax

- Describes specific representations of the modeling language, covering encoding and/or visual appearance

Semantics

- Describing the meaning of the elements defined in the language and the meaning of the different ways of combining them.



A first look

Abstract Syntax

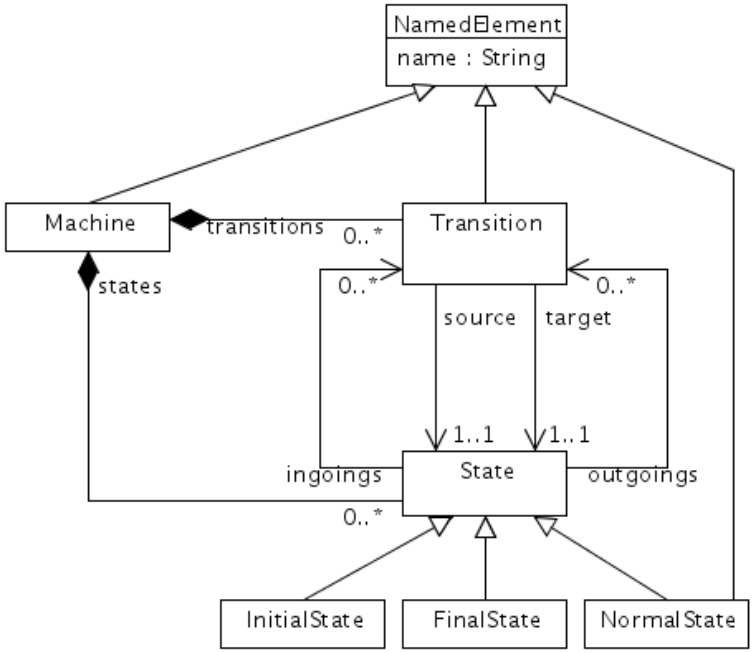
- Describes the structure of the language and the way the different primitives can be combined together, independently of any particular representation or encoding.

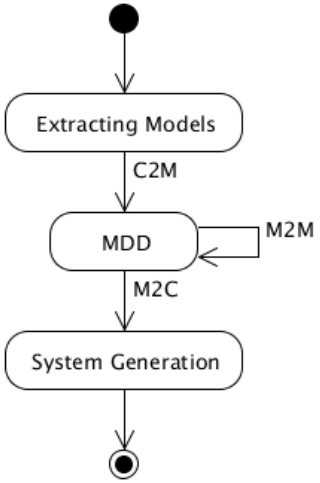
Concrete Syntax

- Describes specific representations of the modeling language, covering encoding and/or visual appearance

Semantics

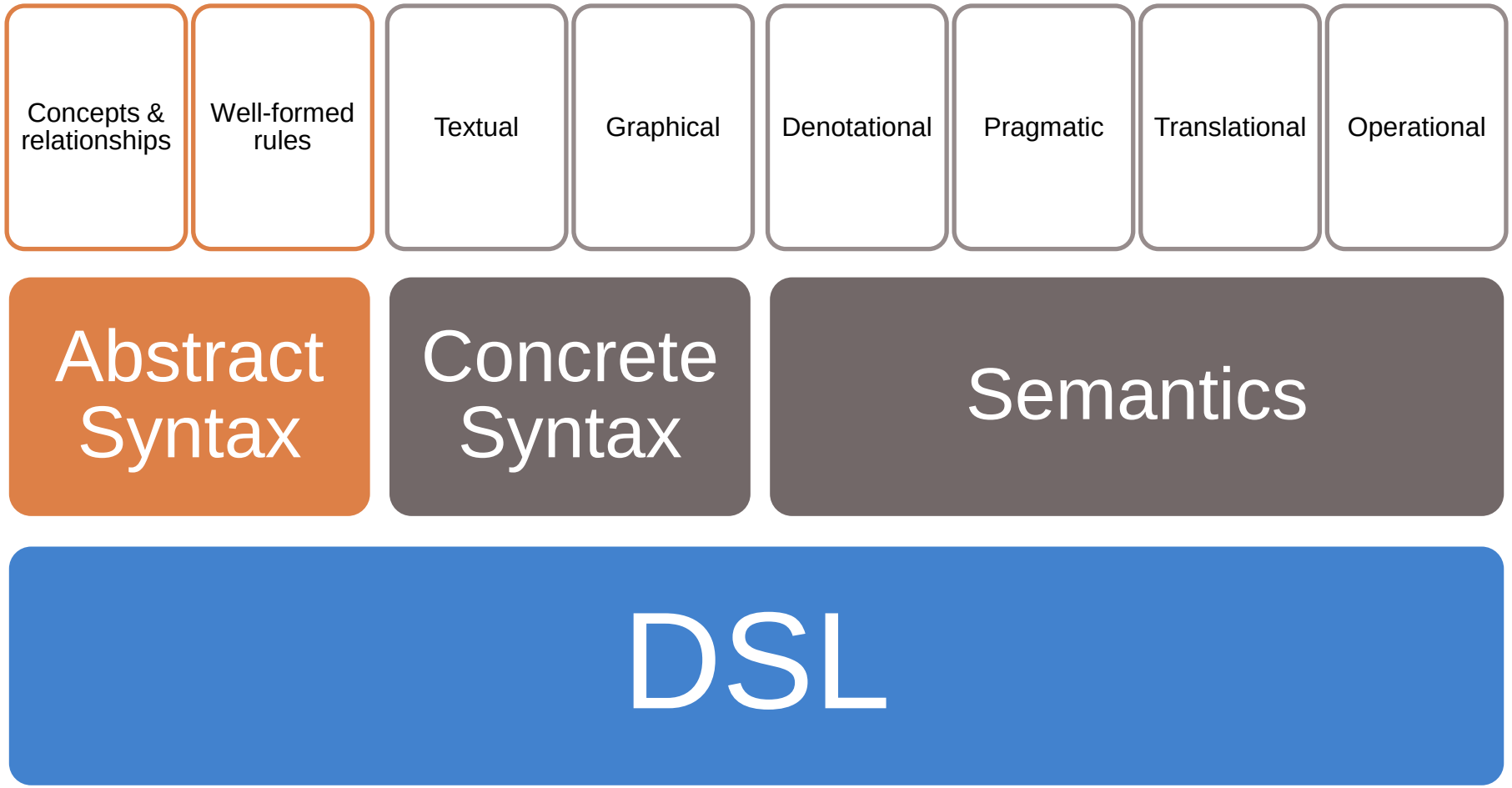
- Describing the meaning of the elements defined in the language and the meaning of the different ways of combining them.





How this is executed?

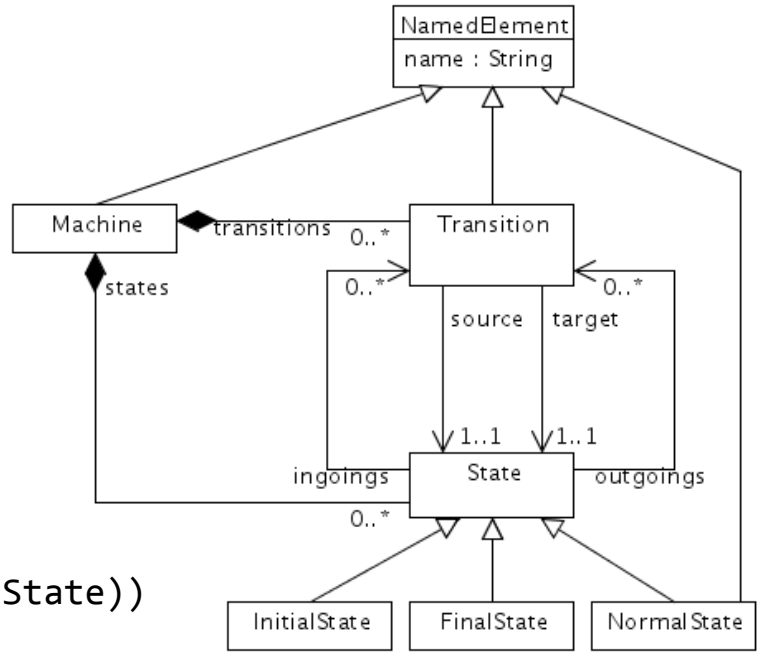
Anatomy


Concepts & relationships
Well-formed rules
Textual
Graphical
Denotational
Pragmatic
Translational
Operational
Abstract Syntax
Concrete Syntax
Semantics
DSL

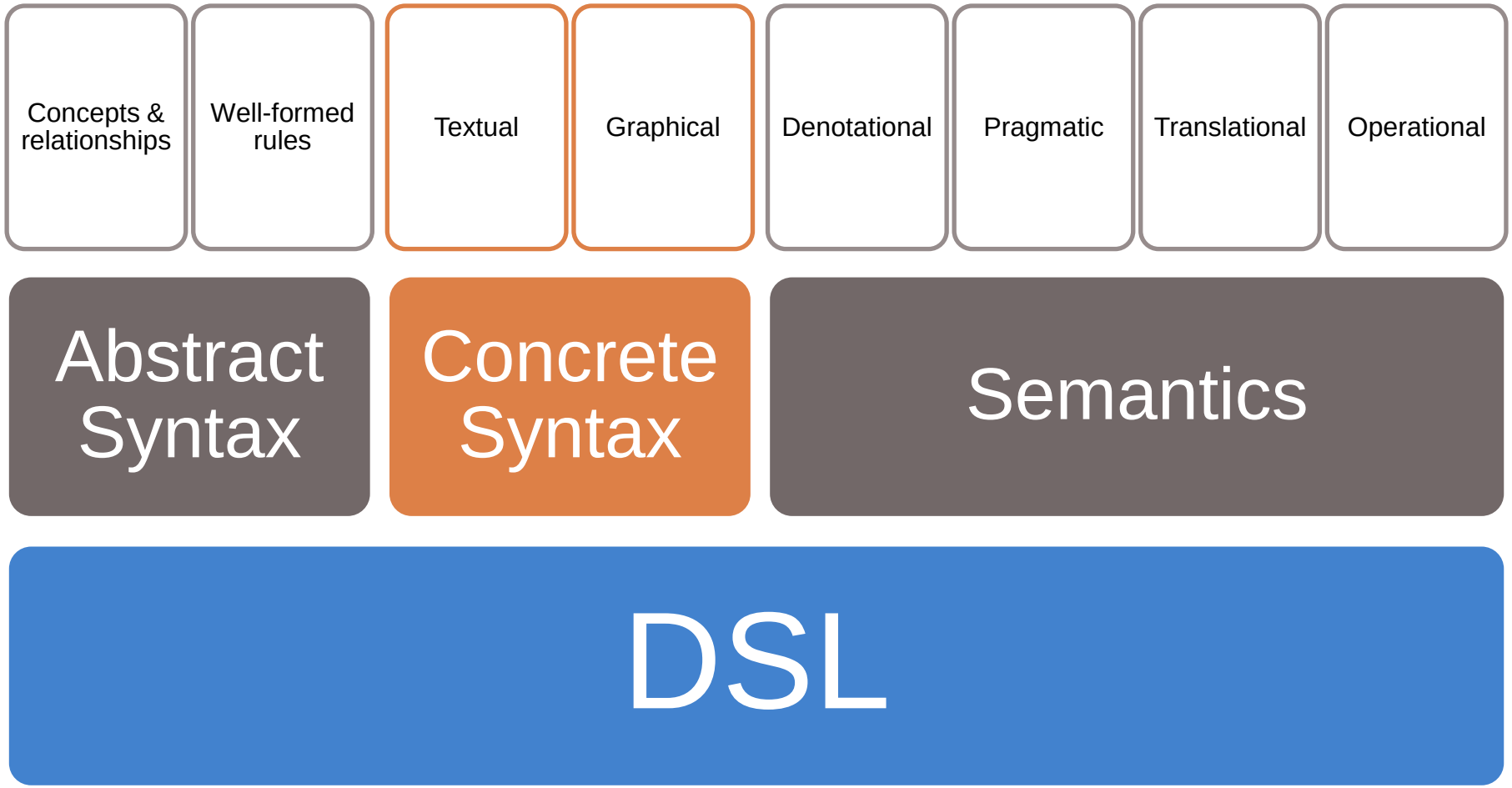
Abstract Syntax

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

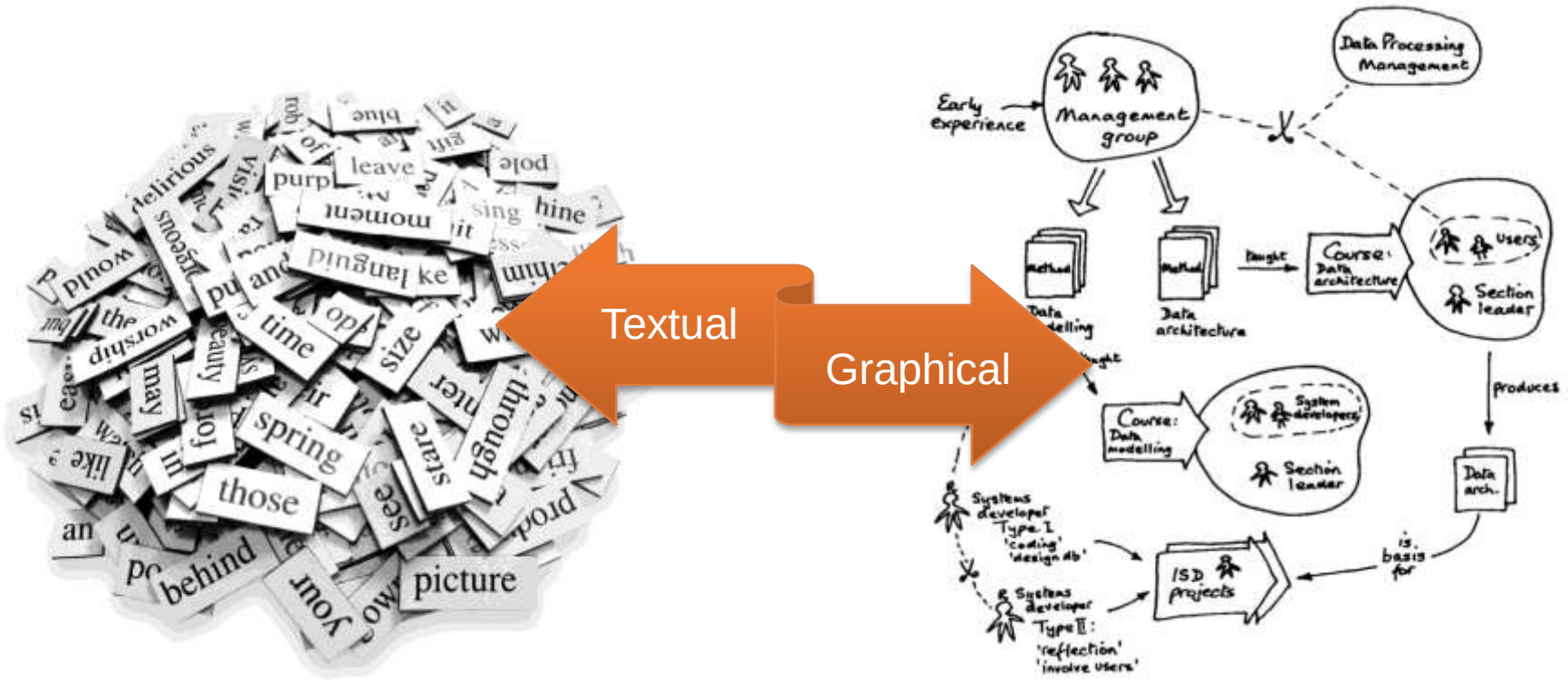
```
context Transition
inv: self.source <> self.target

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



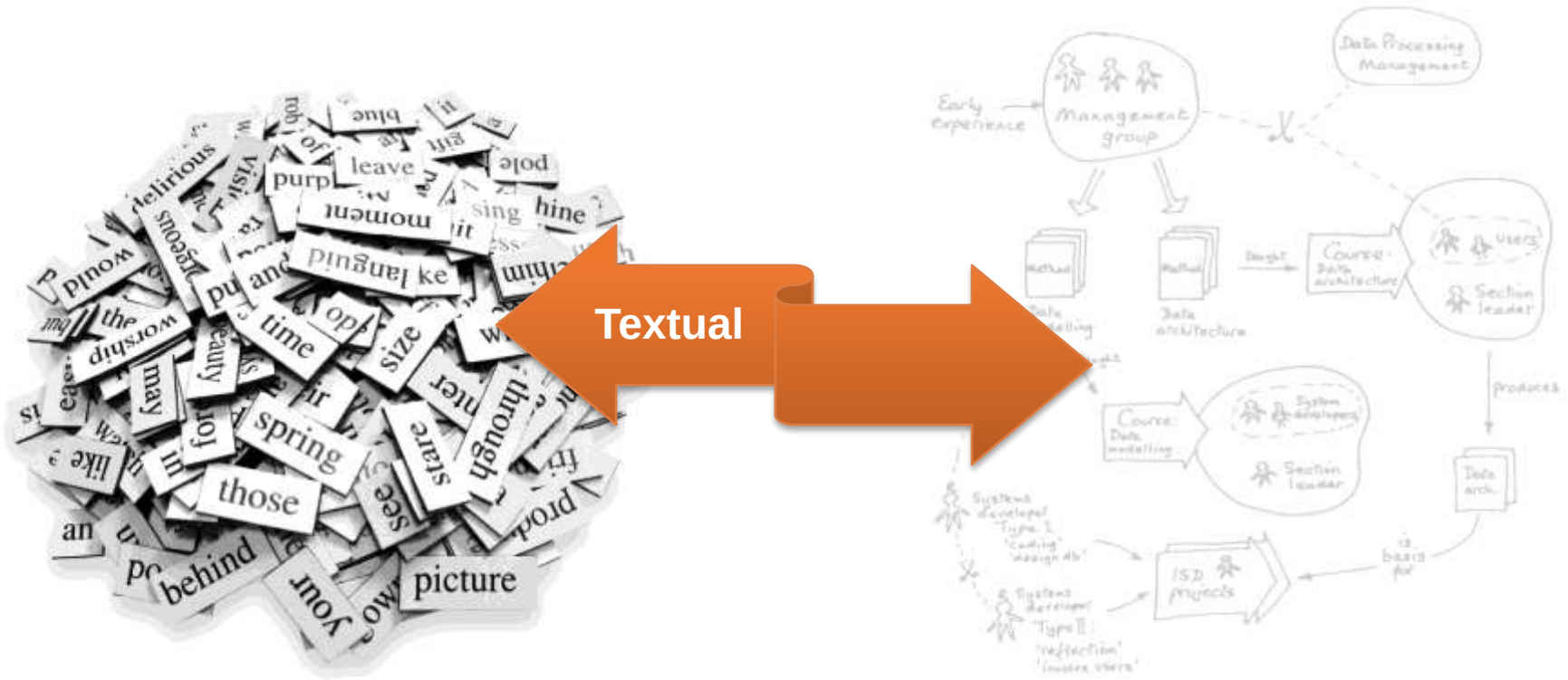
Anatomy


Concepts & relationships
Well-formed rules
Textual
Graphical
Denotational
Pragmatic
Translational
Operational
Abstract Syntax
Concrete Syntax
Semantics
DSL

Concrete Syntax



Concrete Syntax

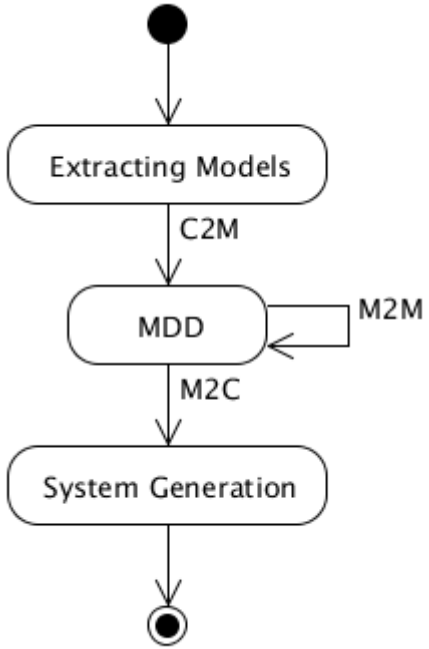


Concrete Syntax

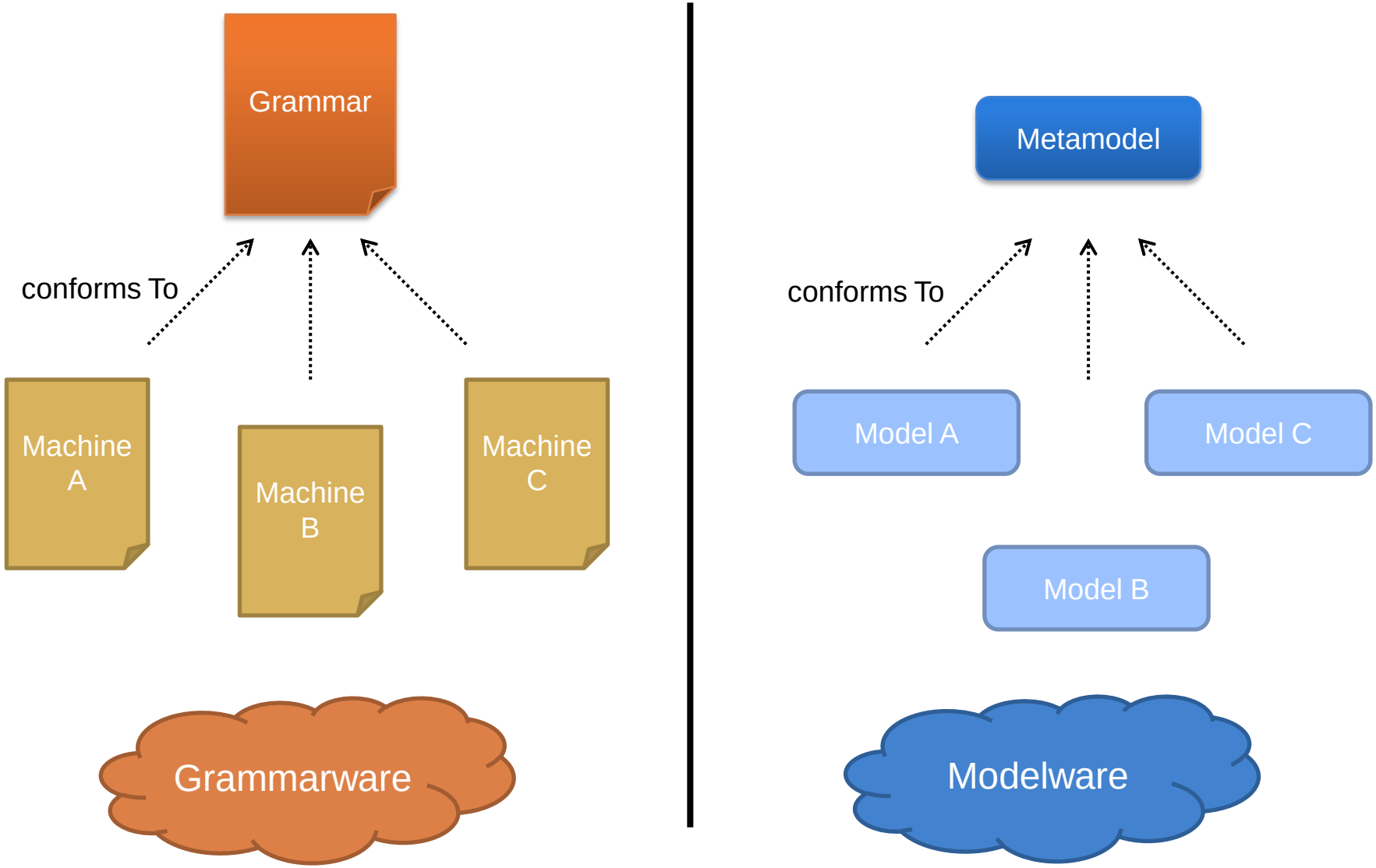
- Textual

