

Model-driven Generation of Graphical Maps for e-Contents



Antonio Natali¹, Enrico Oliva¹, Cristina Bonanni²
¹*Alma Mater Studiorum - Università di Bologna, Cesena, Italy*
{antonio.natali, enrico.oliva}@unibo.it
²*IBM Italia, Tivoli Software, Roma, Italy*
{cristina.bonanni}@it.ibm.com

Outline

- Background
 - Model Driven Software Development (MDSD)
- Motivations
- e-Content Model
- Editor and Grammar Construction
- Generation Process
- Map Visualization
- Conclusions

Model Driven Approach I

- Model Driven Software Development (MDSD) is a strategic issue in the modern software system:
 - it makes explicit knowledge (through the models) that usually remains implicit
 - it promotes automatic generation of code overcoming the gap between model and implementation

Model Driven Approach II

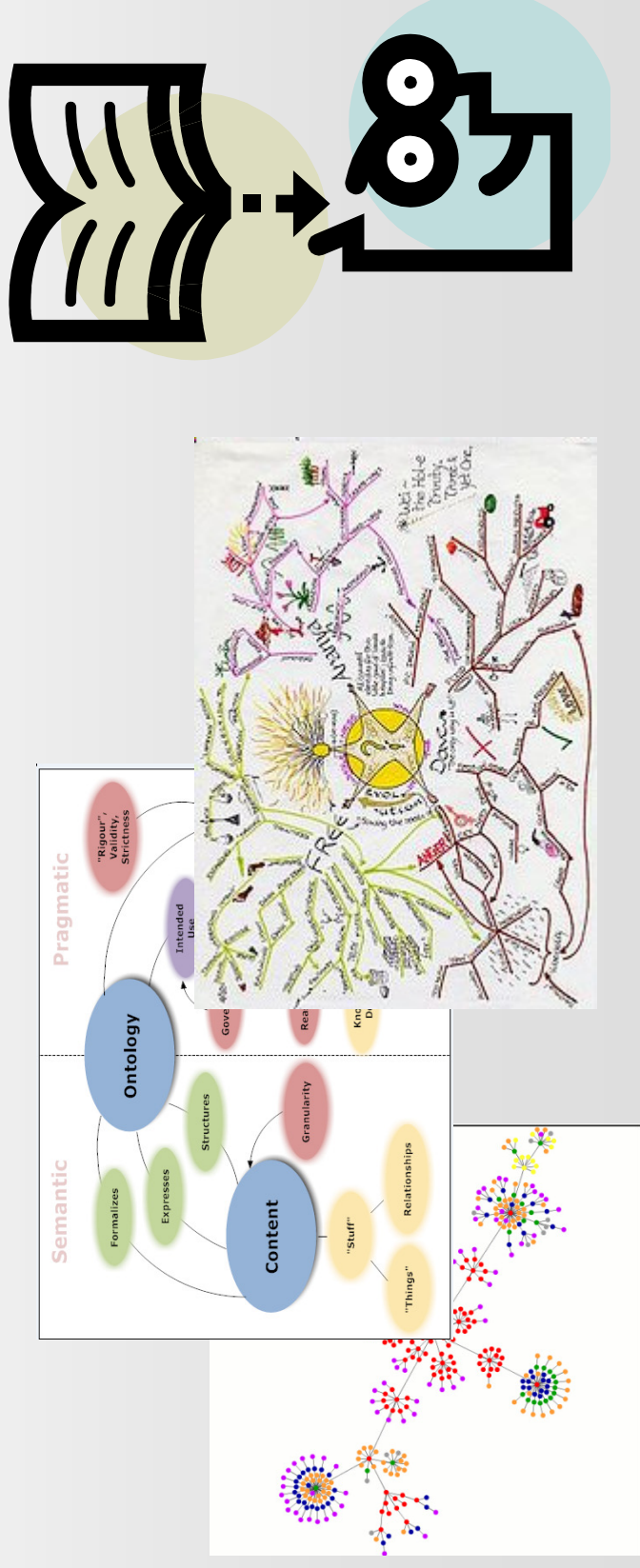
- Eclipse Modeling Framework (EMF) supports the development of *Platform Independent Model* in Ecore providing
 - a runtime support to produce a set of Java classes for the model
 - a set of adapter classes that enable viewing and command-based editing of the model
- Domain Specific Languages (DSL) are the core of MDSD in order to
 - capture the key aspect of the domain
 - promote automatic generation of code

