UNIVERSITY OF CALIFORNIA

SANTA CRUZ

RHIZOME: A FEATURE MODELING AND GENERATION PLATFORM FOR SOFTWARE PRODUCT LINES

A dissertation submitted in partial satisfaction
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Guozheng Ge

December 2008

The dissertation of Guozheng Ge

is approved:

Professor Jim Whitehead, Chair

Professor Ethan L. Miller

Professor Charlie McDowell

Lisa C. Sloan
Vice Provost and Dean of Graduate Studies

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

RHIZOME: A FEATURE MODELING AND GENERATION PLATFORM FOR SOFTWARE PRODUCT LINES

by

Guozheng Ge

Rhizome is a rapid development platform for automatic model-to-code transformation in software product lines. It provides a practical means of filling the gap between an ambiguous and incomplete textual feature description model and a precise source code implementation. This platform takes a feature description model as input and automatically generates source code that implements these specified features. Rhizome includes a feature modeling language FeatureML to describe features, a template language MarkerML which embeds special markers in the source code to instruct code generation, and a template-based generator engine that executes code generation.

To use Rhizome for rapid product line application development, a software designer first uses FeatureML by making various feature design choices and capturing these choices in a textual feature model. These design choices are associated with code

templates which are further interpreted as parameters by a code generator to produce source code. Templates are source code files embedded with special XML tags representing variability. The generator scans each template and replaces XML markers with dynamically generated code blocks to produce final system source code. Templates are composed and maintained by platform developers who are experts in architecture and component implementation. Platform developers manually study and capture similarity and variability for the software product line and implement the software architecture and components (templates) that best fit a software product line. At generation time, variability is resolved using design choices made by software designers. The generator uses an array of different code generation patterns varying from inter-file level to intra-file level, including template file copying, hierarchical file generation, and various forms of marker expansion.

Rhizome is a practical end-to-end solution for feature modeling and generation in a software product line. It allows designers to naturally express their design intent via the selection of domain-specific feature concepts and then automatically convert this feature blueprint into executable system source code with the implementation expertise captured in code templates. The Rhizome platform tackles the core problem facing rapid product line development: a lack of direct connection between a design at the semantic level and a concrete implementation at the code level. The platform has been applied to generate online exam Web applications where each generated application is about 6500 lines of Java code. In addition to generating code from high-level feature descriptions, Rhizome also opens the door to many related topics in feature-based development such as a variability repository, design space modeling, feature dependency modeling and analysis, and change impact analysis at the feature level.

ACKNOWLEDGEMENTS

I would like to first thank my advisor Professor Jim Whitehead for his consistent support, guidance and encouragement during my graduate study at the University of California, Santa Cruz. He taught me how to effectively study the literature, how to perform critical thinking and writing, how to network with researchers at conferences, and how to deliver clear and punchy presentations. Jim gave me and his other students substantial research freedom and allowed us to try and fail instead of forcing his ideas on us. I am very grateful for numerous brainstorming meetings and discussions during which we shared thoughts even at the very primitive stage. To me, Jim is both a mentor and a patient, inspiring and caring friend. He shows me not only necessary research skills and knowledge, but also positive life attitudes and how to become a contributor to the community. There are not many advisors who would hire private tutors to help their students improve spoken and written English during the first year in graduate school. There are not many advisors who would chase department secretaries in order to get their students a TAship when research funding had dried up. I will never forget his effort and sacrifice to help every member of our group during the funding hardship. Without Jim's invaluable input and support, I

could not have completed this dissertation and I will try to pass on this spirit and treat people around me in the same manner that he did.

I would also like to express my thanks to all of my former and current colleagues in the software engineering lab: Jennifer Bevan, Sung Kim, Kai Pan, Elias Sinderson and Mark Slater. They helped me a lot by brainstorming on research ideas, sharing coding tips and revising my paper drafts.

Our research work was made possible with financial support from various sources including grants and fellowships from the National Science Foundation, University of California, and the ACM Special Interest Group on Software Engineering.

Last but not least, I'd like to thank my wife Fang Yu and my family members for their unconditional love, patience and support during this long but fun period of graduate study. I cannot imagine how I would have overcome all the obstacles during my years in school without their encouragements. I am deeply indebted to my parents in China for not being able to stay with them, share family responsibilities and take care of them. My dissertation is the best gift I can give to all my loved ones.

1 Introduction

1.1 Problem definition and project overview

Software products are complex and expensive to develop because they need to accommodate a wide range of user requirements. One common strategy is to bundle hundreds of features, including the most commonly used features along with other special features tailored for particular customer needs. One such example is Microsoft Word 2007 [1], which contains hundreds of common features that enable WYSIWYG text editing. But Word also includes features only useful for particular users: the ability to create, modify, and post blog entries, citation management and generation tools for academic and professional writers, etc.

This packaged software model has many implications. Users have to accept a bloated software application package. This means extra costs for features they may never use, a steep learning curve to master complicated user interfaces and complex workflows, and unexpected software bugs caused by features they do not need. For software developers, providing a maximum set of features means a high degree of complexity and dependency in the software components. The end result is higher development cost, a longer development cycle and a greater prevalence of software bugs.

Customized software products based on unique user requirements provide a desirable alternative to the packaged software model. In this customized software model, users specify the exact feature requirements and developers produce software applications implementing these requirements. However, in industry practice, software development still relies on manual work since there are few systematic platforms and little toolset support for domain-specific knowledge abstraction and component reuse. This makes a customized software model too expensive to develop and maintain unless a systematic software component reuse and automatic code generation platform exists to help reduce manual work. Product line software development [2-5] is one promising effort towards the abstraction of domain knowledge for active software reuse. Automatic code generation technology [6-8] provides the other piece necessary to help realize the customized software development model with rapid speed and low cost.

A product line is a group of products that share a common development platform and vary by the composition and implementation method for these functionalities. Compared with the per-product development approach, product line development has the benefits of higher productivity and shorter time-to-market, more effective component reuse and reduced product development costs. Software product line development (SPLD) promotes component reuse and provides a clean separation of concerns between different layers in system architecture. A product derived from a software product line consists of various components selected from existing component libraries; these components communicate with a common platform to perform specific functionalities. Many successful case studies have been documented and studied on the software product line paradigm such as shipboard command-and-control systems [9], revenue acquisition management software [10], satellite ground control systems [3], engine control software [3], and financial information software [3], etc.

However, in order to develop a software product line, engineers still need to manually integrate existing platforms with component libraries, write configuration files and implement glue code. The manual process of comprehending and translating high-level requirement specifications into low-level source code is still required. Design and implementation knowledge is still buried deep in the source code and in developers' minds. The learning curve is still steep for new developers to master both platform and component development. Current software product lines also lack tool support to trace and explain where, how, and why to make changes in both requirement specifications and source code. It is also difficult to predict how changes made in requirement specifications could impact changes in the source code, and vice versa.

There is still great potential for improvement based on our study of the state of art in software product line development. Other engineering fields such as Electronic Design Automation (EDA) leverage high-level hardware description languages and code generators to provide automation from design to implementation. Electrical engineers provide circuit layout and design parameters using existing devices and components. A code generator then automatically creates low-level hardware code that precisely matches these design specifications. A similar mechanism is needed to capture and reuse development skills with automatic code generation for product line software development. Such an automated development paradigm could greatly simplify and improve the design-to-implementation process, encapsulate functionality structures and enable the connection between semantic specifications and source code segments.