```
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];
}
```

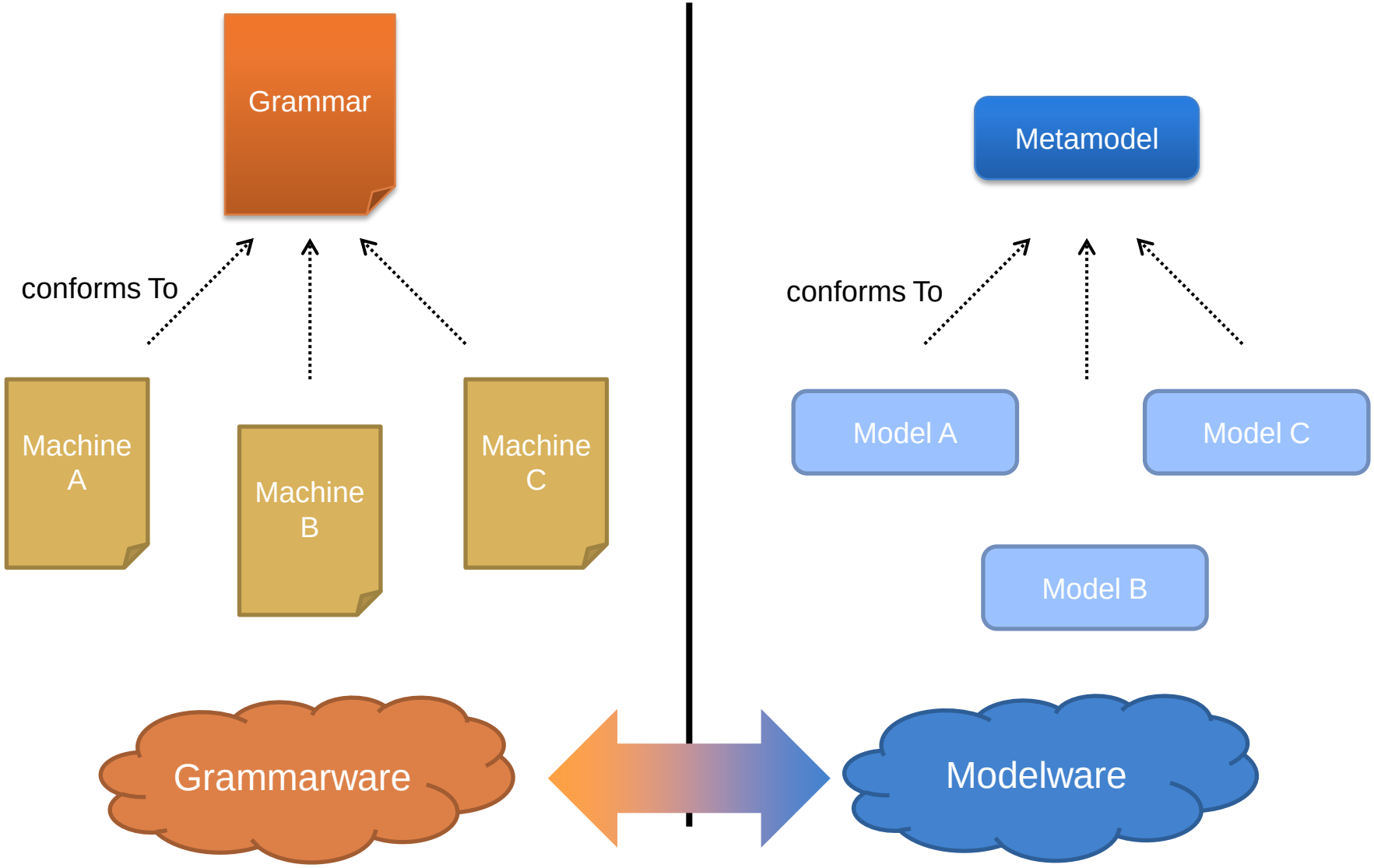
- Graphical



Textual Syntax



Textual Syntax



Textual Syntax

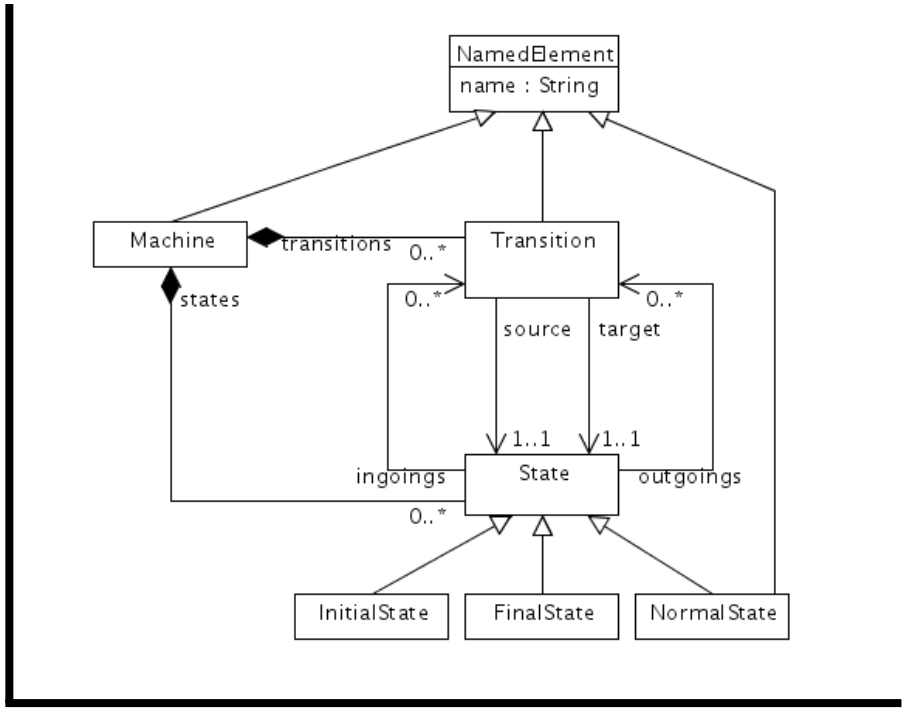
```
machineDefinition:
  MACHINE OPEN_SEP stateList
  transitionList CLOSE_SEP;

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)*;
```



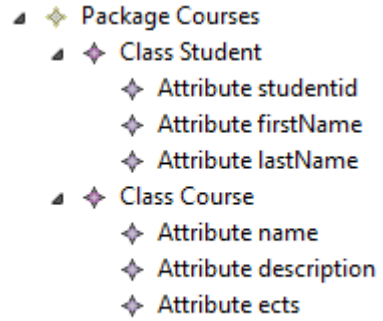
conforms To

```
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
  }
}
```

sintax of



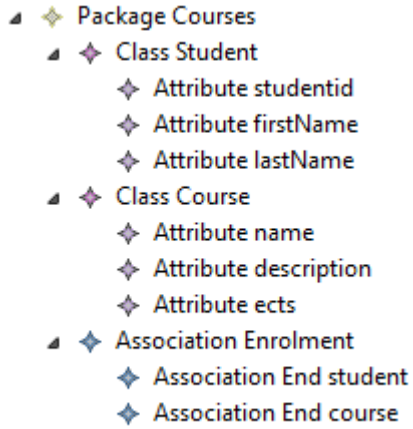
conforms To

Ecore MM

conforms To

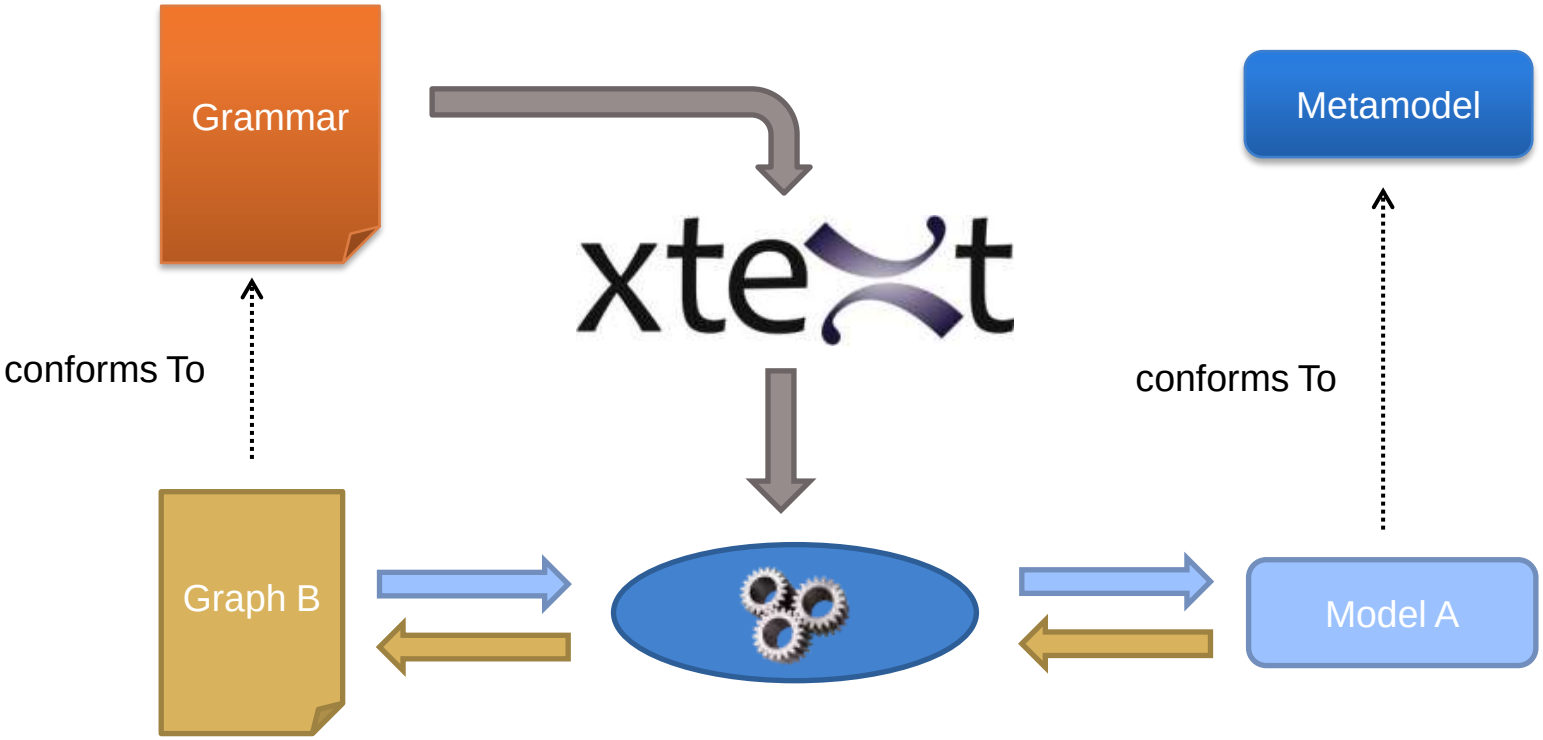
```
package Courses {
  class Student {
    attribute studentid : number key
    attribute firstName : string
    attribute lastName : string
  }

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

  association Enrolment {
    end student : Student [1..*]
    end course : Course [0..*]
  }
}
```

sintax of



Textual Syntax

- Grammar-oriented (Xtext)

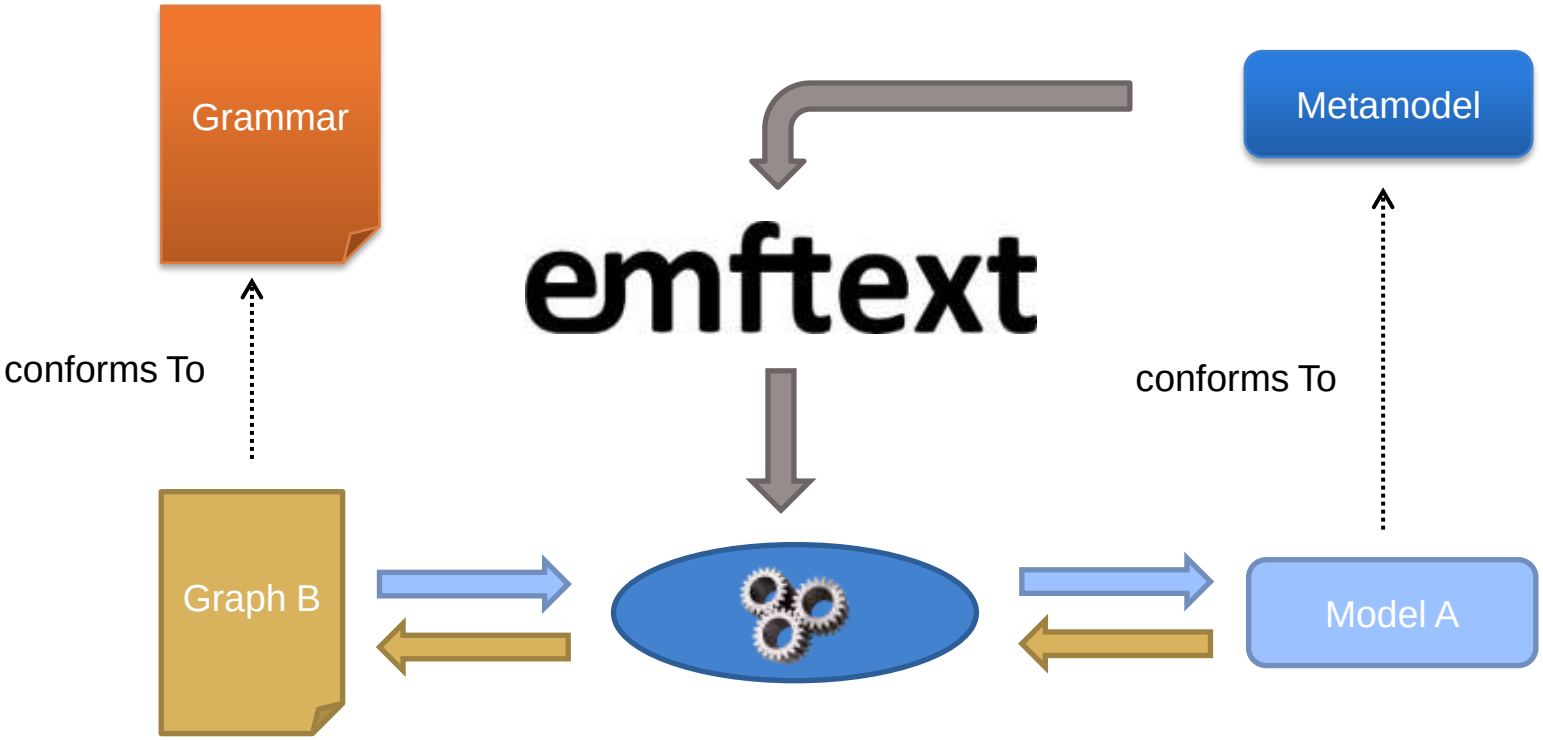


Xtext

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

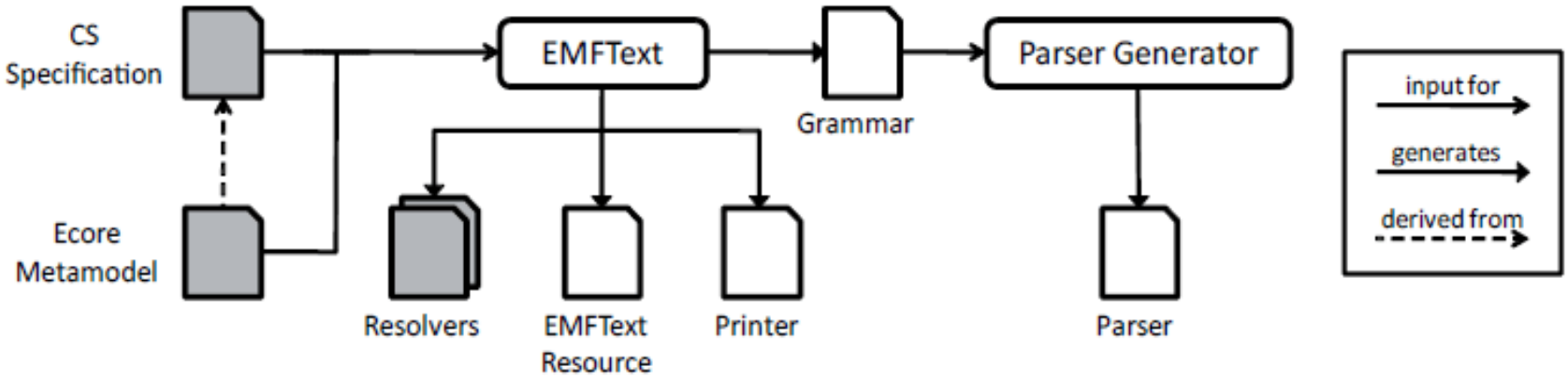
Textual Syntax

- Metamodel-oriented (EMFText)



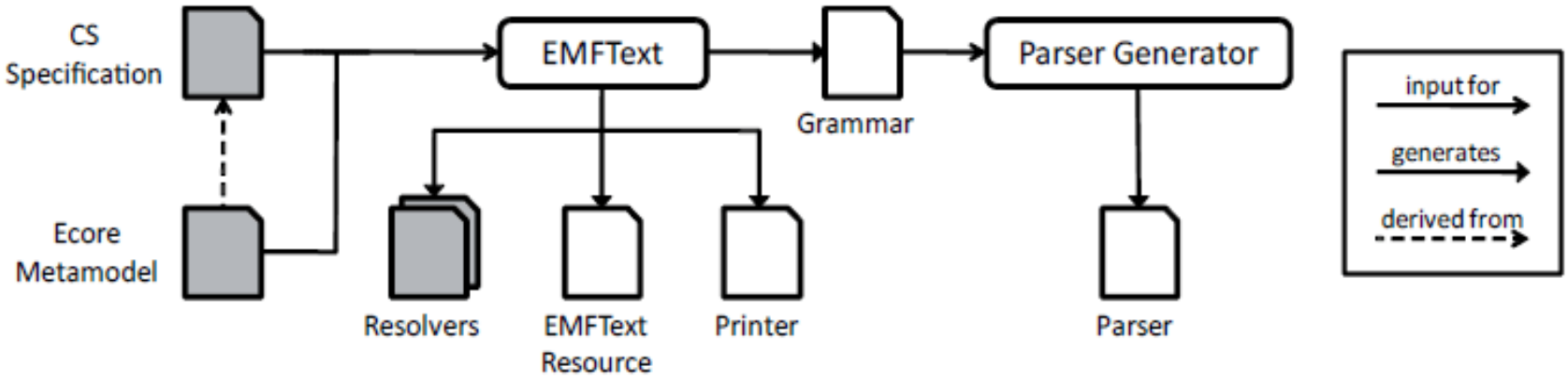
EMFText

- 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

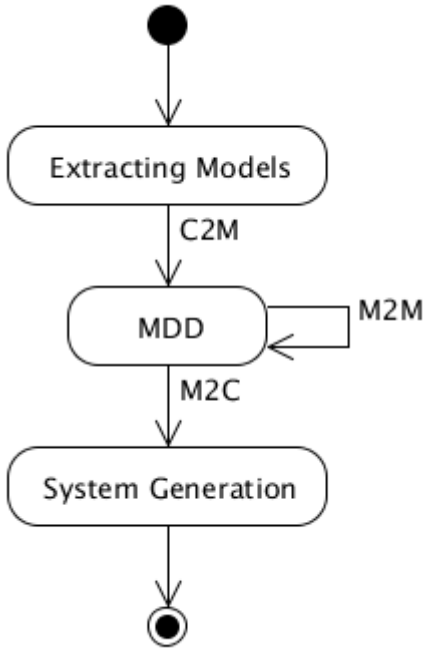
- Model-Driven framework to develop textual DSLs
- Developed as a Eclipse plugin
- Main features:
  - Syntax Coloring
  - Content Assist
  - Validation and Quick Fixes
  - Reference resolving mechanism



Graphical Syntax

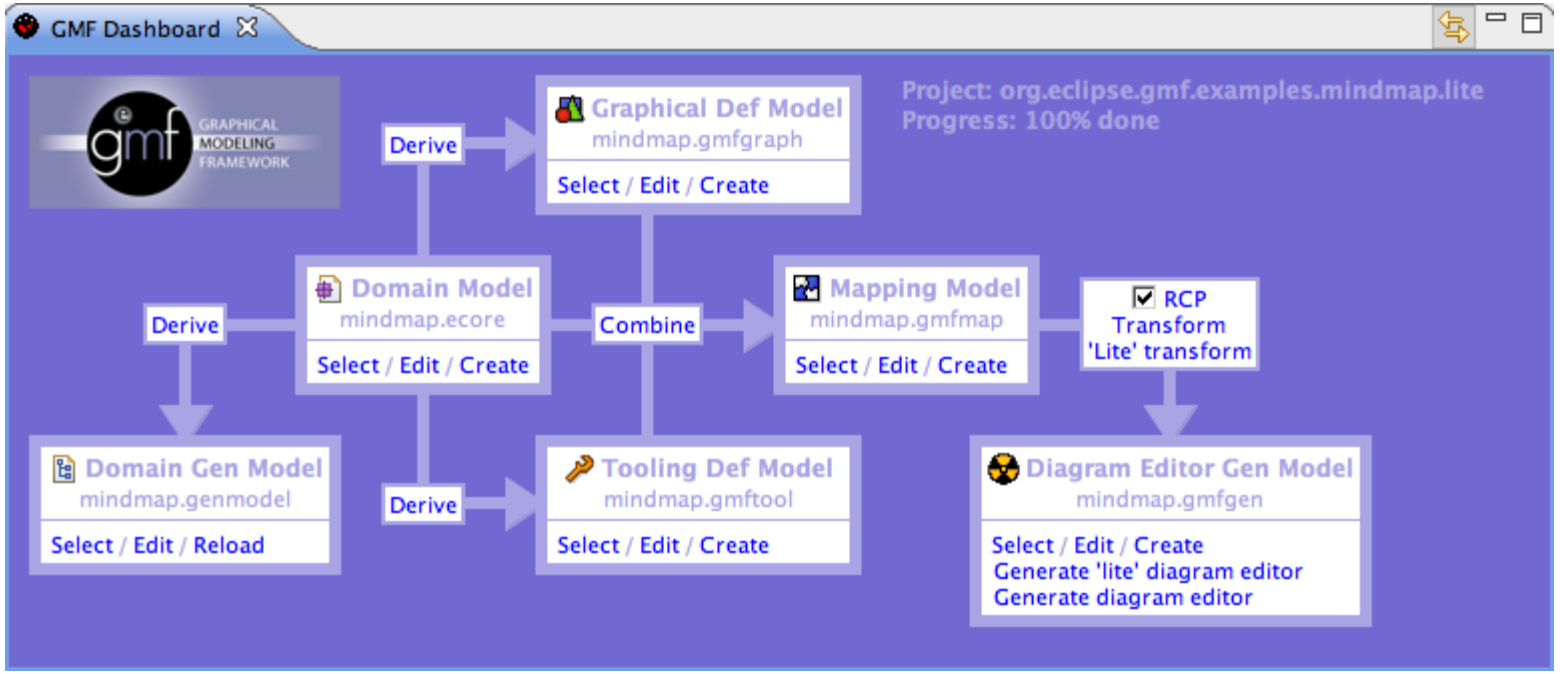
A draw is better than a long explanation

```
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

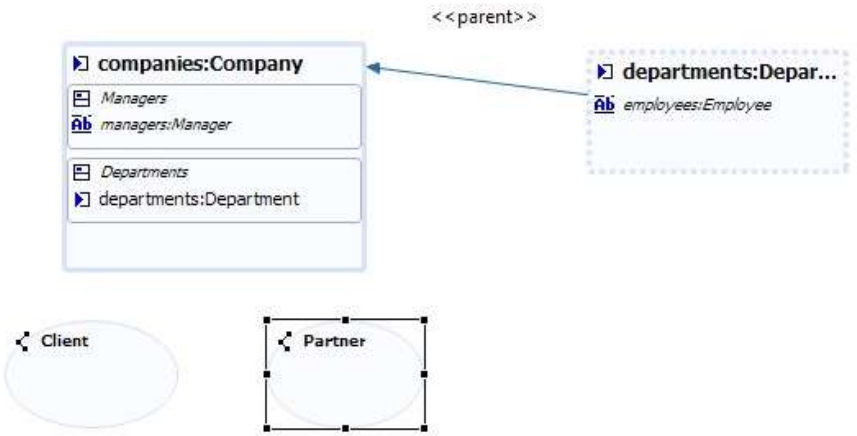
- Graphical Modeling Framework
  - Model-Driven Framework to develop graphical editors based on EMF and GEF
  - GMF is part of Eclipse Modeling Project
  - Provides a generative component to create the DSL tooling
  - Provides a runtime infrastructure to facilitate the development of graphical DSLs