Outline

- Background
- **Motivations/Objectives**
- e-Content Model
- Editor and Grammar Construction
- Generation Process
- Map Visualization
- Conclusions

e-Content Organization I

- The content organization has a dramatic impact on the user ability to acquire knowledge
- We promote the use of graphical maps
 - to structure and organize information
 - to enhance understanding and learning

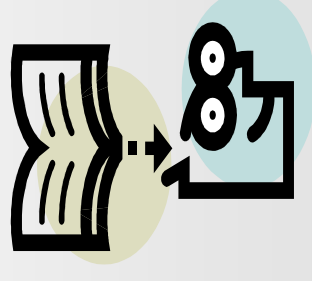


e-Content Organization II

- In our approach the conventional navigation loop



- 1. select a topic from a index,*
- 2. open the topic, and*
- 3. read the content*



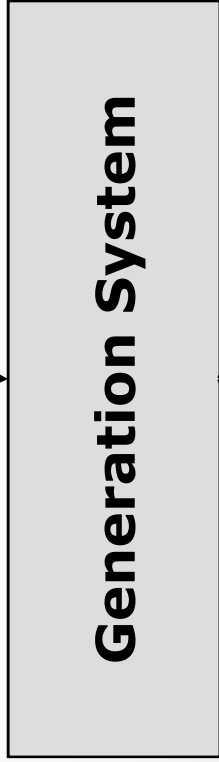
is replaced by

- a semantic navigation that can be customized into
reading path pre-planned by the designer and
semantic map of contents

What have we done?

Content Model

How to construct the model?



How to provide the generation?

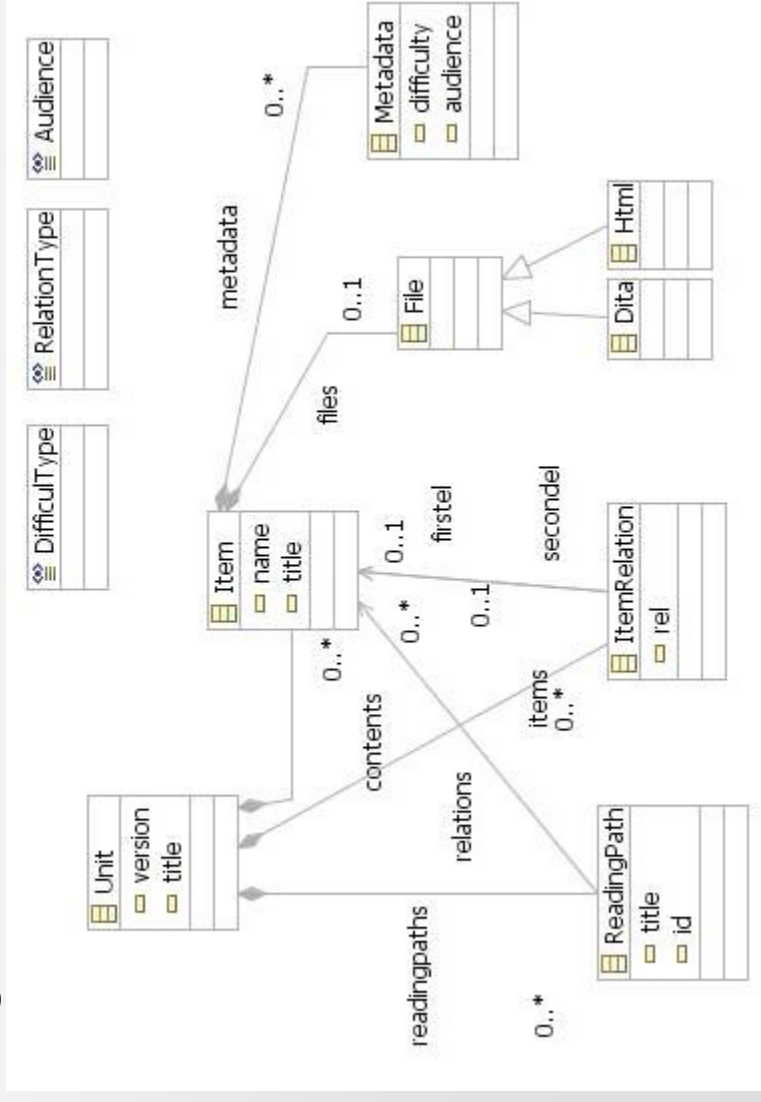
The composite image illustrates the components of a generation system. On the left is a dense network diagram with numerous nodes and connecting lines. In the center is a hierarchical tree structure with nodes labeled 'Loyalty', 'Status', 'Scope', and 'Forms'. On the right is a 3D visualization of a network with nodes represented as colored spheres and connecting lines, set against a dark background.