Our ultimate goal in this dissertation work is to provide a platform for high-level software product line design and automatic development. There are two types of users of this automated product line development paradigm: software designers and platform developers. Software designers are domain experts that understand domain-specific knowledge and design requirements through communication with software users. Platform developers are responsible for implementing the platform components and connecting these components with feature design choices. They also perform maintenance on the code generator and platform assets to accommodate new code generation policies.

Software designers focus on semantic design aspects and leave the implementation work to an automatic code generator and to platform developers. The automated feature generation platform provides reusable components that capture low-level implementation knowledge and expose various design choices for high-level customization. Software designers make decisions regarding customization details such as the data model, the set of features to implement, feature descriptions and structures. These design choices are captured in a design specification called a feature model. Designers work with a feature modeling language and a design wizard program to create this feature model. Once a feature model is complete, the designer’s job is finished and the automatic code generator starts working to produce source code that implements the design choices.

A variability model captures all possible design choices, along with the dependency relationships among various choices, in a software product line. Platform developers are responsible for implementing and maintaining this variability model in order to provide customizable platform components. Design choices made by a software designer are connected with reusable components and translated into a code generation specification. A code generator then uses this specification to produce final source code for an application.

There are three fundamental problems that must be addressed to realize this automated software product line development paradigm: variability and feature model abstraction, code generator design and implementation, and the connection of feature models and code generation. A solution to implement this development paradigm includes answers to such questions as how to define and store feature design choices; how to express design choice dependencies like AND, OR, XOR relationships along with custom dependencies related to particular application domains; and how to create design choice value structures so that they can be queried easily in the code generator. A variability modeling language is needed to support these goals. A feature model is derived from a variability model—a designer selects specific features and methods to implement these features without violating dependency relationships. In other words, a feature modeling language is needed to record the various design choices a designer makes. The variability modeling language and feature modeling language both need language elements to connect with the code generator. Code generators come in different types and employ different mechanisms: some are based on a formal model, some use code assemblers and some use template-based generation. Essentially, there must be connections between a feature model and code generator in order to parameterize high-level design choices and subsequently use them in the code generation process. This connection is the most challenging aspect of our work because feature models are described in an informal and sometimes ambiguous textual language while the generated source code must be a precise representation of a running software system. Finding an approach to close this gap is a challenging task, and one of the issues this dissertation work attempts to solve.

Existing research generally focuses on either feature modeling or code generator development. This dissertation is among the first effort to unite high-level feature models with automatically generated code. Feature modeling work usually defines a feature model using a graphical language or natural language. A feature definition includes feature descriptions, properties, structure, behavior, dependencies and interactions with the external environment. There are many existing graphical feature modeling languages based on the Unified Modeling Language (UML) [11] or based on XML such as XFeature [12], and XVCL [13, 14]. The goal of these existing feature modeling languages is to edit and visualize an abstract software requirement specification for a product line. Software engineers need to manually implement these feature models according to the specification.

On the other side is the automatic code generator that turns a software specification into source code. Many code generators are domain-specific, such as those for GUI programming (Visual Studio Designers [15], WindowBuilder [16]), documentation systems (Javadoc [17], Doxygen [18]), and Web applications (ArcStyler [19], JAG [20], AppFuse [21]). Code generation techniques include template-based, model-based, and assembler-based. In the template-based generation approach, tags are inserted into source code to represent sections of variable code. A code generator scans the tagged code, replacing the tags with information obtained from a requirements specification to resolve the variability and produce compilable source code. In the model-based approach, a requirements specification is turned into models described with a syntax tree, state machine, or UML. Then, a code generator transforms these models into source code. An assembler-based generator is a smart selector that understands how and where to pick the right source code that matches a requirements specification. In this approach, source code files are manually implemented and stored in a source code repository prior to assembly. The code generator translates the requirements specification into queries, and searches for matching source code files. The query result is the set of source code files that implements the specification.

Closing the gap between variability and feature modeling on the one side, and code generation on the other side, is a challenging research problem. There is little work addressing this problem so far. A feature model contains a semantic description about features that need to exist, internal structures these features have, and behaviors these features exhibit. Implementation details are not included since they are not the focus of a feature model. So, from an implementation point of view, a feature model is an ambiguous textual model best suited for a human audience to capture design ideas and high-level system requirements. Source code, on the other hand, is an accurate and complete representation of both the static structure and dynamic behavior for a software application. It describes exactly how features are implemented using a formal programming language and how these features interact with each other at runtime. Its target audience is the computers that execute the application. This gap between ambiguous human specifications and accurate code implementation is the primary reason for which there has been little research into connecting feature modeling with code generation. The implementation knowledge needed to fill the gap between a feature model and its implementation code resides in software engineers' minds and it is quite difficult to represent human development skills using a formal machine-readable language.

Source code templates provide an easy and practical way to fill this gap. A template is a source code file embedded with special tags, where each tag represents one variability dimension in the source code. The code generator uses information from a feature model to create a block of code that replaces a tag in a template. After all tags in the template files are replaced, the resulting source code files can be generated. Notice that templates are still prepared in advance by platform developers: platform developers still need to study the existing software product lines and abstract out commonality and variability in the source code. The templates encapsulate development knowledge and skills to strategically control how variability is exposed to feature model designers via the template tag mechanism. For example, if User properties such as username, real name, email, etc. should be available as a design choice in the feature model, the code that implements the User entity needs to include corresponding tags to accommodate this variability so that design choices made at the feature model level can be reflected in the generated code. On the other hand, if the User entity properties are not supposed to be exposed in the feature model, a platform developer can simply use code that implements a set of default User properties in the template without using tags. In this way, tags can be used in places where design decisions should be made by feature model designers. For variability that should not be exposed to the feature model designer, we simply use code that implements the default behavior.

Template tags are connected with a feature model through Code Generation Units (CGUs). A CGU provides instructions for the code generation process: it defines template locations, code generation types, parameters to be used for generation, and queries for finding parameter values. There are many different types of CGUs, such as a file copy CGU that copies a template file to create new source code files without modifying content, a snippet join CGU that joins code generated by other CGUs to create a new source code file, and a tag processing CGU. The tag processing CGU is one of the most important CGUs, since it provides a direct connection to the tags embedded in a template file. Each tag processing CGU has an id that matches a tag with the same id in a template file. A tag processing CGU also defines a list of parameters and query expressions for fetching parameter values from the feature model—design choices are represented as parameter values in the code generator. Parameters in a tag processing CGU match parameters in the template file that have the same parameter names. In this way, design decisions made by software designers in a feature model are passed through CGUs into the template tags to generate the final source code.

The advantage of our template-based approach is its simplicity of implementation and its flexibility with respect to maintenance and upgrade. A CGU contains the code generation instructions and data definitions required to produce source code. A code generator simply processes CGUs associated with a feature model to create final source code: the generator scans related templates to create code blocks, and performs other code generation activities, such as global string replacement, file copy, etc. The code generator does not need to understand the semantic meaning of each variability factor. Compared with model-based code generation, in which the code generator needs to understand the modeling language and perform complex model-to-code transformations [22], our template-based code generator is much simpler to build.