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;
}
```

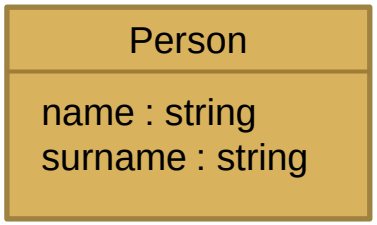


Graphical vs. Textual

- Success depends on how the notation fits the domain

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

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

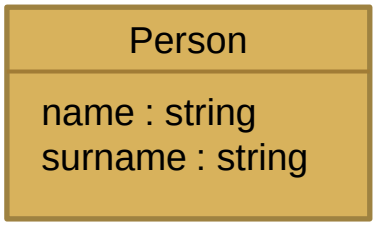


Graphical vs. Textual

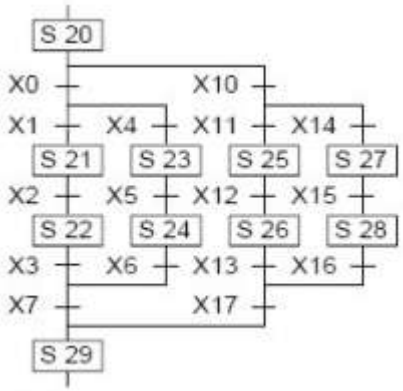
- Success depends on how the notation fits the domain

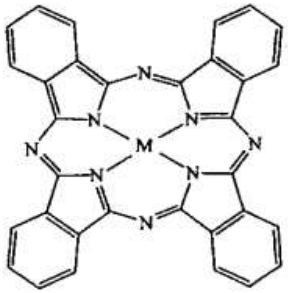
```
class Person {
  private String name;
  private String name;
}
```

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



- Graphical DSLs are not always easier to understand





phthalocyanine

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

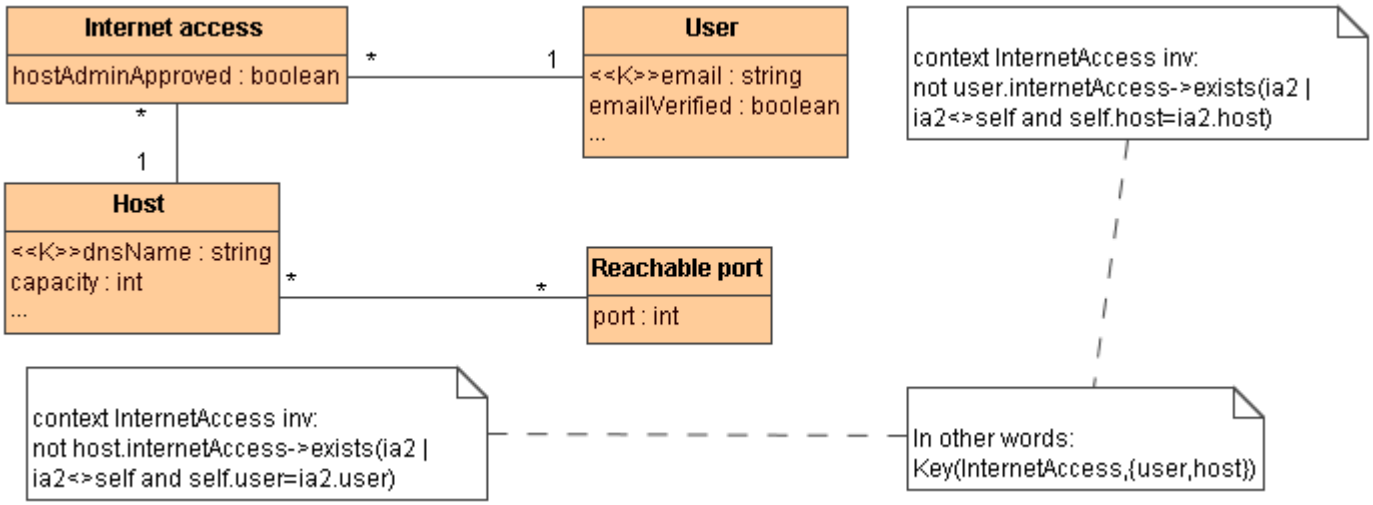
- Verbosity for less common concerns

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

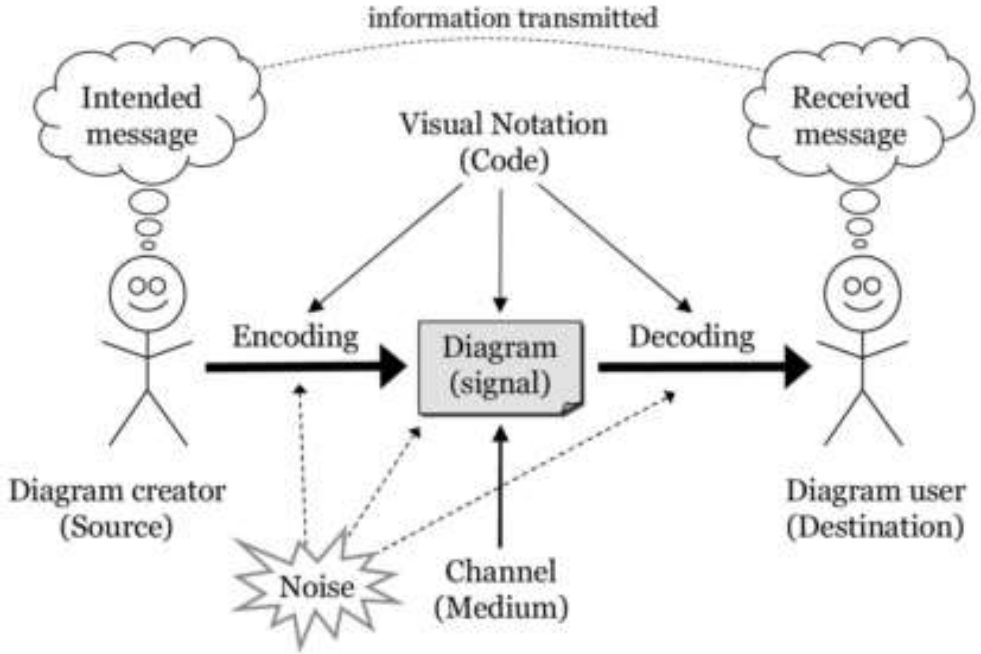
```
Person has (name, surname)
  extension
    name is String
    surname is String
```

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

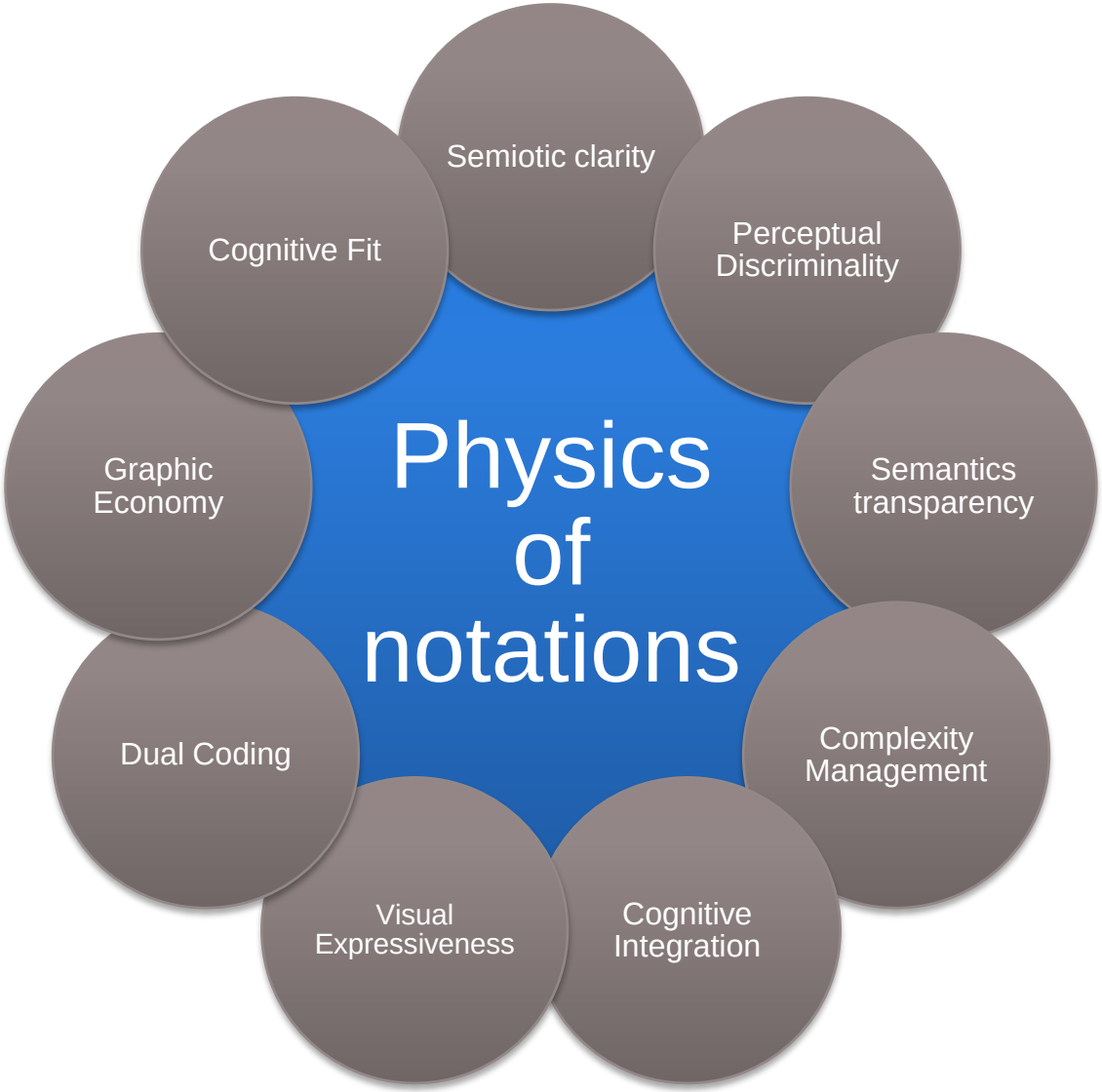
Graphical AND Textual



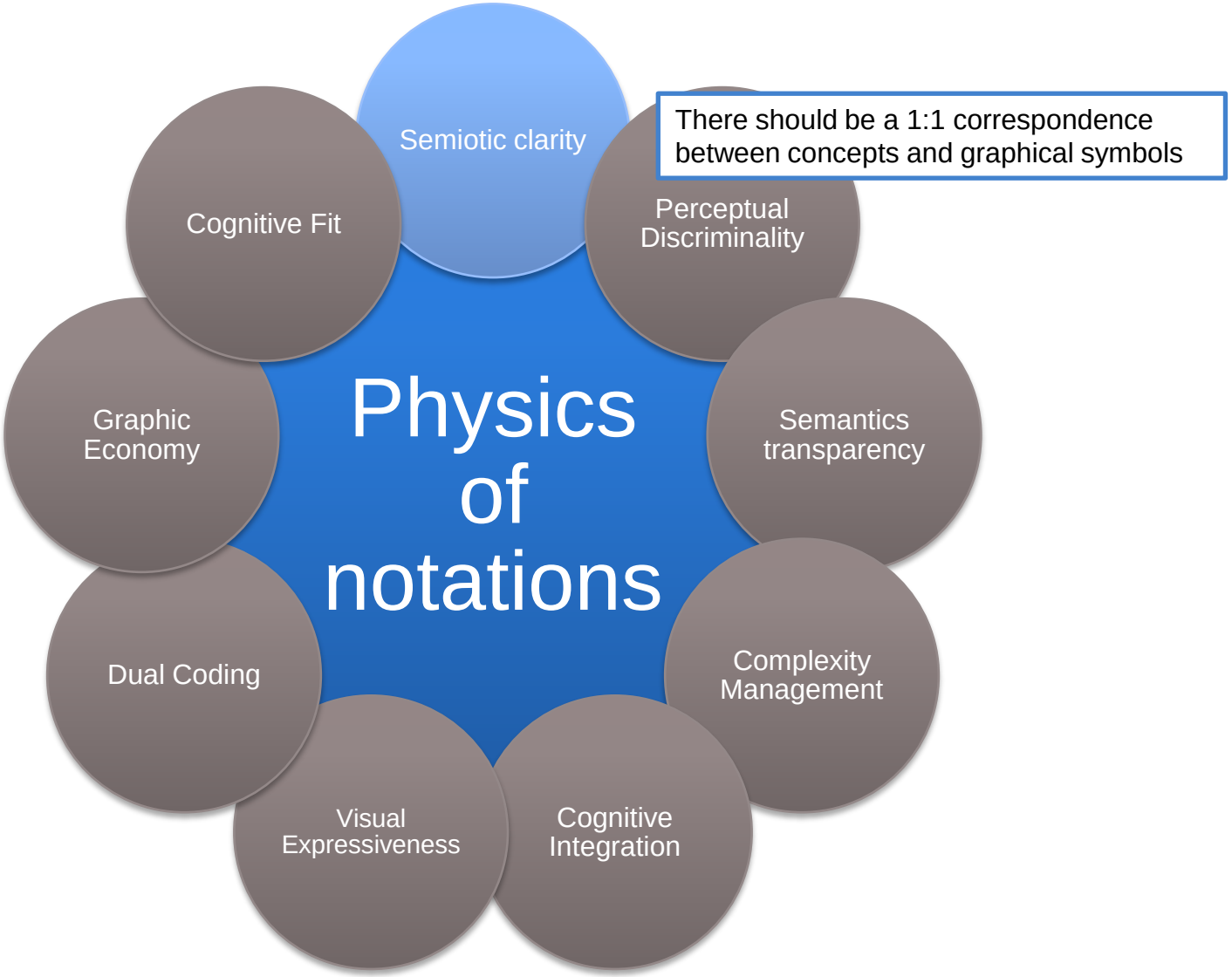
Recommendations



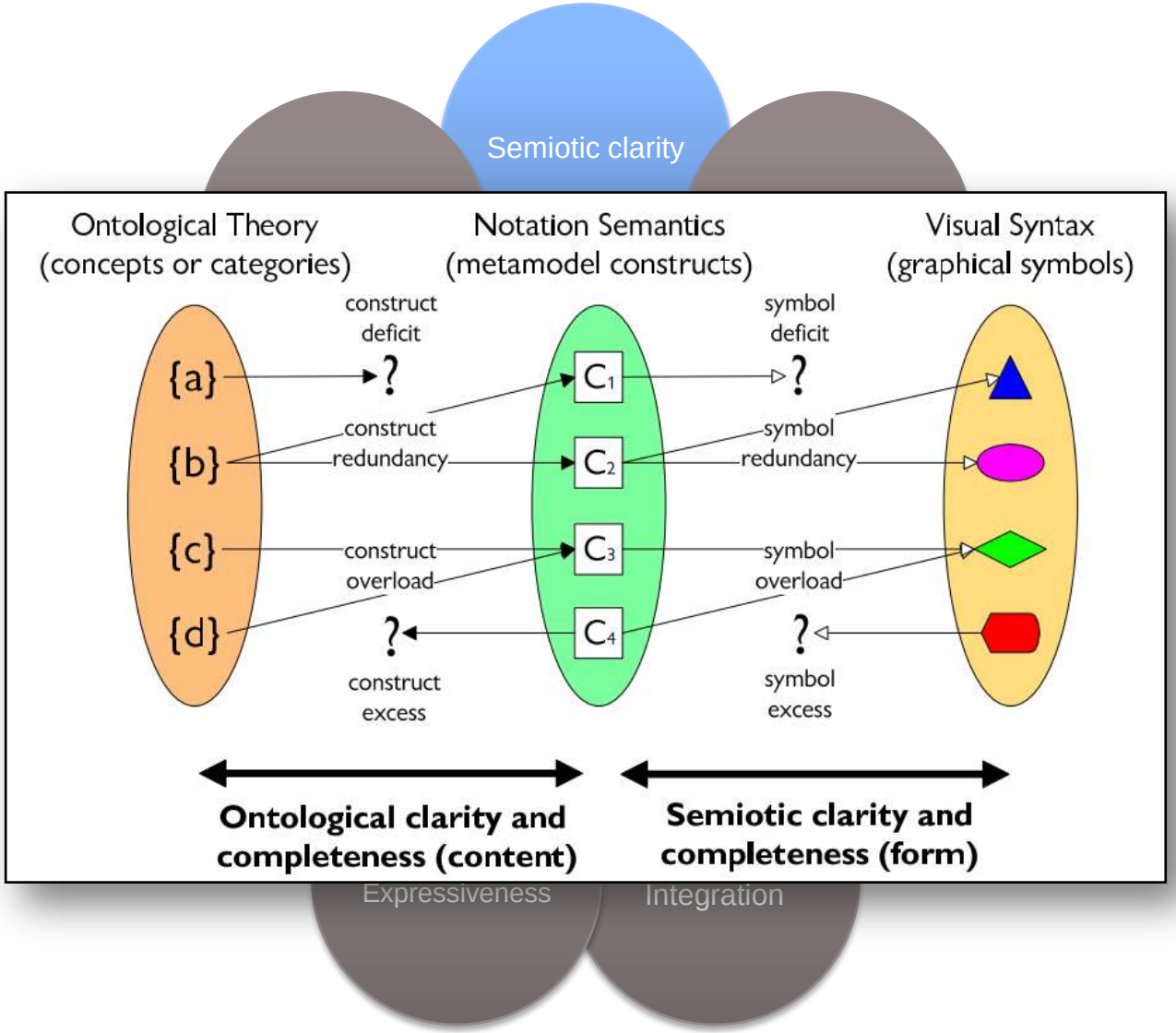
Recommendations



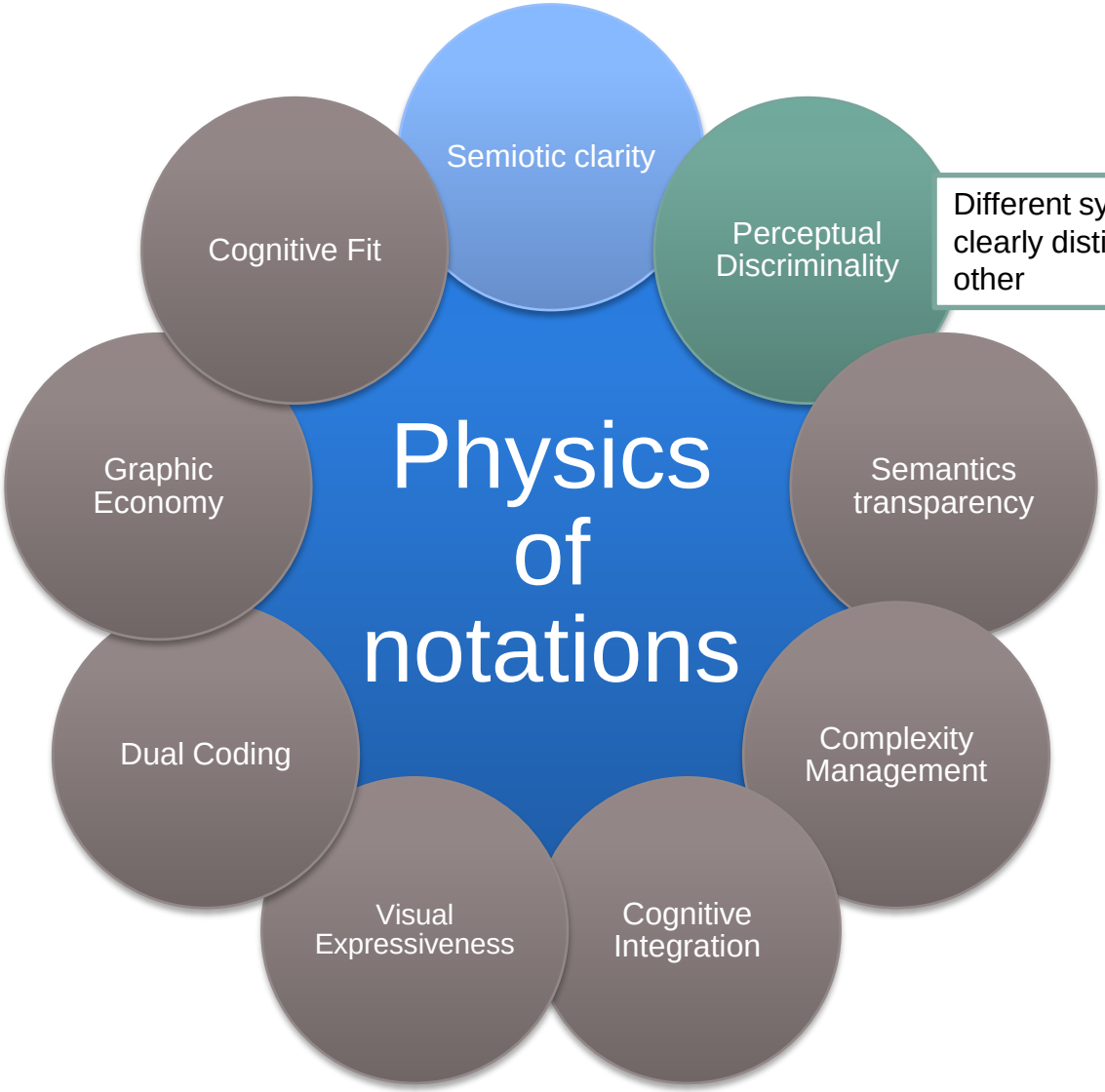
Recommendations



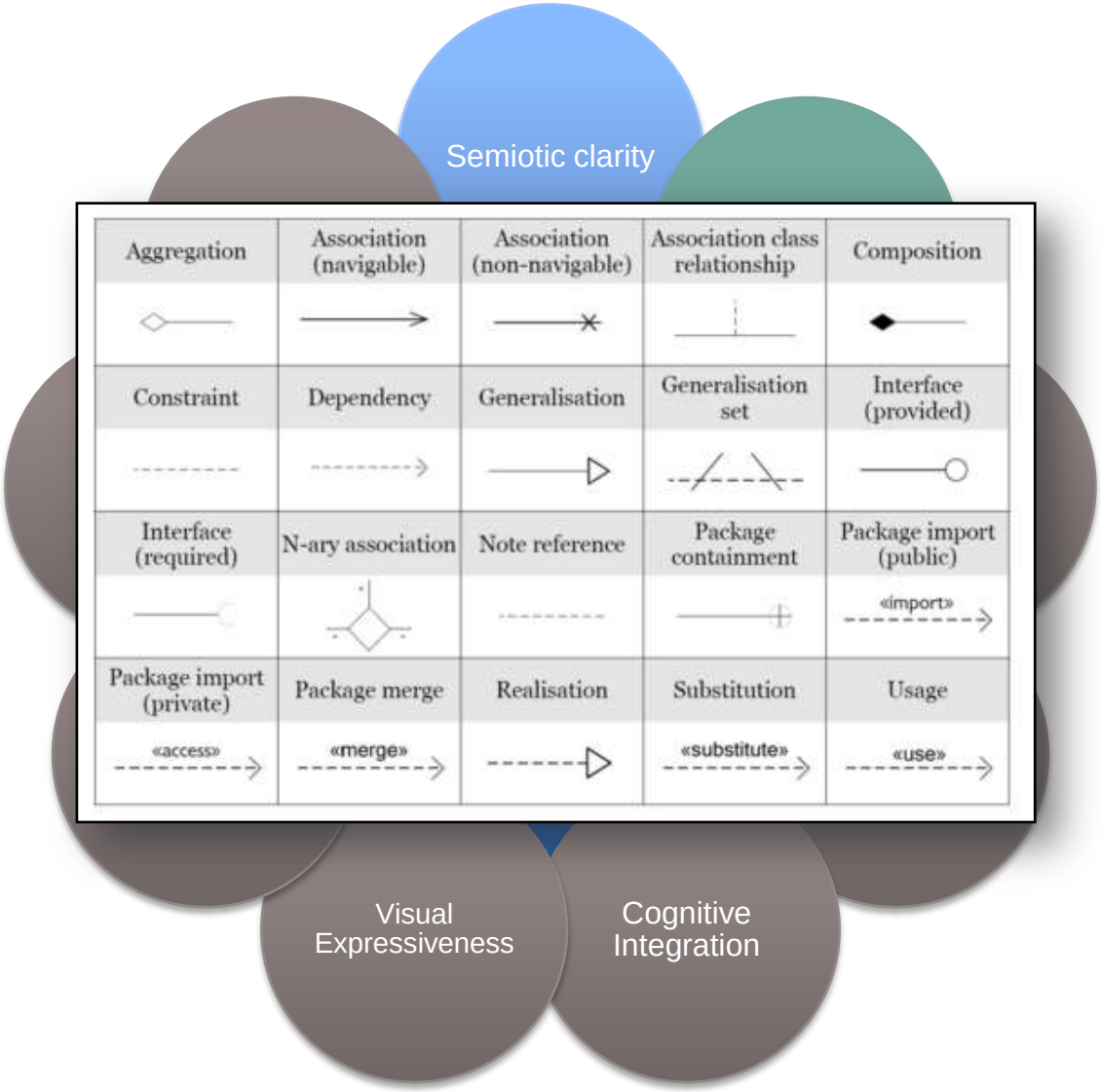
Recommendations



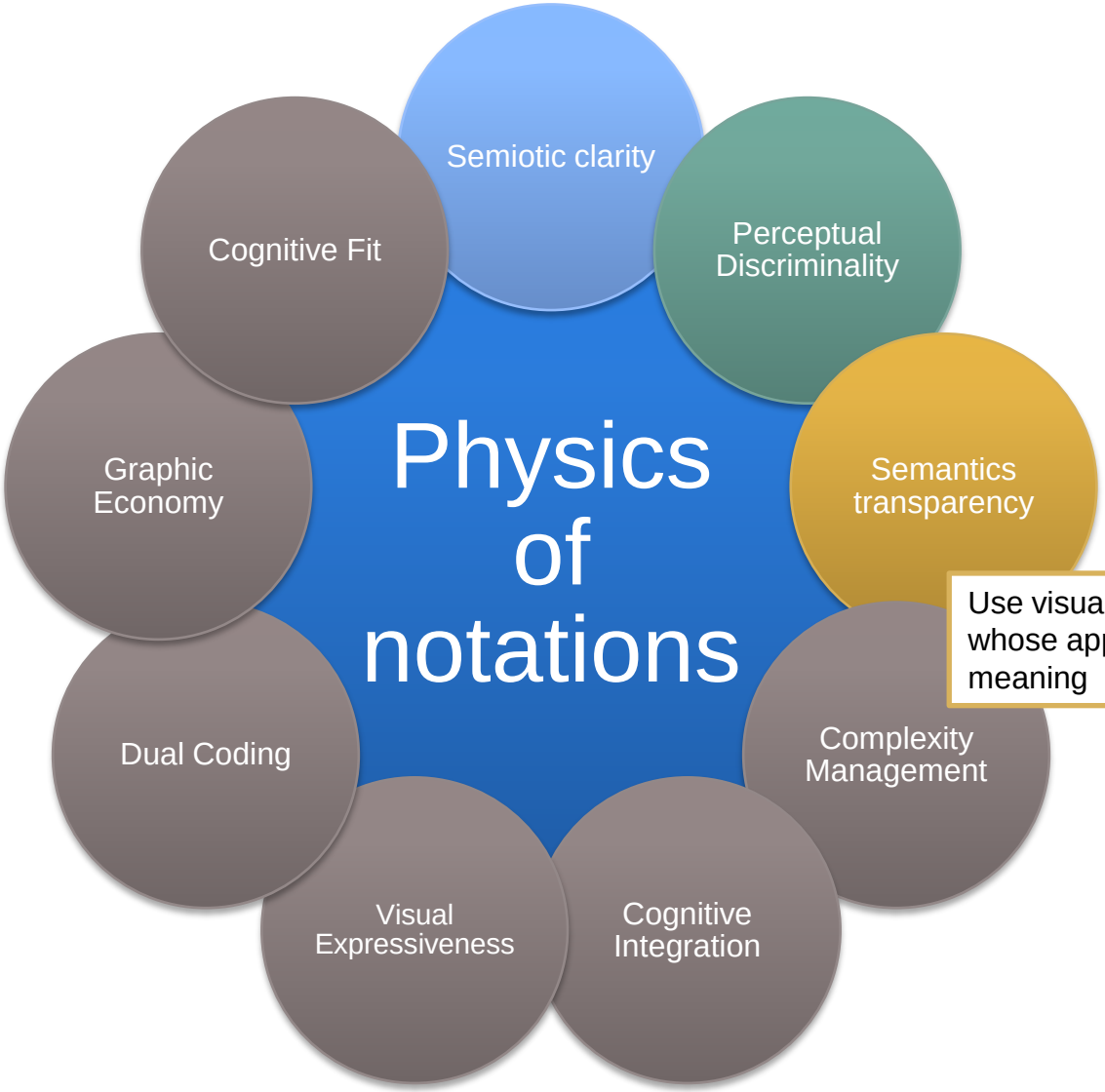
Recommendations



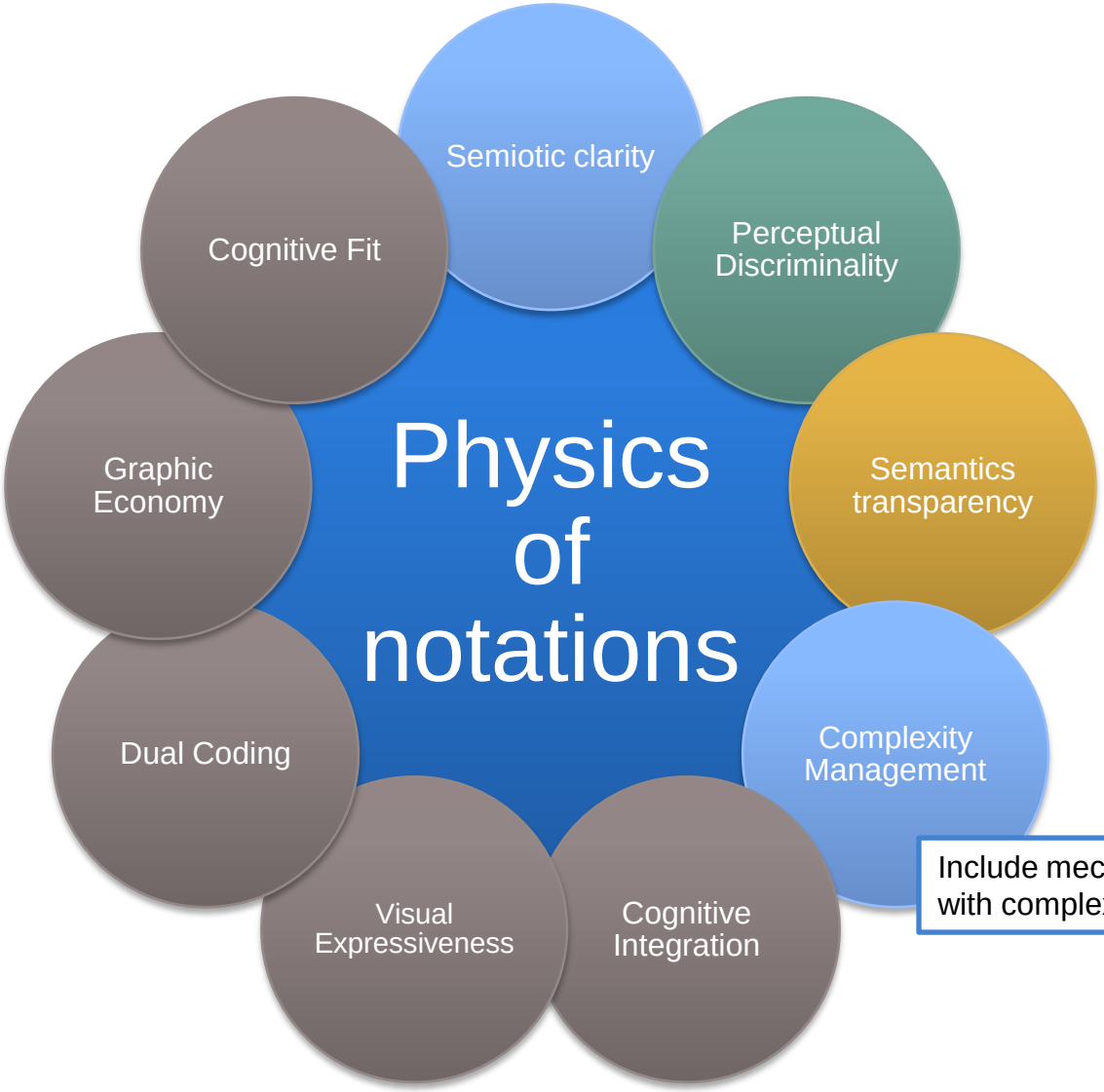
