

Rhizome

A Feature Modeling and Generation Platform for Software Product Line

Ph.D. Defense Talk

Guozheng Ge

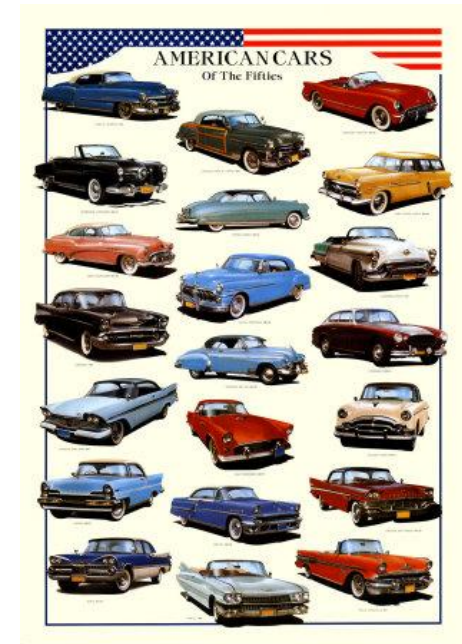
Department of Computer Science
University of California, Santa Cruz

Product Line Concept



Blackberry

Mac



Cars

Product Line: a group of products that share a common development platform and varies by actual feature configuration.


Product Line is Good

- Higher productivity and shorter time-to-market
 - Effective code reuse at platform and component levels
 - Reduced product development cost
- ...
- So, people applied this idea to **Software Development**

Software Product Line Example

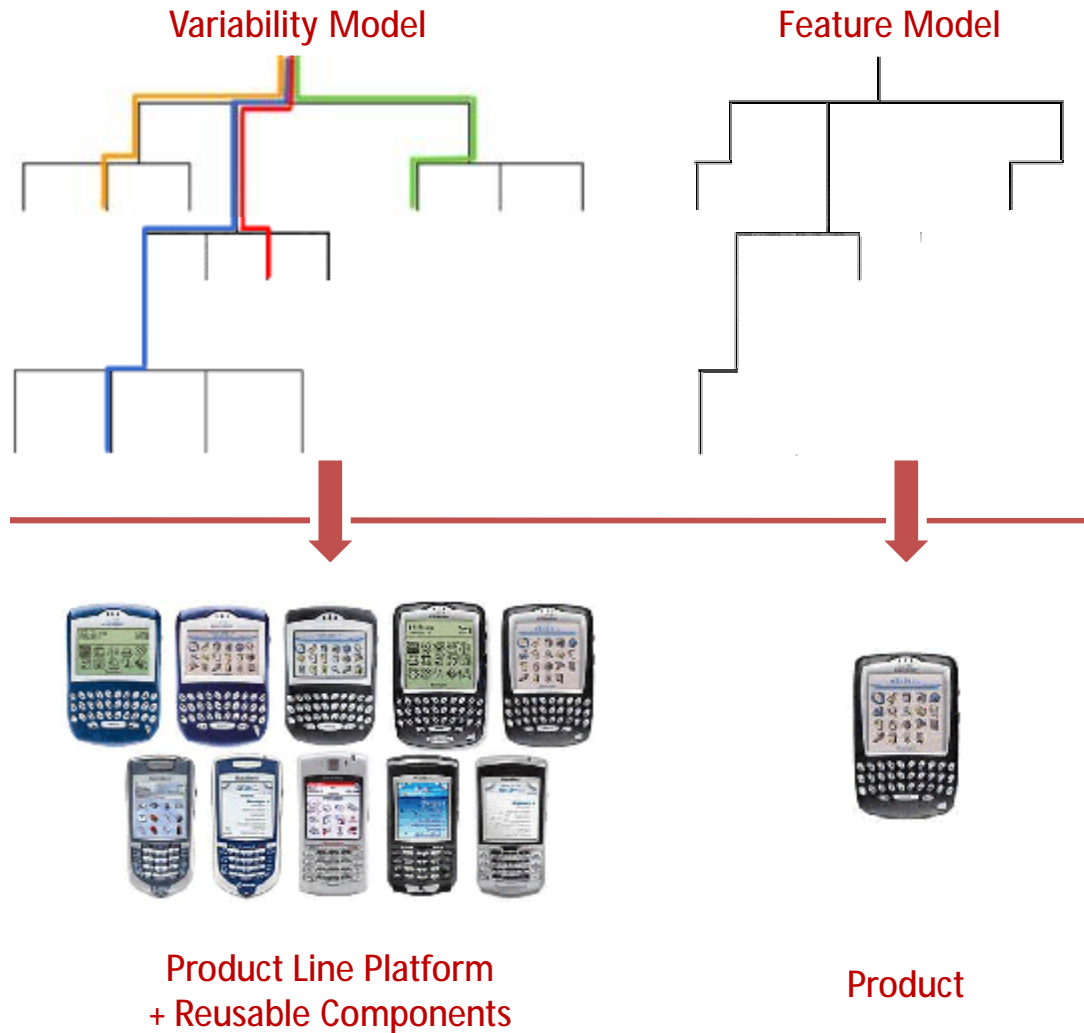
Google Earth

Product Variants: products with different feature configuration and functionality

	Free Edition	Plus Edition	Pro Edition
Zoom in on locations	✓	✓	✓
3D terrain and buildings	✓	✓	✓
Point of interest	✓	✓	✓
High resolution printing		✓	✓
GPS data import		✓	✓
Address data import using CSV format		✓	✓
Area measurement tools			✓
Movies maker			✓
GIS data import for SHP files, GeoTiffs			✓

Features: structural or behavioral system qualities that provide usability to users

Terminology



Design Level

- **Design Space:** conceptual
- **Variability Model:** collection of all possible features design choices
- **Feature Model:** one fixed set of feature design choices

Implementation Level

- **Product or Product Variant**
- **Product Line Platform:** basic system level services: file system, network I/Os, repository CRUD
- **Reusable Components:** domain-specific components to implement features

Software Product Lines are good, but they could be better...

- Not easy to learn and use
 - Design and implementation knowledge still buried in source code and developers' minds
 - Learning curve to master a Software Product Line Development platform and component programming
 - Difficult to understand how and where to make changes with existing products
- Need better tool support
 - Lack of automated tool support for feature model design and product code implementation
 - Little tool and language support to resolve dependencies among features
 - Manual work to build component code and glue code
- Missing link between semantic level design and code level implementation
 - No support for change impact analysis

Proposed Solution

- Separation of developer roles
 - Feature model designers
 - Platform developers
- Solidify knowledge for easy reuse
 - Store design knowledge in variability model
 - Store implementation knowledge using code generation patterns and code templates
- Automated feature-to-code generation tools
- Connecting high level feature model design with low level code generation in modeling language

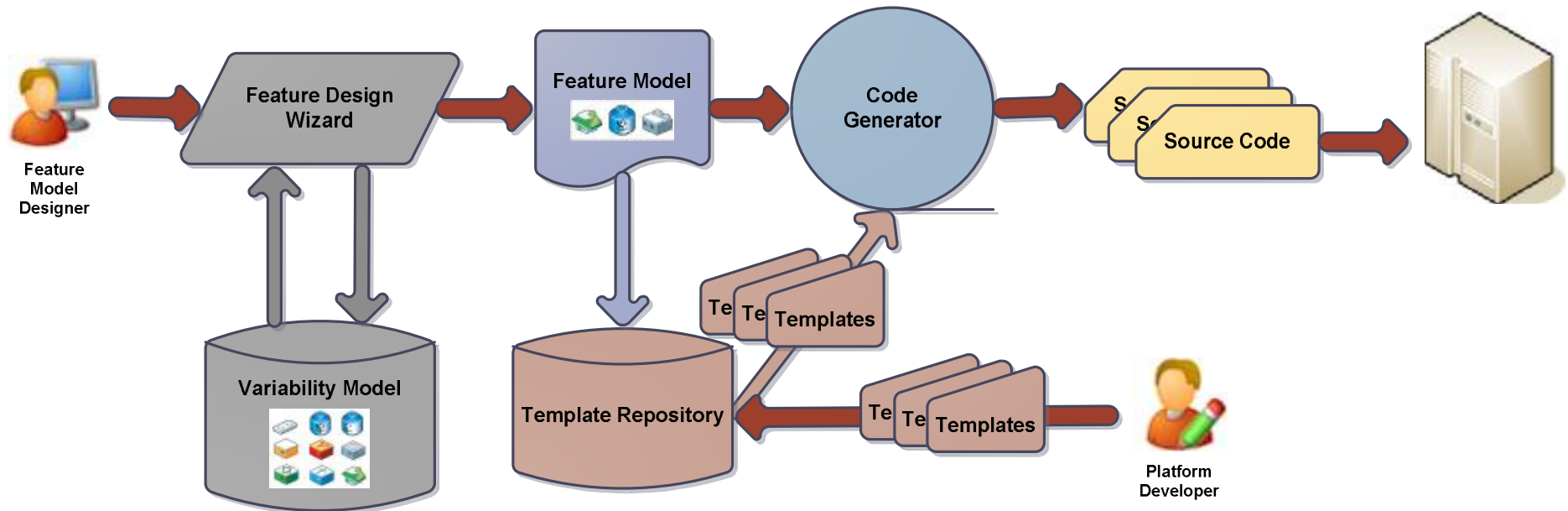
Approaches and Contributions

- A platform for high level software feature design and code generation (Rhizome)
 - A feature modeling language (VarML)
 - Other feature modeling languages exist, ours happens to be a particularly good one
 - A template-based code generation language and code generation engine that interprets this language
 - Permits direct generation of code from a feature model
 - This is the most substantial novel contribution of this dissertation
- Identification of key code generation patterns
- Proof-of-concept implementation of Rhizome
 - Online exam systems in Web application domain

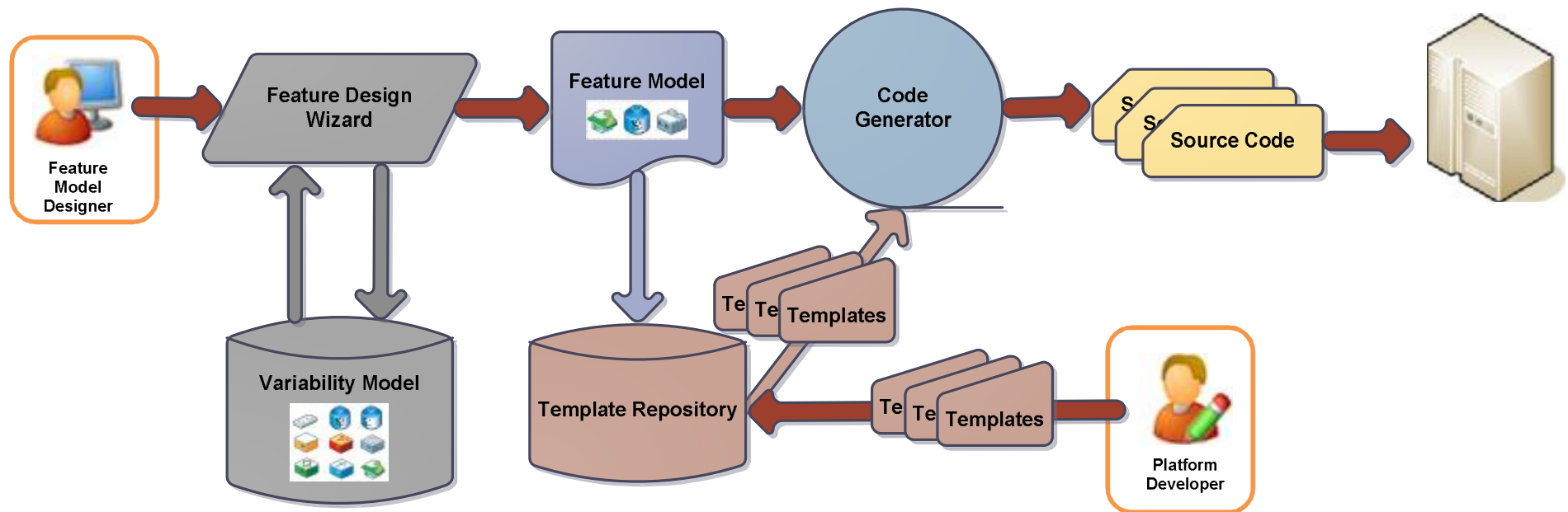
Presentation Outline

- Rhizome Workflow Overview
 - How users work with Rhizome
 - Variability model and feature design wizard
 - Feature modeling language (VarML)
 - How code generation works using template-based Code Generation
 - Template language
 - Code generation process
- Proof-of-concept System Generation
- Conclusion and Future Work
- Related Work