Outline

- Background
- Motivations
- **e-Content Model**
- **Editor and Grammar Construction**
- Generation Process
- Map Visualization
- Conclusions

Meta Model

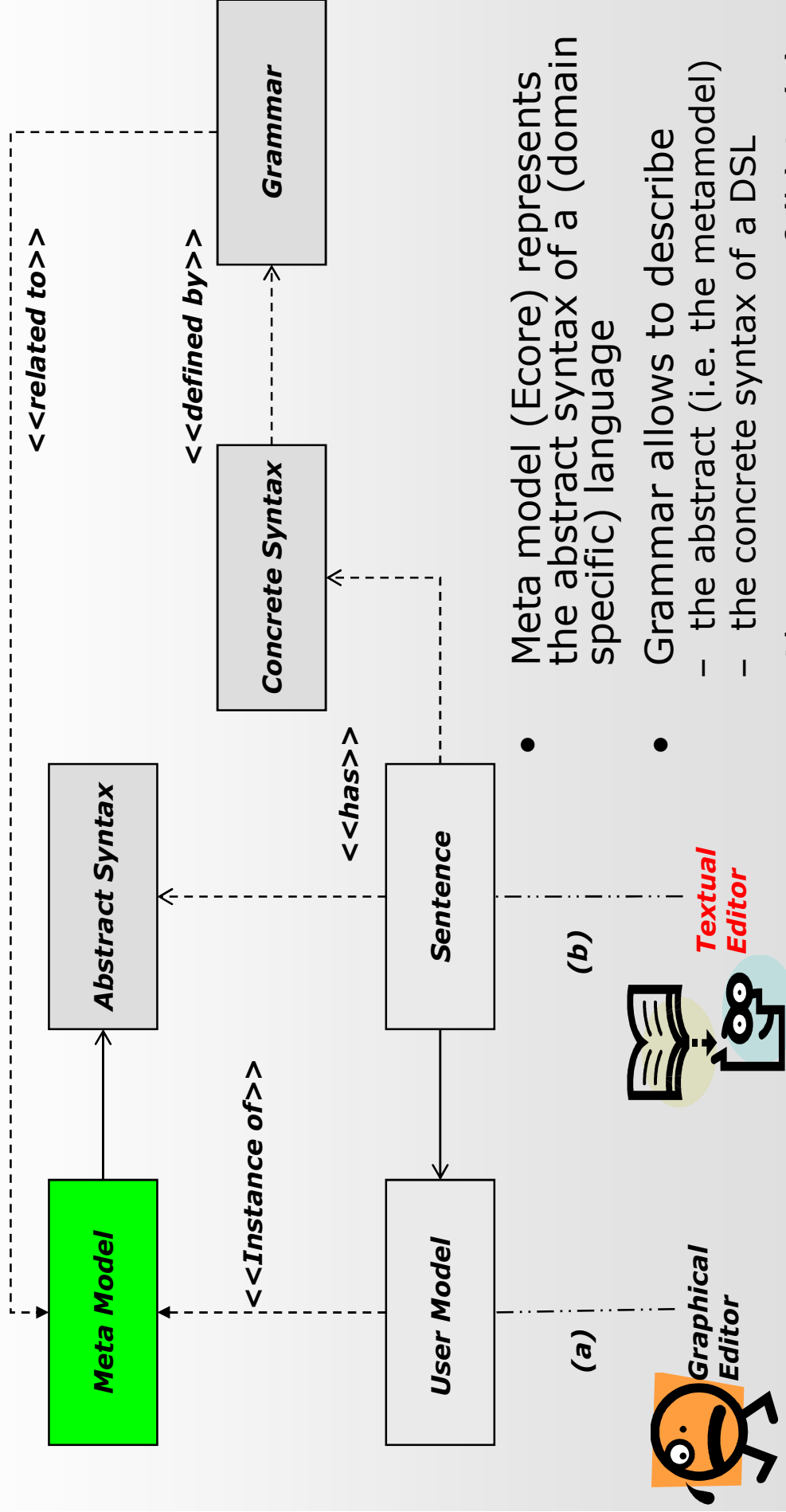
- In [Natali et al. 2007] the MDSD approach is adopted which define a formal model for its content:
 - The **meta model** defines (the abstract syntax of) a *domain specific language* (an extension of DITA concepts) for content organization



Textual Editor

- Text model representation
 - is useful for large systems (by experience)
 - is stored in a human-readable format (it is possible to exploit tools as CVS and SVN)
 - is makes easier the integration/transformation between different DSL
- Textual Editor (`xText`) provides
 - DSL specific **constraint checks**
 - **languages support** (code completion, syntax error ...)