Maintenance in Rhizome is simple, as new feature variability can be easily added so long as the associated templates are updated. However, a down side is also apparent: platform developers are responsible for the manual work to implement these templates. If a designer wants some feature variability that is not covered by an existing template, then platform developers need to either modify existing templates or create new ones. As a consequence, platform developers need to be experts on many details related to product line architecture design, the frameworks and libraries used in implementation, as well as feature modeling and generation platform components. Developing a set of templates for a product line is much more difficult than developing a product line or a single system—there are source code patterns to observe, template tag design strategies to learn and various CGU processing scenarios to implement. As a consequence, this approach involves a tradeoff, where more up-front engineering effort can lead to substantially reduced downstream effort. In general, this tradeoff is only worthwhile in software product lines where there are many products over which to spread the development costs for templates and the code generator.

In addition to the CGU that bridges a feature model and the source code that implements this model via automatic code generation, this dissertation also makes contributions to the feature modeling and template-based code generation field. The feature modeling language can describe hierarchical feature structures, where a parent feature typically consists of a group of sub-features. It classifies feature variability into two categories: *entity-related variability* models static aspects and *application-related variability* models dynamic or behavioral aspects of a feature model. In Rhizome, a unique feature modeling process is also proposed to identify entity and application variability, which works naturally with the FeatureML modeling language. The contributions to the template-based code generation research area include textual code generation patterns and a template language to specify embedded XML tags. The code generator architecture has a flexible structure that allows easy integration of new code generation patterns.

The domain of online exam systems is used as a sample domain throughout this dissertation. Online exam systems have a rich set of features and each system is different from others in terms of feature configurations. Individual online exam systems are often customized to contain different user types, different question and exam types, different grading policies and various other features. That is why this online exam system domain is selected as a proof-of-concept demo of automatic system generation using the Rhizome platform.

An online exam system consists of multiple entities such as various user types including teacher, student, teaching assistant (TA), and administrator, various question types including multiple choice (MC), single choice (SC), short answer (SA), etc., and various exam types that incorporate different question types and different exam attributes. Using the feature modeling language elements in this example, entity variability includes property value types, value length limit, value patterns like those for time or phone number, user types, user properties for each user type, question types, question properties, exam types and properties, and GUI widgets associated with each property. Application variability includes user activities such as taking an exam, answering questions, grading an exam, reading an exam report, etc. Feature variability can have multiple options, and each such option is a design choice made by a software designer. For example, the user type feature may have four variability options: Student, Teacher, TA, and Administrator. So, a feature model can be viewed as a collection of feature variability choices knit together by the software designer making various design choices. The feature modeling language helps records designers' choices.

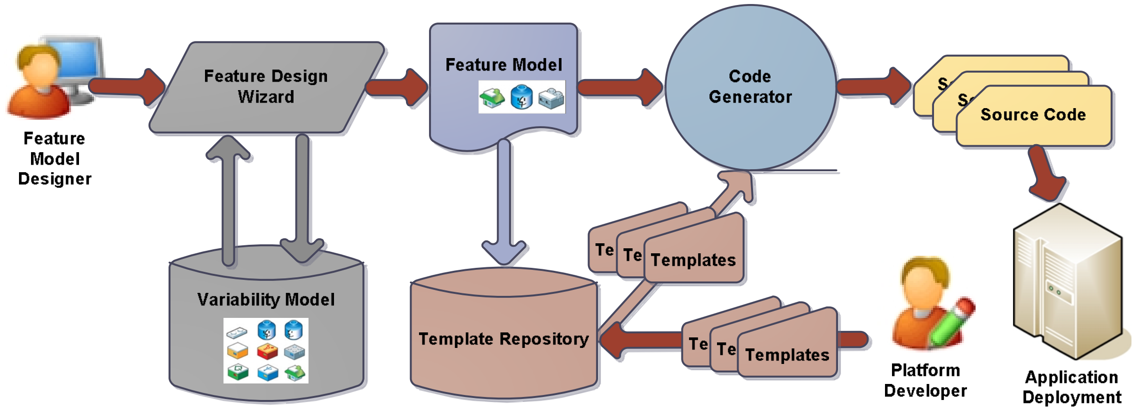


Figure 1-1. Rhizome platform overview.

Our realization of this conceptual approach for variability, feature modeling, CGUs and the code generator is called the Rhizome platform. Now, let us consider a typical scenario using the Rhizome feature platform (Figure 1-1). First, platform developers study existing software variants in an existing software product line and define source code templates. They study both the commonality and variability among these software variants and define templates using tags to capture variability. Usually, platform developers can find repetitive text structures in the source code for software variants in a product line. These recurring text structures are known as “code generation patterns” and they are used to model different forms of variability.

Our study has revealed several code generation patterns and there are undoubtedly more to discover. For example, a *list item* pattern can be used to generate a list of items with similar structures, such as a list of field declarations in a class definition. Another example is the *if-else* pattern in which the first item is an *if* conditional block, the last item is an *else* conditional block, and the rest of the items are *else-if* code blocks. Besides these intra-file level code generation patterns, there are also inter-file level code generation activities, including file copy (copy template file contents to newly generated files), and snippet join (join several code snippets together to create a new file). These patterns and activities are modeled as different CGU types. Platform developers need to implement various CGU processing modules in the code generator for these CGU types. When new code generation patterns or inter-file generation activities are found, platform developers simply add a corresponding CGU processing module to the code generator. Templates are stored in a template repository and each template is associated with CGUs and feature variability choices defined in feature models. Through a query mechanism to the template repository, the template files specified in the CGUs are used for code generation. In addition to creating template files and maintaining the template file repository, platform developers implement the code generator and continue to update its CGU processing modules when new generation patterns and activities emerge during product line source code analysis.

When the fundamental code generation platform is ready, a software designer, the expert on high-level feature design, composes a feature model that captures various feature design choices. The software designer can directly use the feature modeling language, possibly with the help of a GUI feature design wizard. The wizard helps validate feature dependencies and provides to the designer hints regarding correct design choices. A software designer is presented with available design choices for each feature and his job is to select the design choices to include into the final feature model. The validation work happens in the background with the design wizard. When all design choices are determined, a valid feature model is produced by the design wizard with a set of associated templates. The final step of using the Rhizome platform is executing the code generator using the feature model and templates as input. After the code generation process finishes, it outputs the generated source code that implements the selected feature design choices.

1.2 Contributions

1.2.1 Connecting Feature Modeling with Code Generation

Feature modeling and code generation serve different purposes in the software development. A feature model gives a high-level overview of the system functionality and data model. It is mostly used during the design phase of software development to help users and designers better understand system requirements. Code generation is a technique that is often seen in the implementation phase to reduce manual coding effort by leveraging recurring code patterns. Most existing research work focuses solely on either feature modeling or code generation. One of the biggest contributions of this dissertation is that the Rhizome connects these two pieces of work and permits automated code generation from high-level feature models.