Rhizome Workflow Overview



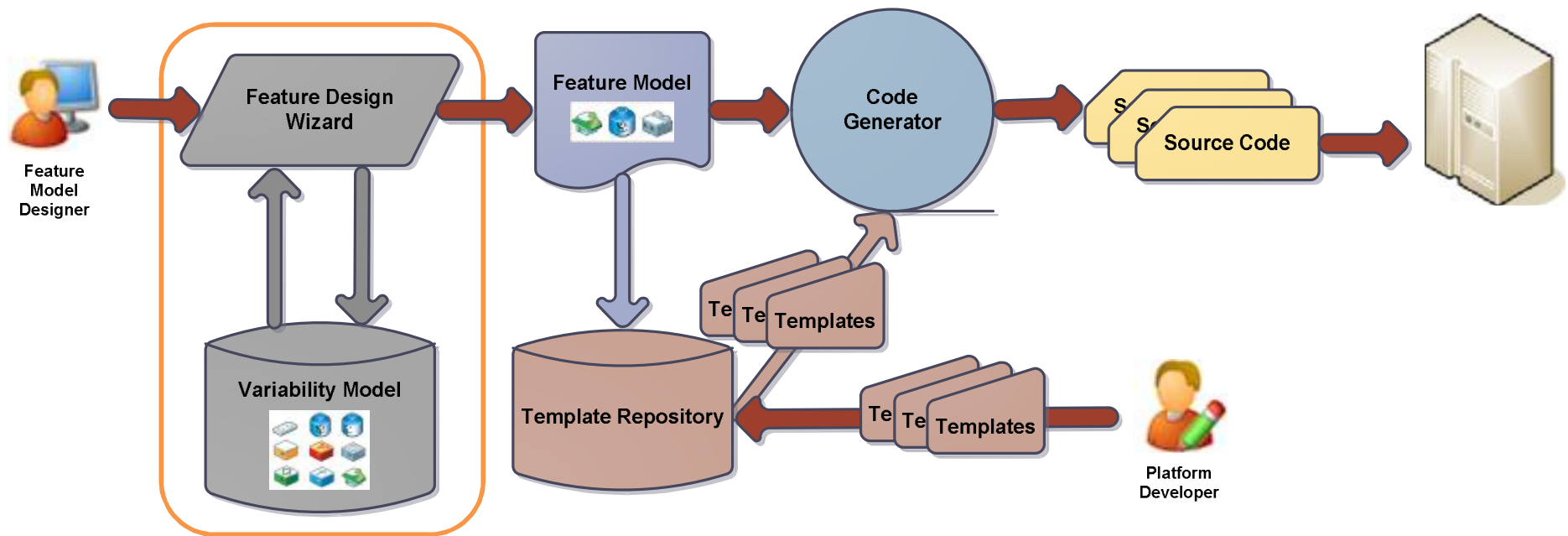
Rhizome Users



Feature Model Designer: select features, define feature structures and behaviors

Platform Developer: implement platform and component code, write code templates, expose template parameters, associate code generation units (CGU) with design choices

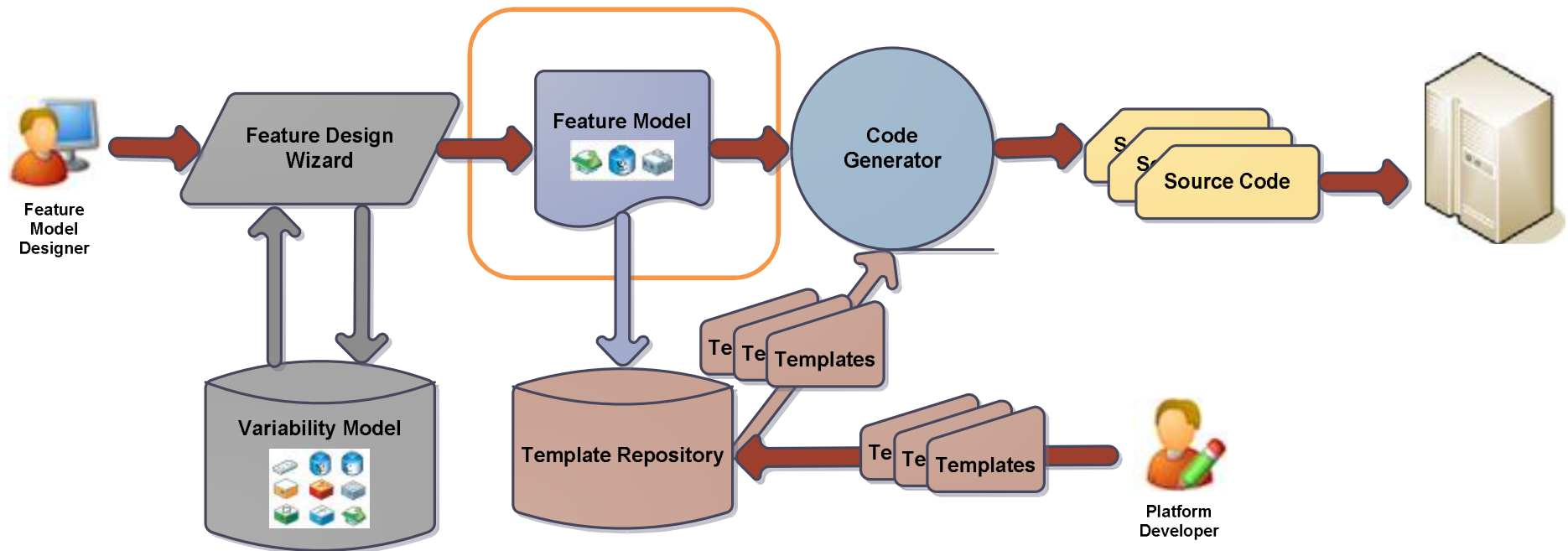
Variability Model and Feature Design Wizard



Variability Model: all features and various feature design choices, dependency and constraints, association with code generation units

Feature Design Wizard: graphically compose a feature model, provide design hints, automatically check dependencies, automatically resolve CGU association

Feature Model



Feature Modeling Language VarML:

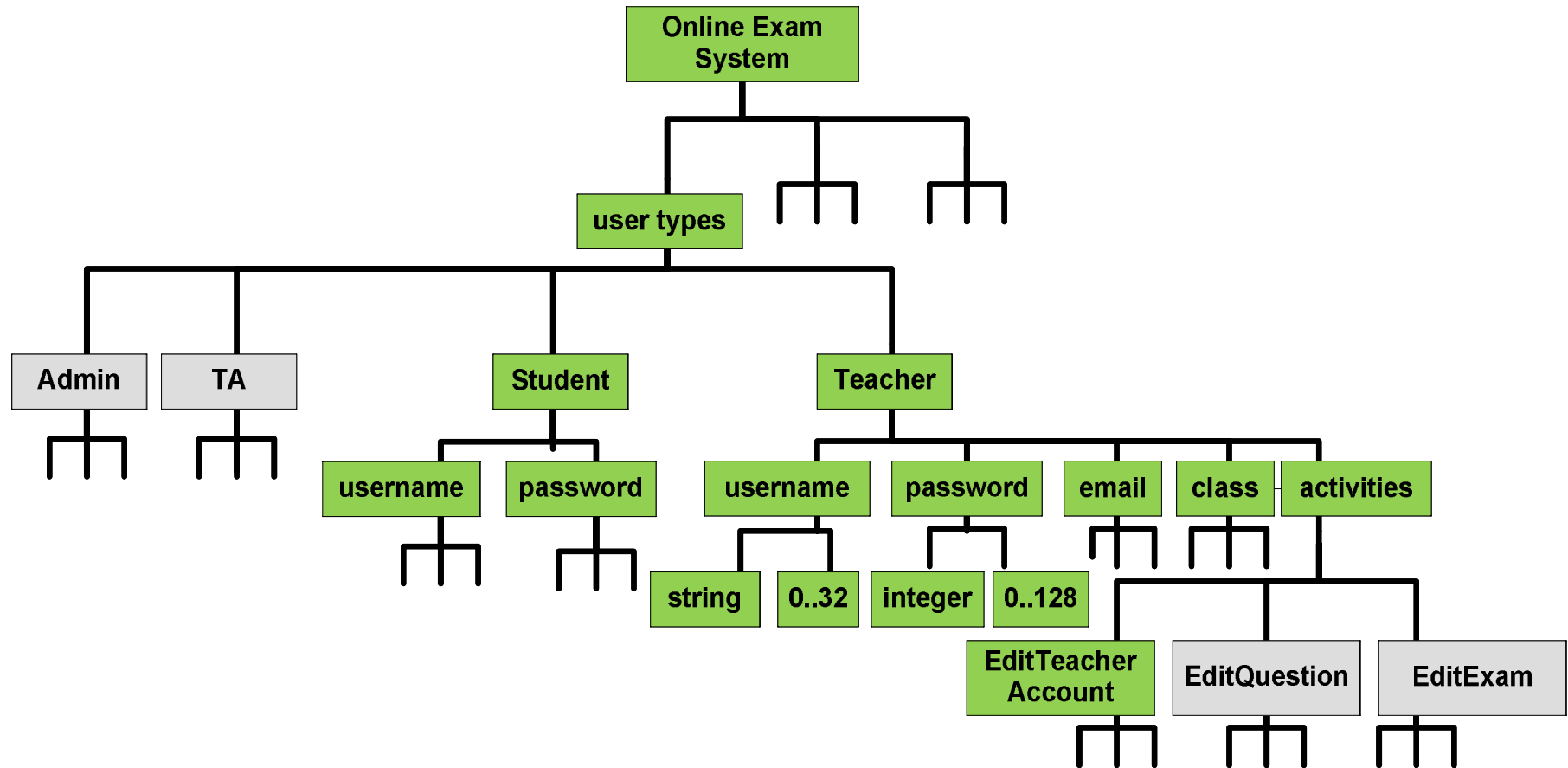
- defines feature structure
- design choice structure
- code generation instructions
- trace information to code generation

Example: Online Exam System

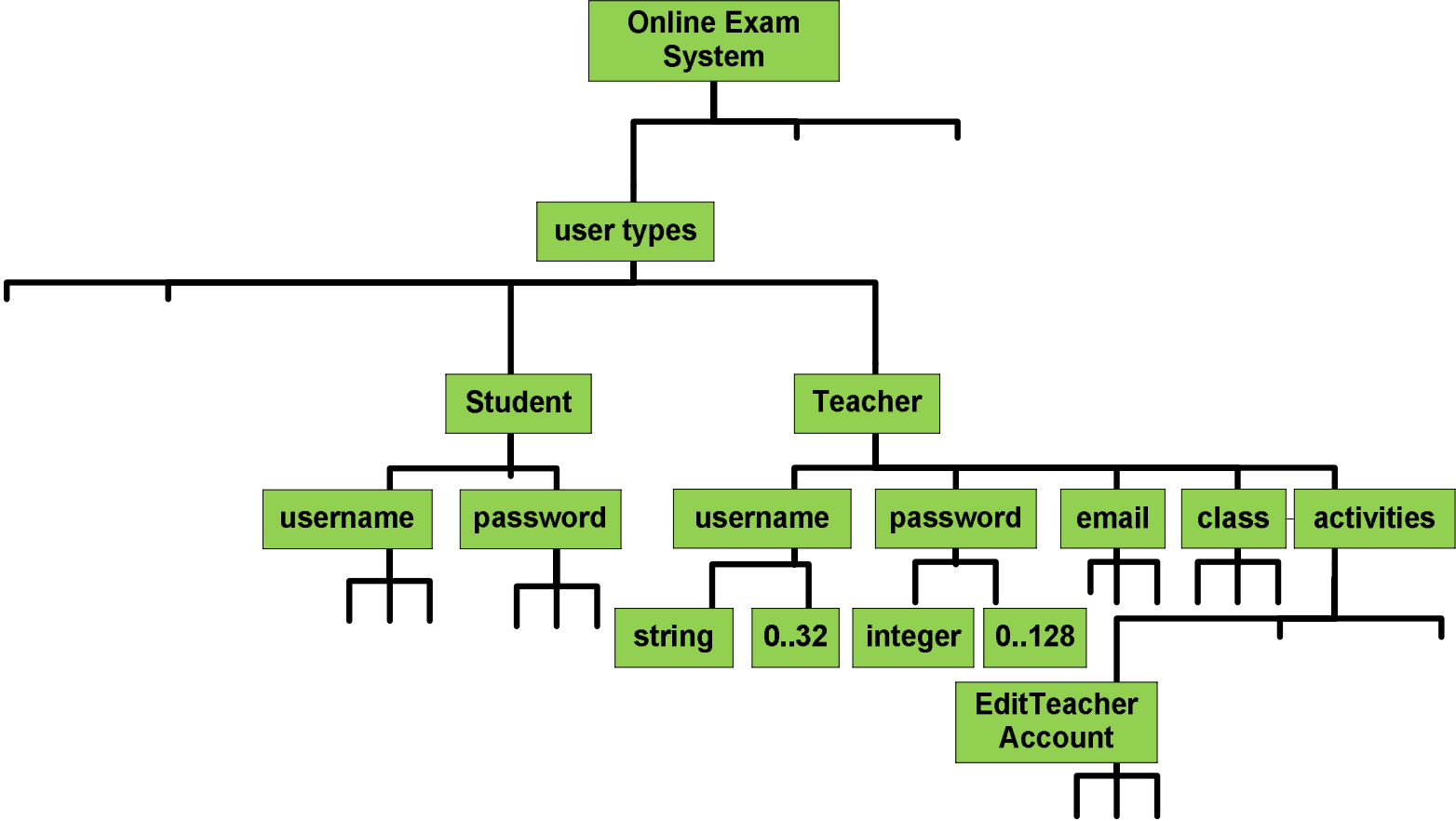


- Multiple user types: student, teacher, admin, TA, ...
- Multiple question types: multiple choice, true or false, short answer, essay, ...
- User activities: register new user, edit account info, log in/out
 - Teacher, TA: manage students, create new questions, edit existing questions, adding questions to create new exams, edit exams, preview exams, release/close exams, grade exams
 - Student: take open exams, view exam grades
 - Admin: all above plus teacher, TA management

Variability Model for Online Exam System



One Possible Feature Model for Online Exam System



Feature Modeling Language Requirements

A Sample Feature Model

- Two user types:
 - teacher and student
- Both types share some properties:
 - username, password
- Additional properties for teacher:
 - email, class
- Basic data types:
 - integer, string
- Data value length limit:
 - 0..32, 0..128
- Teacher activities:
 - EditTeacherAccount