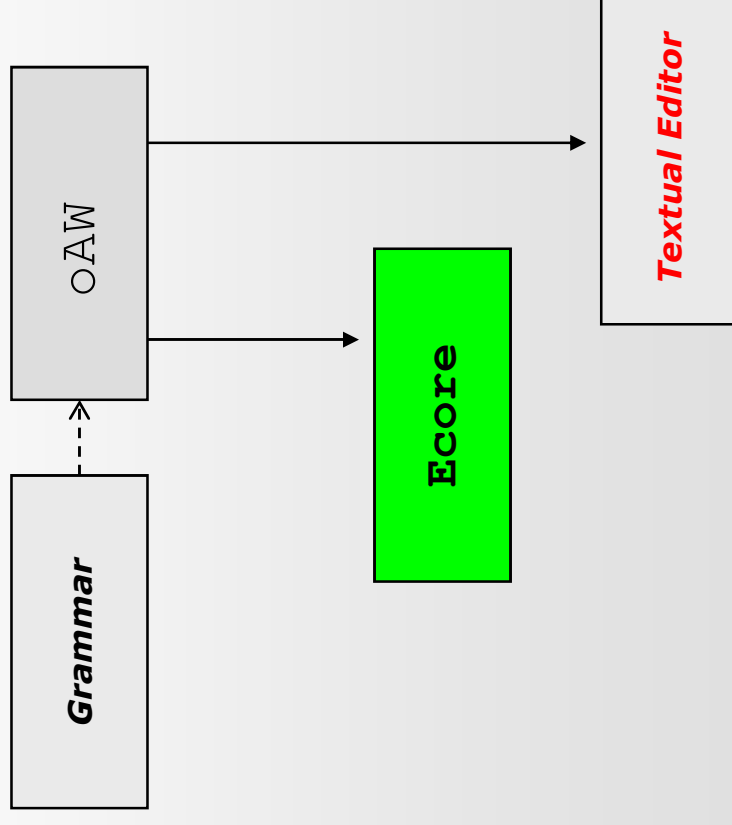
From Model to Grammar



- Meta model (Ecore) represents the abstract syntax of a (domain specific) language
- Grammar allows to describe
 - the abstract (i.e. the metamodel)
 - the concrete syntax of a DSL
- Abstract parser tree of (b) is (a)

Technology

- openArchitectureWare (oAW) is Eclipse Project for model driven software development; it supports:
 - the parser of models
 - the check and transformation of models
 - the code generation
- `xText` is a component of oAW . It supports:
 - text definition of DSL in `EBNF` notation
 - automatic generation of **textual editor** for Eclipse
 - automatic generation of `Ecore` meta model



Example of eContent organization specification

```
econtent Computer Science

item func
  title "Function"
  meta difficulty high audience beginner
  file "course\lp_function.dita"
item stat
  title "Statement"
  meta difficulty low audience beginner
  file "course\tp_statement.dita"
...
readingpath labpath { java prog jadt adt list }
readingpath thpath { func stat val adt list arr }

relation func preknowledge-of prog
relation stat preknowledge-of prog
...
```

Domain Specific Language

- To support the new contents organization our DSL provides the following concepts:
 - Each content `Unit` is composed of a set of logical contents called *Items* or *Topics*.
 - Each `Item` is associated to one or more resource `File` that can be written in any format readable by a conventional browser.
 - Each `Item` can be associated to a (empty) set of *metadata* including binary logical relations with other items.
 - The author can define one or more *reading paths*.
 - *Graphical maps* are used to arrange the items into custom-related highly-readable contexts.