Recommendations



Recommendations

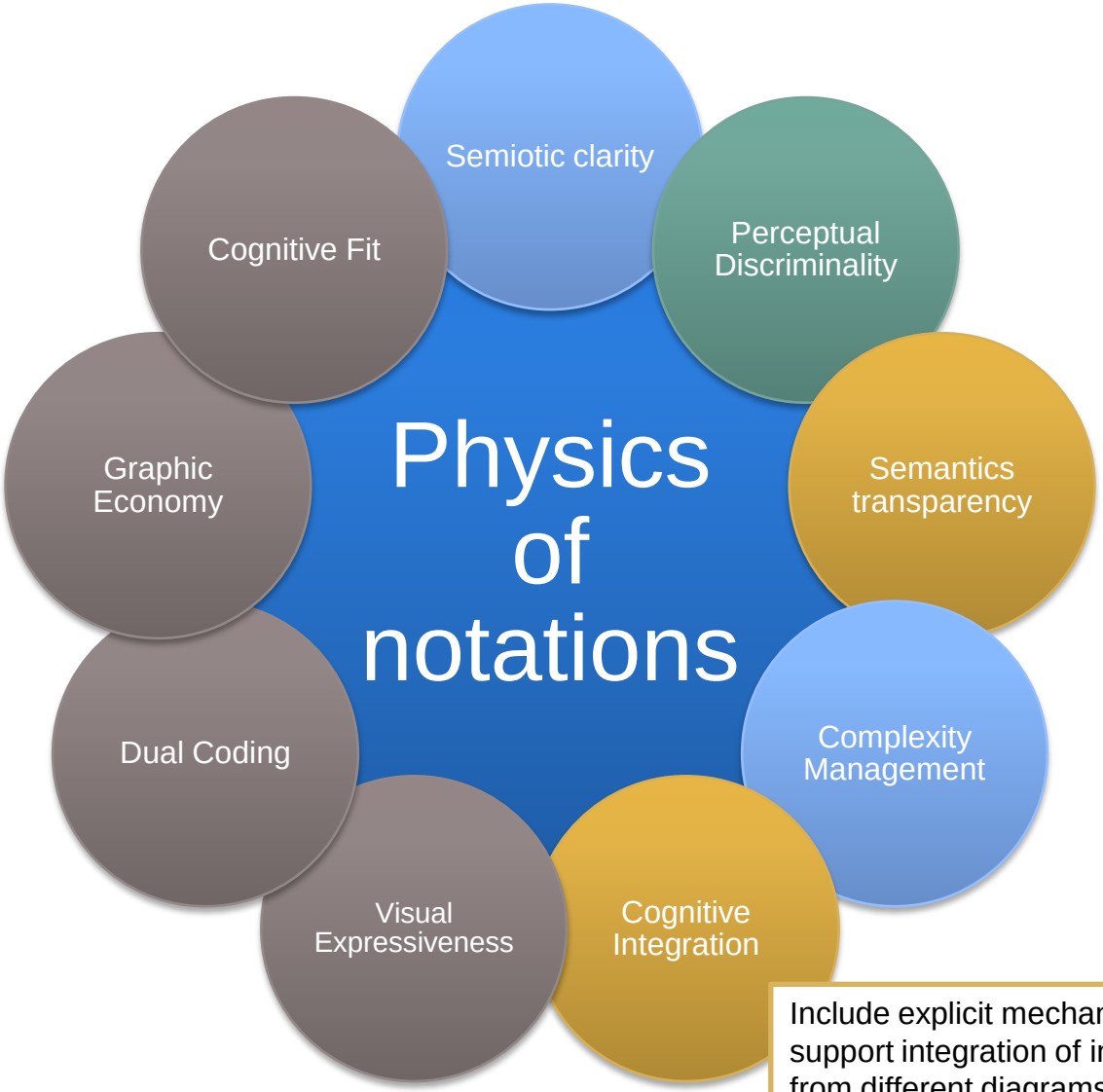


Recommendations



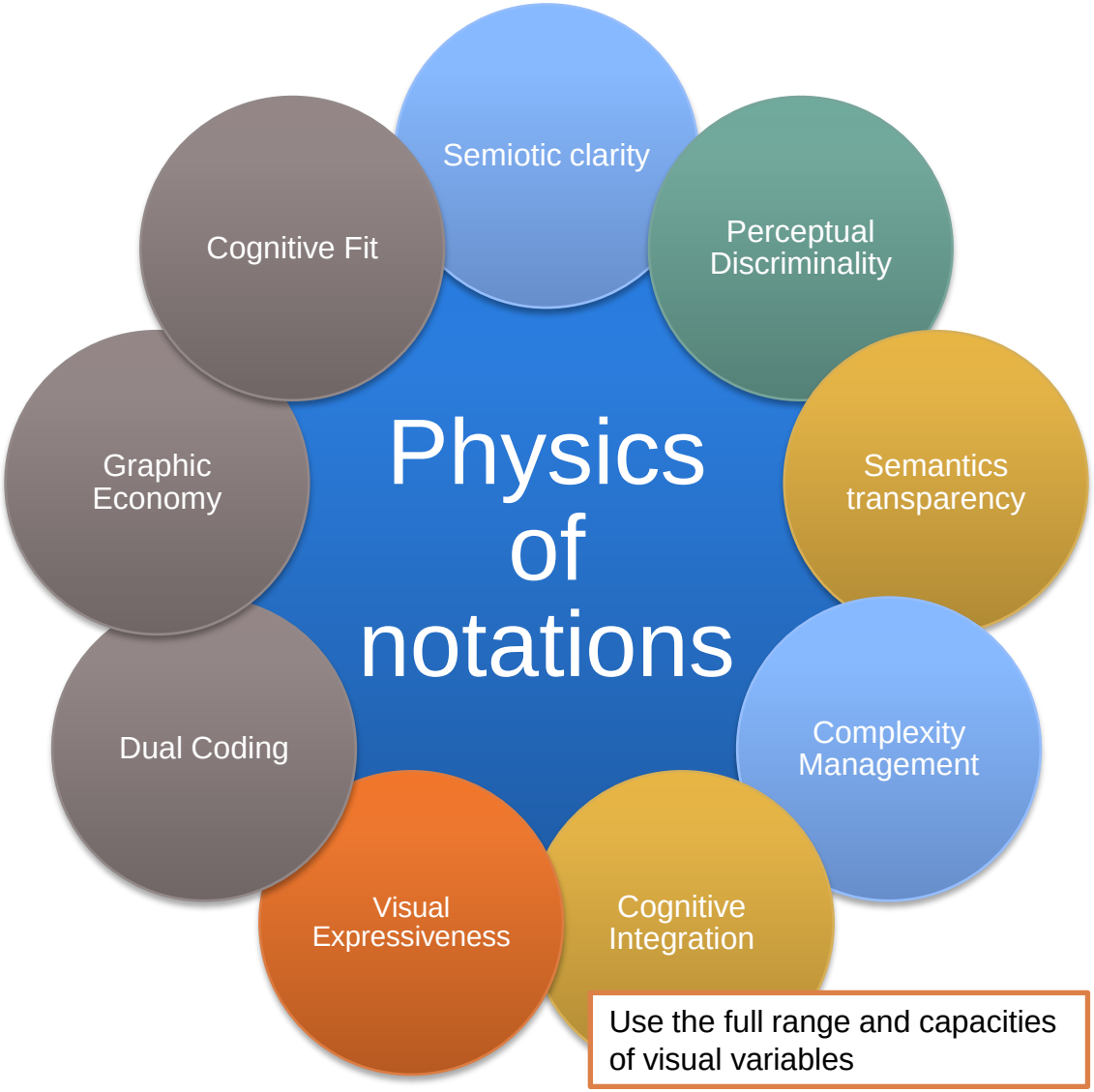
Include mechanisms for dealing with complexity

Recommendations



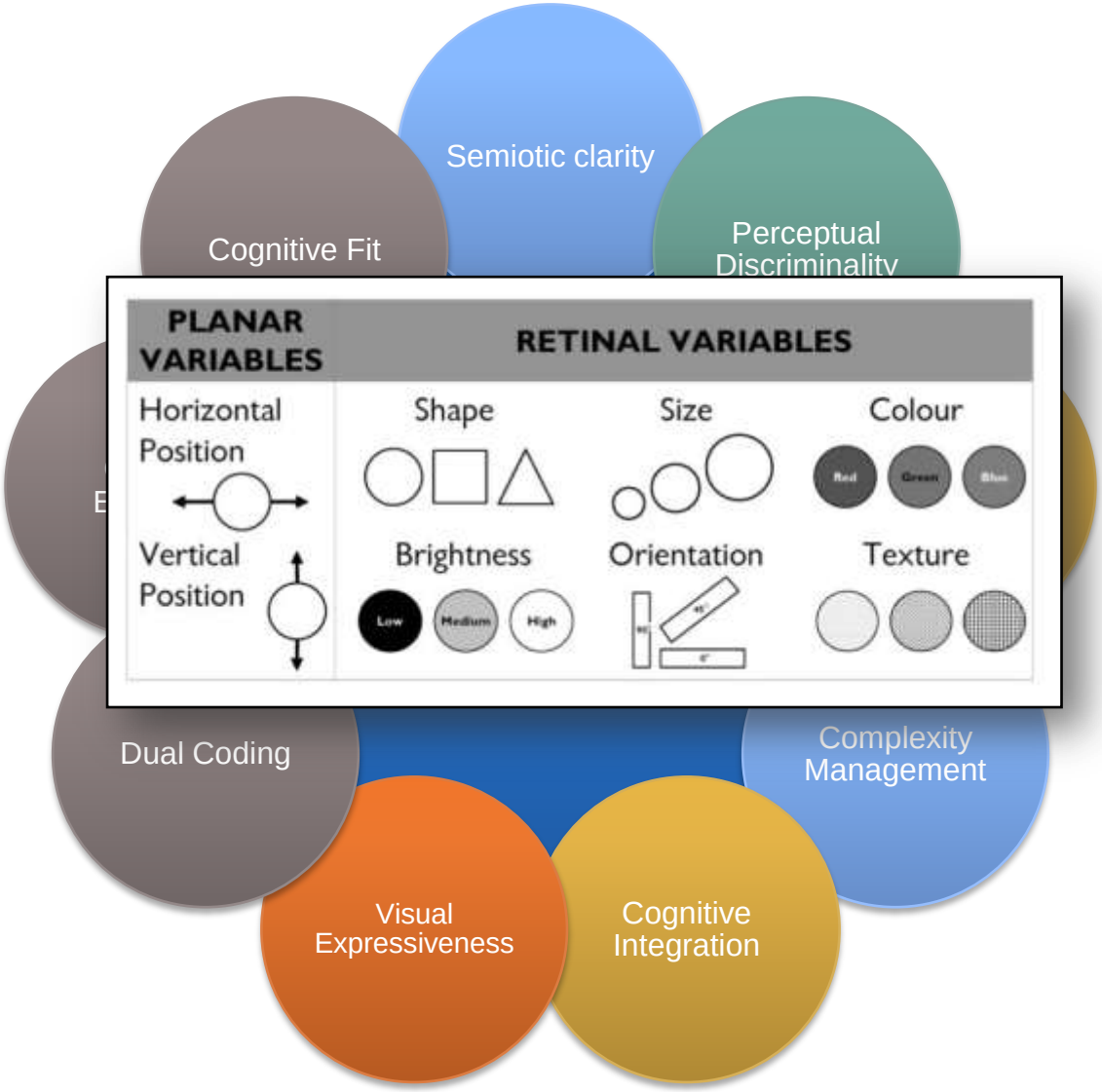
Include explicit mechanisms to support integration of information from different diagrams

Recommendations

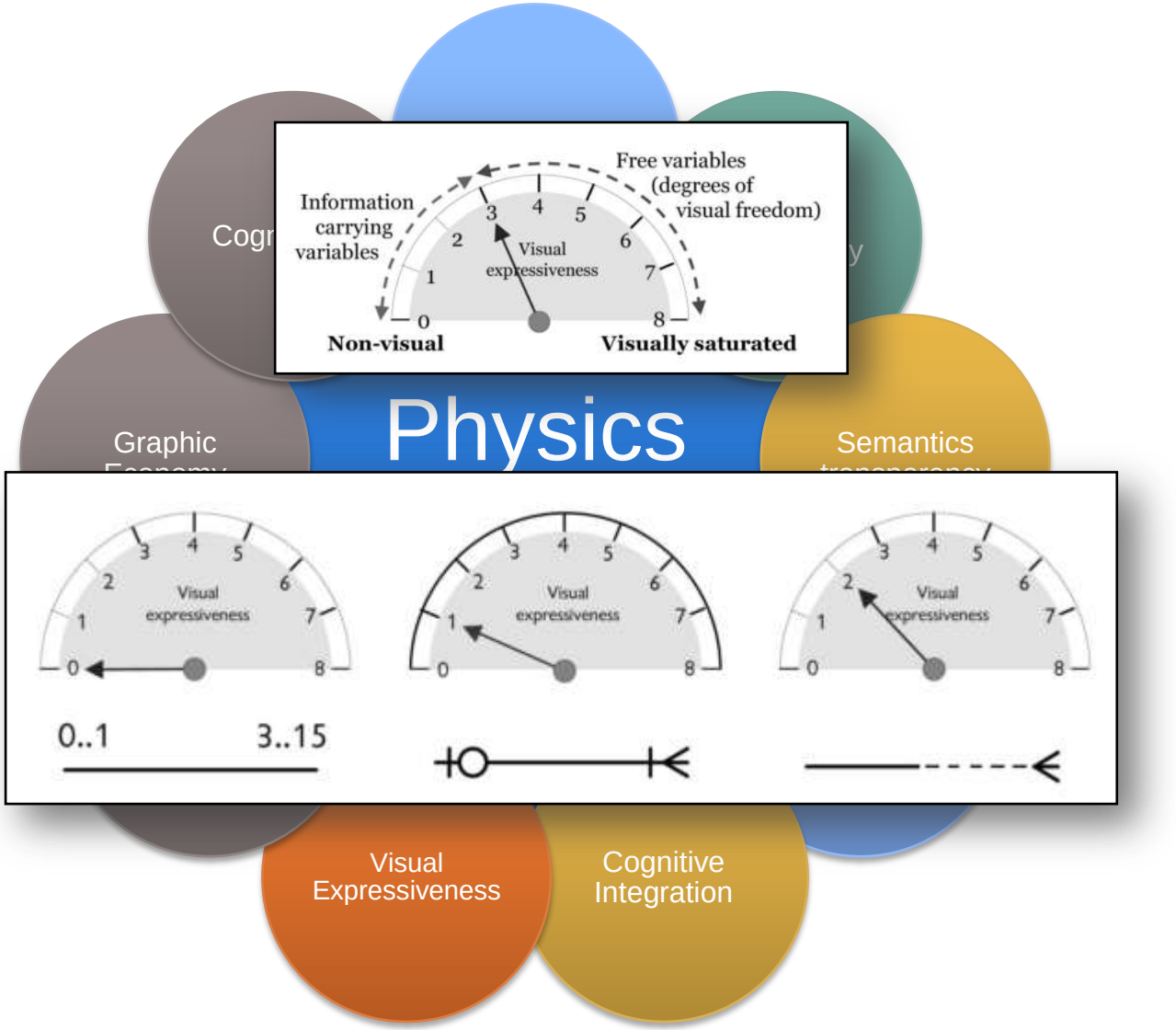


Use the full range and capacities of visual variables

Recommendations



Recommendations



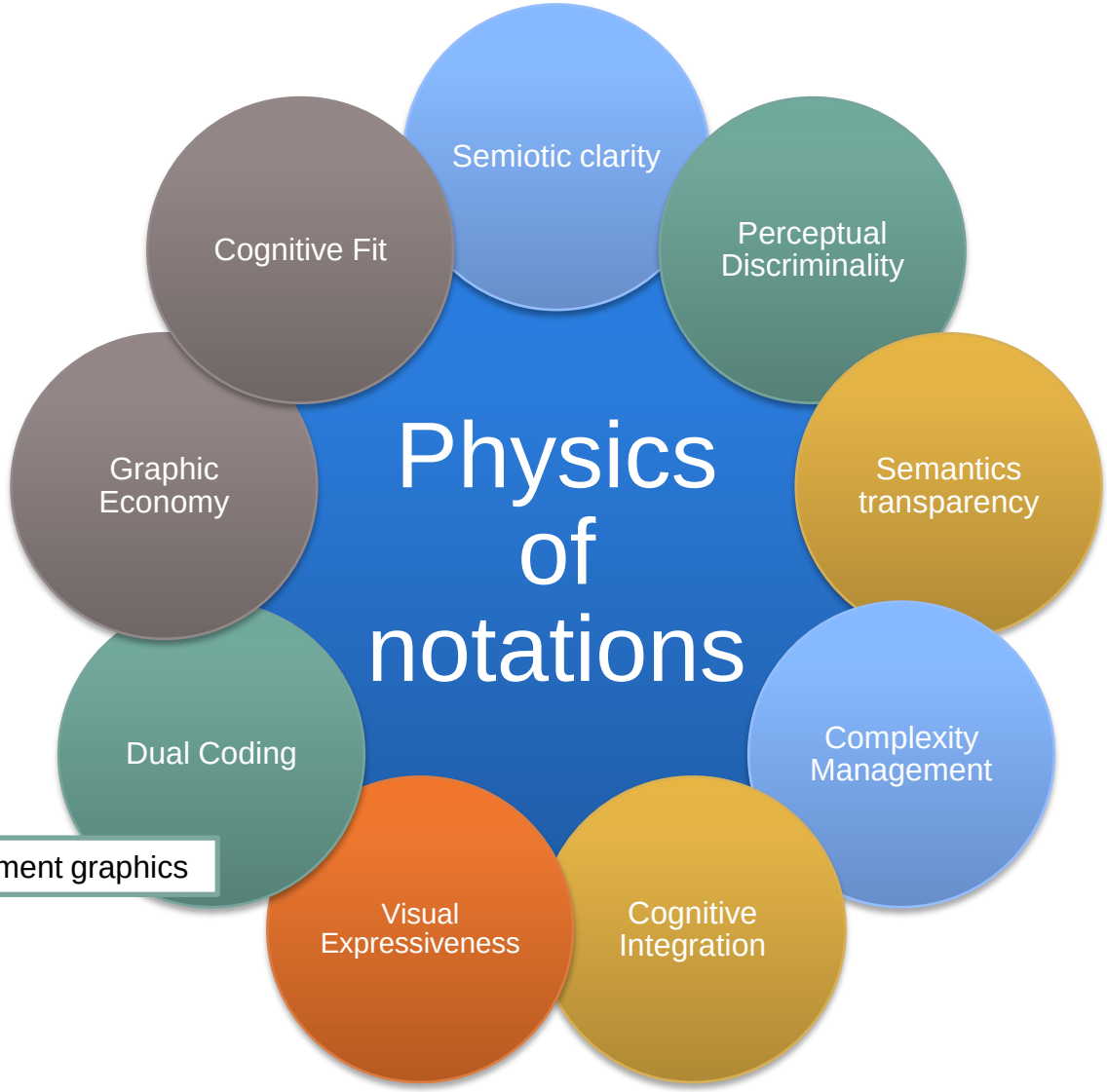
Recommendations

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

Visual Expressiveness

Cognitive Integration

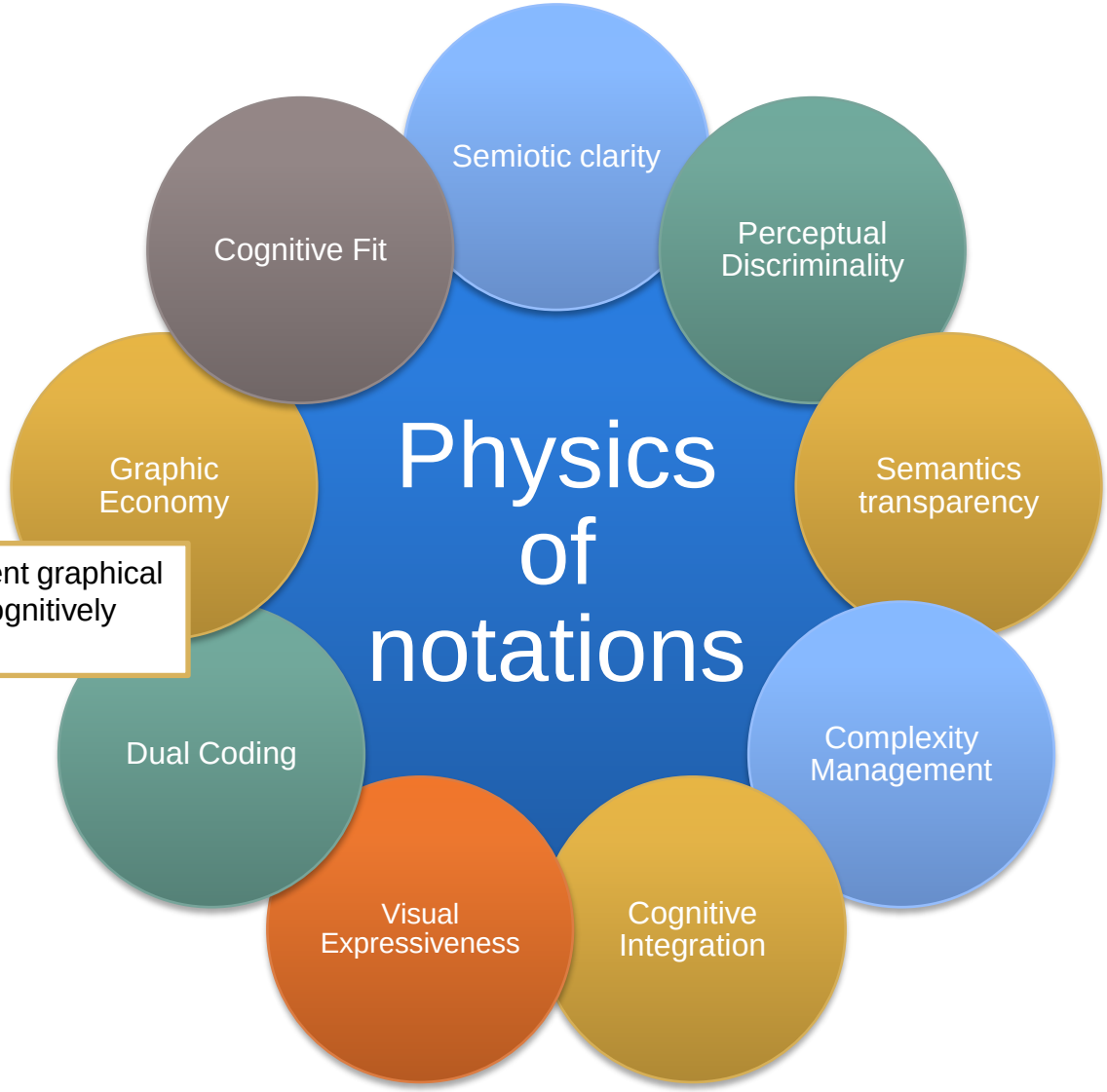
Recommendations



Use text to complement graphics

Recommendations

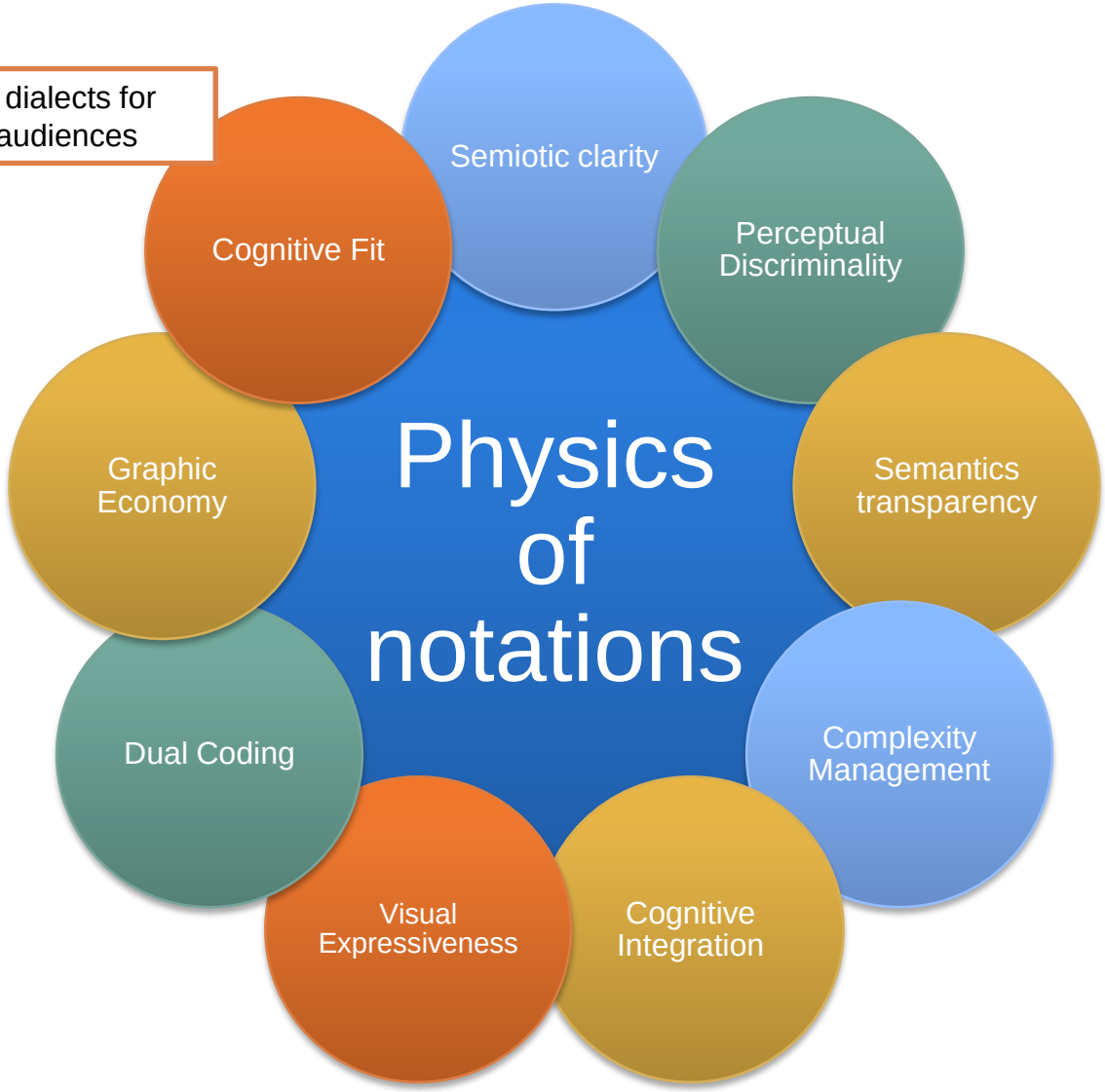
Recommendations



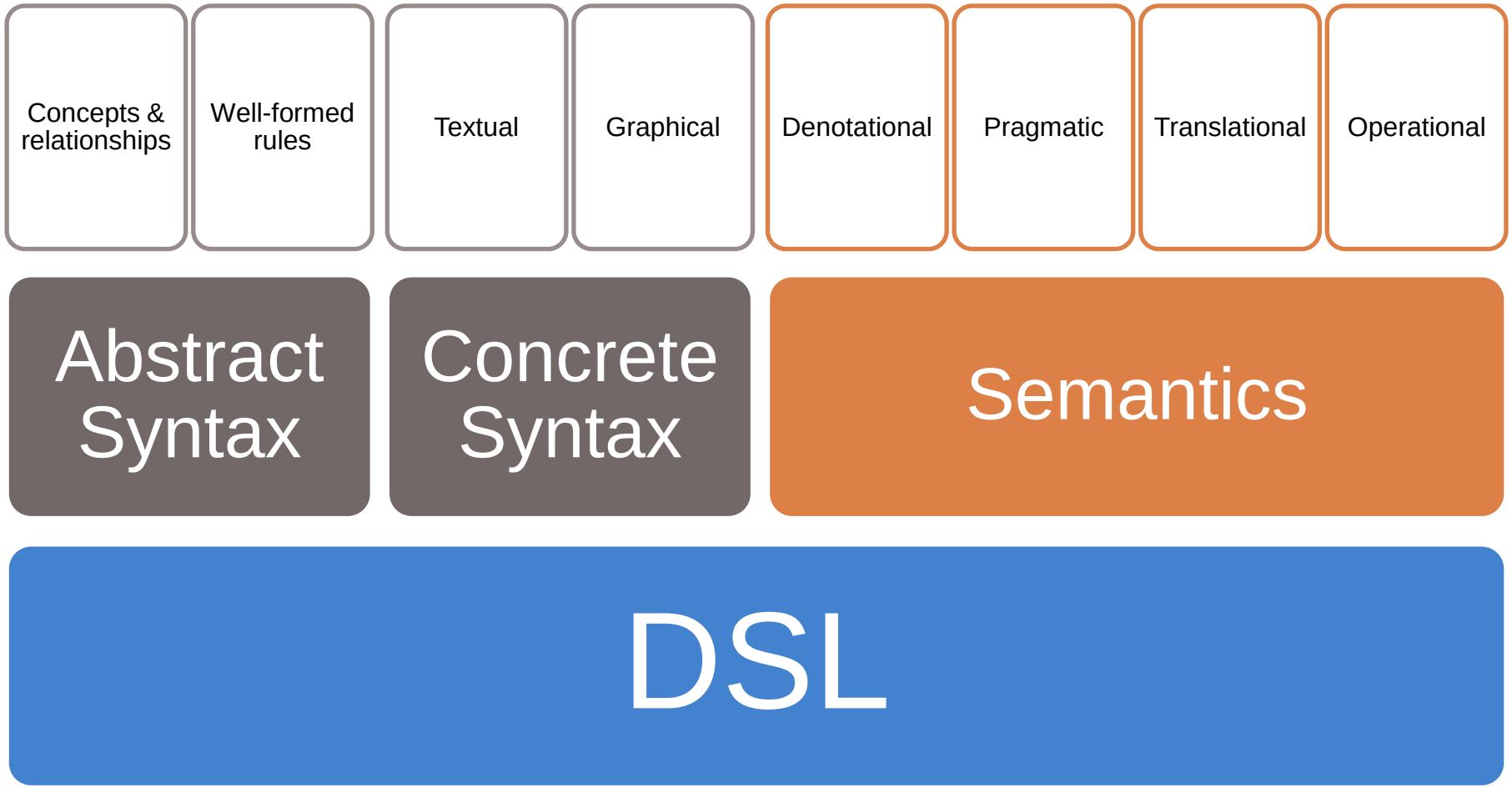
The number of different graphical symbols should be cognitively manageable

Recommendations

Use different visual dialects for different tasks and audiences



Anatomy


Concepts & relationships
Well-formed rules
Textual
Graphical
Denotational
Pragmatic
Translational
Operational
Abstract Syntax
Concrete Syntax
Semantics
DSL

Semantics

Denotational

- Use of mathematical formalisms
- Complex

Pragmatic

- Use of examples to illustrate the language
- Input / outputs
- Hard to deduce the real semantics

Traslational

- Translation to other well-known language
- Similar to compilation

Operational

- Sequence of operations