The key abstraction connecting the modeling elements, which carray and translate design choices, with the generated source code is the code generation unit (CGU) element. Each CGU has three basic functions: converting feature design choices into parameters to be used in the code generator, querying the template repository to find matching templates for a given feature model and providing code generation instructions concerning what code generation patterns should be used and how to use parameters. The development of the CGU, and the associated CGU types associated with code generation patterns, is the most significant contribution of this dissertation.

The combination of templates and CGUs yields a practical and flexible solution that fills the gap of implementation knowledge and connects high-level feature models with low-level code generation activities. This is a general solution that can be applied to different application domains with low cost and high flexibility.

1.2.2 Feature Modeling Markup Language FeatureML

FeatureML is our feature modeling language used for recording feature design choices made by a software designer. This language contains elements for hierarchical feature diagram structures and design choice dependency relationships similar to those found in FODA [23, 24], FORM [25] and their extensions [25-28]. In addition, it also supports language elements for behavior features (data features that provide data values and structures are differentiated from behavior features that define dynamic activities), trace information between feature design choice and code generation, and code generation instructions via CGUs. This modeling language can express variability structure at any level of complexity. The modeling language is general and can be used independently of domain of interest or code generation framework. As for the variability model for a particular software product line, currently only an architecture design is available. The basic idea is that a central feature repository stores entity and application variability and their dependencies. A feature design wizard queries this feature repository during the design process to suggest valid design choices by resolving dependency relationships.

1.2.3 Entity-Oriented Feature Modeling Process

The Rhizome platform also makes a unique contribution to the feature modeling process as well. This feature modeling process is entity-oriented and different from the existing feature modeling processes in typical FODA [23, 24], FORM [25] and their extensions [25-28], where the design process mostly focuses on feature model structures, feature dependencies, and reference architectures that implement the feature model. The Rhizome platform targets not only feature modeling but also automatic generation of the feature model elements. One of the key design principles of the Rhizome feature modeling process is that the feature model should fit seamlessly with the automatically generated code implementation. Many modern software applications developed today follow the object-oriented design methodology and source code is implemented around the notion of objects. Using the Rhizome entity-oriented design process, feature design choices associated with each entity can be interpreted as parameters to generate code that implements the corresponding object.

The modeling process is centered on system entities, their behaviors, and applications that involve these entities. First, a designer identifies entities, entity properties, and entity relationships for a feature model. For example, in an online exam system, there is a *student* user entity. Each student entity has properties like *name*, *username*, *password*, *email*, *address*, etc. A student entity has associations with *classes* he is taking, *exams* he needs to take, *exam answers* he creates, and *exam reports* with grades for his answers.

Once the data and structural aspects of a feature model have been defined, a feature model designer tries to identify activities and applications associated with entities. For example, a student entity can take exams, view completed exam reports, etc. An entity can be a subject, an object, or both, for application services. For example, the application of grading an exam involves entities like *teacher* as the subject of the application and *exam answers* and *question answers* as the objects of the application.

During the second step (identifying entity activities and applications), it is likely that new entities not initially included from the entity definitions created during the first step. If this happens, a new round of iteration occurs until the inclusion of all the entities, their activities, and their applications into the feature model.

1.2.4 Code Generation Patterns and the Code Generator

During the process of code generator development, many interesting code generation patterns were identified. These generation patterns contribute to the general research field of template-based code generation. We classify generation patterns into two categories. One category includes patterns used for file-level code generation, such as content replication from one file to the others, the assembly of a new file using the content of other files, etc. These file-level code generation patterns can form a hierarchical generation structure, e.g. file 1 is assembled using the contents of files 2 and 3, which might be assembled using other files. The other generation pattern category is at the inter-file level. These patterns vary from the simplest string swap to more complicated ones such as list item generation and counter-related collection item generation. Details about these patterns will be discussed in later chapters.

The Rhizome code generator is a simple text processor that executes code generation instructions defined in CGUs. It processes each CGU and executes code generation instructions based on the CGU type. The code generator architecture is quite expandable—when a new code generation pattern is discovered, a new module in the generator is added to handle the new pattern and output proper source code text. Because Rhizome uses purely text-based processing, the generated code is in the same programming language as the template. The output code is not bound to any particular programming language.

The dissertation is structured in the following way. Chapter 2 defines a list of key terminology used to explain concepts in product lines, variability and feature modeling, and template-based code generation, throughout the dissertation. Chapter 3 provides related work and compares it with the Rhizome platform. Chapter 4 provides a retrospective view of the Rhizome project, including its predecessor Bamboo, to clarify key design requirements. Chapter 5 is a comprehensive domain analysis for our sample domain of online exam systems—it provides context for the various design choice options that have been implemented in the online exam system feature model and code templates. The next three chapters give a detailed discussion of these components including the feature modeling language (chapter 6), the template language and code generation unit (chapter 7), and the code generator architecture and generation workflow (chapter 8). Chapter 9 discusses future directions for the Rhizome platform and lists contributions to the generative programming and software product line research fields. Chapter 10 concludes the dissertation with a revisit of core contributions and lessons learned during the project.

2 Terminology

In this chapter, key concepts that will be used throughout this dissertation will be presented. These concepts are arranged from broader and more general terms to narrower and more specific terms. Related items are grouped together to make comparison and comprehension easier. The chapter starts with an explanation for abstract terms like product line, domain, and domain analysis. Then, it goes into detail about terms like feature, design choice, variability, variants, etc. Following, terms related to modeling variability and features are introduced, such as variability model and feature model structures. Later, we present terms related to feature modeling in the Rhizome platform, such as the FeatureML modeling language, code generation unit and template-based code generator. Other miscellaneous items that do not belong to specific groups are listed at the end of this chapter.