Item Relation

- **Item Relations** are the key-concept introduced in the formal model in order to
 - specify semantic relations among parts of the content
 - represent content in term of conceptual map
- Set of defined relation types
 - **Preknowledge-of:** S assumes that the reader knows what is written in T
 - **Clarification-of:** T should make the content of S more clear;
 - **Conceptualization-of:** T presents the content of S in a more formal way;
 - **Widening-of:** T is a study in depth of S;
 - **Experiment-of:** T is an experiment related to S;
 - **Exercise-of:** T is an exercise related to S;
 - **Test-of:** T is an evaluation of the student understanding of S.

Grammar Rules in xText

```
Unit :
    "econtent" title=STRING
    ("version" version=ID)?
    (contents += Item)*
    (readingpaths += ReadingPath)*
    (relations += ItemRelation)* ;

Item :
    "item" name=ID
    "title" title=STRING
    ("path" files += File)+
    (metadata += Metadata)* ;

Metadata : "meta"
    ("difficulty" difficulty=Difficulty)?
    ("audience" audience=Audience)? ;

File : STRING(Dita | Html) ; Dita : ".dita";
Html : ".html";

ReadingPath :
    "readingpath" id=ID title=STRING "{
    (items += [Item])+ }";

ItemRelation : "relation" firstel=[Item]
    rel=RelationType secondel=[Item];

Enum RelationType : A1="widening-of" | ...
Enum DifficultyType : dt1="low" | ... ;
Enum AudienceType : at1="expert" | ...;
```

- Production rules are RuleName:Description
- RuleName is both
 - the name of the rule and
 - the name of the class (metatype) in the metamodel
- Description is made up of tokens that can be
 - a built-in token ID
 - **KeywordTokens**,
 - IdentifierTokens,
 - **AssignmentTokens**

Conversion Criteria

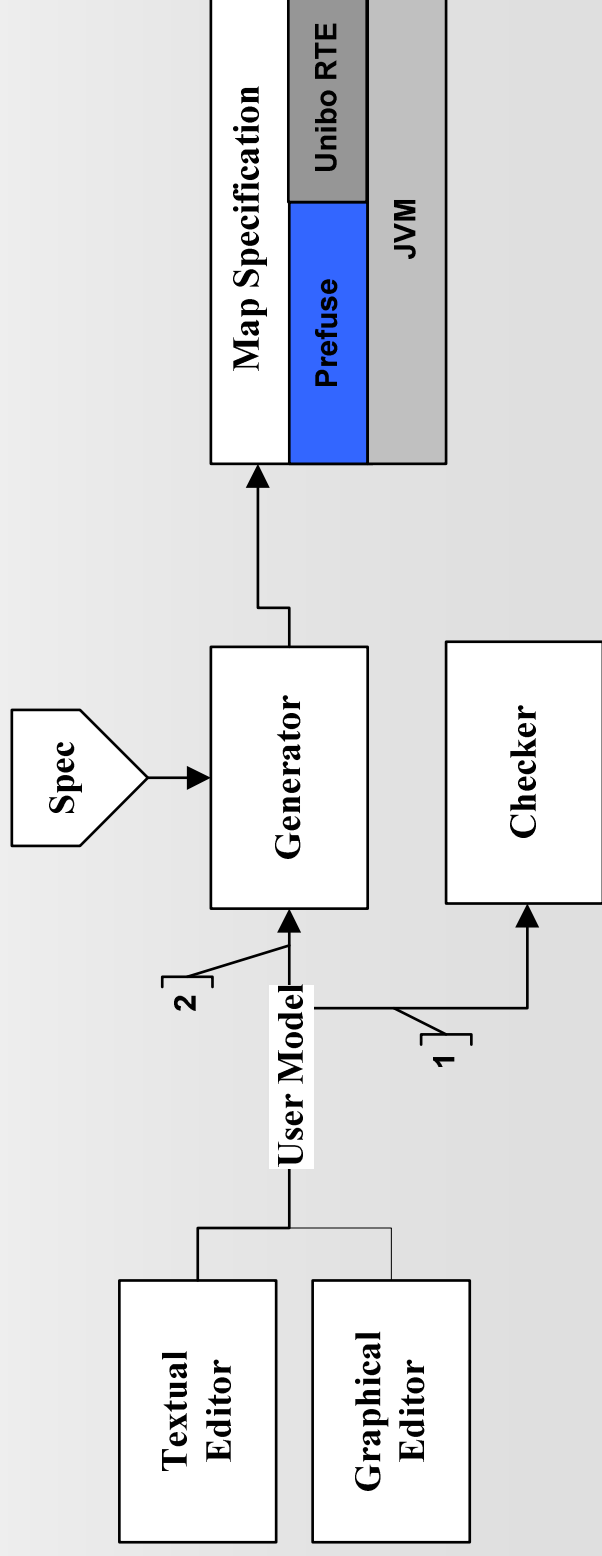
- For each *attribute* a keyword token (which is exploited by the editor to suggest text completion) must be defined through a standard assignment.
- For each *composition* you must define a += assignment in which the Rpart is the rule associated to the related metaclass.
- For each *association* (cardinality 1) you must define a standard assignment that makes reference to the rule associated to the related metaclass.
- For each *aggregation* you must define a += assignment in which the Rpart is a cross-reference to the rule associated to the related metaclass D (denoted with[D]).
- For each *generalization* you must define an abstract rule made of a sequence of alternatives that reflects the types hierarchy;
- For each *enumeration* type, you must define an enumeration rule Enum composed of a sequence of choices with all literals as terminal symbols.