- Virtual machine

DSL

- The big picture
- What is a DSL? Why?
- Anatomy
- Development
- Actors
- Best practices
- Worst practices

Development process

Decision

- Create a new one or reuse an existing one
- Enable non-technical users to participate

Analysis

- Problem domain identified
- Domain knowledge gathered from end-users
- Domain model: Scope, vocabulary, concepts, commonalities and variabilities

Design

- Define the DSL components (abstract and concrete syntaxes, semantics)
- Decide best approach to implement the language

Implementation

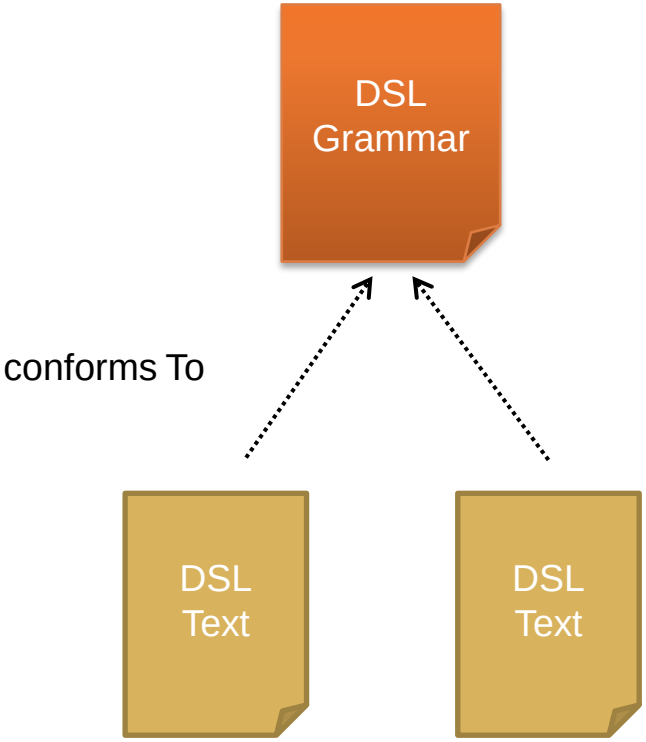
- Develop DSL components (Xtext, EMFText, GMF, etc…)
- If it is executable, develop the corresponding interpreter or compiler

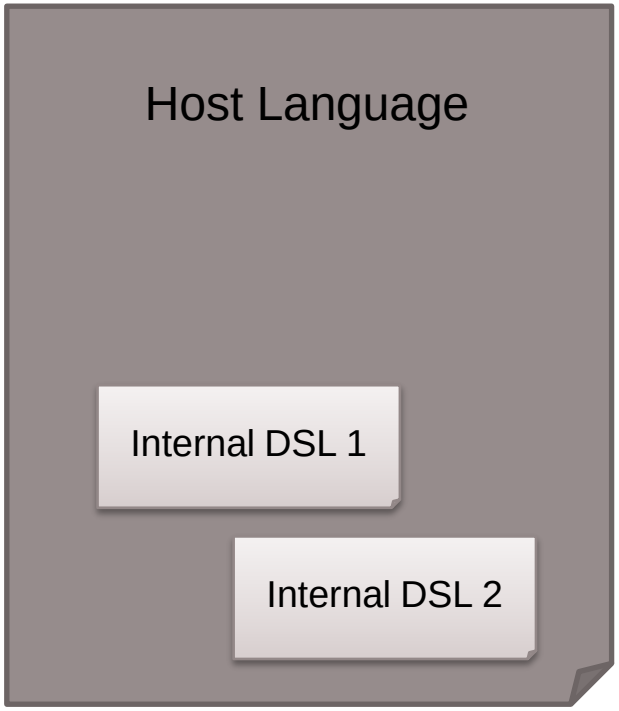
Validation

- Test the language
- Validate the language with the end-user

Regarding the implementation

External vs Internal

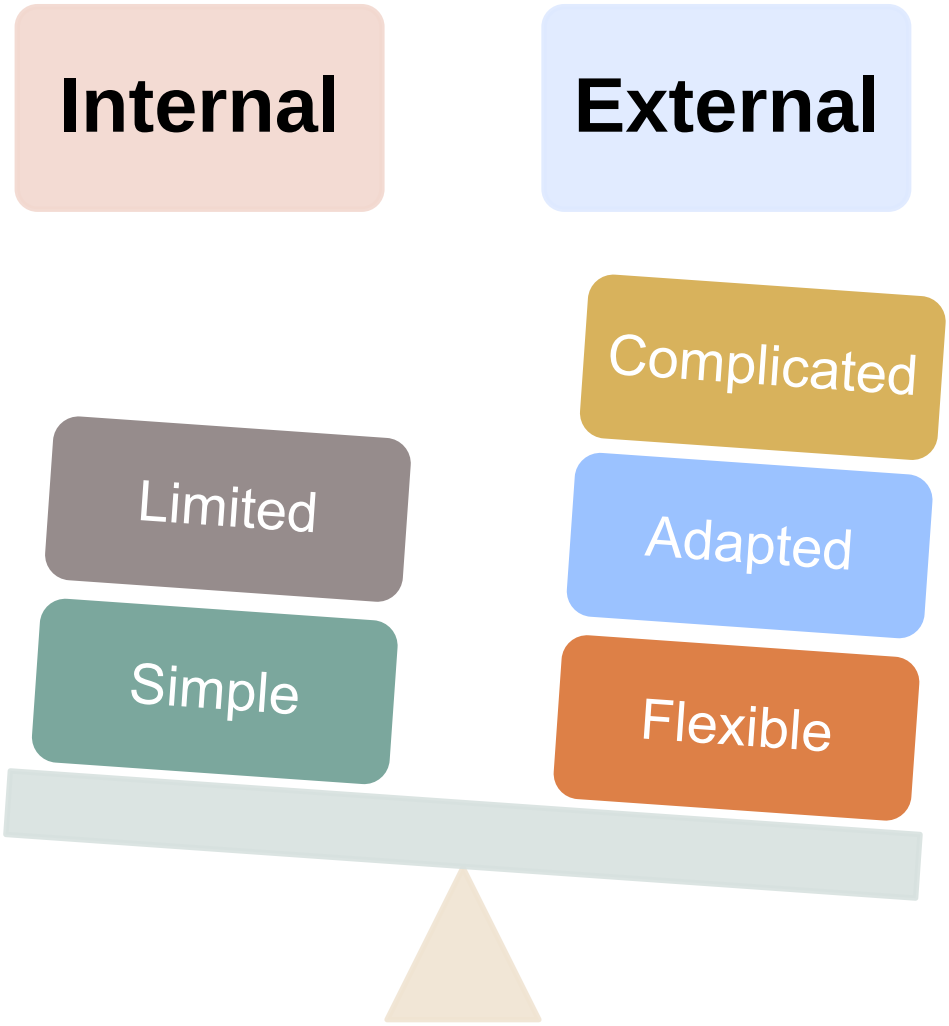




Internal DSL

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

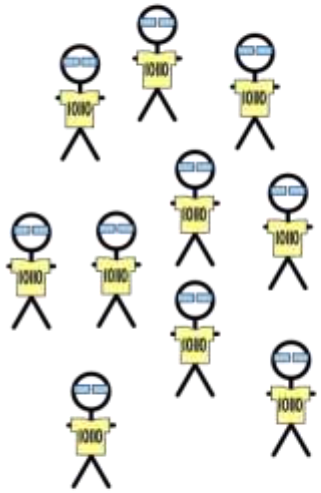
External vs Internal



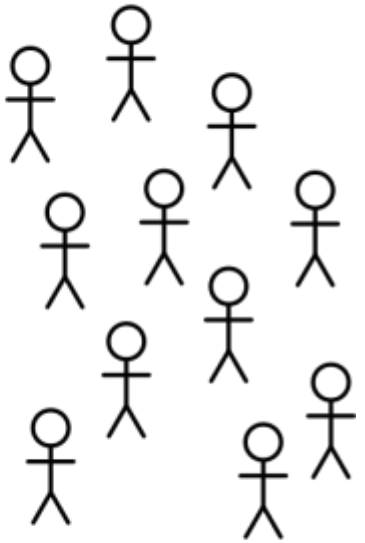
DSL

- The big picture
- What is a DSL? Why?
- Anatomy
- Development
- Actors
- Best practices
- Worst practices

Actors

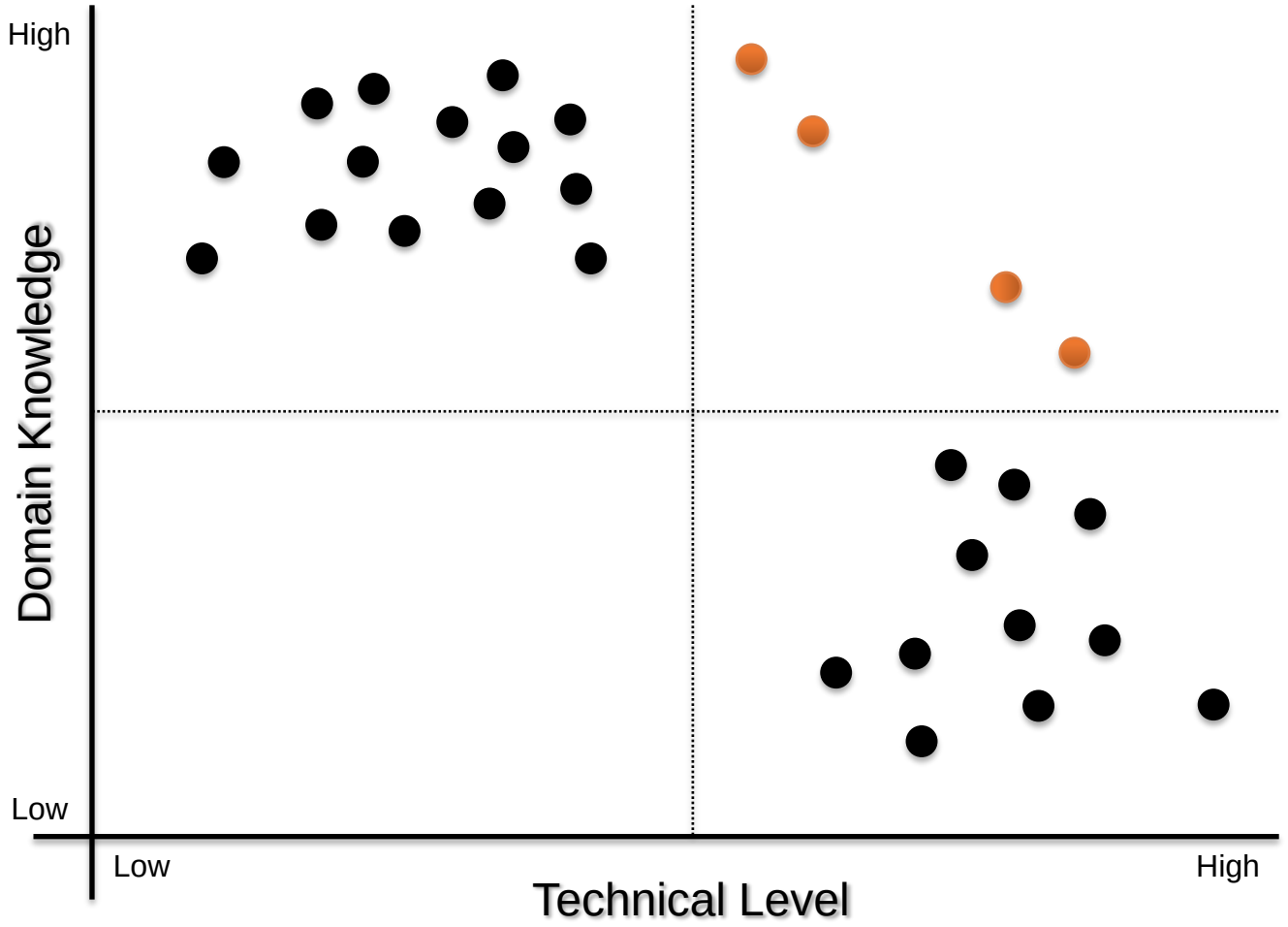


Developers

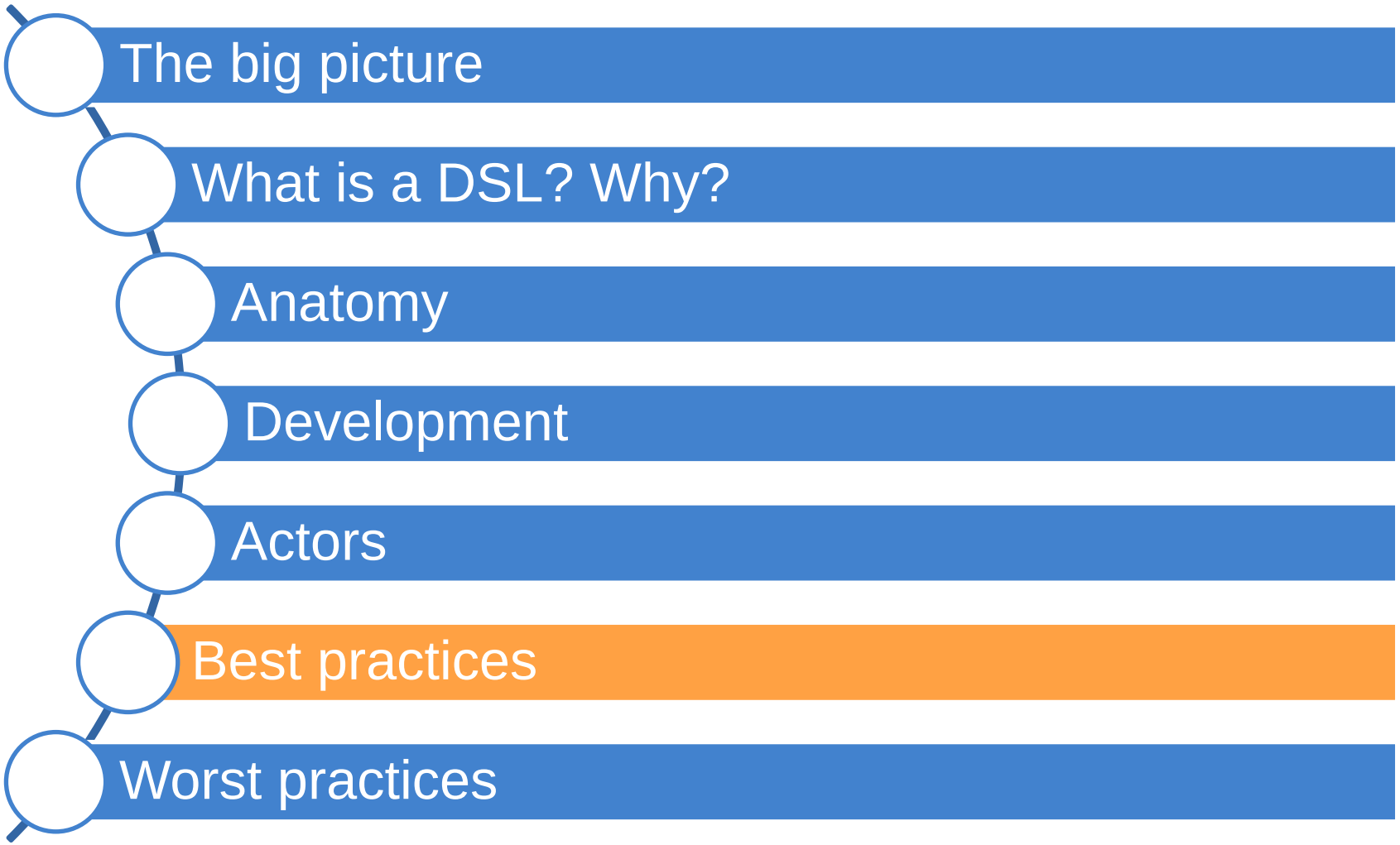


End-Users

Actors



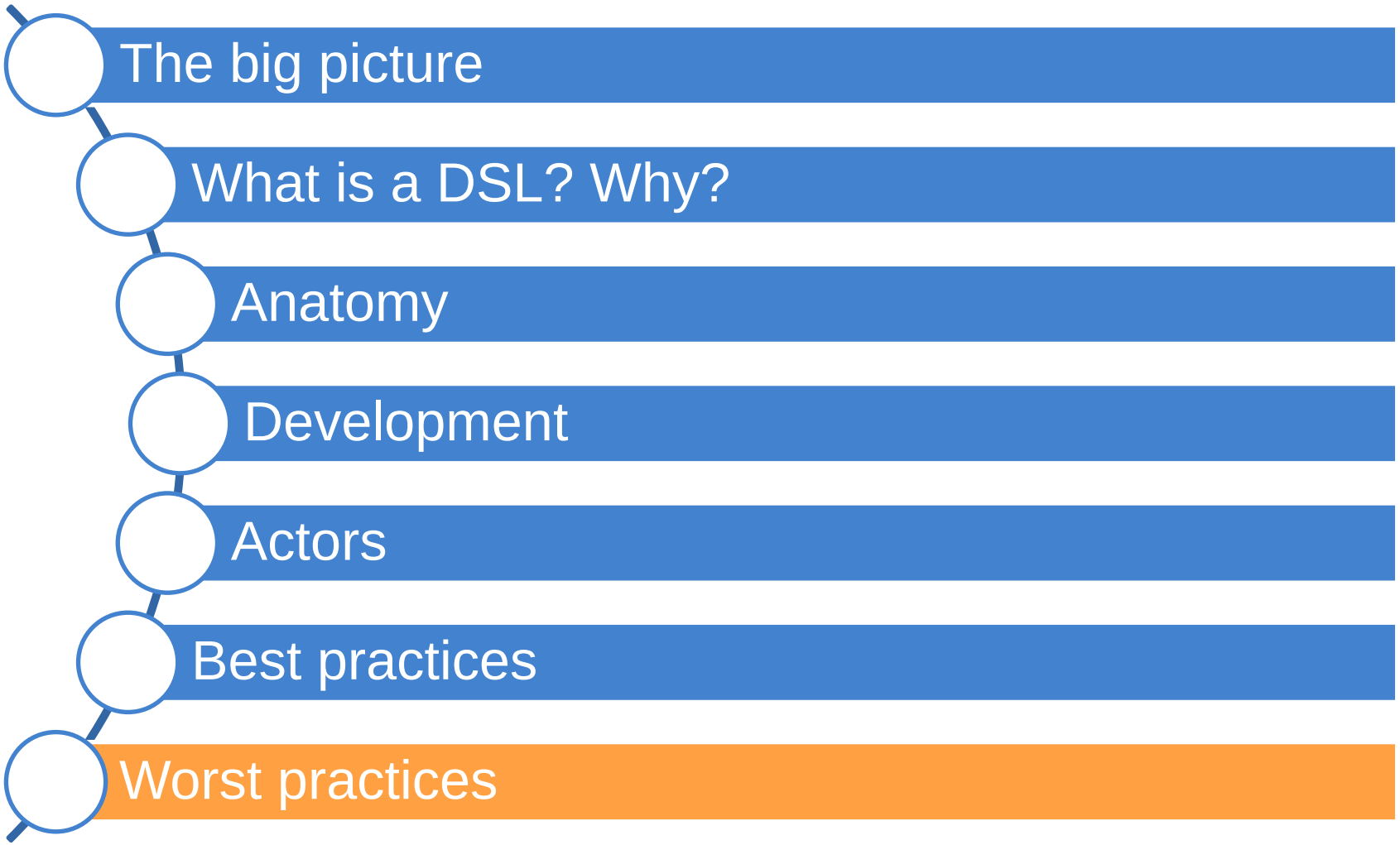
DSL


The big picture
What is a DSL? Why?
Anatomy
Development
Actors
Best practices