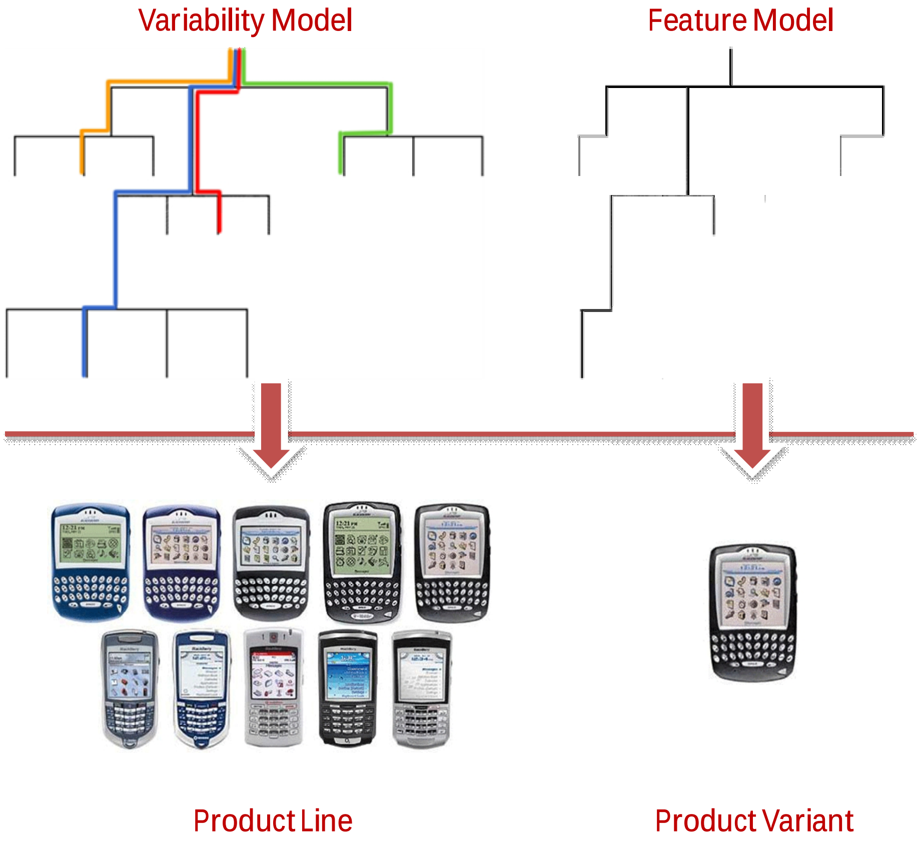


Figure 2-1. Product line concepts.

Product Family and Product Line: These two terms are used interchangeably and represent a group of systems that share common development platform components and have varying feature configurations. A product line provides high productivity and quality, shorter time-to-market, effective component reuse and reduced development costs for a group of similar products. Figure 2-1 uses a product line of

Blackberry mobile devices as an example for the product line concept. These mobile devices share common set of features such as core operating system, push emails, mobile phone calls, instant messaging, Internet faxing, Web browsing, etc. They also have certain unique features based on their configuration such as GPS, camera, video camera, Wi-Fi support, etc.

Software Product Family and Software Product Line: When the product line concept is applied to software development, the resulting set of related applications is a software product family [4] or software product line.

Domain: An abstract space where a set of related software systems can share common requirements, functionality, behavior, and terminology. A domain has a clear definition for the domain boundary to identify what systems are considered to be in the domain and what are considered to be outside of the domain.

Domain Analysis: The process of analyzing a group of related software systems to identify variability and commonality. Domain analysis produces various design artifacts, such as data-flow diagrams, feature models, data models, functional models, and reference system architectures. Together they describe the domain boundary, the static and dynamic aspects of a product, and a reference blueprint for implementation. This term can be used interchangeably with *Product Line Analysis* [2].

Feature: A characteristic system quality, either structural or behavioral, that provides capability to its users. There are two types of features based on their values: structural features and simple features. A structural feature has a hierarchical tree-like structure value: it consists of *child features*, which in turn might contain their own child features. These child features can also be called ***sub-features***. A simple feature is an attribute like color, shape, data type, etc., which can be assigned with simple values. Simple features either exist as dangling features (those without parent or child features) or as the leaf child features of a structural parent feature.

Feature Design Choice or Feature Design Option: There are usually multiple ways to design a feature and each way is called a feature design choice or option. There are two types of feature design choices: simple and structural, corresponding to their feature types. A simple design choice is selected from a list of values, such as an automobile color choice from a list of Athens Blue, Ivory Pearl, Platinum Graphite, etc. A structural design choice is obtained by making design choices for all of its sub-features. For example, a structural feature called car design can have two sub-features: color and engine type. One structural design choice for the car design could be Ivory Pearl color and V6 engine. Another possible structural design choice could be Black color and V8 engine.

Feature Design Space: This is an abstract term that represents a virtual space that contains that set of all possible feature design choices. This is the representation of the design knowledge for a particular domain. The process of creating a feature model is to explore this design space step-by-step until each feature design choice has been fixed. The physical realization of the feature design space is a variability model.

Variability: The word has its origin from Latin *variabilis* and *variare*, meaning "to vary". It is an abstract term to describe the property that a feature can have different design choices and adapt to different environment settings and user requirements using these design choices.

Variation Point: This is the location or place where variability can happen. For example, in a software product line, those features that can have multiple design choices are variation points. In source code templates, the tags representing various possible ways to implement a block of code are also variation points.

Variant: This term can have different meaning in different settings. A feature variant is equivalent to a feature design choice or a feature option. The phrase "a variant for a product line" means one product from a family of similar products in the product line. For example, Microsoft Office Suite [1] has a Student and Teacher version, a Professional version and an Enterprise version, each version exemplifies a variant for the Microsoft Office product line.

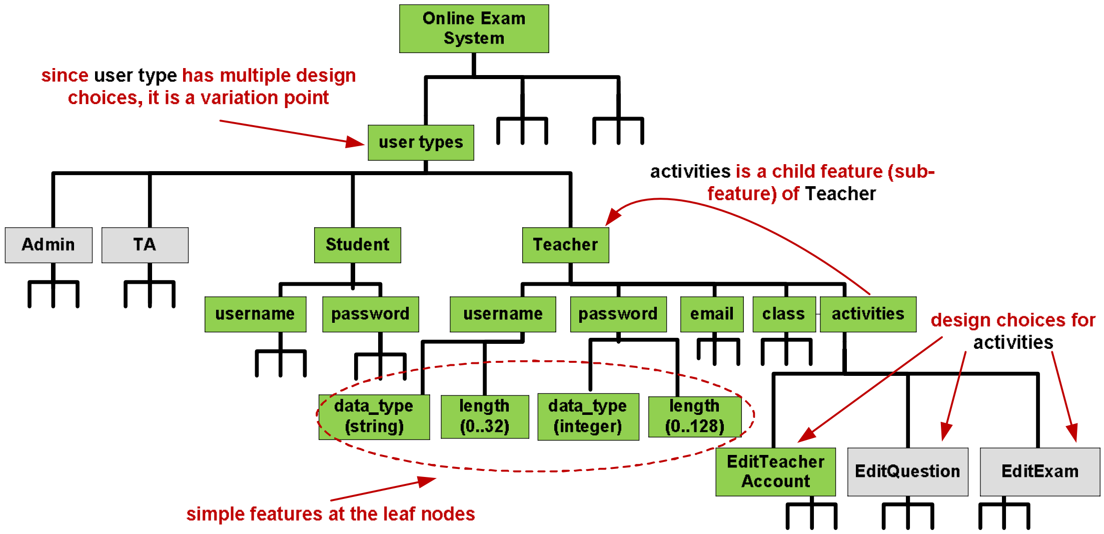


Figure 2-2. Variability model and feature model for online exam systems. Green rectangles represent design choices made for a specific feature model.

Variability Model: A variability model represents the set of all possible feature models for a product family. It is the realization of the product design space and it captures design knowledge about how to construct a valid feature model. Besides feature definitions, a variability model also contains structure, dependency relationships among features and constraint rules. Constraint rules are customizable dependencies based on feature containment structure, feature existence and feature options, etc. Figure 2-1 illustrates a variability model concept that has a tree-like structure. When this variability model is implemented in the domain, it corresponds to concrete product variants—different Blackberry models. Figure 2-2 shows a more detailed example for an online exam systems product line. It represents all possible feature design choices to design an online exam system. For example, there are four possible user types: Admin, TA, Student and Teacher and each user type has a different set of properties to select from.