Outline

- Background
- Motivations
- e-Content Model
- Editor and Grammar Construction
- **Generation Process**
- Map Visualization
- Conclusions

Generation System

- The **User Model** is the input to
 - the **checker** in order to validate the model based on OCL-like expression (`xextend`)
 - the **generator** in order to produce the run-time support for the chosen platform and a set of graphical maps

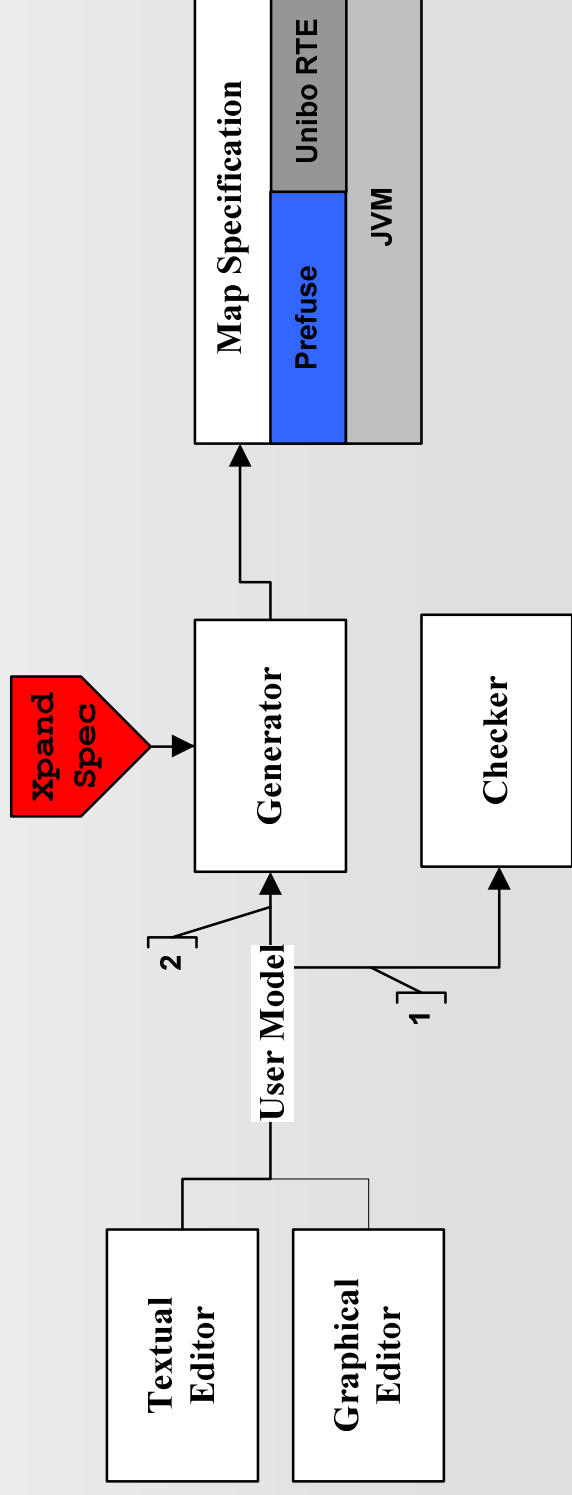


Reference Platform

- Reference platform is Prefuse (<http://prefuse.org/>)
 - It is an open source project to simplify the creation of custom visualization
 - It is a highly reconfigurable rendering engine, with abstraction to
 - filter data
 - make actions over contents
 - assign layout