Worst practices

Best practices

- Limit Expressiveness
- Viewpoints
- Evolution
- Learn from GPLs
- Support
- Tooling

DSL


The big picture
What is a DSL? Why?
Anatomy
Development
Actors
Best practices
Worst practices

Worst practices

- Initial conditions
  - Only Gurus allowed
    - Believe that only gurus can build languages ir 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
    - Duplicanting 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

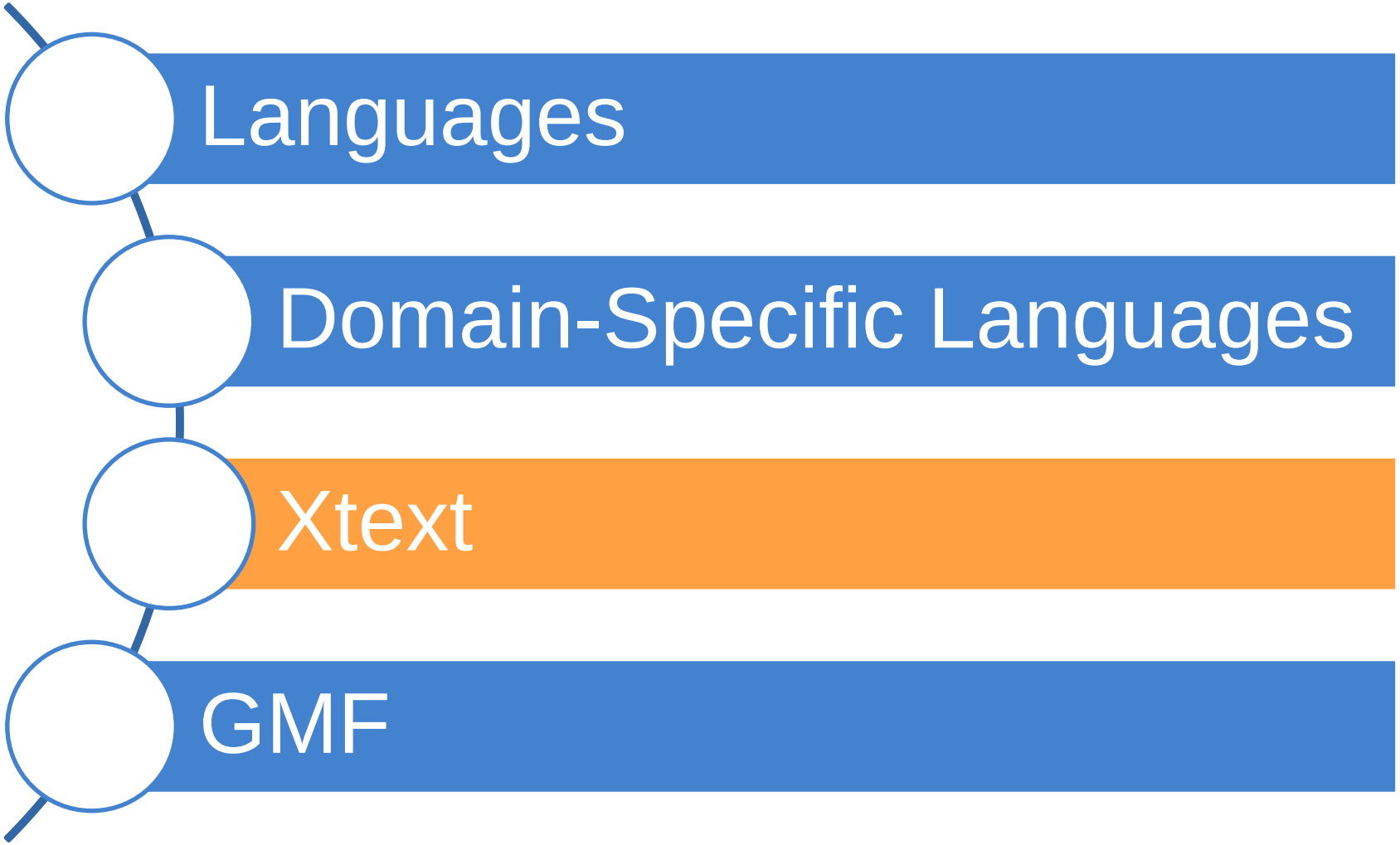
Worst practices

- 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

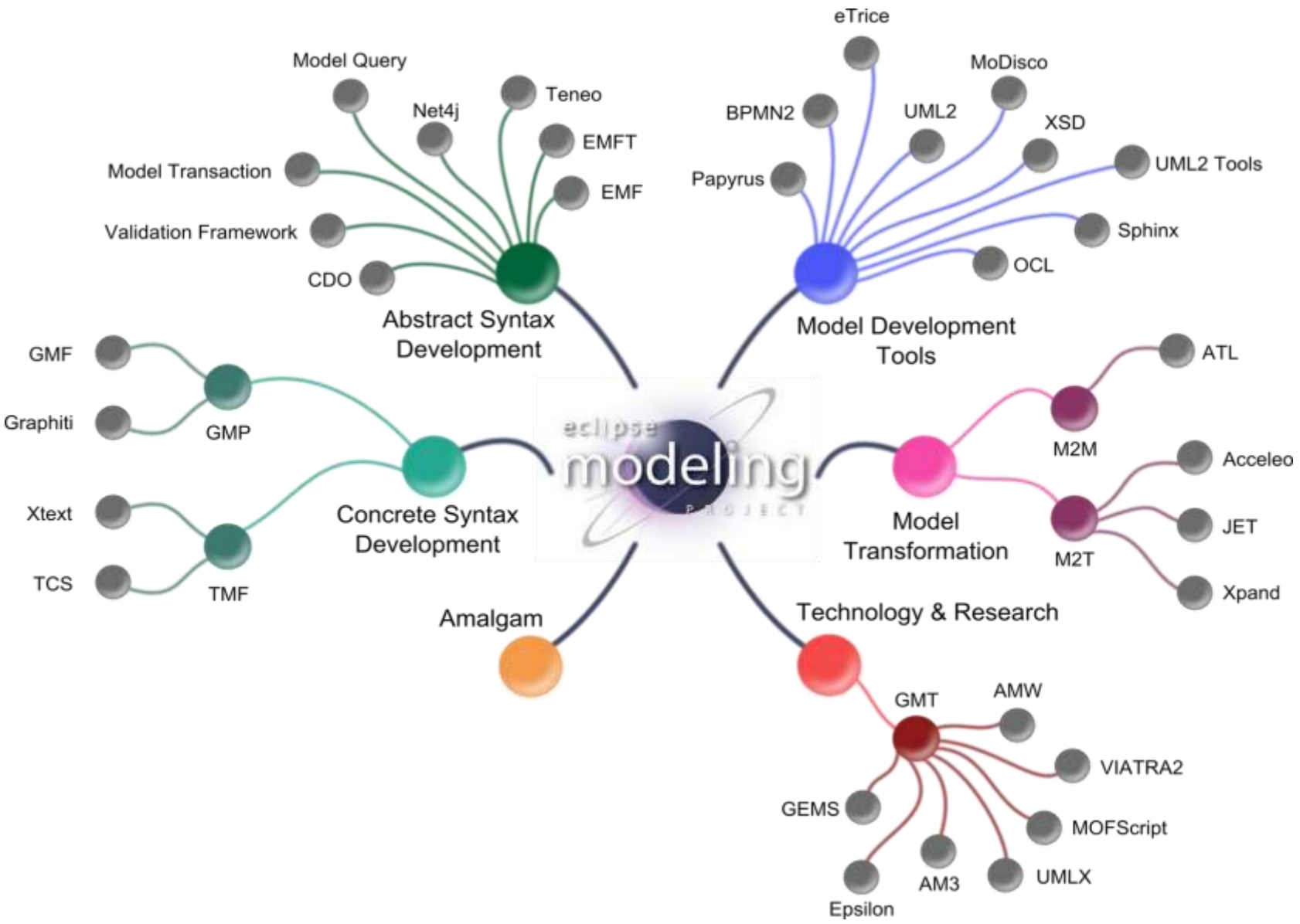
Outline



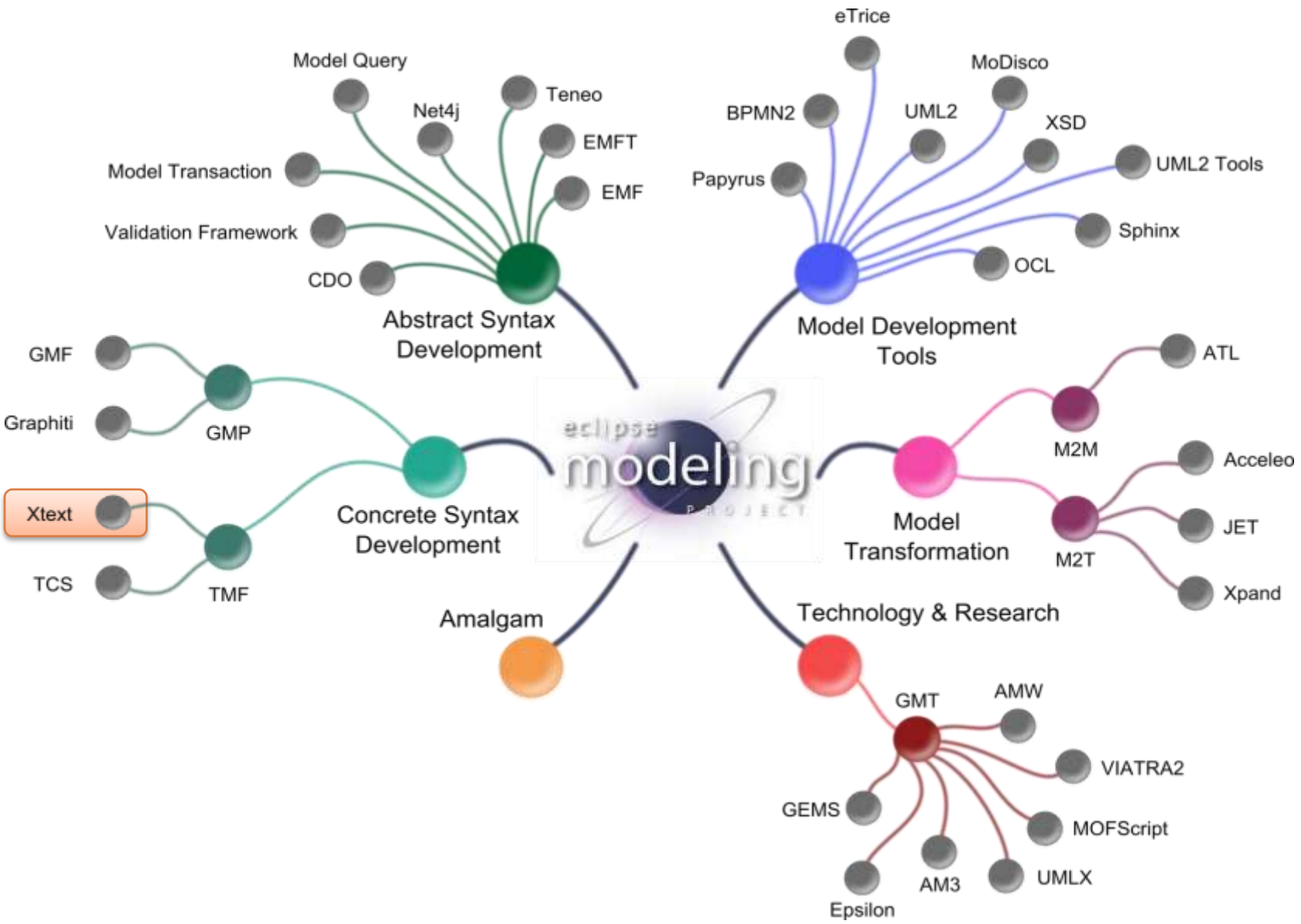
Xtext

- 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



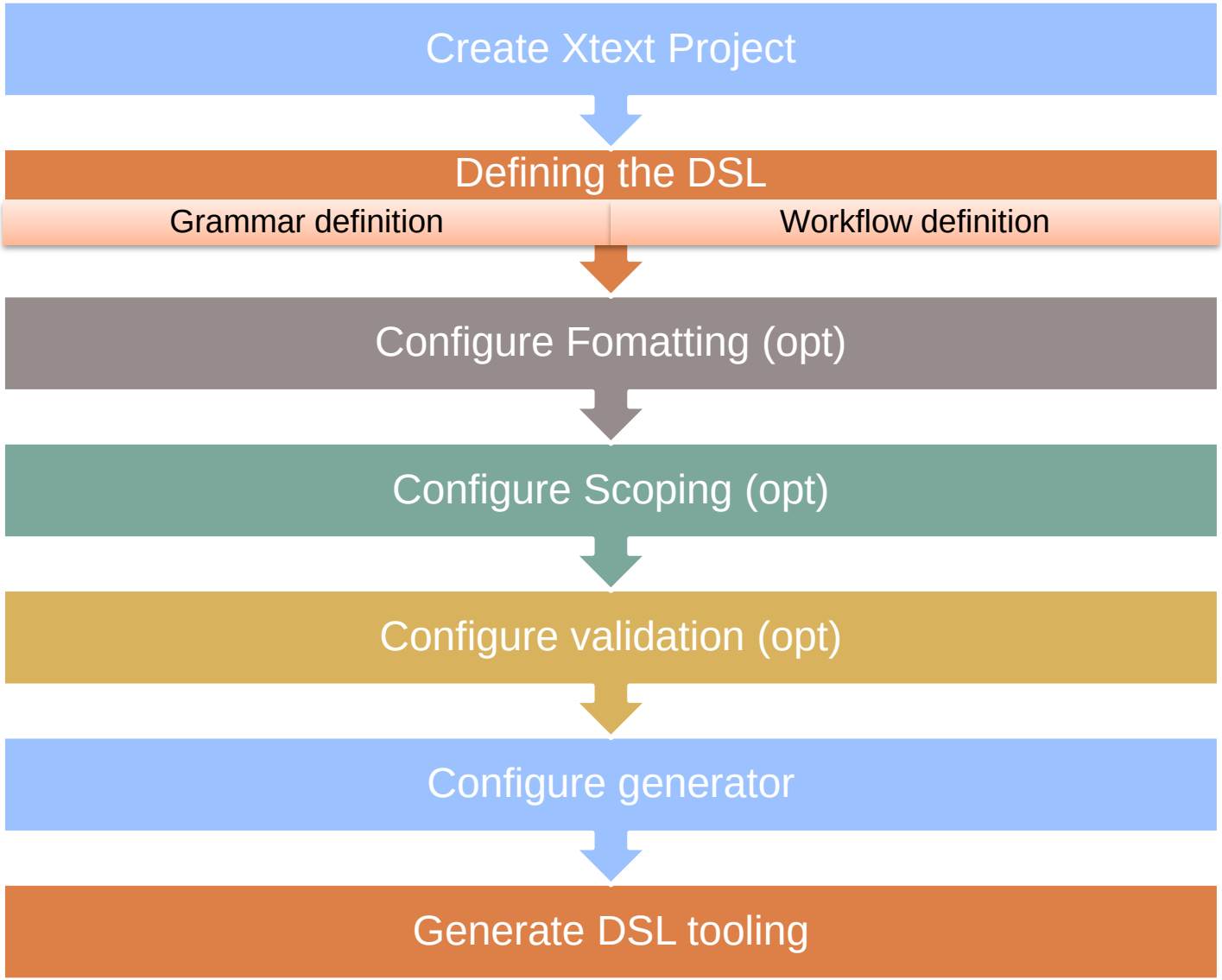
The grammar language

- 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

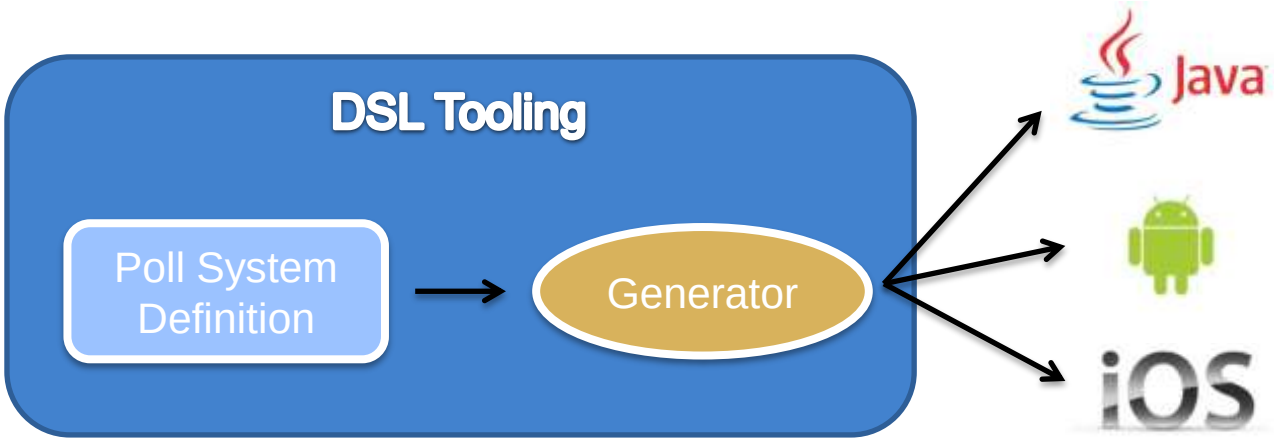
- Consistent look and feel
- Textual DSLs are a resource in Eclipse
- Open editors can be extended
- Complete framework to develop DSLs
- Easy to connect to any Java-based language

Proposed development process

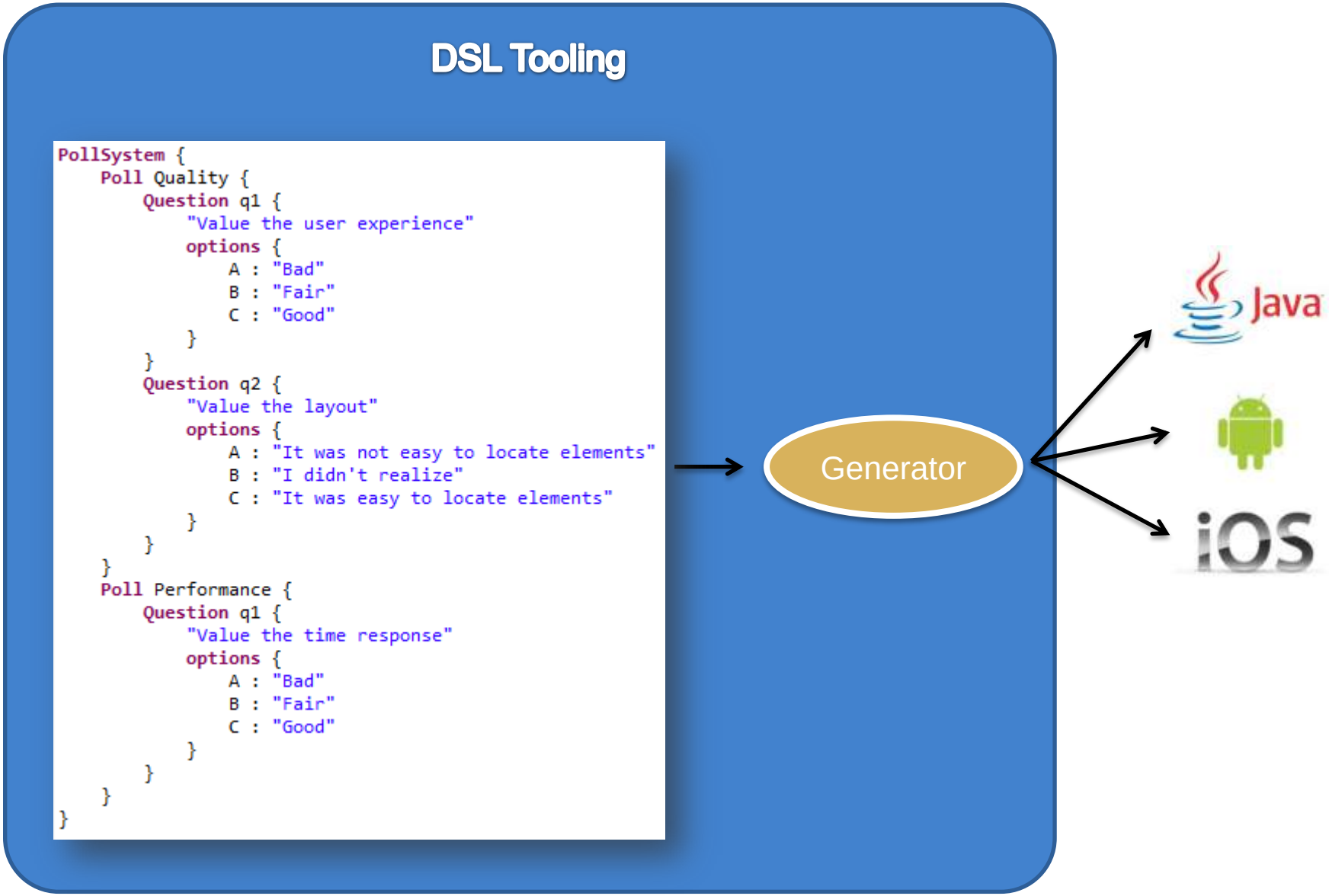


Example DSL

- 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



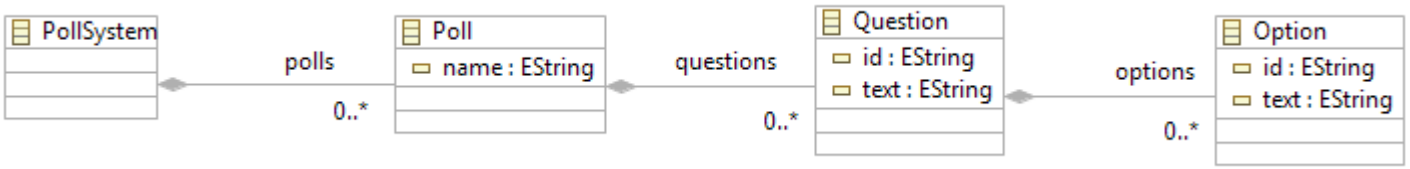
Example DSL


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
Java
iOS

Grammar Definition

Grammar definition →

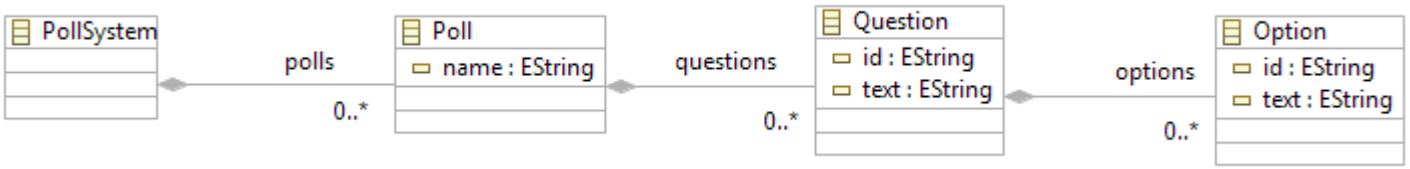
```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