Feature Model: A feature model consists of a collection of feature design choices. Each design choice corresponds to a particular feature. A feature model is derived from a variability model by making proper design choice selections for each feature. Because the variability model stores all possible ways to create feature models, a feature model is also referred to as an instance of a variability model. Figure 2-1 shows the relationship of a feature model to a variability model. Colored lines represent the feature design choices that a designer makes. Collecting together all of these design choices yields a feature model corresponding to one Blackberry model that implements the selected design choices. Similarly, in the example presented in Figure 2-2, the green boxes represent the set of feature design choices made for a particular feature model. A feature model is also known as a product model in the feature modeling literature.

Feature Dependency: A feature dependency is a dependency relationship among features. Dependencies are stored as part of the design knowledge in a variability model. The following types of dependencies are typical:

- *Existence dependency*: required feature and optional feature
- *Cardinality dependency for a feature group*: a parent feature must contain m..n child features ($0 \leq m \leq n,\ m, n \in Z$)
- *Customized dependency*: the most general form of dependency usually described using a formal dependency language. This type of dependency is a superset that covers existence dependency, cardinality group dependency plus domain-specific dependencies that involve conditions on feature design choices. For example, a customized dependency may require that if feature A exists and uses option 1, then feature B must exist and pick option 1 too.

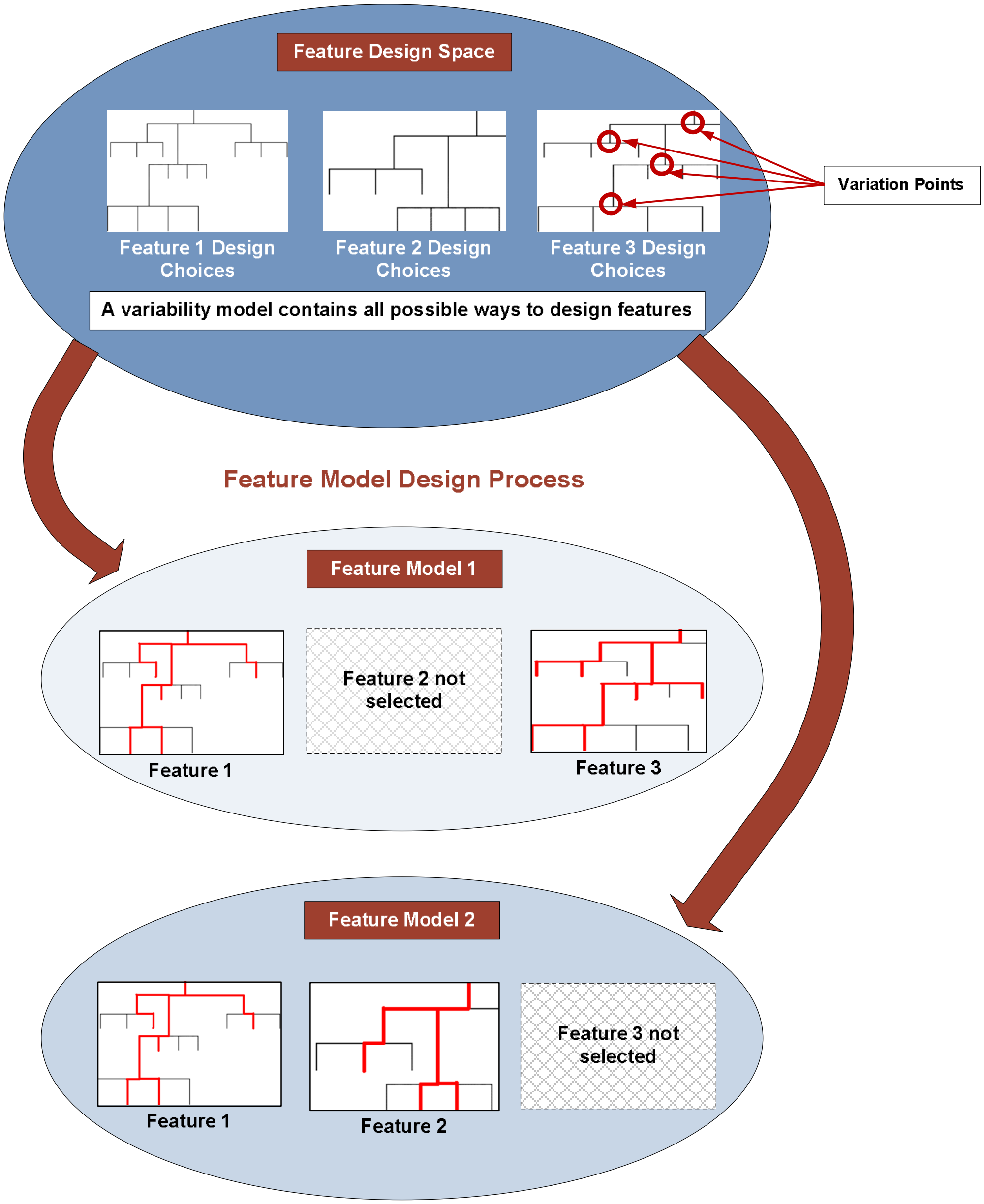


Figure 2-3. Feature modeling concepts overview.

Figure 2-3 gives an overview of the core feature modeling concepts such as variability model, feature model, feature design space, variant, variation points, etc. A variability model contains all possible feature design choices and implements the abstract concept of a feature design space. A variability model contains all possible features and when a designer wants to create a feature model, he just selects which features are to be included and how to design each feature. In Figure 2-3, there are two feature model variants: one with feature 1 and feature 3 selected and the other with feature 1 and feature 2 selected. For each selected feature, the designer has to make specific design choices at variation points (red circles) and his design choices are represented as red lines.

Variability Modeling Language: A language to define feature structures and record all possible feature design choices along with feature dependencies for a particular domain.

Feature Modeling Language: A language to define a feature model derived from a variability model. The language is used to record specific feature design decisions a designer makes during the feature model design process. Feature models described using feature modeling language must satisfy designated feature dependencies.

FeatureML Document: A specific feature model described in the Rhizome feature modeling language FeatureML. Feature model concept is an abstract term and feature model configuration is the physical representation (text files, XML documents or model files in specific languages) to implement a feature model. In Rhizome platform, a feature model configuration is an XML document that must conform to the FeatureML language.

Code Generation Unit (CGU): A code generation unit connects a feature model configuration file with the code generator. It contains instructions to the code generation engine to carry out generation activities. Each CGU includes information such as template file paths and parameter expressions, which will be used by the CGU processing module in the code generator to produce final source code. There are many different CGU types, each with its own specific way of generating code.

Code Template: A template file is a special text file to capture both commonality and variability in source code. Commonality is represented as original source code contents extracted from domain applications and these contents stay unmodified in the generated code. Variability is expressed using special markers such as place holders and tags. During the code generation process, these markers are replaced with generated source code corresponding to a selected feature model configuration. There is a specific CGU type called “tag expansion” that handles this kind of template-based generation.

Parameter: A variable definition that is used widely in the Rhizome platform. It carries design decision data translated from design decisions in the feature model. The code generator uses the parameter values to generate code using the templates.