Automatic Code Generation

- To overcome the gap between the model (course) and the platform (prefuse)
 - we use automatic code generation of oAW based on Xpand, which
 - is a declarative language
 - is based on DEFINE blocks associated to the metamodel



Template (Xpand)

```
«IMPORT courseds1»
«DEFINE main FOR Unit»
«FILE title+".xml"»
<?xml version="1.0" encoding="UTF-8"?>
<!-- Radial graph -->
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
<graph edgedefault="directed">
  <!-- data schema -->
  <key id="name" for="node" attr.name="name" attr.type="string"/>
  <key id="type" for="edge" attr.name="type" attr.type="string"/>
  <key id="diff" for="node" attr.name="diff" attr.type="Real"/>
  <key id="audi" for="node" attr.name="audi" attr.type="string"/>
<!-- nodes -->
«EXPAND node FOREACH this.contents»
<!-- edges -->
«EXPAND edge FOREACH this.relations»
</graph>
</graphml>
«ENDFILE»
«ENDDFINE» ...
```

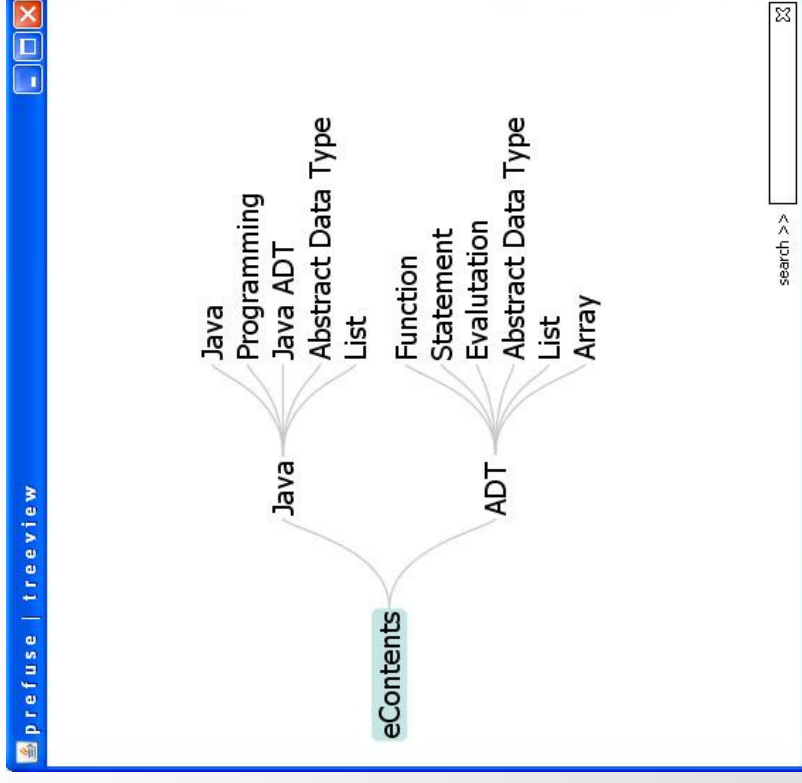
**Declarative specification
of the rules (like XSLT)**

Outline

- Background
- Motivations
- e-Content Model
- Editor and Grammar Construction
- Generation Process
- **Map Visualization**
- Conclusions