Language Requirements

a feature structure to hold sub-features

subclass-superclass-style feature
structure for sharing

basic enumeration values as choices

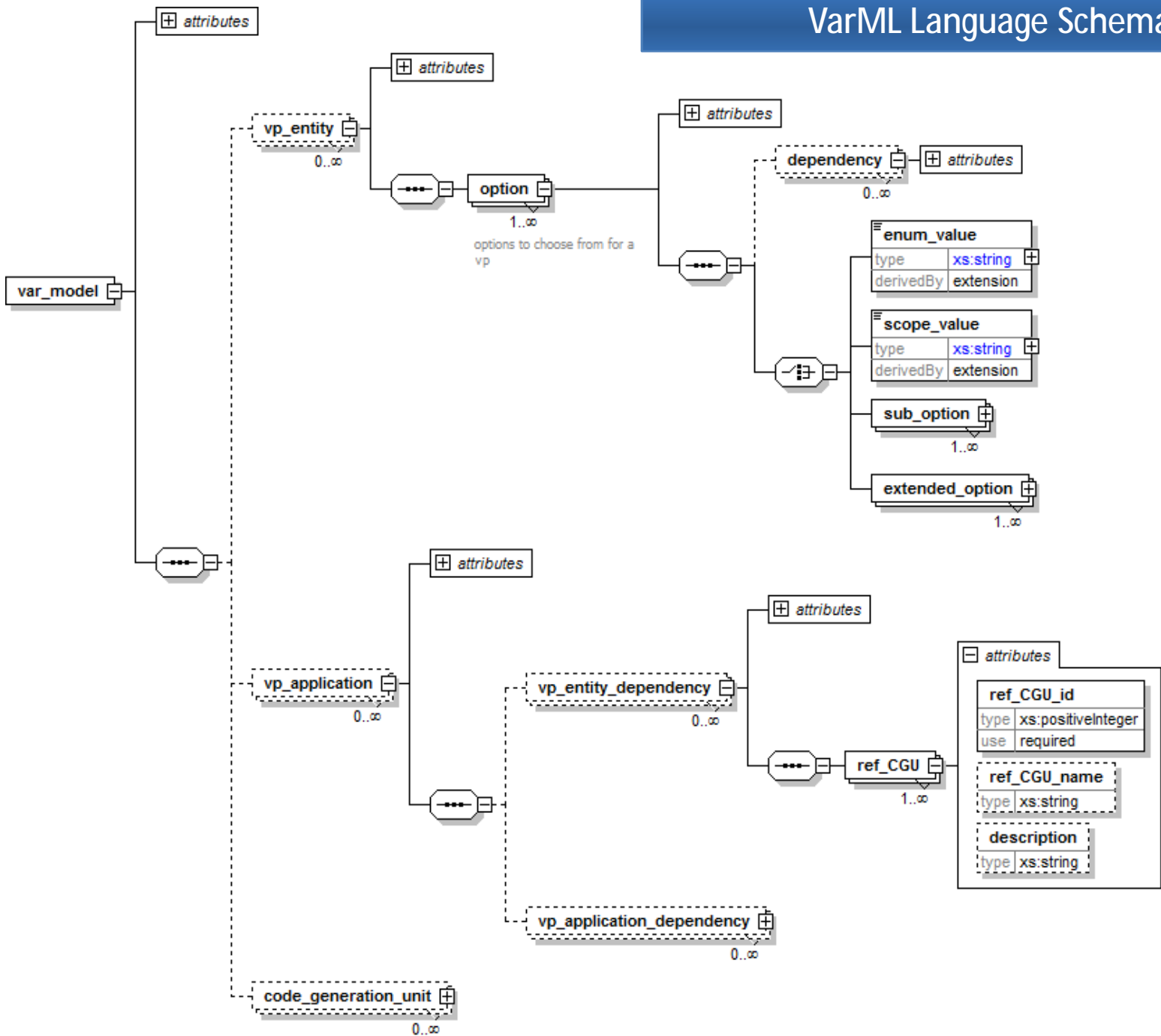
basic scope values as choices

separate data features from behavioral features

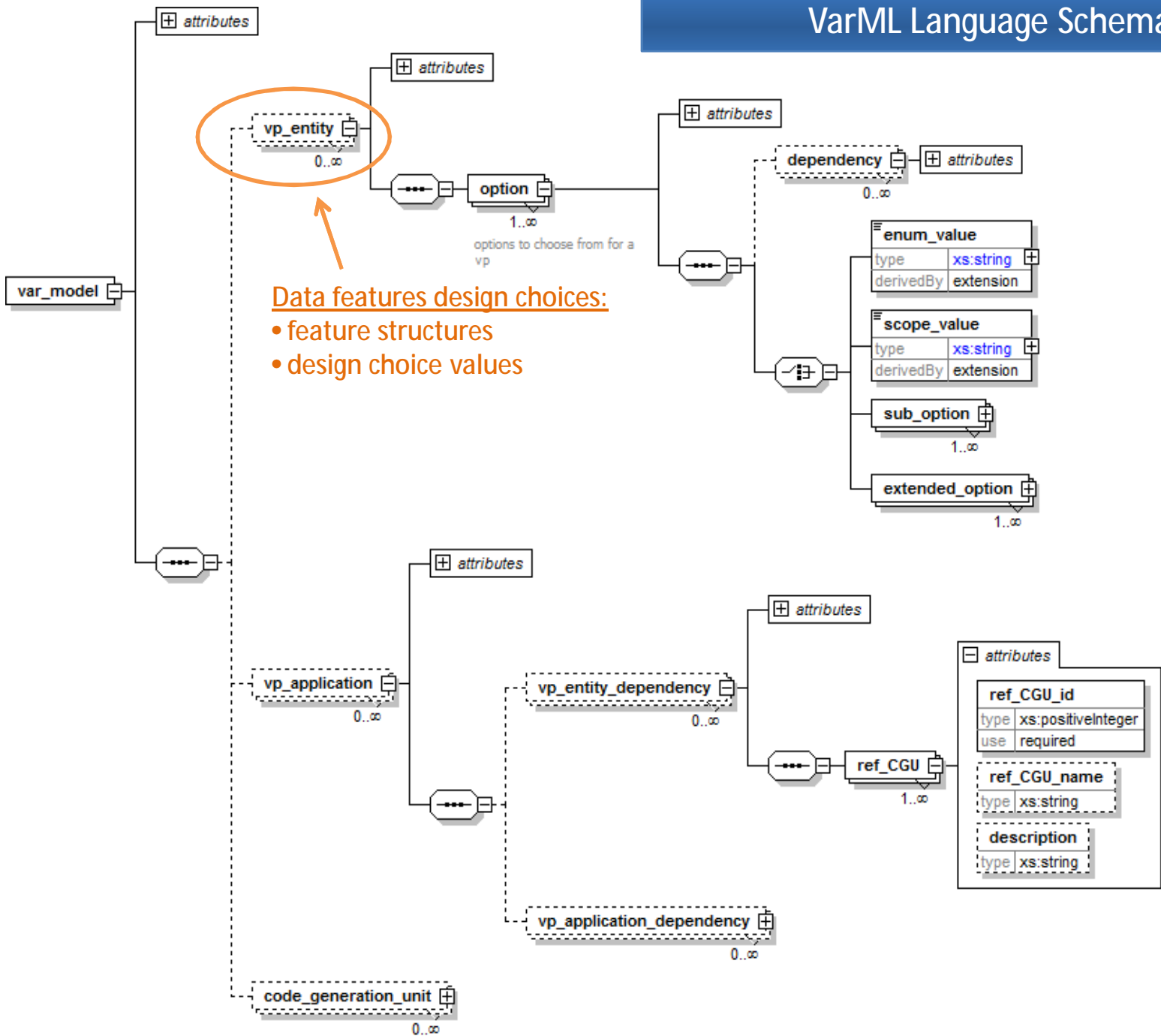
Feature Modeling Language: VarML

- Three major elements:
 - **vp_entity** for data features and their design choices
 - **vp_application** for behavioral features and design choices
 - dependency relationship with vp_entity and other vp_application elements
 - references to CGUs
 - **code_generation_unit** for instructions to do code generation
- Each feature model is an XML document conforming to VarML language schema

VarML Language Schema



VarML Language Schema



Data features design choices:

- feature structures
- design choice values

```
<vp_entity vp_id="1" vp_name="basic_value_types" description="different
value types used for object properties, we are using Java types here">
  <option op_id="1" description="Integer type">
    <enum_value type="String">java.lang.Integer</enum_value>
  </option>
  <option op_id="2" description="String type">
    <enum_value type="String">java.lang.String</enum_value>
  </option>
</vp_entity>
```

enum_value example: basic types of Integer and String

```
<vp_entity vp_id="2" vp_name="value_length_limit" description="length
limit for string parameter value">
  <option op_id="1" description="no more than 16">
    <scope_value type="String">0..16</scope_value>
  </option>
  <option op_id="2" description="exactly 12 for phone number">
    <scope_value type="String">12..12</scope_value>
  </option>
</vp_entity>
```

scope_value example: length limit of 0..16 and 12..12

```
<vp_entity vp_id="100" vp_name="username" description="username property">
  <option op_id="1" ...> ...
</vp_entity>
```

```
<vp_entity vp_id="108" vp_name="password" description="password property">
  <option op_id="1" ...> ...
</vp_entity>
```

```
<vp_entity vp_id="41" vp_name="User_props" description="properties for a general user type">
  <option op_id="2" op_name="commercial_system_creditcard" description="commercial system
user has extra credit card information">
```

```
  <sub_option vp_entity_id="100" vp_entity_option_id="1"/>
```

```
  <sub_option vp_entity_id="108" vp_entity_option_id="1"/>
```

```
</option>
```

```
</vp_entity>
```

Feature sub-structure example: a user has username and password properties

```
<vp_entity vp_id="42" vp_name="Teacher_props" description="properties for teacher type">
  <option op_id="2" op_name="commercial_system_creditcard_extend" description="extending the
commercial_system_creditcard user entity by adding department, office and phonenum properties">
```

```
  <extended_option parent_vp_entity_id="41" parent_vp_option_id="2">
```

```
    <additional_sub_option vp_entity_id="67" vp_entity_option_id="1"/>
```

```
    <additional_sub_option vp_entity_id="71" vp_entity_option_id="1"/>
```

```
  </extended_option>
```

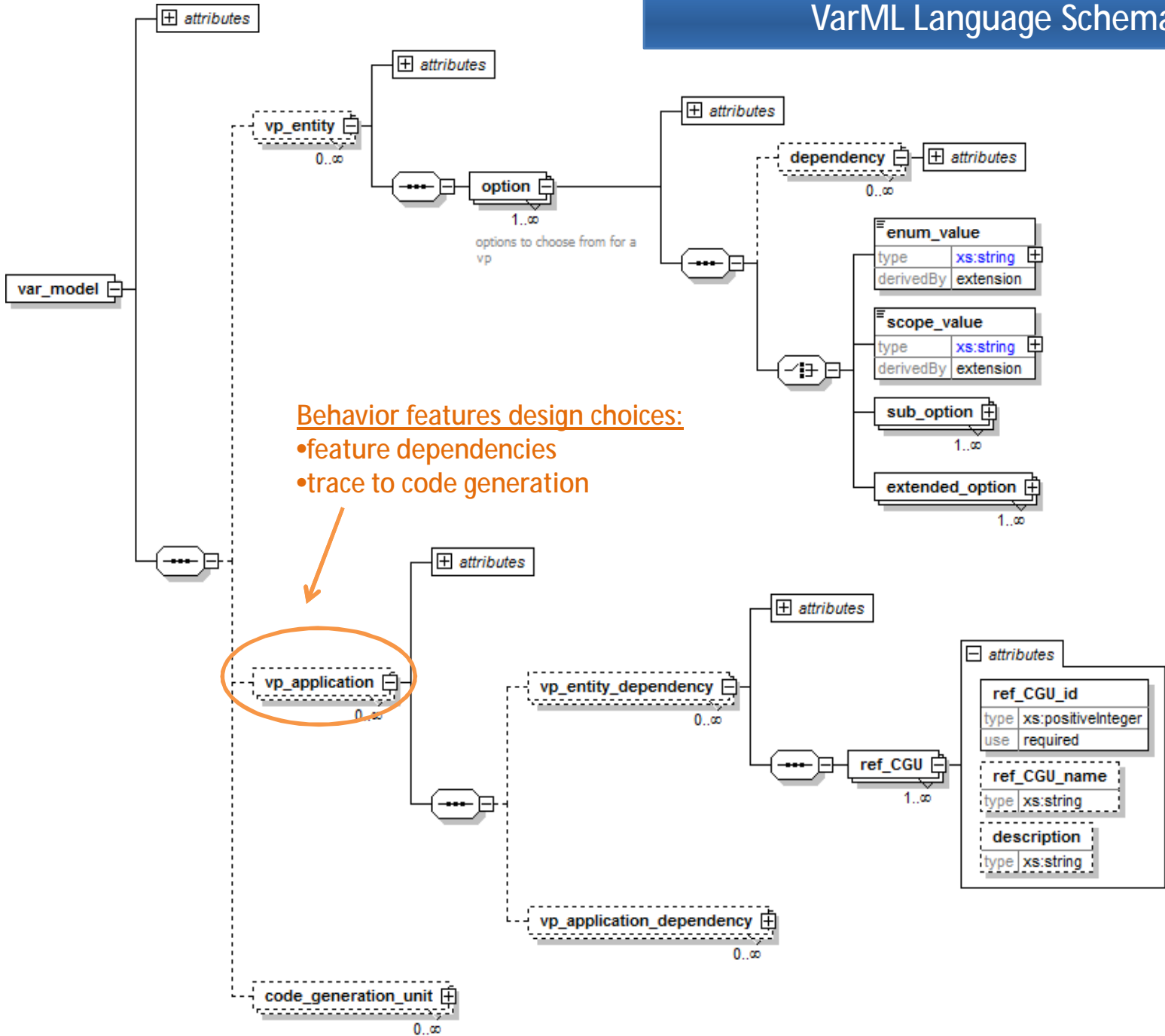
```
</option>
```

```
</vp_entity>
```

Feature extension example: a teacher extends user and adds more properties

...