Placeholder: A special string format embedded in a template. It is used for global string replacement, one of the code generation patterns. The placeholder has a format of $placeholder$$ or $placeholder.operation$$. The operation is the extra textual processing function to be applied to the placeholder value. For example, $UserEntityName$$ is a placeholder to be replaced by the name of a user entity; $UserEntityName.LowerCase$$ performs an extra function to change all characters to the lower case for the value of UserEntityName placeholder.

Tag: A special XML document embedded in a Rhizome template to represent places where dynamic content is going to be generated. The dynamic content replaces the tag in the final generated code. A tag is constructed using a template language.

Code Generator: A program that takes a feature model configuration file and generates source code that implements the feature design choices contained in the model configuration.

Model Driven Development (MDD): This is a software development paradigm that treats semantic software models as first class entities and allows developers to work on high-level models that precisely describe system functionality. There can be different levels of abstraction for these models. Low-level models are automatically generated from high-level models using model transformation and code generation technologies. One best known piece of work is Model Driven Architecture (MDA) [22] from Object Management Group (OMG) [29] and Eclipse Foundation has implemented many tools and frameworks such as Eclipse Modeling Framework (EMF) [30], Graphical Modeling Framework (GMF) [31], etc.

Meta-modeling: This term is often seen in MDD work and it refers to the process, syntax notations and utilities to design modeling languages in a layered architecture. Meta-modeling helps create models that are extensible, portable and exchangeable. Although the standard does not enforce how many layers you can have, MDD typically contains four layers [32]: meta-metamodel (M3), metamodel (M2), model (M1) and model object instance (M0), where models at each layer are defined using a modeling language constructed from its upper layer. For example, in an online exam system, an instance of this system filled with actual user and exam data is at M0 layer, the code that implements this system is at M1 layer, Java programming language that is used to implement the system is at M2 layer, and mathematic syntax to define programming languages is at the M3 layer.

3 Background and Related Work

Substantial prior work has been performed in the field of software product lines, software feature modeling, automatic code generation, and model-based automatic software development. In this chapter, we present representative work related to the Rhizome platform and discuss its commonality and difference. First, each related work is introduced and compared with Rhizome. Then, an "apples-to-apples" comparison of the related work is presented using various criteria.

3.1 Related Work Overview

3.1.1 Domain Analysis

Domain engineering studies the methodologies and processes used to create and manage product lines. Parnas [4] was among the first ones to shed light on the design and development of similar software systems called "program families". He compared three methods for developing program families: the classical method of sequential development and two new methods of stepwise refinement and information hiding module specification. Jim Neighbor's work [33] first addressed the problem of domain analysis and domain modeling. McCain [34] proposes a product-oriented model where reusable components are developed as products and later classified and categorized for future reuse. Prieto-Díaz provided a domain analysis method and survey study for software domain analysis with examples, definitions and key research issues [35, 36]. The Prieto-Díaz method defined a domain boundary, user roles, as well as inputs and outputs for the analysis process. He also emphasized the continuous refinement nature of the process where new systems are developed by reusing existing components and these new systems update an existing component library to enhance reusability.

Combining previous work in the product line research field, researchers at the Software Engineering Institute developed Feature-Oriented Domain Analysis (FODA) [23, 24] as a domain analysis process to collect domain knowledge, study product family structure, and create reference architecture for product family applications. It has become the best-known domain modeling approach. The basic modeling concepts in FODA are abstraction and refinement. Abstraction is the process of studying existing software applications and generalizing the functionality and designs into generic domain elements. Refinement is the process of creating specific software applications from generic domain elements by adding factors that are unique for individual applications. The FODA process emphasizes the identification of user-visible features in a software product line. The assets such as various domain models and the reference architecture produced during a FODA process focus on features—this is why the process is called “feature-oriented”.

FODA consists of three consecutive steps: context analysis, feature analysis and architecture analysis. Context analysis produces a context model that includes a structure diagram (describing relationships between the target domain and other connected domains) and a data-flow diagram (describing data flows between the target domain and connected domains with communication entities on the arcs) to define the domain boundary and data exchange between the target domain and its environment. Feature analysis creates a feature model (describing feature structures and dependencies), an Entity-Relationship (E-R) model, and a functional model. The E-R model defines the entity structures using two relationship types: consist-of type for aggregation and is-a type for generalization. The functional model graphically depicts dynamic behaviors using a data-flow diagram and a finite state machine. The architecture analysis creates abstract architectural reference models as the blueprint for system implementation. This reference architecture has four layers from top to bottom: domain architecture, domain utilities, common utilities and systems. The domain architecture layer shows the concurrent domain processes and their inter-connections. The domain utilities layer describes the packaging of functions and data objects into modules and the inter-connections between them. The common utilities layer contains utilities that can be used across different domains, e.g. a message queue, an event handler, etc. The system layer includes utilities and services provided by the operating system and programming languages.

The Feature-Oriented Reuse Method (FORM) [25] and Feature Oriented Product Line Software Engineering (FOPLSE) [28] projects extended FODA to promote component reuse in product line software development. FODA focuses on feature modeling in the requirements identification and composition phase. FORM and FOPLSE build connections between a feature model and software components so that feature reuse automatically maps to component reuse. FODA and its extensions have been used in applications such as telecom systems [37], and bulletin board systems [25].

Feature dependency and interaction have emerged as a key research problem in the feature modeling community. For example, the original FODA and FORM approaches can express simple relationships like “requires”, “exclusive or” and “optional”. Czarnecki [26] extended FODA notations and formalized these relationships using cardinality in the form of <m..n> where m is the lower bound and n is the upper bound. Feature grouping, feature diagram reference and feature attributes are also added in this work.

Czarnecki implemented this extended modeling capability in the Feature Model Plugin (fmp) [27, 38] for the Eclipse Integrated Development Environment (IDE), which acts like a design wizard for feature modeling activities. The Eclipse Modeling Framework (EMF) [39] is used in fmp to represent the feature diagram with a tree-based editor to select what features to include or exclude and what design choices to pick for each feature. Global constraints are defined using the Object Constraint Language (OCL) [40] and a valid feature solution space is computed using a Binary Decision Diagram (BDD) [41] algorithm to address Boolean satisfiability problem, also known as SAT problem. For example, consider a feature model containing four features, A, B, C, and D, where the designer has already made a design choice for A but not for the other three features. At this point, the feature solution space contains all valid combinations of feature design choices for these three features. However, some of these combinations may not be feasible or reasonable. Based on the OCL constraint computation, fmp can automatically select or deselect feature options for the designer, and can provide guidance until all feature design choices are selected.

Building on the cardinality-based notations for feature modeling, Czarnecki next worked on the partial feature modeling problem, where a partial feature model is a feature model that still has some of its feature design choices unfixed. Czarnecki proposed a staged configuration process to perform step-wise refinement on a feature model until a fully specialized feature model is created [42, 43]. The intermediate partial feature models between the initial variability model (with none of its variability resolved) and the final feature model (with all of its variability resolved) are called “specializations”. An Eclipse plugin called “FeaturePlugin” for feature modeling [44] is a proof-of-concept for this specialization idea.