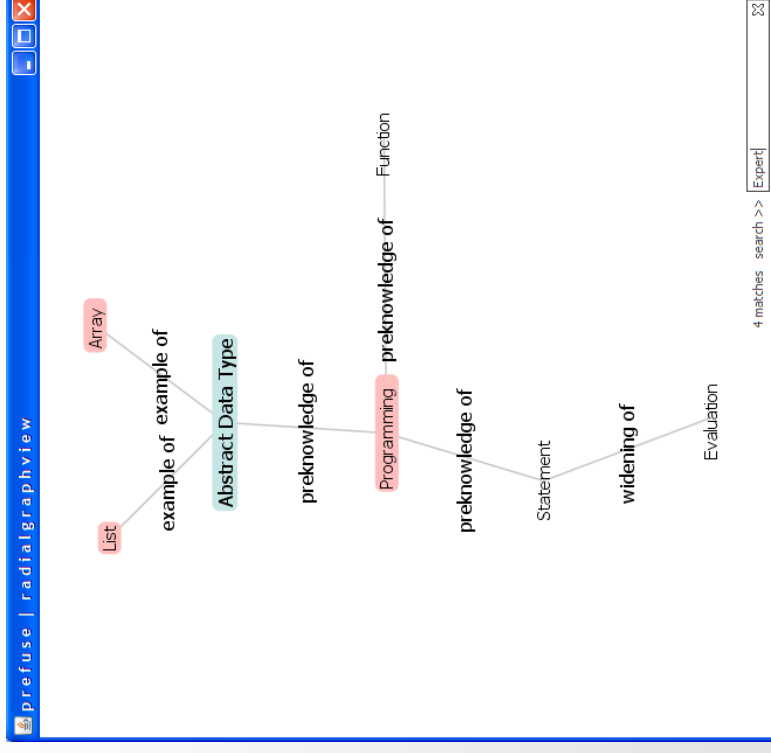
The Map: *tree view*

- Tree view: is a map style organized like classical hierarchical index
 - contents are dynamically opened and closed according to the user's requests



The Map: *radial graph*

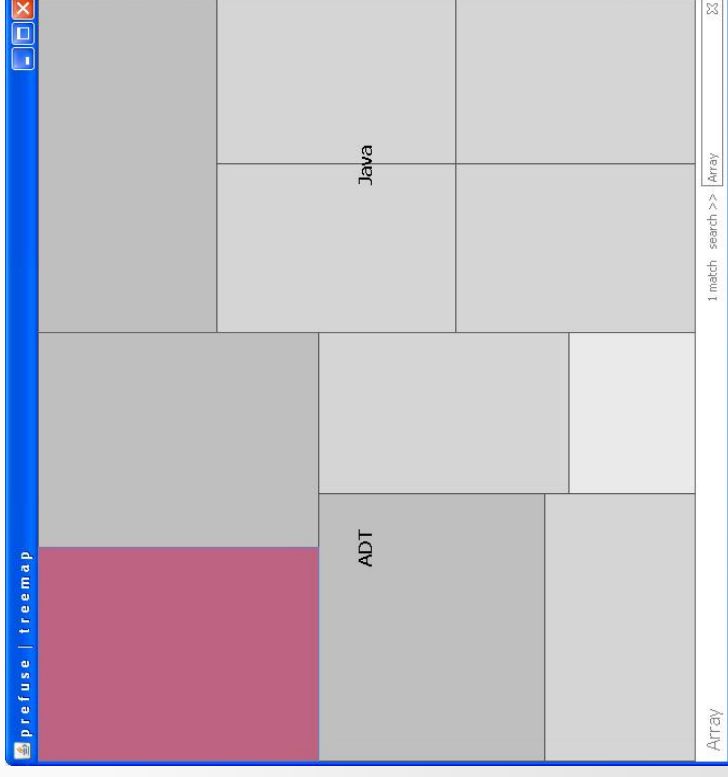
- Radial graph
 - highlights hierarchies or semantic networks of information centered on a specific item
 - produces a reconfiguration of the map by focusing on the selected item
 - promotes a qualitative elaboration of the information



Centered on the `Programming` item that highlights the items written with expert metadata

The Map: *tree base*

- Tree base map is a space-filling approach
 - to display data as a set of nested rectangles
 - to organize and represent complex structure of information
 - with several connections, categories and users
 - to recognize patterns or content characteristics in few seconds
 - to organize the views keeping into account metadata values



Dimension and colour of rectangles are related to difficulty metadata

Conclusions I

- Definition of a domain specific language for the description of e-Contents with textual syntax
- Definition of a specification for automatic generation of “back-end” equivalent respect to the execution platform based on abstract syntax of the language (`Ecore`, `EMF`, `MOF`)
 - Thanks to Eclipse and oAW!

Conclusions II

- Thanks to Eclipse and oAW!
 - we focus the attention on business logic
 - we have a generative DSL process and automatic code generation that could be successfully applied to other environments

Model-driven Generation of Graphical Maps for e-Contents



Antonio Natali¹, Enrico Oliva¹, Cristina Bonanni²
¹*Alma Mater Studiorum - Università di Bologna, Cesena, Italy*
{antonio.natali, enrico.oliva}@unibo.it
²*IBM Italia, Tivoli Software, Roma, Italy*
{cristina.bonanni}@it.ibm.com

OCL expression in Xtend

- We exploit Xtend
 - to overcome the limit of meta model expressivity
 - to express and verify model constrains

- Example: *"an instance name must begin with lower case"*

```
Context Instance Error ="instance error"  
: name.toFirstLower () ==name;
```