VarML Language Schema



```
<vp_application vp_id="2100" vp_name="EditTeacherAccount" description="input property values to create a new Student or edit an
existing Student">
  <vp_entity_dependency vp_entity_id="42" vp_entity_option_id="2" description="for each student property, create its widget on the
edit page">
    <ref_CGU ref_CGU_id="129" ref_CGU_name="EditUserAccountPage_java_string_swap" description="string swap for edit
user account java page"/>
  </vp_entity_dependency>
  <vp_entity_dependency vp_entity_id="42" vp_entity_option_id="2" description="for each student property, create its widget on the
edit page">
    <ref_CGU ref_CGU_id="130" ref_CGU_name="EditUserAccountPage_java_tag_expansion" description="expand tags for
edit user account page"/>
  </vp_entity_dependency>
  <vp_entity_dependency vp_entity_id="42" vp_entity_option_id="2" description="for each student property, create its widget on the
edit page">
    </vp_entity_dependency>
</vp_application>
```

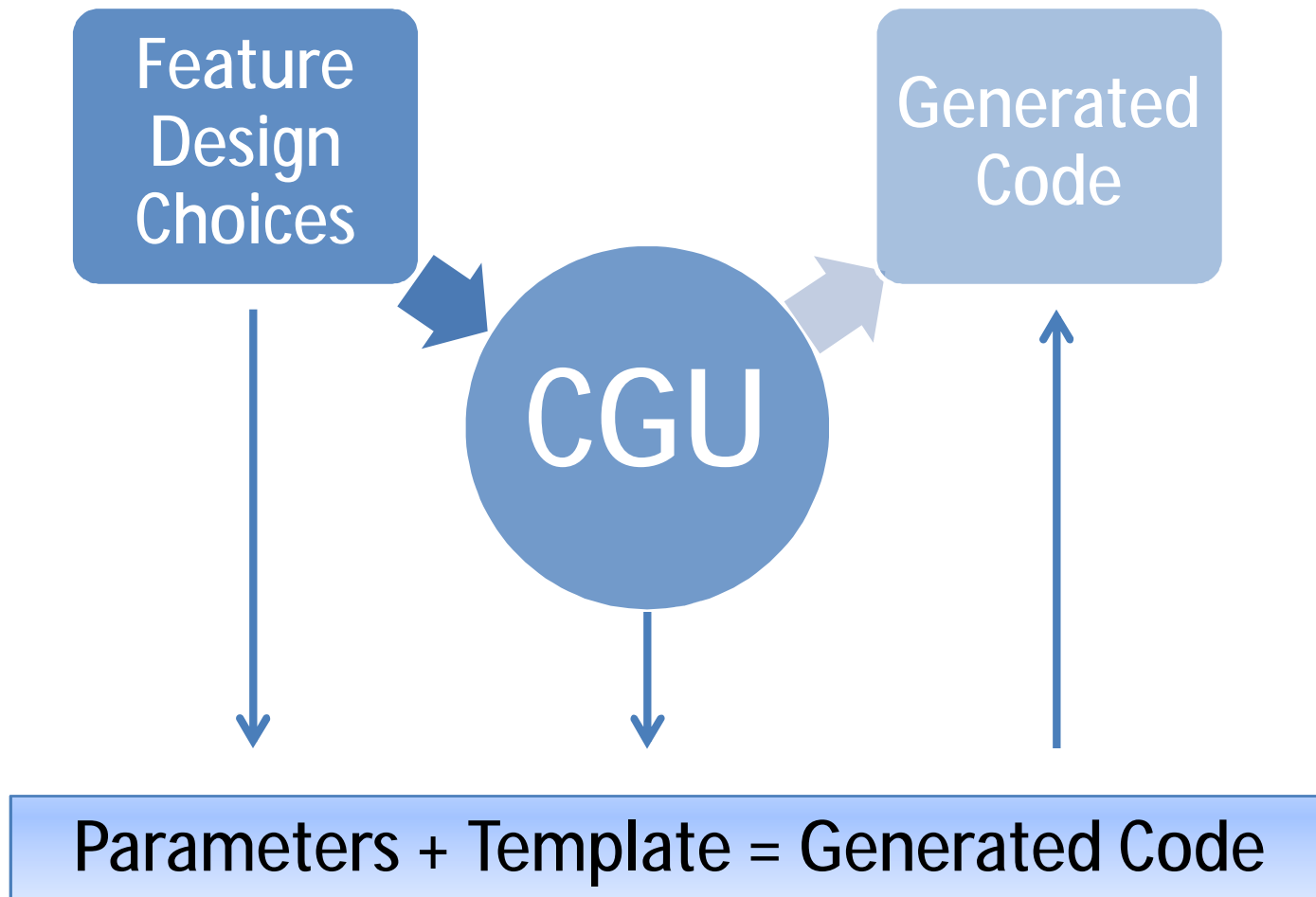
Behavioral feature example: EditTeacherAccount activity

Recap

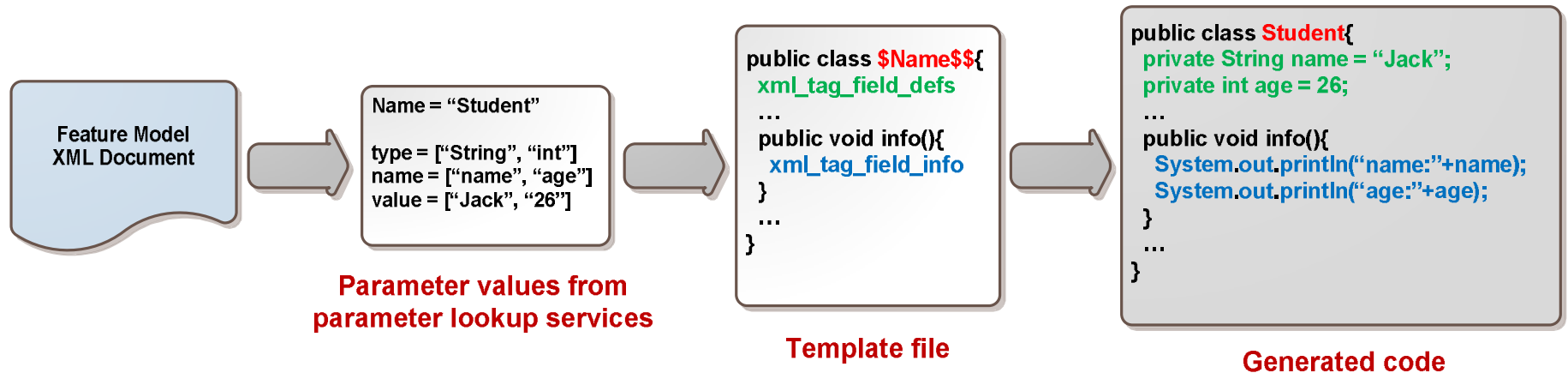
- A feature model represents the choices made (out of a larger variability space) for one specific product instance
 - VarML modeling language represents these feature choices, and connections to code generation
- But how to generate code from here?

Code Generation Unit (CGU) Element

- Bridge between Feature Model and Code Generator



Template-based Code Generation

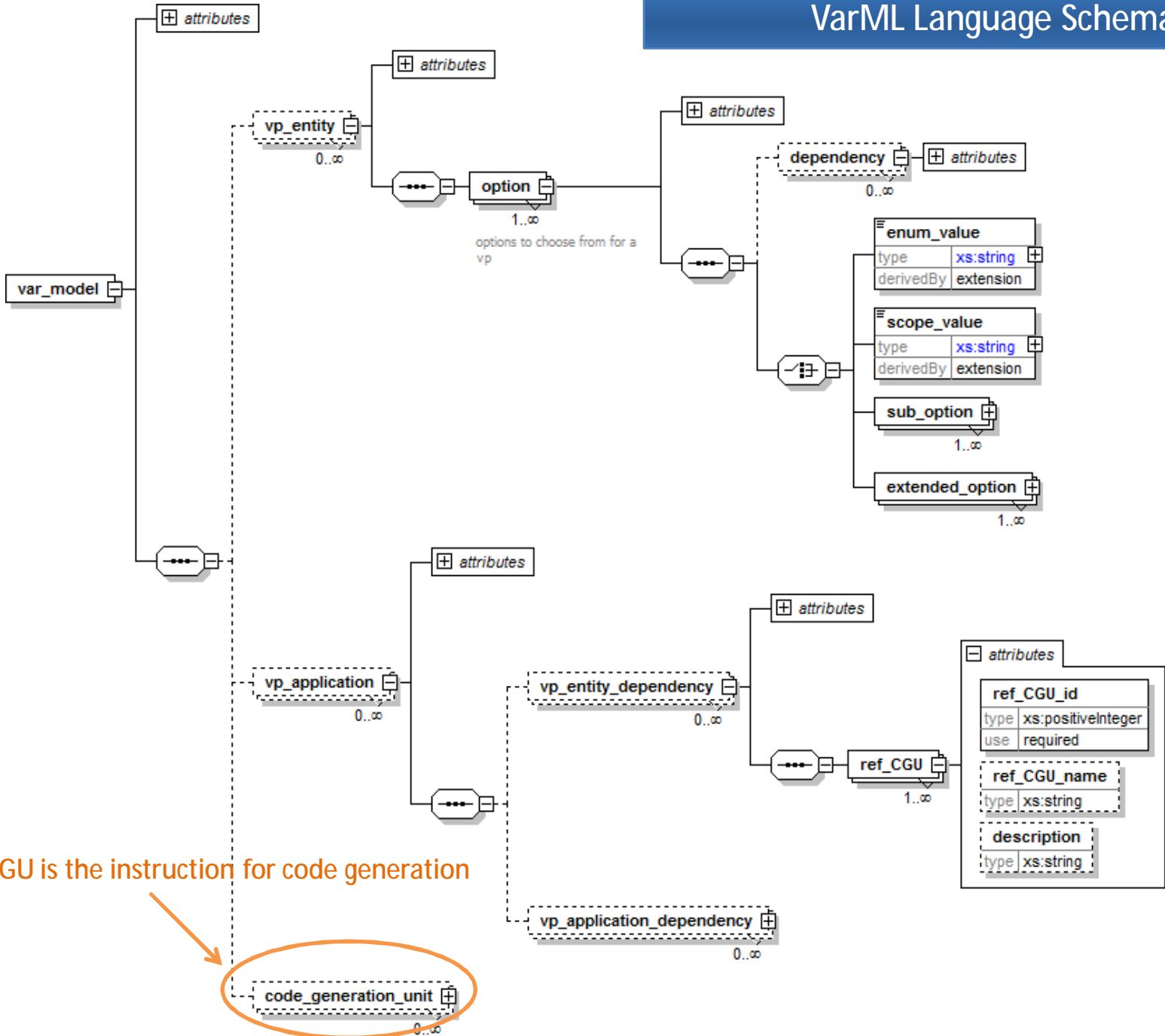


It's still like magic, how does the generator know:

- Where and how to get parameter values?
- How to use parameter values in the template to create code?