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

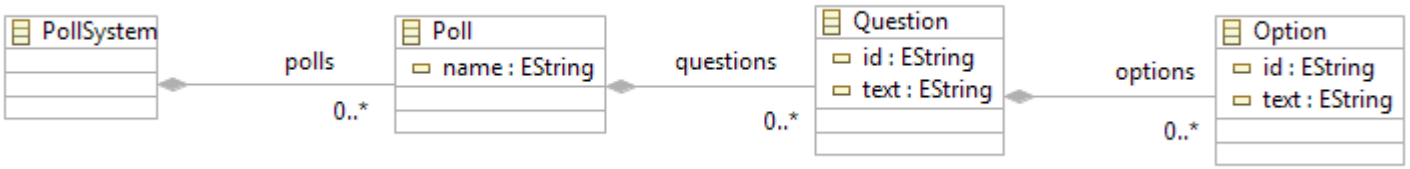
```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

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 →

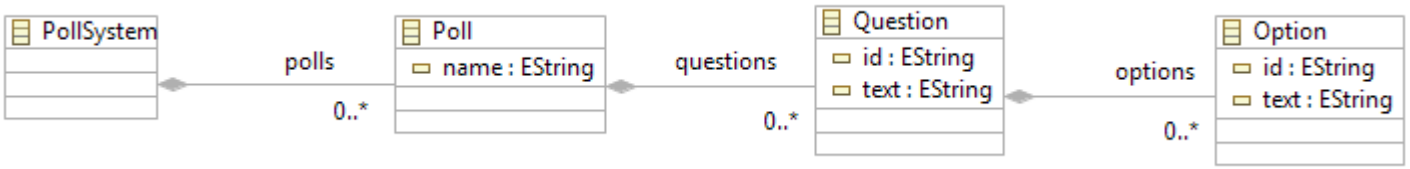
```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

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

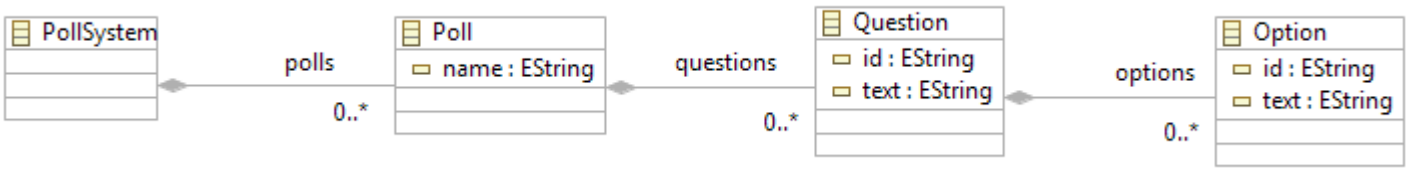
```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

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

Keywords

```
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;
```



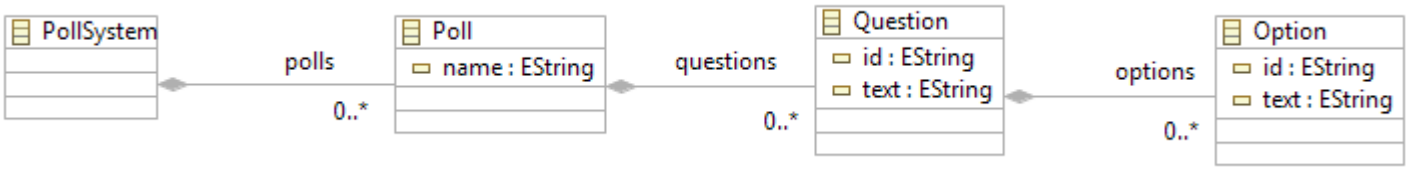
Grammar Definition

```
grammar 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;
```

Multivalue asignment

Simple asignment

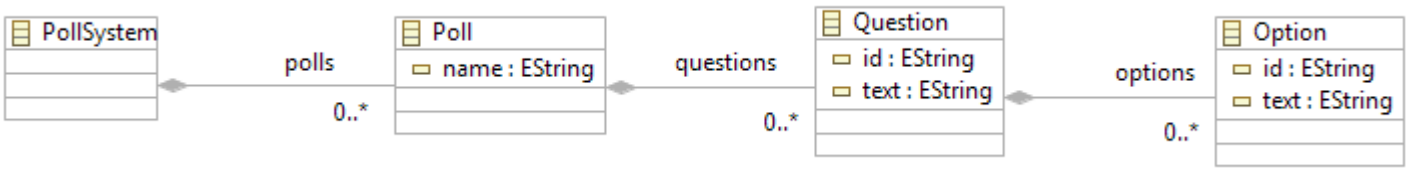
?= Boolean asignment



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;
```

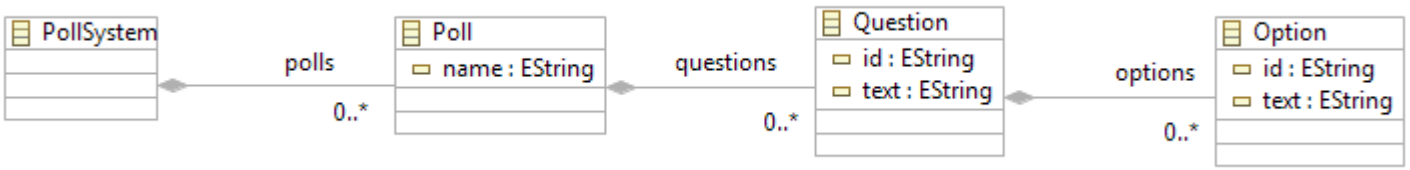
Cardinality (others: * ?)



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;
```

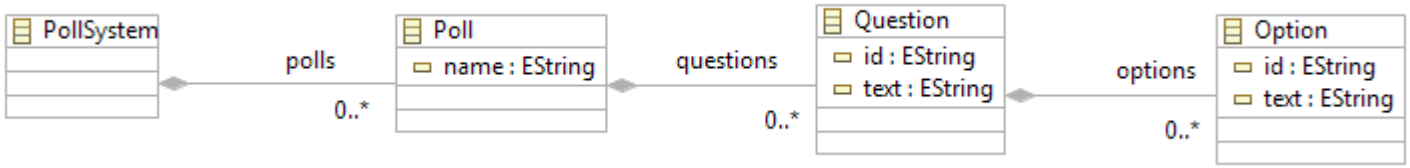
Containment



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;
```

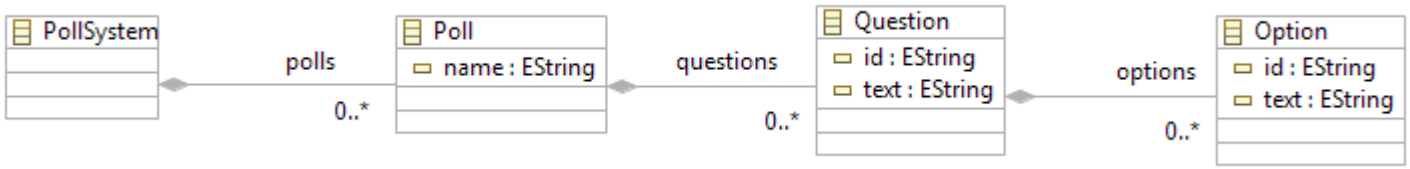
```
PollSystem {
    Poll Quality {
        Question q1 {
            "Value the user experience"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
        Question q2 {
            "Value the layout"
            options {
                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;
```

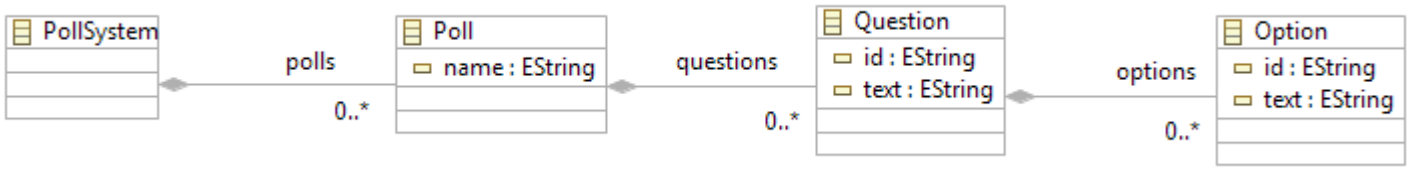
```
PollSystem {
    Poll Quality {
        Question q1 {
            "Value the user experience"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
        Question q2 {
            "Value the layout"
            options {
                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;
```

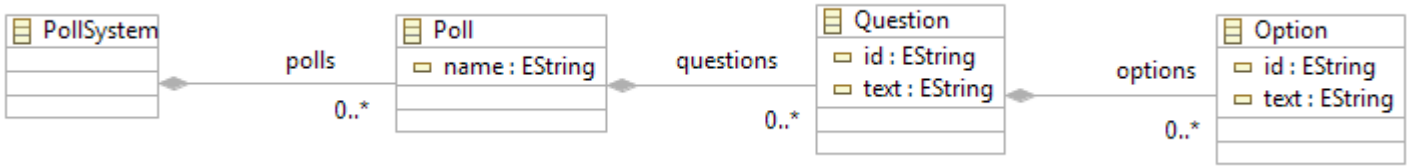
```
PollSystem {
    Poll Quality {
        Question q1 {
            "Value the user experience"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
        Question q2 {
            "Value the layout"
            options {
                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

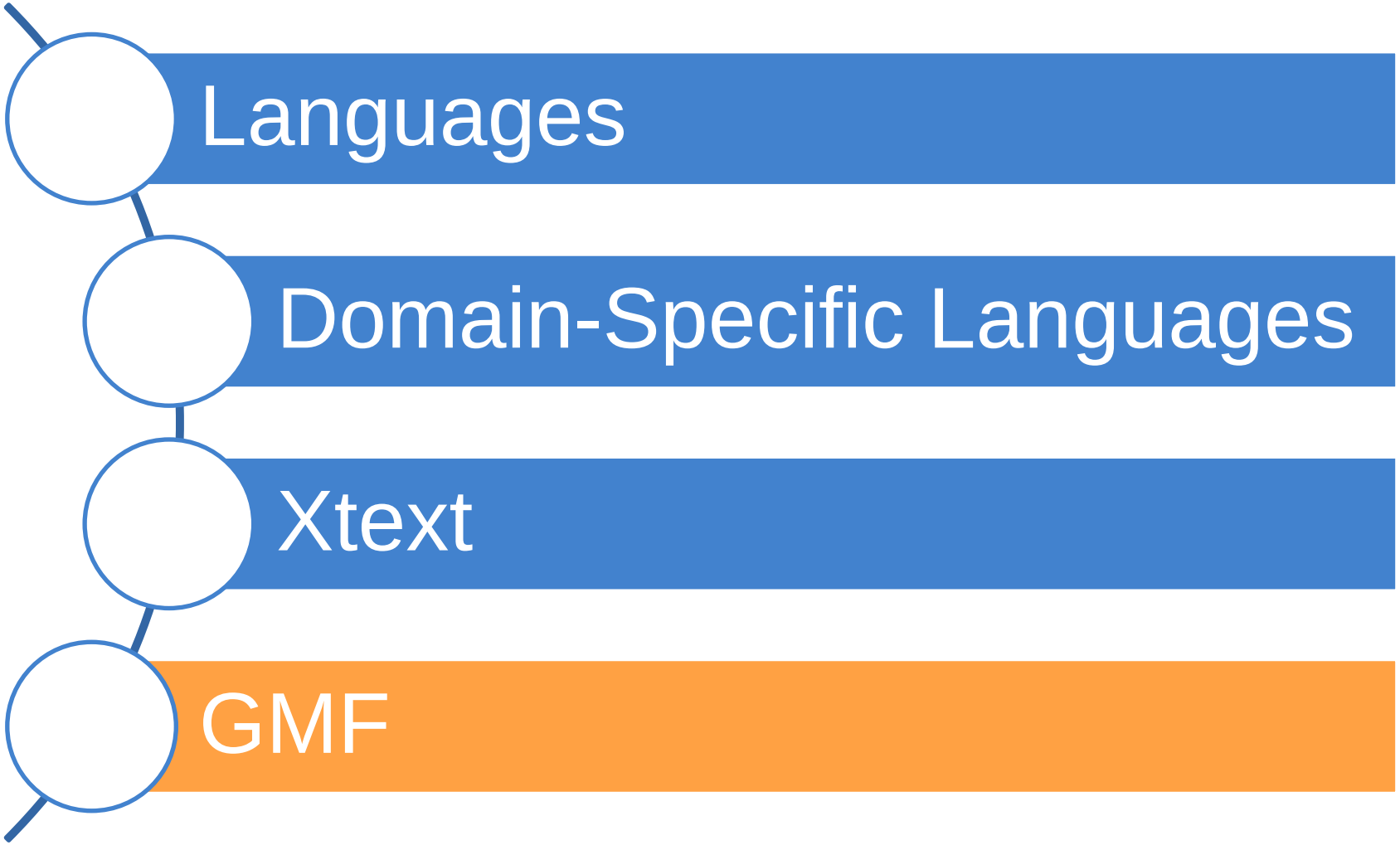
```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