The answer is the 3rd element in VarML language:
code_generation_unit

VarML Language Schema



CGU is the instruction for code generation

CGU Types Overview

Inter-file Level CGUs:

- file_copy
- snippet_join

Inner-file Level CGUs:

- string_swap
- tag_expansion
 - COPY
 - LIST_GEN
 - LIST_GEN_CONDITIONAL
 - LIST_GEN_BY_INDEX
 - COUNT
 - LIST_GEN_COUNTER

Tag_expansion CGU has many tag types, each tag type is defined using our **Template Language**

Inter-file CGU Type I: file_copy

file_copy: copy exactly the same content from template file to create source code



Exam.hbm.xml.tpl



Exam.hbm.xml

Case 1: one template generating one source code file



$\$EntityName\$.hbm.xml.tpl$



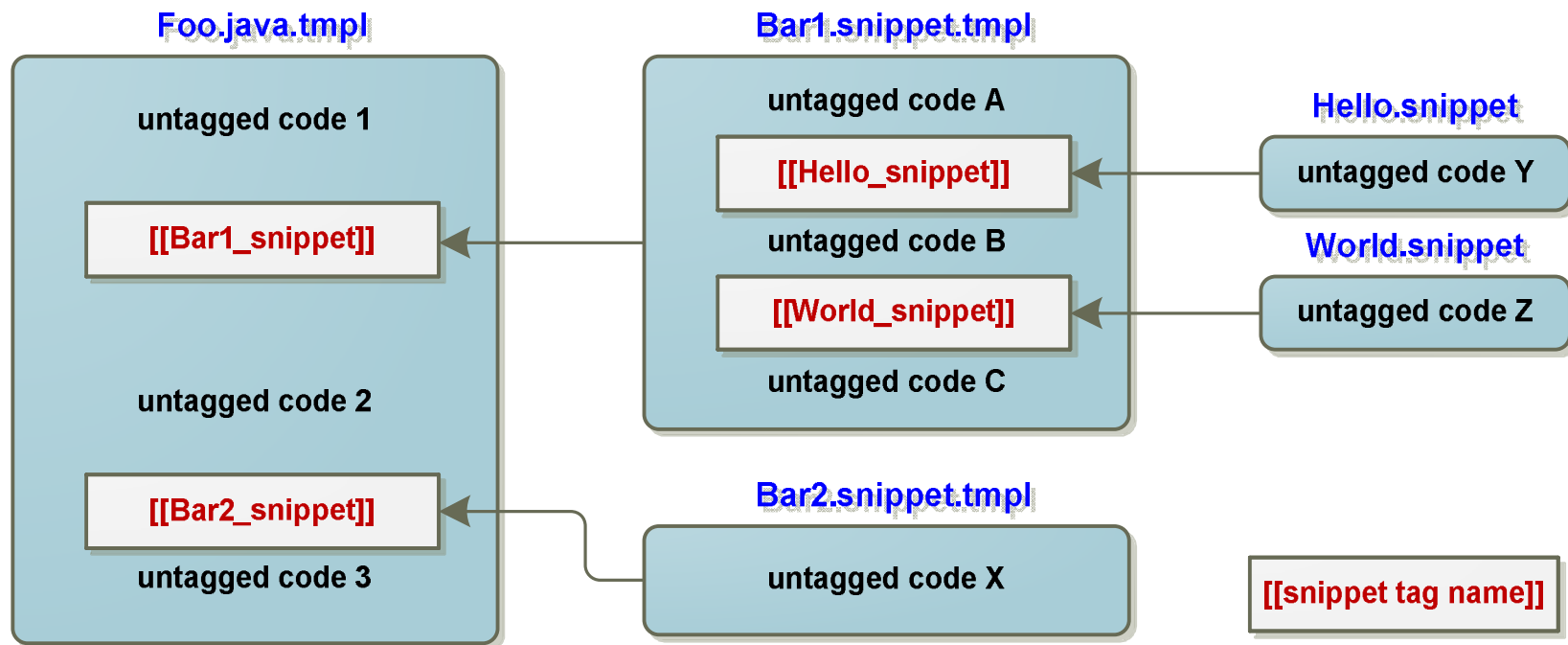
Teacher.hbm.xml
Student.hbm.xml

Case 2: one template generating multiple source code files

Inter-file CGU Type II: snippet_join

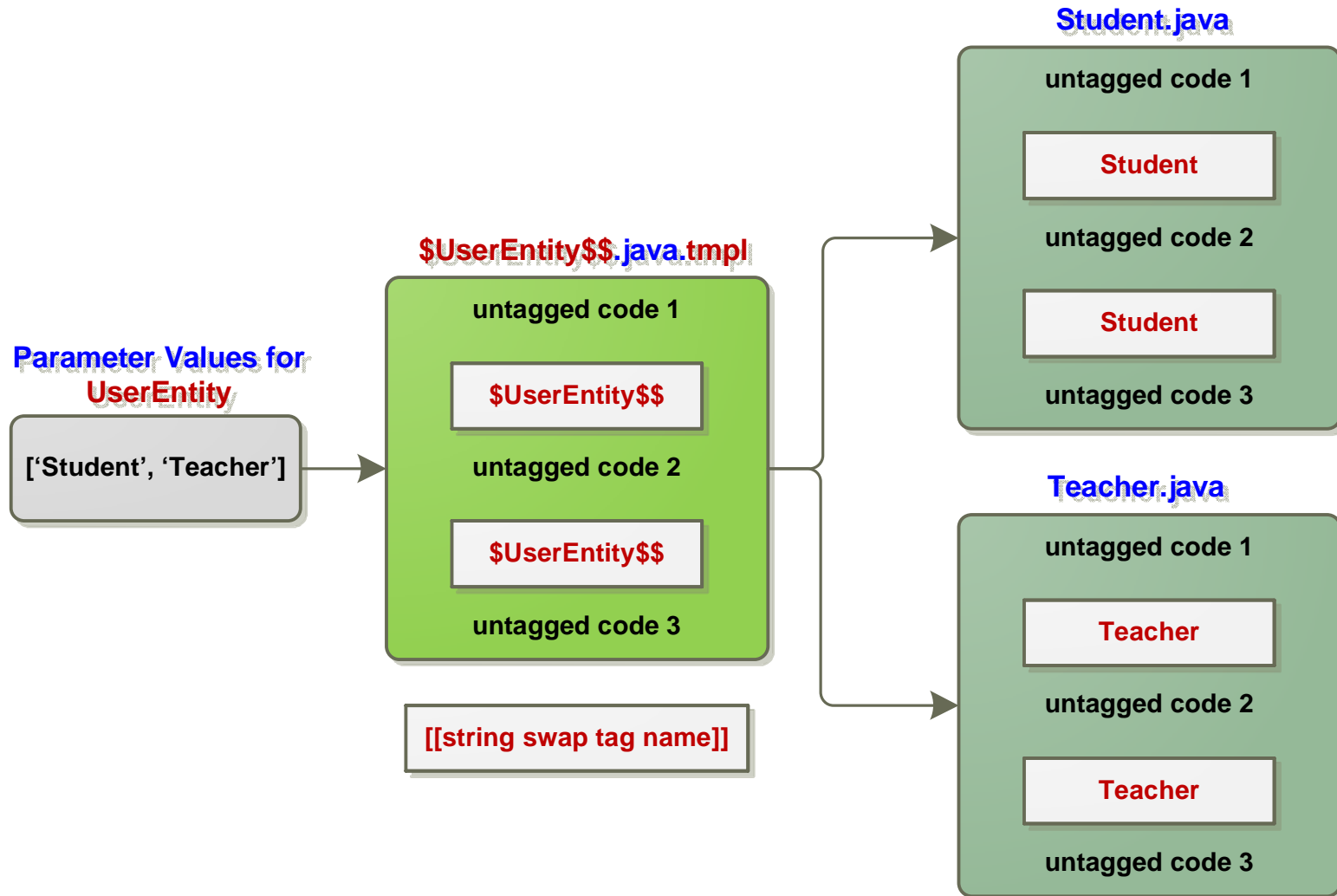
snippet_join:

- Generating a code file depends on generating of other code files called snippets
- Snippet files are generated using template in the same way as source code files
- This CGU type forms a hierarchical dependency among CGUs



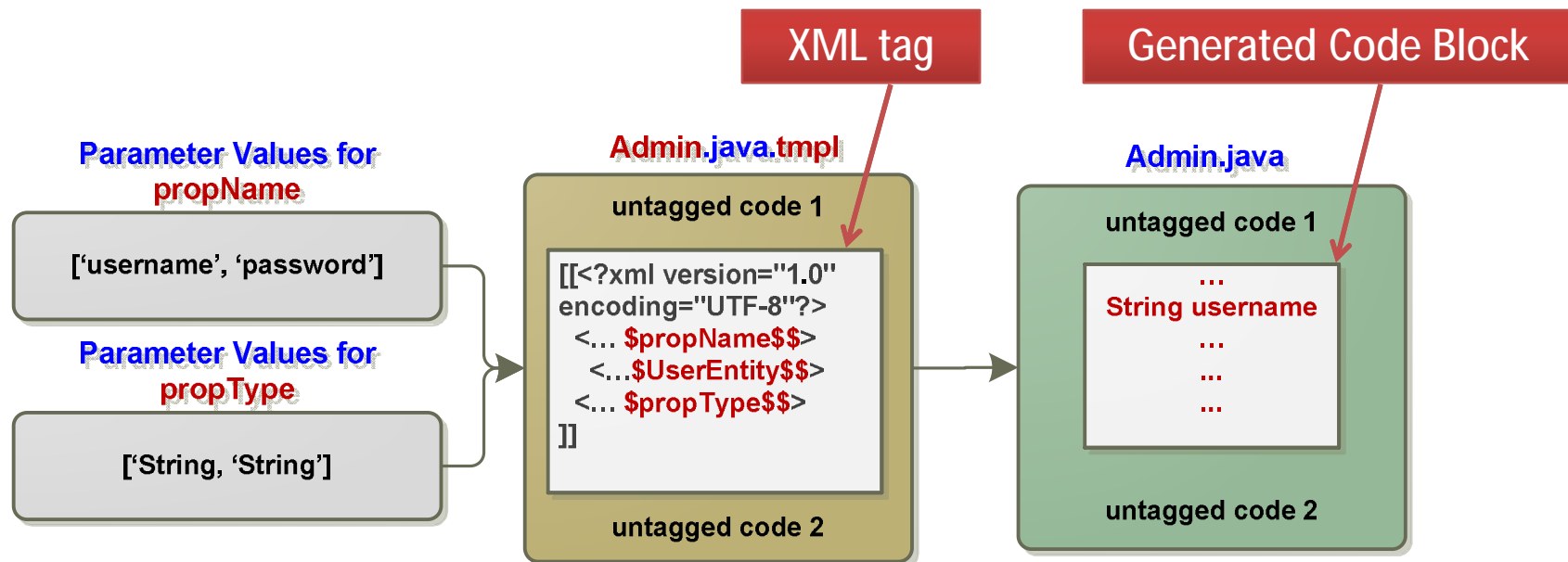
Inner-file CGU Type I: string_swap

string_swap: replacing string swap tags using parameter values



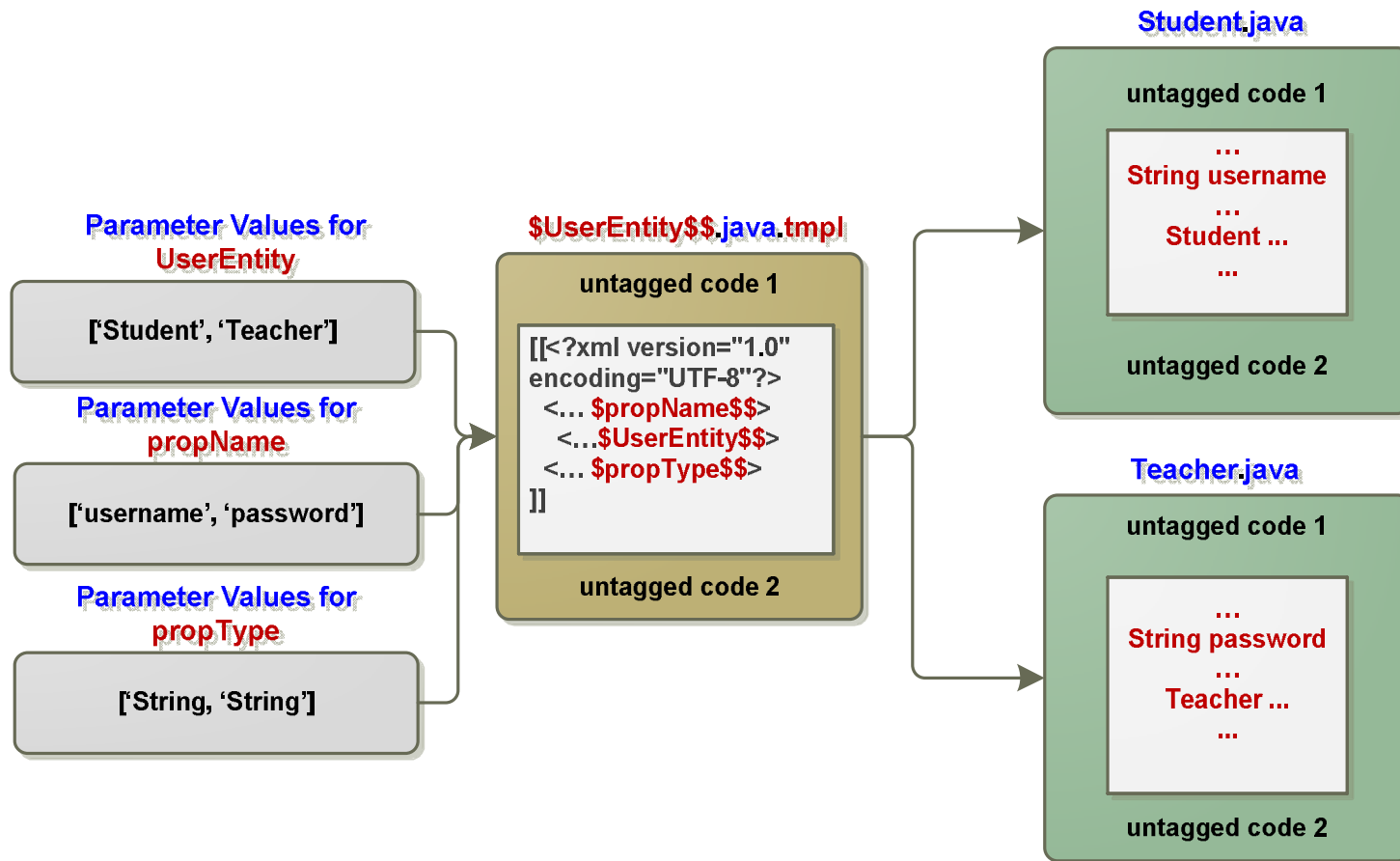
Inner-file CGU Type II: tag_expansion

tag_expansion: replacing xml tags with generated code blocks



Case 1: one template generating one source code file

Inner-file CGU Type II: tag_expansion (cont.)



Case 2: one template generating multiple source code file

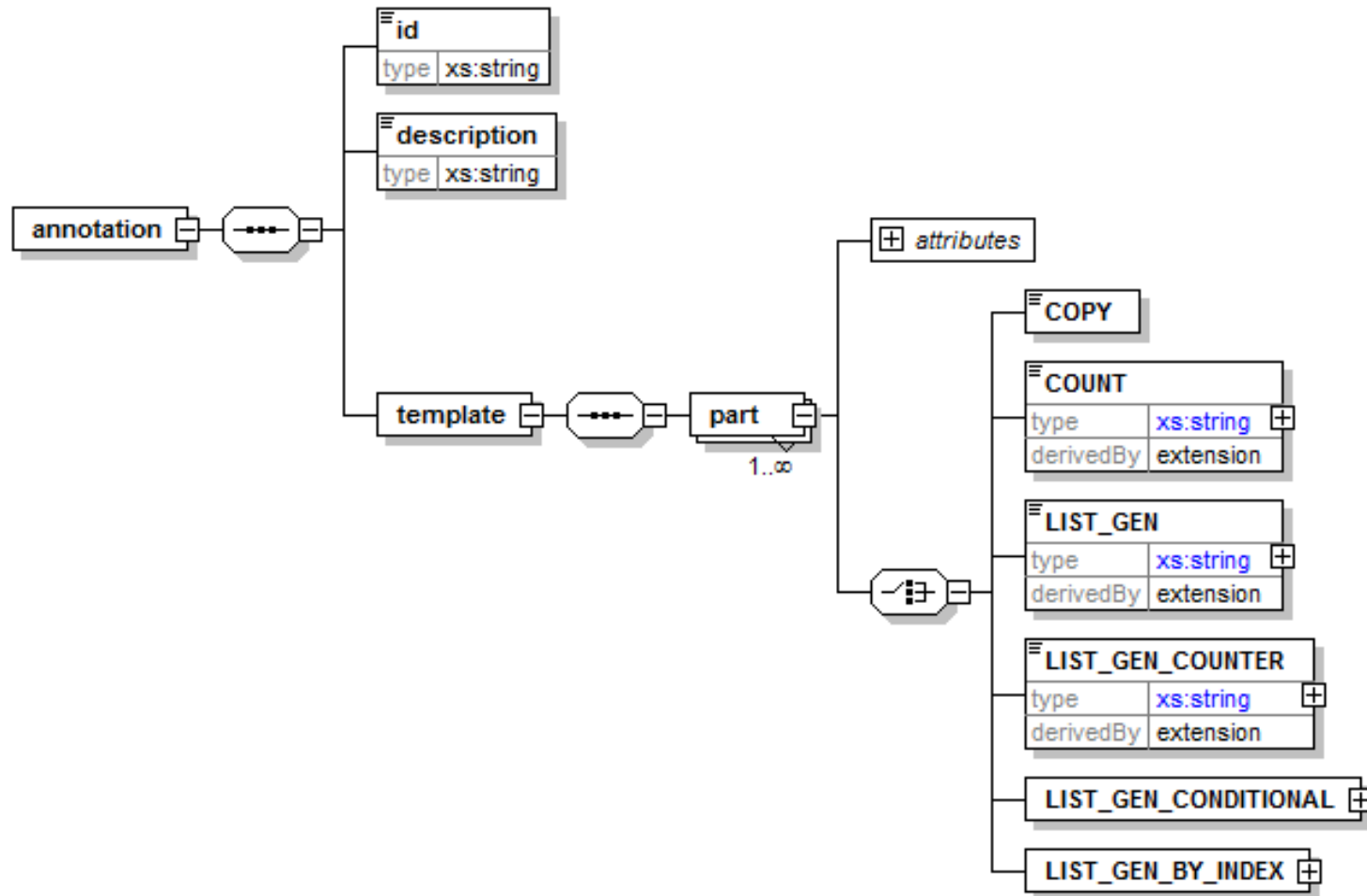
How tag_expansion CGU works

- Each tag is an XML document described using our template language (an XML schema)
- Each tag contains “tag templates”, for some code block in list structure, we also call them “list item templates”
- Tag templates contain parameters
- Code generator replaces these parameters with their values and generate code blocks
- Code blocks replace tags
- Then we got the final generated source code, Bingo!

tag_expansion CGU XML tag types

- **COPY:** simple copy the tag content to the generated code block
- **LIST_GEN:** generate a list of items using a same item template, but different parameter values
- **LIST_GEN_CONDITIONAL:** add conditions for each list item generation, so different item template can be used based on condition evaluation
- **LIST_GEN_BY_INDEX:** select list item template by the item index, useful for if...else if...else structure
- **LIST_GEN_COUNTER:** useful when the template content contains item index, e.g. refer to one array item at a time
- **COUNT:** useful for declaration of array, list, etc. where collection size is required

XML Template Language Overview



A Tag Expansion Example

- *generating an if-else block*

```
[[<?xml version="1.0" encoding="UTF-8"?>
<annotation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="template_annotation.xsd">
  <id>homepage_redirection</id>
  <description>redirect to proper user homepage based on user
  type</description>
  <template>
    <part separator_to_next_part="\n">
      <LIST_GEN_BY_INDEX separator_after_last_item="true"
        parameters="userEntityName">
        <content separator="\n" index_range="0..0">
          if(session.is$userEntityName$$LoggedIn())
          setResponsePage(new
          $userEntityName.UpperCase$$HomePage());
        </content>
        <content separator="\n" index_range="1..1">
          else if(session.is$userEntityName$$LoggedIn())
          setResponsePage(new
          $userEntityName.UpperCase$$HomePage());
        </content>
      </LIST_GEN_BY_INDEX>
    </part>
  </template>
</annotation>
]]
```

XML tag

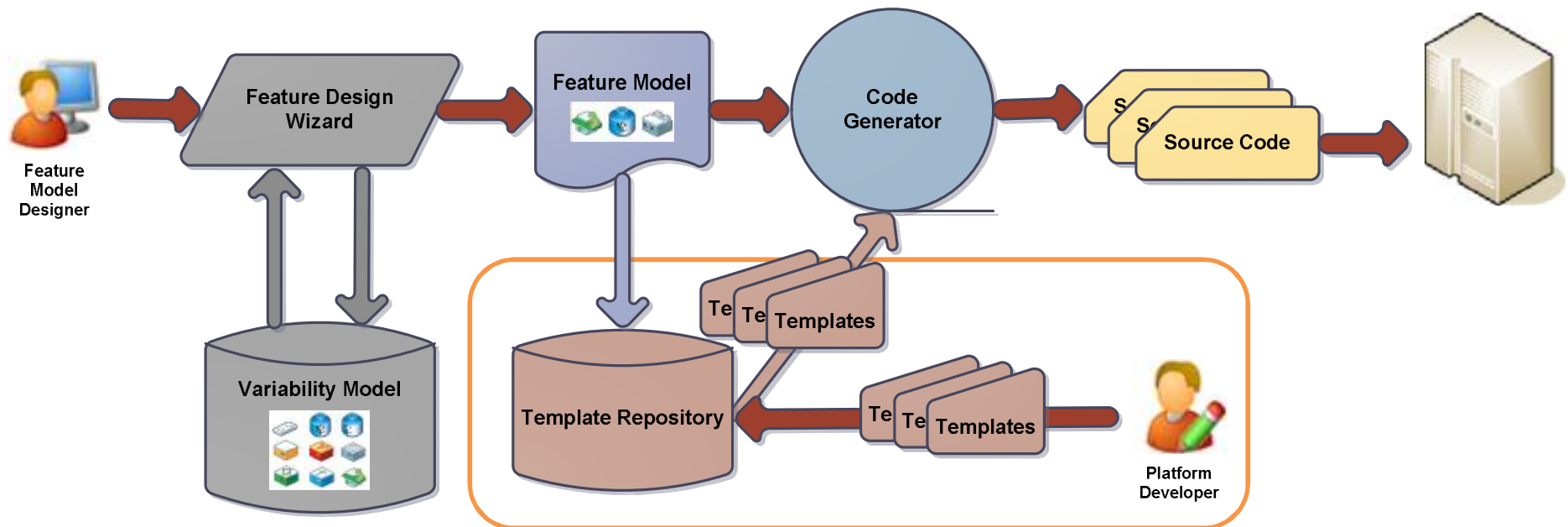
Parameter Values for "userEntityName":
['Student', 'Teacher']

```
If(session.isStudentLoggedIn())
  setResponsePage(new STUDENTHomePage());

Else if(session.isTeacherLoggedIn())
  setResponsePage(new TEACHERHomePage());
```

Generated Code Block

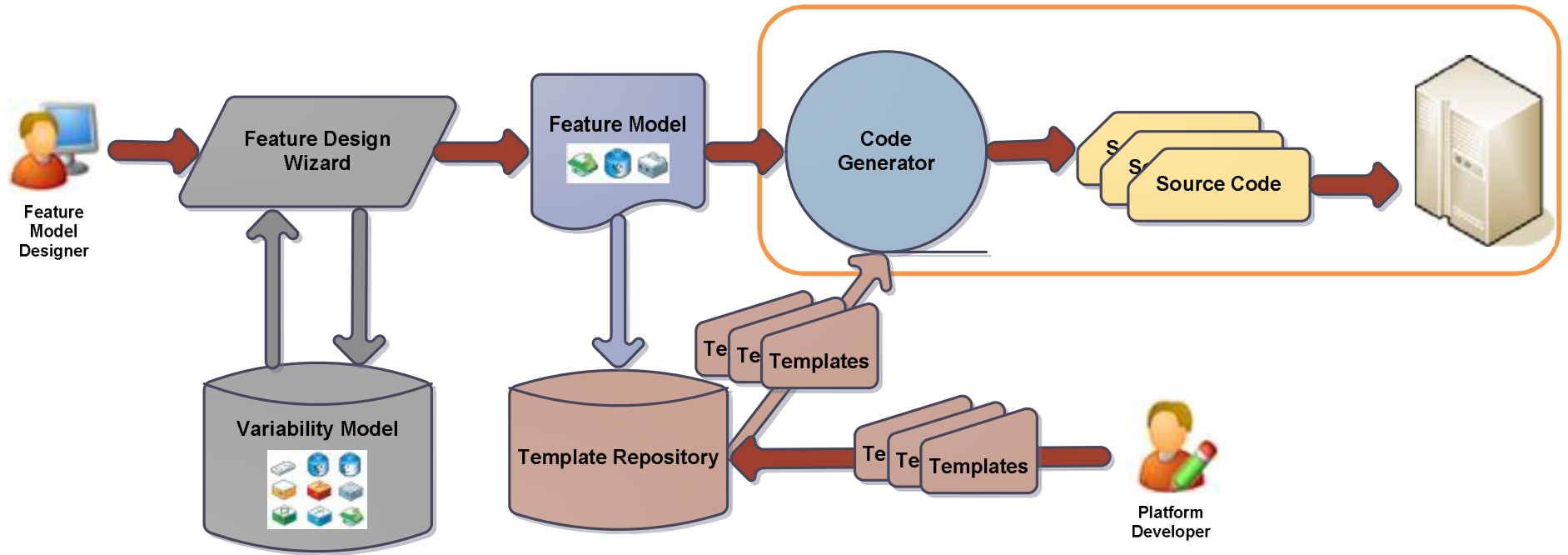
Template Language and Template Composition

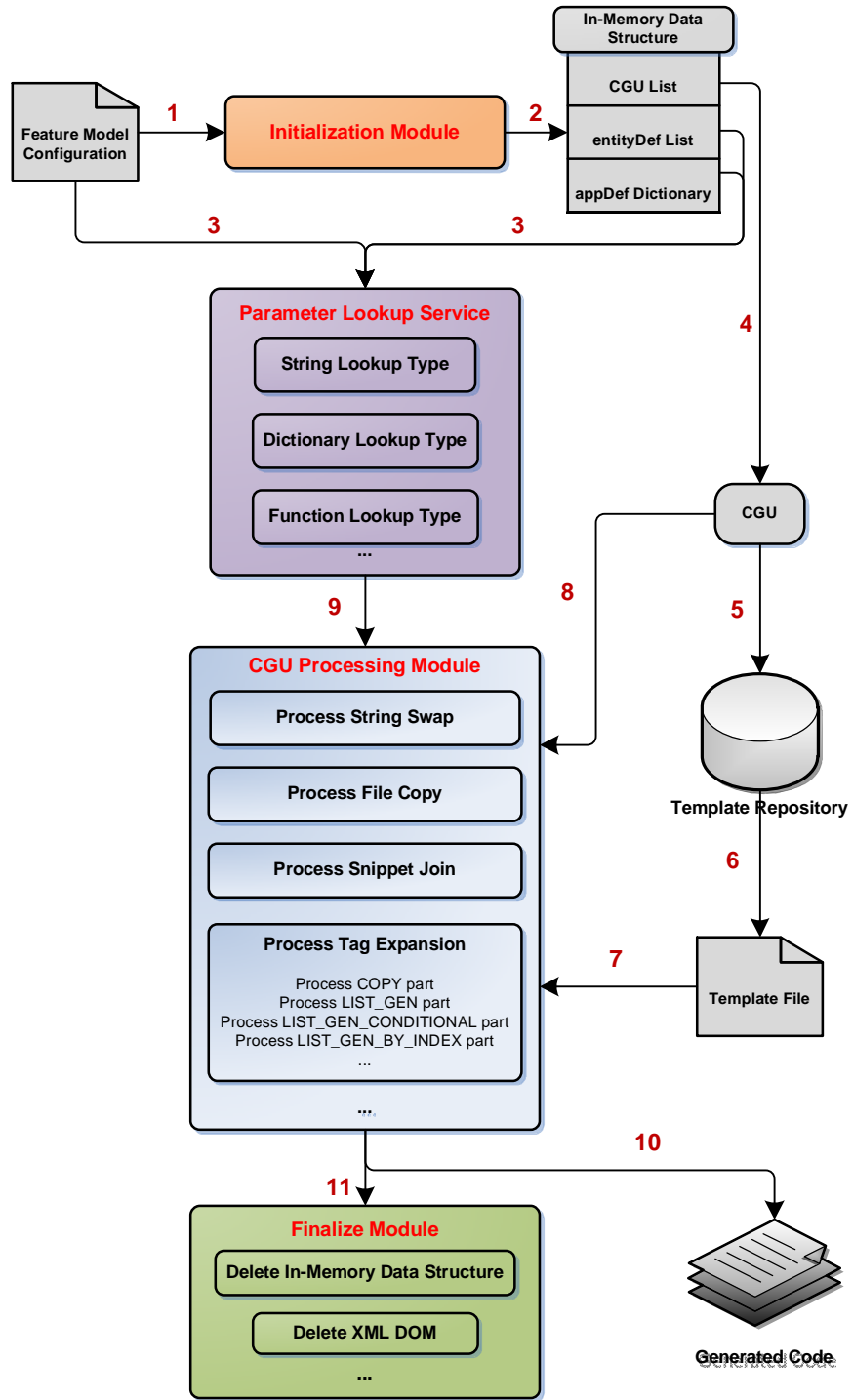


Where do templates come from?