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"
            }
        }
    }
}
```

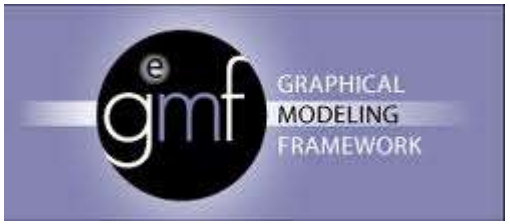


Xtext Demo

Outline

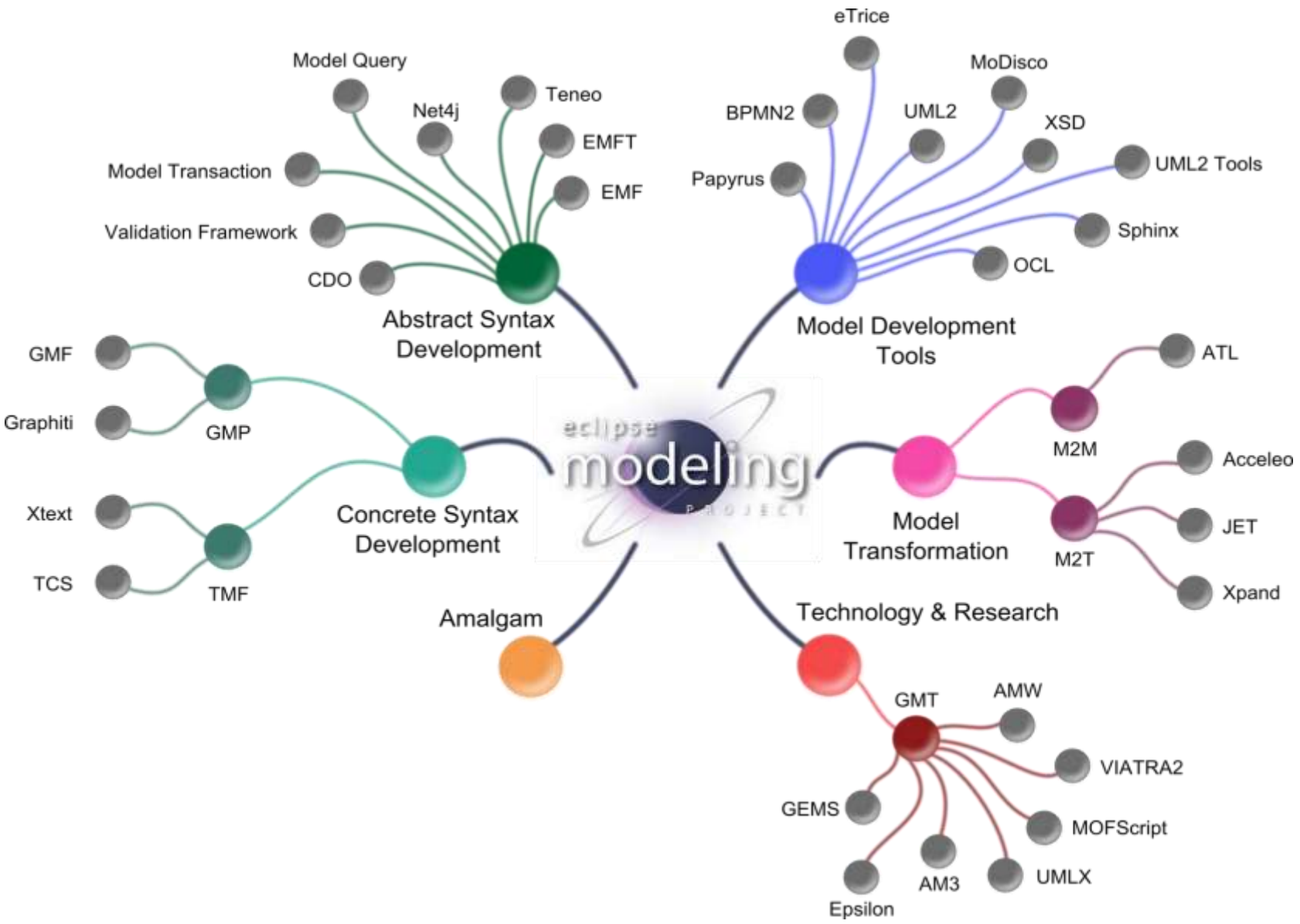


GMF

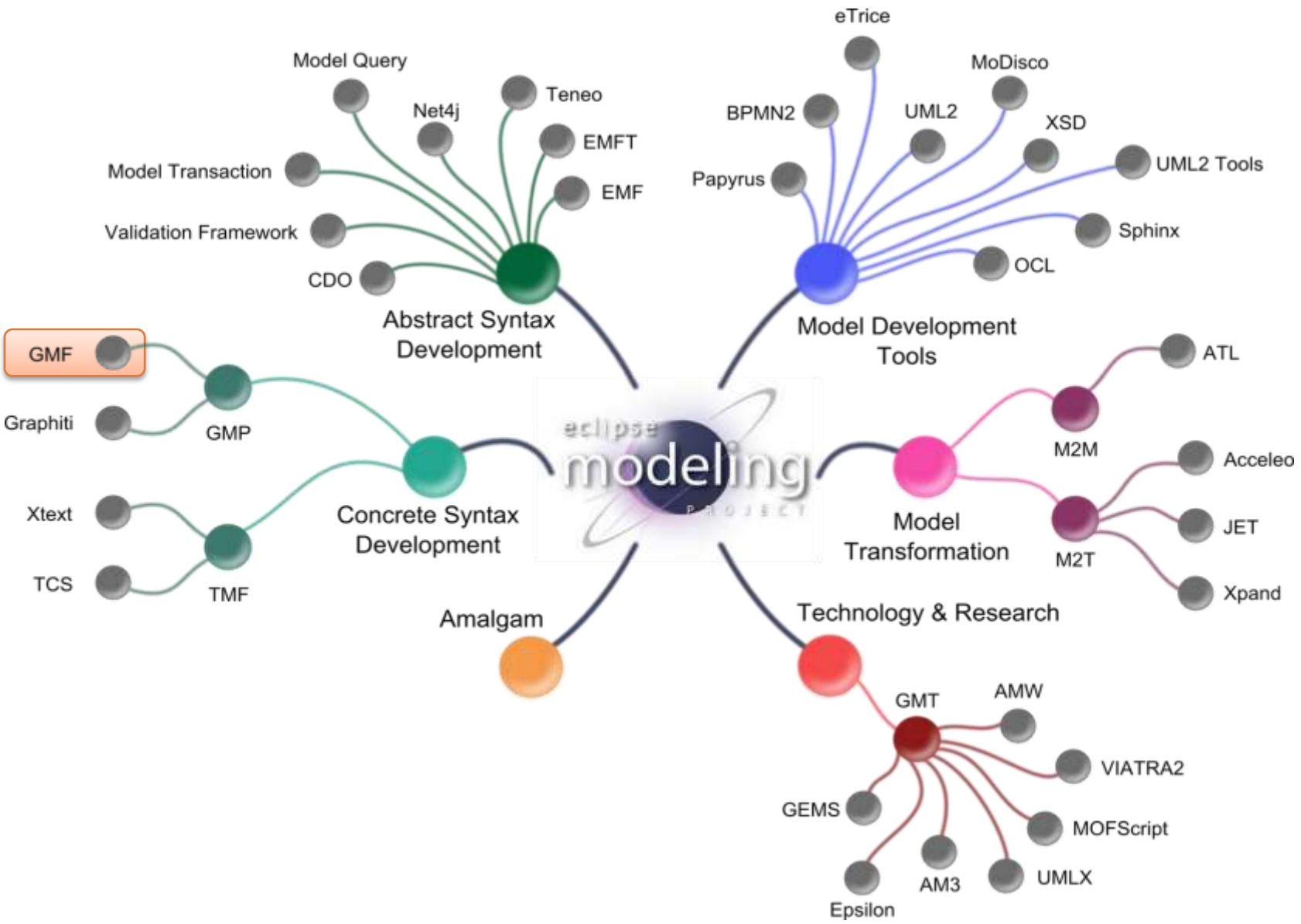


- Eclipse project
  - Eclipse Modelling components
  - Uses
    - EMF (Eclipse Modeling Framework)
    - GEF (Graphical Editing Framework)
- Model-driven framework for Graphical DSLs
  - Everything is a model
- Still under development (perpetual beta)
- The only alternative for now
- DSL definition easy,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

Eclipse Modeling Project



Eclipse Modeling Project



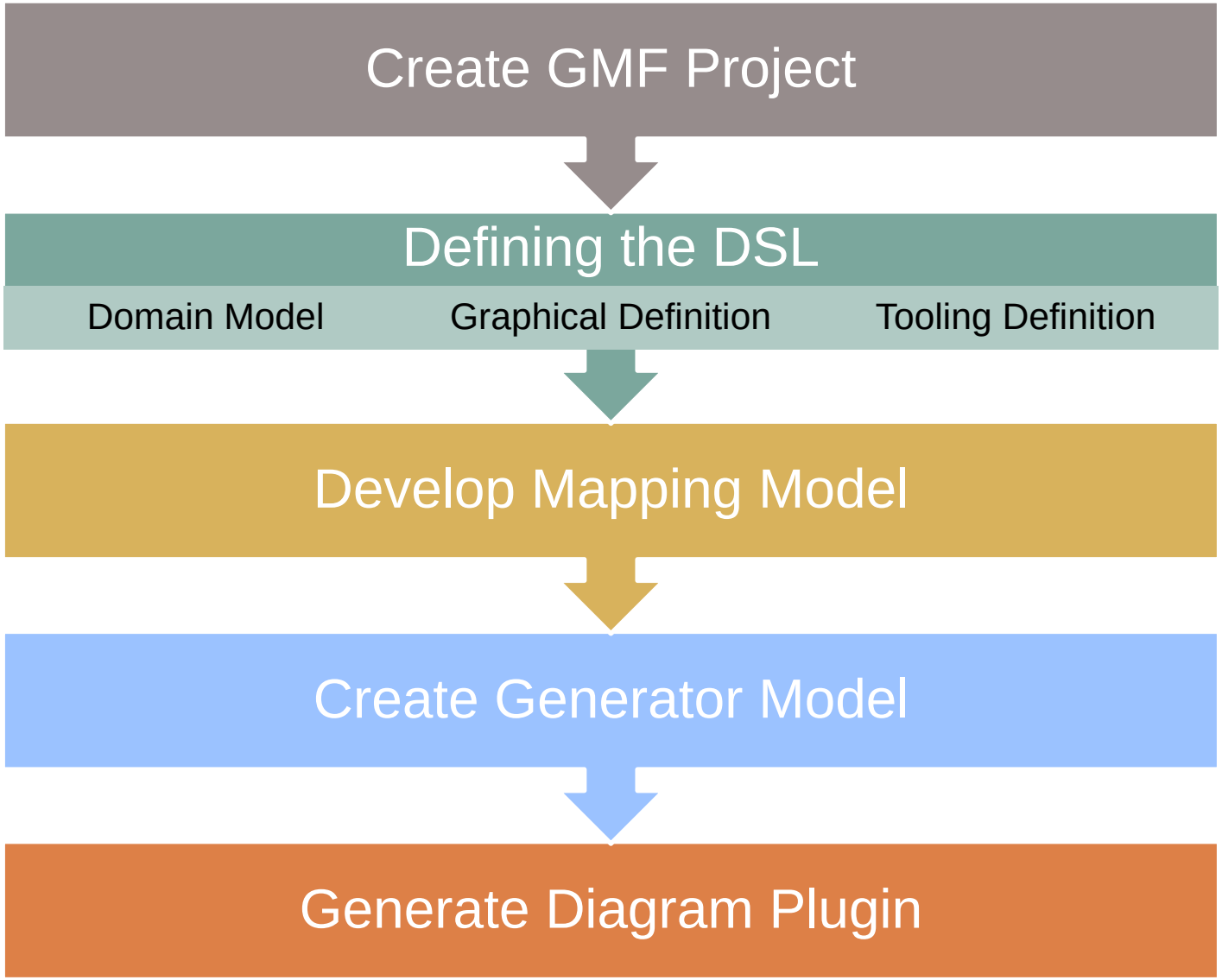
Parts of GMF

- 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

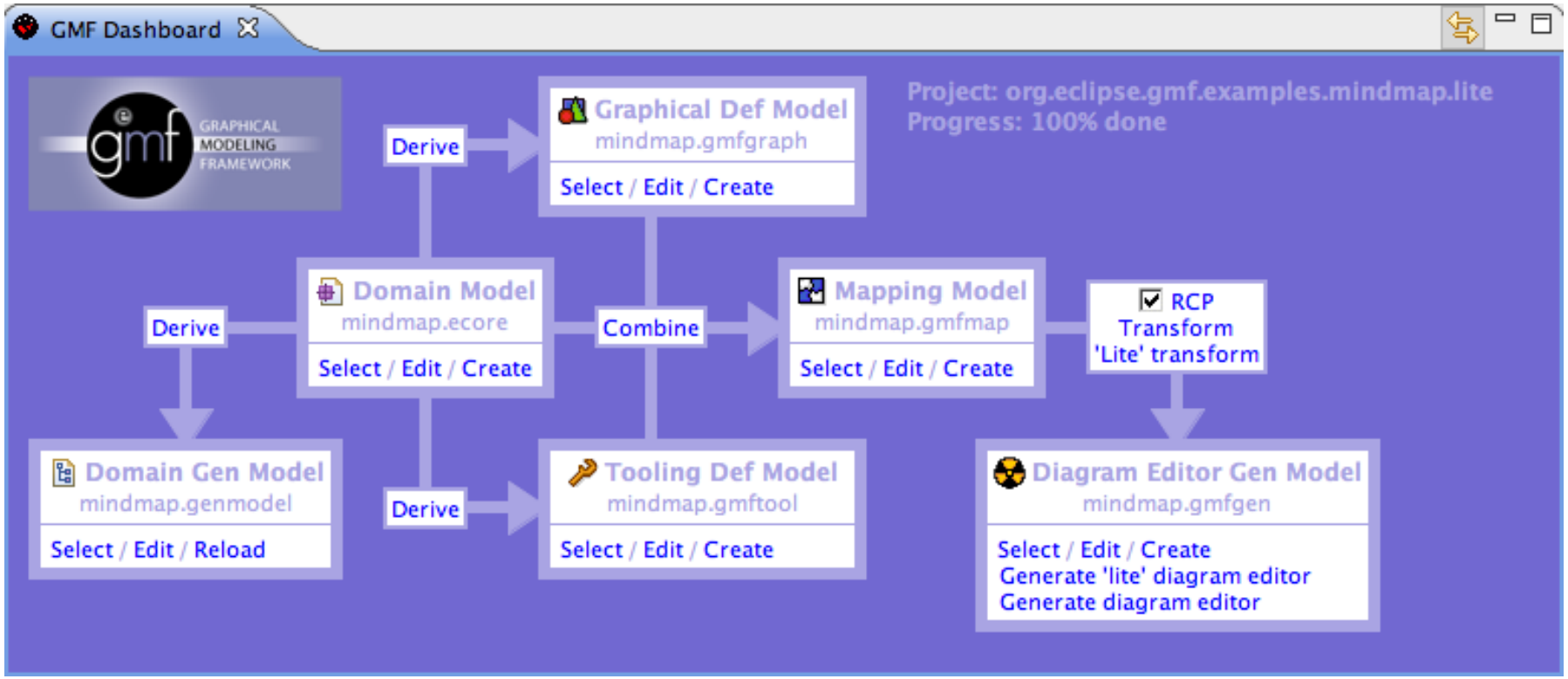
Main advantages

- 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

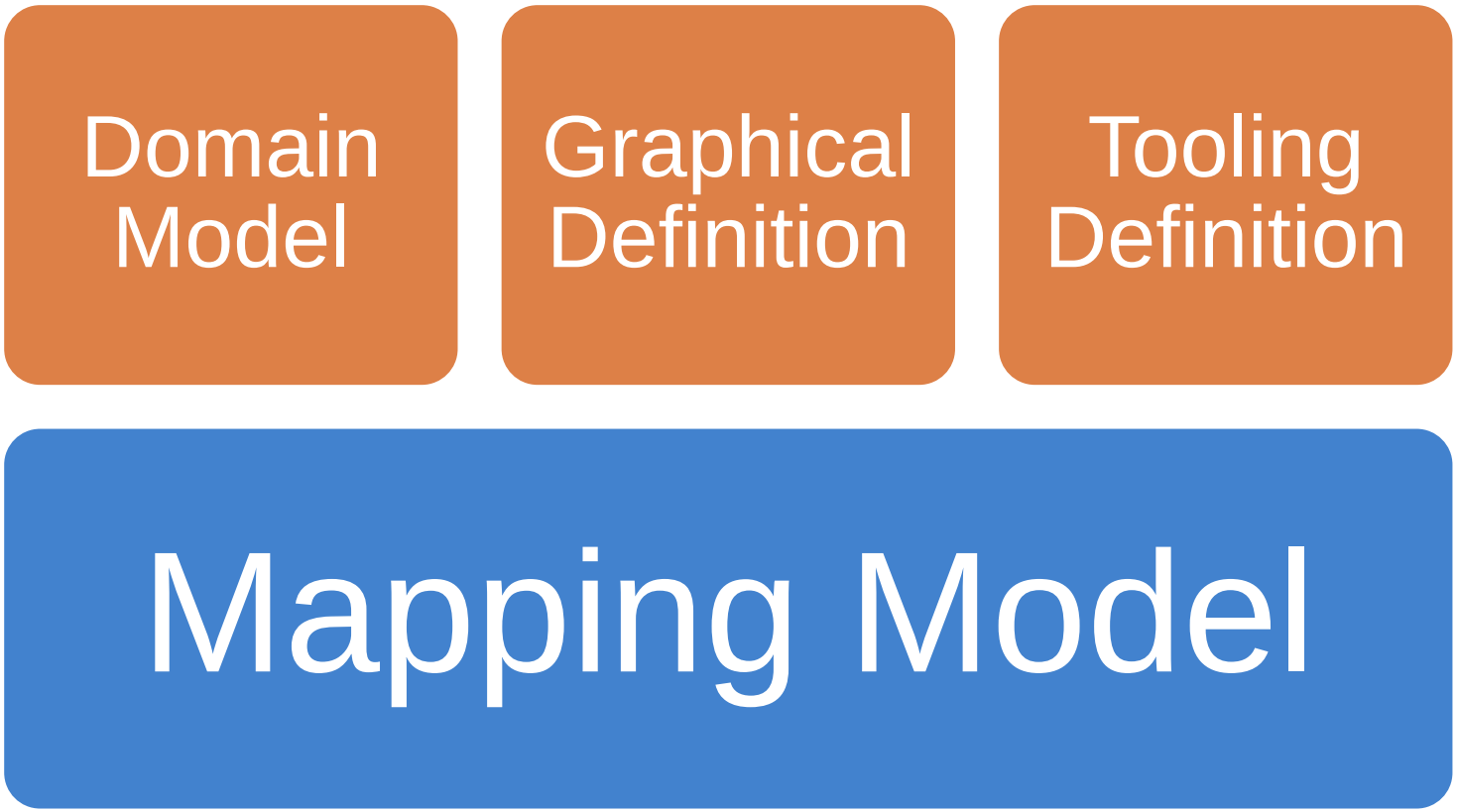
Proposed development process



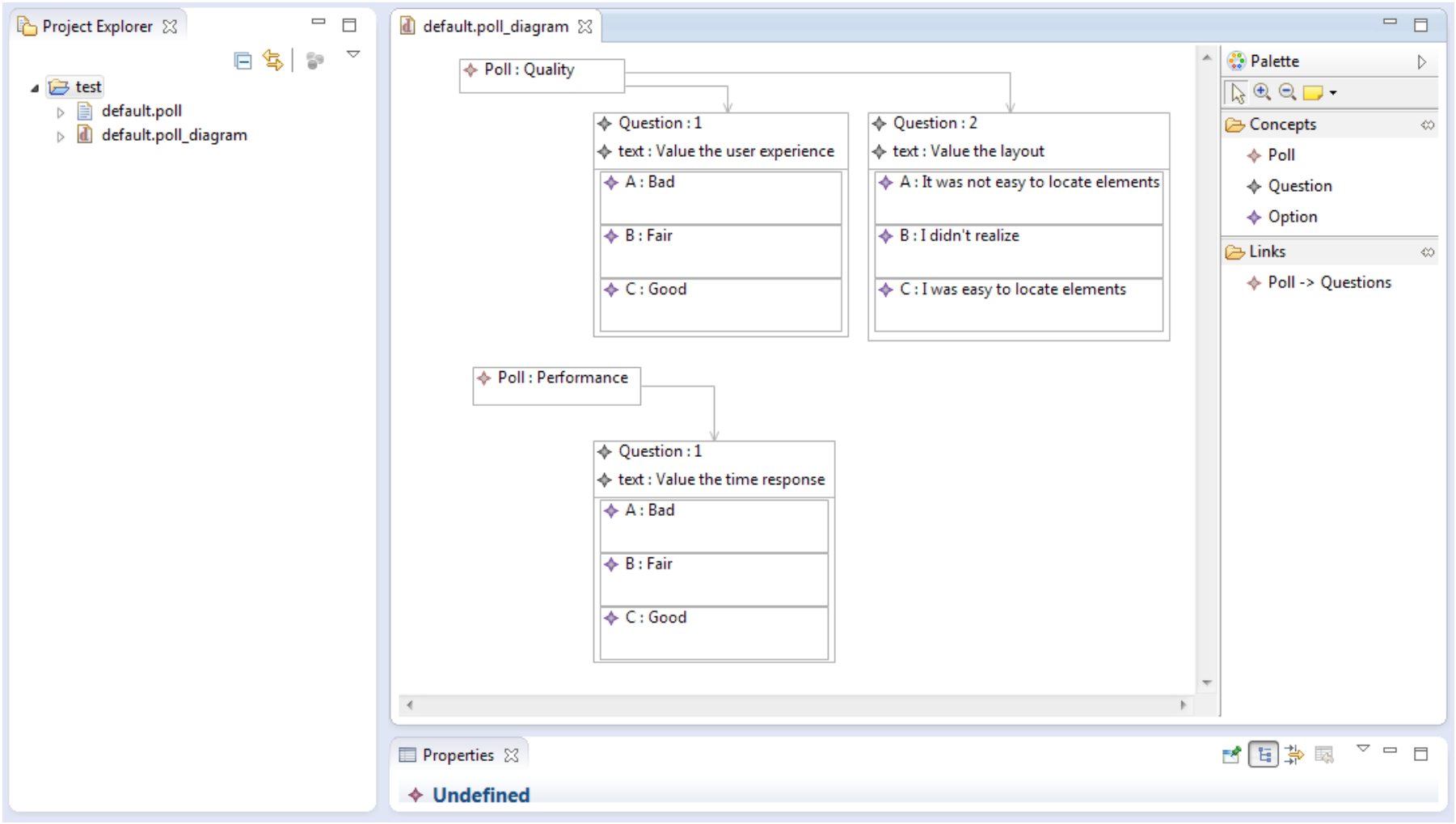
Proposed development process



Defining a graphical DSL

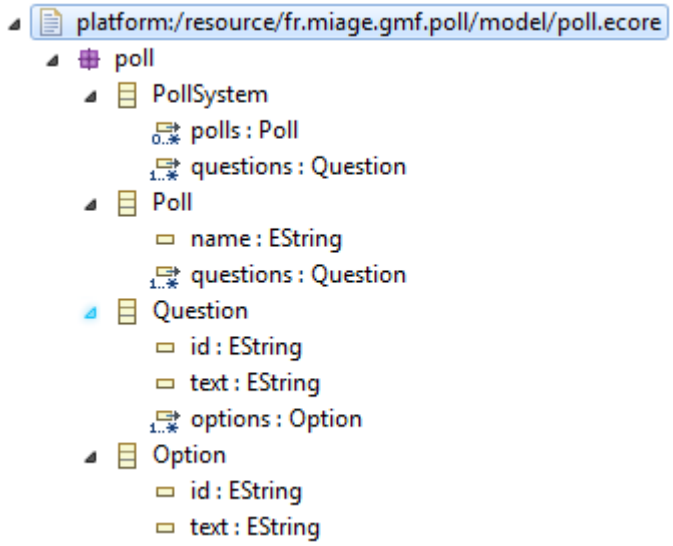


Example DSL


Project Explorer
test
default.poll
default.poll_diagram
default.poll_diagram
Poll : Quality
Question : 1
text : Value the user experience
A : Bad
B : Fair
C : Good
Question : 2
text : Value the layout
A : It was not easy to locate elements
B : I didn't realize
C : I was easy to locate elements
Poll : Performance
Question : 1
text : Value the time response
A : Bad
B : Fair
C : Good
Palette
Concepts
Poll
Question
Option
Links
Poll -> Questions
Properties
Undefined

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

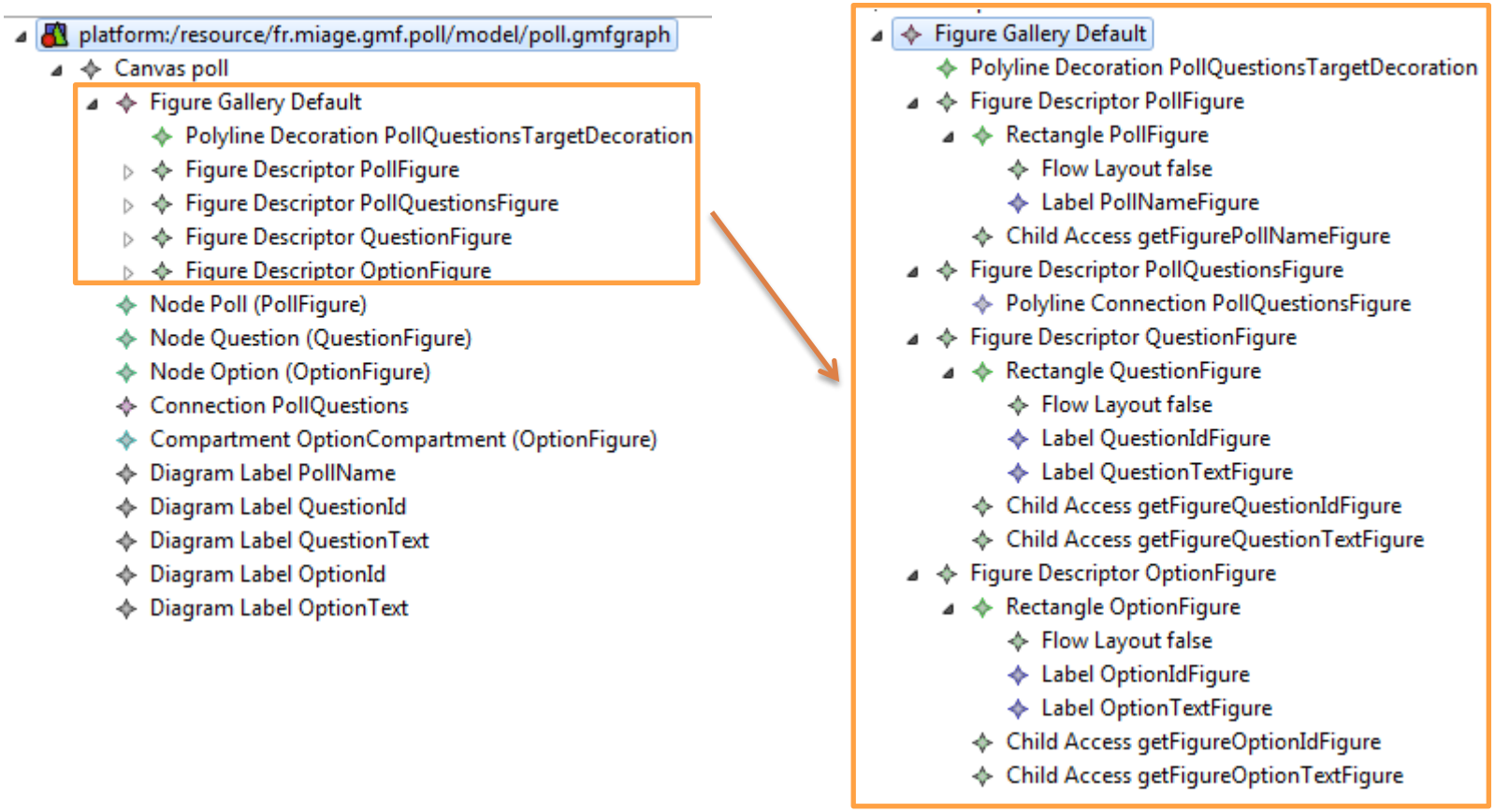


Graphical Definition

- 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

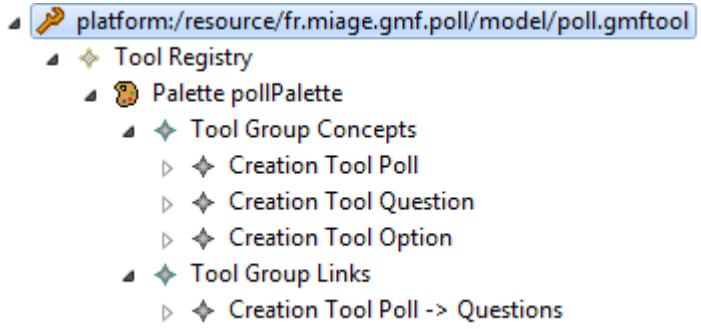
Graphical Definition

- 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



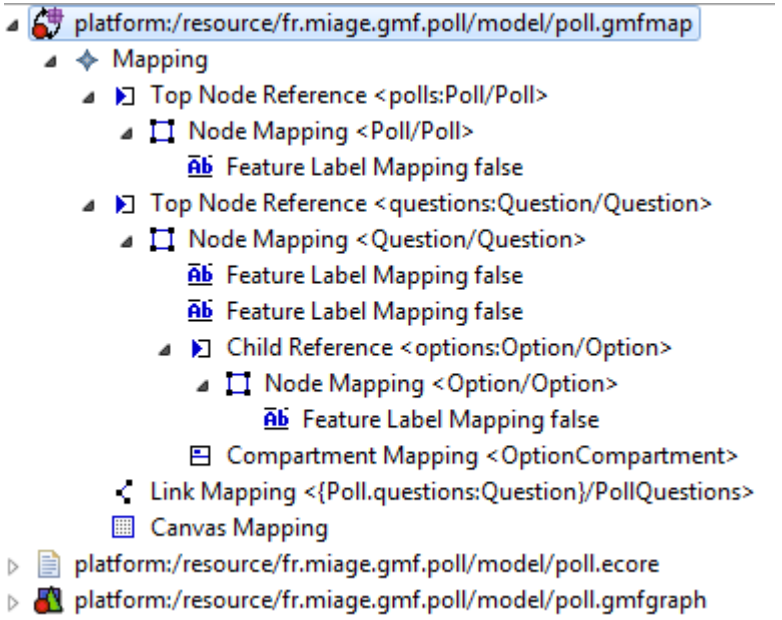
Tooling Definition

- 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 correspondeces 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 concet 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