- Platform developer study existing software products and compose templates
- Templates are stored in repository and associated with feature design choices
- When new features or feature design choices are added, new templates need to be built and added too

Code Generation










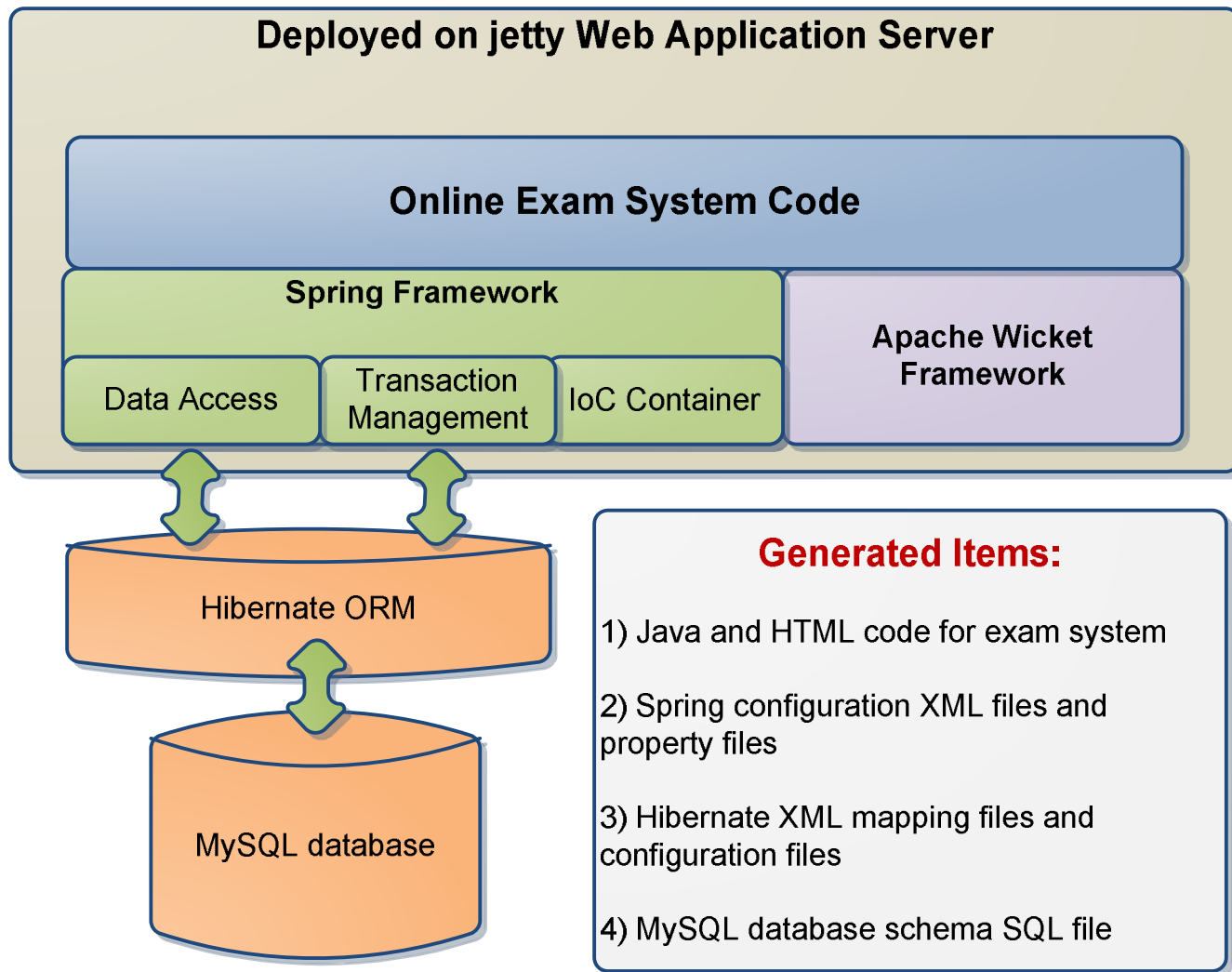
Rhizome Platform Implementation

- Feature modeling language VarML, template language built using XML schema language
 - Easy to add new language elements
 - Rich set of XML utilities to choose from
- Code generator built using Python
 - Around 2.2K LOC
 - Easy to add processing function for new CGU types or tag_expansion types
 - Little dependency on product line domain

Proof-of-concept System Generation

- Online Exam Systems
 - Users
 - Teacher: Create questions, exams, release and grade exams, manage students of the class
 - Student: Take exams, view exam reports
 - Admin: All system functionality
 - Questions:
 - MC: Multiple choice questions
 - TF: True or false questions
- System Variants
 - System 1: MC only, w/o Admin user
 - System 2: MC+TF, w/ Admin user (demo at <http://www.soe.ucsc.edu/~guozheng/Rhizome-demo-video/Rhizome-demo-video.html>)
- Each variant is built/generated using
 -  WICKET  Spring  HIBERNATE  MySQL  Jetty
 - Wicket Web application framework
 - Spring Framework
 - Hibernate Object Role Mapping
 - MySQL Relational Database
 - Jetty Web application server

System Architecture for Generated Online Exam Systems



Suitability for Validating Rhizome

- Non-toy domain
 - Web app has 3-tier architecture
 - Many different technologies such as Spring Framework, Hibernate ORM, Wicket Web Application Framework, leading to non-trivial code structures
 - Mid-size scale, about 6K LOC for each variant Java code, 3K LOC for feature model XML document
- Simple, easy to understand variations
 - Familiar entities such as student, teacher, admin, etc.
 - Typical Web application behaviors like registration, login, form filling and submission, etc.

Contributions

- First substantial effort to link variability modeling with code generation for non-toy example
 - An end-to-end Feature Modeling and code Generation Platform
- Feature Modeling Language
 - Has necessary hooks for supporting code generation & change impact analysis
- Template Language
 - Designed to support multiple kinds of source files, including configuration files, database schemas, OO code, etc.
 - Well modularized, supports addition of new code patterns
- Code Generation Engine
 - Flexible enough to generate a wide variety of code
- Identification of Template-based Code Generation Patterns
 - Which code patterns are most needed to create a web application

Future Work I: Variability Model

- Variability Model or Variability Repository:
 - Store existing design choices for feature structures and behaviors
 - Textual description for features, design rational about why and when to choose a design choice
 - Dependencies among design options and design rules
 - Existing feature models
 - Code Generation Unit (CGU) associations
- Implementation Ideas:
 - Use XML Schema to define variability modeling language
 - Implement the model using native XML database
 - Use XML Schema built-in functions for simple dependencies like “required”, “optional” and special attributes for feature group dependencies
 - Use Schematron language for advanced dependencies

Future Work II: Feature Design Wizard

- Currently, feature model XML documents are composed manually
 - Easy to make errors
 - Semantic errors only identifiable at code generation time
 - Lack of real-time feature dependency checking, e.g. a validation daemon running the background to eliminate invalid design paths silently
- We need a GUI-based design wizard
 - To allow graphically edit a feature model using widgets like checkbox, textbox, etc. and save it in textual format using a feature modeling language
 - To provide real-time suggestion about available design choices and eliminate invalid choices
 - To block further design activities if no valid design paths leading to a final feature model
 - To help create new design choices and update variability model
 - To inform platform developers about design choice changes and ask them to update template repository accordingly

Other Future Work

- Design and code change impact analysis
- Code generator improvement
- Applications to other domains

Related Work

- Feature and Variability Modeling
 - Graphical languages
 - FODA feature diagram and its extensions
 - UML-based variability modeling in SPL
 - Textual languages
 - XFeature project
 - Shape grammar and feature structure
- Code Generation Frameworks Supporting Variability
 - Hamilton 001 Tool Suite
 - XML-based Variant Configuration Language (XVCL)
 - Feature-Oriented Programming (FOP) and AHEAD toolkit
 - Model Driven frameworks like Eclipse Modeling Framework

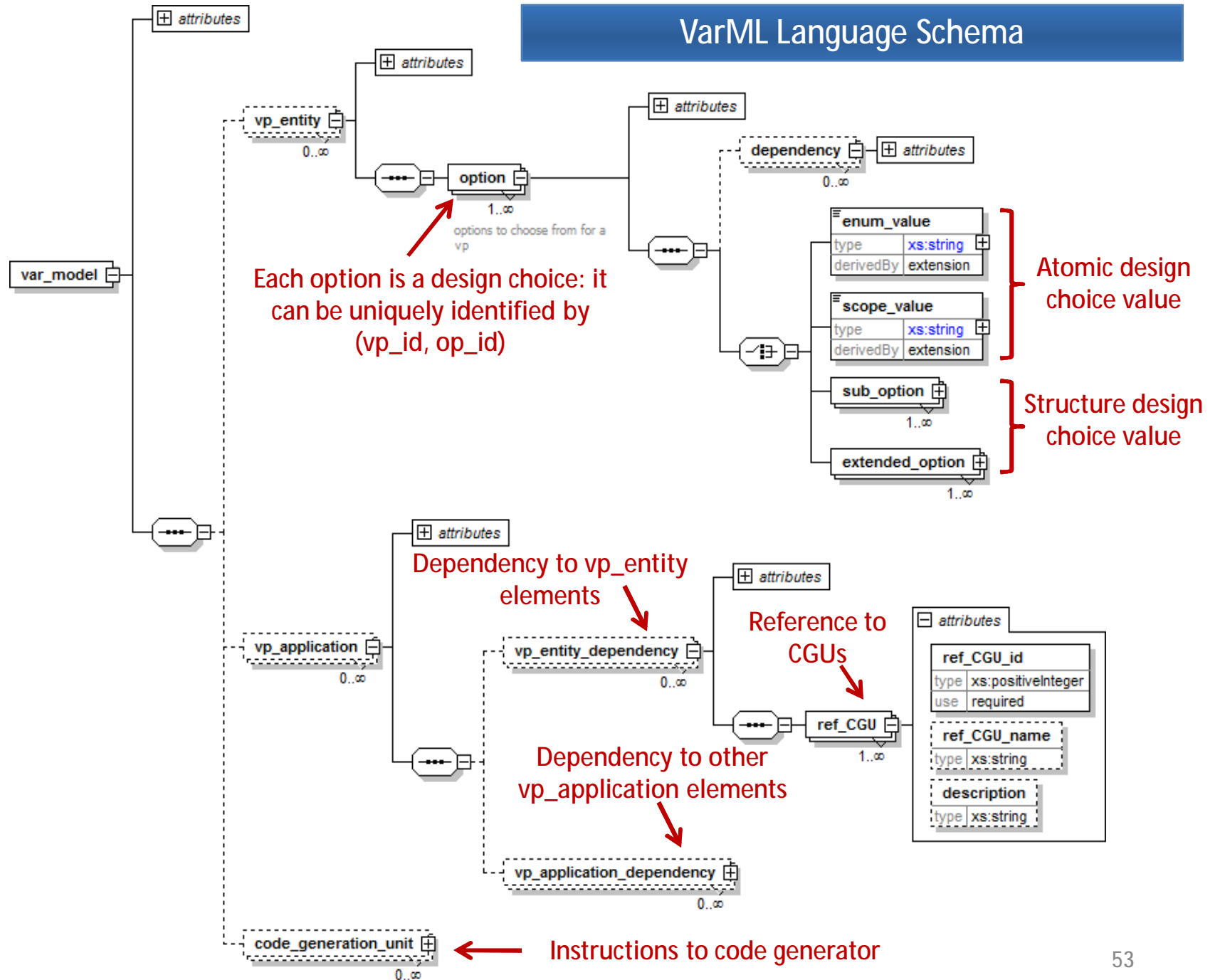


Question?

Platform Design Requirements

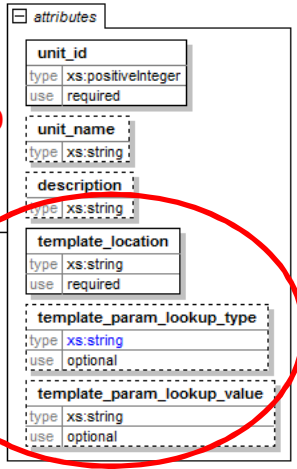
- Lessons learned from Bamboo Containment Modeling Framework
- Bamboo can:
 - Generate content management system repository and explorer based on a graphical modeling language
 - Generate features using domain roles and feature implementation patterns on top of repository model
- Bamboo is limited:
 - Code assembler using prepared code snippets for feature generation
 - Feature implementation pattern not scalable and no dependency
 - Existing patterns cannot cover all possibilities
 - Lack of feature implementation pattern language, CMF language is only good for containment models, not dynamic system behaviors

VarML Language Schema



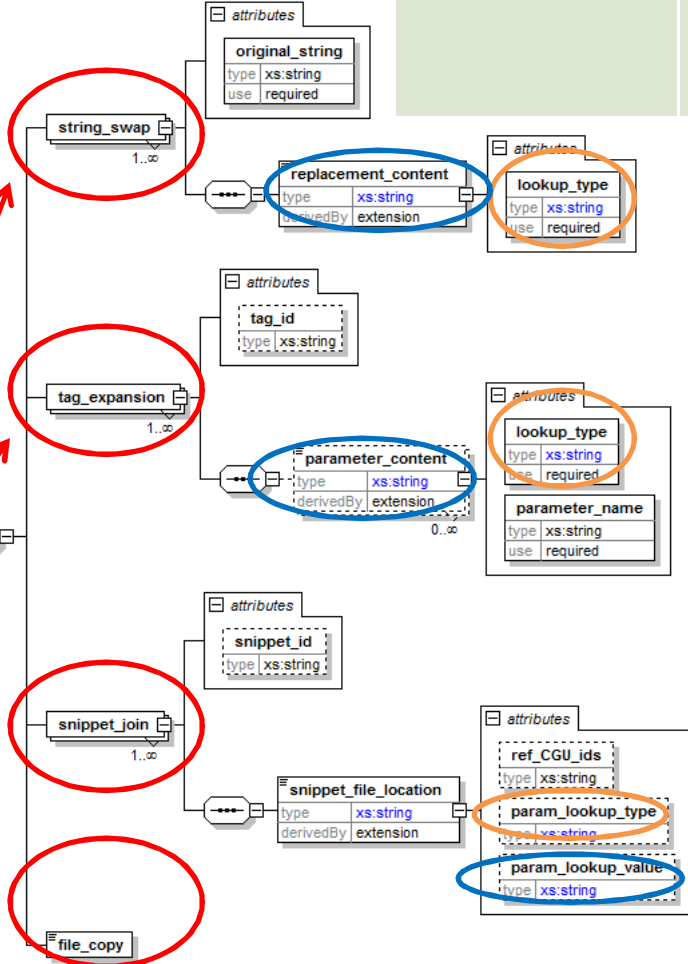
CGU element in VarML Language Schema

Template Info



Lookup Type	Lookup Value	Result
String	"Student"	"Student"
Function	findSubclassUserEntityNames	"Student", "Teacher"
Dictionary	userDef.Student.props.propName	"username", "password", "firstname", "lastname"

code_generation_unit



Code Generation Unit Types


Param Lookup Content

Param Lookup Type

Parameter Lookup Applications

Application 1: Template Filename Resolution *(one template can generate multiple code files)*

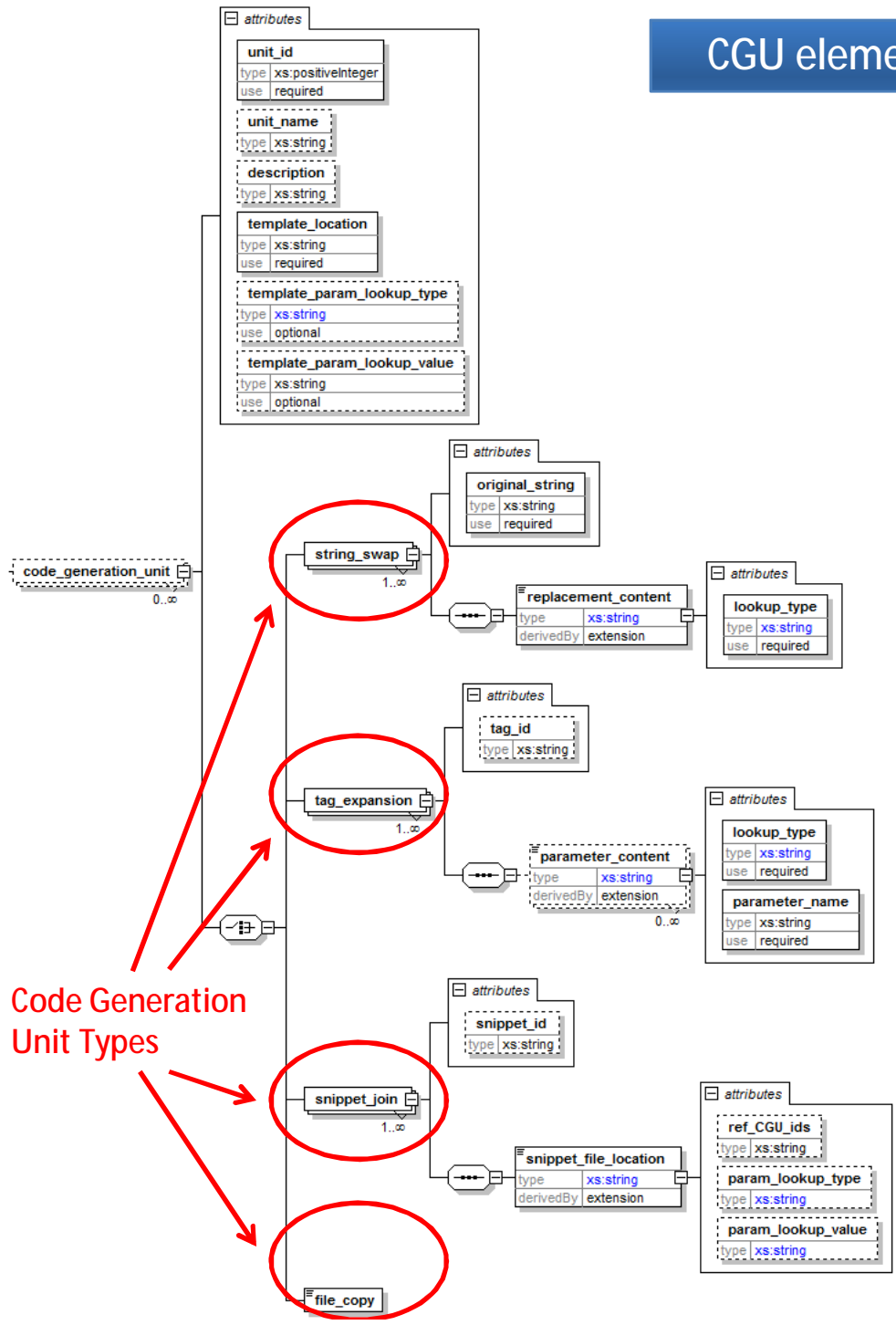
Lookup Type	Lookup Value	Resolved Template Filename(s)
String	"MC"	MC.hbm.xml.tpl
Function	findSubclassQuestion returns ["MC", "TF"]	MC.hbm.xml.tpl TF.hbm.xml.tpl
Dictionary	questionDefs.MC.Template name	MC.hbm.xml.tpl \$.hbm.xml.tpl



Application 2: Parameter Replacement in Templates to Generate Code (we will see later)

- Now, the first question about “how and where to find parameter values” is answered
- Let’s see “how to use parameters to generate code” using `code_generation_unit` element and templates

CGU element in VarML Language Schema



Mechanism for Function and Dictionary Parameter Lookup

Code Generator Parameter Data Store

```
entityDef = ['name':name, 'parent':parent, 'props':props, 'collections':collections, 'mappings':mappings]
    //we have userDefs, questionDefs, examDefs, etc.

props = [prop, ...]

prop = {'propName':propName, 'propType':propType, 'lengthLimitLower':lengthLimitLower,
    'lengthLimitUpper':lengthLimitUpper, 'valuePattern':valuePattern, 'propSQLType':propSQLType,
    'skippedTemplates':skippedTemplates}

skippedTemplates = [templatePath, ...]

collections = [collection, ...]

collection = {'colName':colName, 'colType':colType, 'colTypeImpl': colTypeImpl,
    'colMemberType':colMemberType}

mappings = [mapping, ...]

mapping = {'type':type, 'name':name, 'targetClass':targetClass, 'table':table, 'column':column}

appDefs = {'entityName':apps, ...}

apps = {'appName':props, ...}

props = [prop, ...]

prop = {'propName':propName, 'propType':propType, 'GUIWidget':GUIWidget}
```

Mechanism for Function and Dictionary Parameter Lookup

Dictionary Lookup: Path Expression Grammar

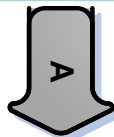
Format	Applicable Values	Example	Explanation
list*.key	list=userDefs, questionDefs, etc. key=name, parent	userDefs.*.name	List all user types
list.entityName.key	list=userDefs, questionDefs, etc. entityName=any entity name key=name, parent	list.Student.parent	Show the parent user types for Student user type
list.entityName.key4List.key4ListItem	list=userDefs, questionDefs, etc. entityName=any entity name key4List=props, collections, mappings key4ListItem=any property for prop, collection, mapping dictionary	list.Student.props.propName	List all property names for the Student user type
dictionary.entityName.AppName.props.key	dictionary=appDefs entityName=any entity name appName=any app name key=propName, propType, GUIWidget, etc.	appDefs.Teacher.DeleteUserConfirm.props.propName	List property names to appear on DeleteUserConfirm page for Teacher

Function Lookup: Write Search Functions in Code Generator

String Swap, COPY and LIST_GEN Example: Step 1

```
<code_generation_unit unit_id="1" unit_name="Subclass_User_Entity_string_swap" description="swap strings for global parameters in a user entity such as Student and Teacher that extends a parent User entity definition" template_location="edu/ucsc/cse/exam/datamodel/$UserEntity$.java.tpl.Subclass" template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
  <string_swap original_string="UserEntity">
    <replacement_content lookup_type="Dictionary">userDefs.$UserEntity$.name</replacement_content>
  </string_swap>
</code_generation_unit>

<code_generation_unit unit_id="2" unit_name="Subclass_User_Entity_tag_expansion" description="generate a class constructor for subclass user entities such as Student and Teacher" template_location="edu/ucsc/cse/exam/datamodel/$UserEntity$.java.tpl.Subclass" template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
  <tag_expansion tag_id="constructor">
    <parameter_content lookup_type="Dictionary" parameter_name="propType">userDefs.$UserEntity$.props.propType</parameter_content>
    <parameter_content lookup_type="Dictionary" parameter_name="propName">userDefs.$UserEntity$.props.propName</parameter_content>
  </tag_expansion>
</code_generation_unit>
```



Find Parameter Values using CGU Definitions



UserEntity = "Student"
propertyType = ["Long", "String", "String"]
propertyName = ["id", "name", "email"]

UserEntity = "Teacher"
propertyType = ["Long", "String", "String", "String", "String", "String"]
propertyName = ["id", "name", "email", "classes", "office", "phonenum"]

String Swap, COPY and LIST_GEN Example: Step 2

/edu/ucsc/cse/exam/datamodel/**UserEntity\$.java.tmpl.Subclass**

```
package edu.ucsc.cse.exam.datamodel;

[[<?xml version="1.0" encoding="UTF-8"?>
<annotation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="E:\slugforge_repos\doc\online_exam_design\xml\
template_annotation.xsd">
  <id>constructor</id>
  <description>create the constructor for the subclass user entities such as Student and Teacher</description>
  <template>
    <part separator_to_next_part="\n">
      <COPY>public class UserEntity.CamelCase$$</COPY>
    </part>
    <part separator_to_next_part="\n">
      <LIST_GEN separator="\n" separator_after_last_item="true" parameters="PropertyType PropertyName">
        private PropertyType$$ PropertyName$$;
      </LIST_GEN>
    </part>
    <part separator_to_next_part=" ">
      <COPY>public UserEntity.CamelCase$$ </COPY>
    </part>
    <part>
      <LIST_GEN separator="," separator_after_last_item="false" parameters="PropertyType PropertyName">
        PropertyType$$ PropertyName$$
      </LIST_GEN>
    </part>
    <part>
      <COPY></COPY>
    </part>
    <part>
      <LIST_GEN separator="\n" separator_after_last_item="false" parameters="PropertyName">
        this.PropertyName$$ = PropertyName$$;
      </LIST_GEN>
    </part>
    <part>
      <COPY></COPY>
    </part>
  </template>
</annotation>]]

public void packageInfo(){
  System.out.println("package name: edu.ucsc.cse.exam.datamodel");
}
}
```

Generate Source Code using Template

/edu/ucsc/cse/exam/datamodel/**Student.java**

```
package edu.ucsc.cse.exam.datamodel;

public class Student{
  private Long id;
  private String name;
  private String email;

  public Student(Long id, String name, String email){
    this.id = id;
    this.name = name;
    this.email = email;
  }

  public void packageInfo(){
    System.out.println("package name: edu.ucsc.cse.exam.datamodel");
  }
}
```

A

/edu/ucsc/cse/exam/datamodel/**Teacher.java**

```
package edu.ucsc.cse.exam.datamodel;

public class Teacher{
  private Long id;
  private String name;
  private String email;
  private String classes;
  private String office;
  private String phonenum;

  public Teacher(Long id, String name, String email, String classes,
String office, String phonenum){
    this.id = id;
    this.name = name;
    this.email = email;
    this.classes = classes;
    this.office = office;
    this.phonenum = phonenum;
  }

  public void packageInfo(){
    System.out.println("package name: edu.ucsc.cse.exam.datamodel");
  }
}
```

B