Our work was initially inspired by the variability modeling aspects in the FODA work. Compared with the FODA process for domain analysis, our feature modeling process is lightweight. Our entity-oriented modeling process defines domain entities first, and then determines dynamic behaviors related to these entities. These behaviors introduce new entities and the new entities may exhibit new behaviors at a finer grain. So, entities and behaviors are the only two elements to shape the feature model and there are no other FODA models, diagrams or reference architectures. Our entity-oriented analysis approach is iterative, and keeps running until no new entities or behaviors can be added.

Instead of using graphical FODA modeling elements, a textual feature modeling language called FeatureML is used in Rhizome. It defines static entity structures, dynamic entity behaviors and dependency relationships. Graphical and textual modeling languages serve different audience. Human designers may feel more convenient to work with graphical modeling elements since graphical models are intuitive and editing a graph is much easier than editing text with complex syntax. But from a code generation perspective, a textual feature model is the only way as input specification to drive the code generator. A graphical model is difficult to parse and edit using computer programs.

After a feature model is created, automatic code generator uses it to produce a running software system that implements the feature model. The FODA work focused just on the feature modeling aspect and did not substantively engage automatic software system generation.

3.1.2 Variability Modeling using Unified Modeling Language (UML)

Most of the work in variability research for software product lines uses UML extension mechanisms such as stereotypes, tagged values and OCL constraints to add variability modeling support to existing UML models. This is because UML is the most popular standard for representing high-level system design and there is a rich set of tools based on the UML standard. By extending UML to add variability notations, existing UML models are not affected, and designers familiar with UML do not need to learn another variability language and existing tools can be leveraged to add variability modeling capabilities. Clauß [45] introduced stereotypes and tag values to class diagram and component diagram to express four types of feature dependencies: mandatory, optional, alternative and external (external represents context information, such as platform, programming language, etc.). Gomma [11] introduced two stereotypes, extend and include, to the use case diagram. Extension use cases can be derived from a base use case through an extension point using the extend stereotype. The include stereotype is used to share common functionality in use cases.

Another line of software product line research explored making variation point (VP) and variant (V) first-class entities in UML diagrams so that variability information will not be localized in feature models and become difficult to track. Work has been proposed for meta-models that include VP, V, product artifacts and dependencies as first-class entities. The meta-model and graphical notations have also been integrated into UML use cases in [46]. Pohl et al. extend the idea and apply this modeling approach to all development stage and all UML diagrams to manage variability in requirement, design, implementation, and testing artifacts [5].

Compared with our work, this UML-based software product line variability work focuses on the high-level designs of the product family, not on customization, generation and composition of software applications. In our work, substantial effort has been put into the implementation of the code generation that has built-in variability support. Generated source code can be directly traced back to the design choices a software designer makes. However, our variability modeling and code generation approach is based on purely text template processing without any semantic models. Therefore, when our code generator needs semantic knowledge to produce code, our approach will simply hit its limit and we need to manually provide prepared code templates and snippets. UML models, on the other hand, carry a lot of semantic information and can act directly like an abstract representation of the application. So, once general code generation tools that create code directly from UML models can incorporate variability modeling capabilities, UML-based software product line will be the a promising solution and handle even most complicated code generation cases that involve domain semantic knowledge.

3.1.3 XFeature: An XML Feature Modeling Framework

Besides graphical feature modeling languages, there are also textual languages. Graphical languages are more convenient for human to understand while textual languages are easier to implement in modeling tools. In this section, we talk about XFeature [12, 47, 48 2004], a feature modeling and variability modeling framework built using various XML technologies. XFeature is an environment to automate software application development in a product family. XFeature has the typical four-level architecture (M3-M0) seen in many model driven development frameworks. For example, a meta-meta-model (M3) is the abstract syntax to define programming languages. A meta-model (M2) defines concrete syntax for a particular programming language such as Java. A model (M1) is a system implemented using a particular programming language such as Java programming language used to implement our online exam system. The model instance (M0) is an actual running system that contains actual data.

In XFeature, a family meta-model (M3) is an XML schema that defines abstract facilities to describe product families. A family model (M2), an XML document is derived as an instance of the family meta-model. This family model describes an actual product family such as the online exam systems. Then, an XSL program automatically generates application meta-model, an XML schema, using the family model as an input. The application meta-model (M2) represents the variability model for the product family and it includes all possible feature design choices and in various scenarios. Finally, an application model (M1) that represents a fixed set of design choices for a real application can be derived from the application meta-model.

In XFeature, constraints are classified into two types: local constraints and global constraints. Local constraints are those defined on combinations of sub-features that are children of the same feature, e.g. a parent feature must have one of the three child features. Local constraints are implemented using the XML Schema language. Global constraints can involve any features at any vertical or parallel structures. Global constraints are defined separately from the variability model using a separate XML schema document known as a constraint meta-model. Designers use this constraint XML schema to define their global constraints in an XML document and then a global constraint compiler will create an XSL checker program using that XML document as input. This XSL checker will then run on the application feature model to see if it meets those global constraints, responding that either passes the validation or fails the validation.

There are four types of global constraints: require (AND), exclude (XOR), if-then, and custom constraints. If-then constraint means if “constraint A holds” then “constraint B must hold as well”. Custom constraints use XPath expressions to define arbitrary constraint relationships. XFeature uses a dialect of XML schema language called Schematron [49], which is a rule-based schema language good for expressing rule-like constraints.

The implementation of XFeature is an Eclipse plugin and graphical editor based on the Graphical Editing Framework (GEF) from Eclipse Foundation. It has a visualization customizer that allows users to pick icons and lines for their modeling elements. This visualization customizer is implemented using a meta-display-model that defines all possible visual elements and an application display model derived from this meta-display-model.

There are similarities between XFeature and our project. It uses the full stack of XML technologies and tools to represent meta-model and model: XML schema to define the meta-models and XML documents to define the feature model and application model. Its implementation as an Eclipse plugin has a very appealing and easy-to-use GUI interface for users to build feature model with rich visual customization. The primary difference is that XFeature is exclusively on feature modeling tool and does not have an automatic source code generator that translates a feature model to corresponding application source code. The way XFeature handles global feature constraints is more powerful than our approach—Rhizome only defines feature dependency relationships, which means one feature must exist if another existing feature is dependent on it. XFeature externalizes constraints using an XSL document instead and it can handle more constraint cases than our "requires" constraint.

3.1.4 Design/Shape Grammar and Typed Feature Structure

In the world of Computer Aided Design (CAD), researchers have been long studying the nature, structure and expression of design spaces. A design space is a multidimensional combination and interaction of variables and rules and a variability model is one way to capture and present a design space. A design space consists of starting states, terminal states, design rules and design elements. The design process becomes a process to explore the design space, and the path from a starting state to a terminal state becomes a design. Design/shape grammars and typed feature structures are two closely related design space research fields described below.

The basic idea of a design/shape grammar is to automatically generate designs using rules (rules represent existing design knowledge) and design elements. All possible designs in the space can be generated by following the rules. A designer composes design rules to match design requirements and generate the final optimal design solution. Design/shape grammars have applications in the fields of fine arts, architecture, landscape design, etc. Andersson [50] gives a formal definition for a design grammar (DG) and a set of terminologies. In [51], a formal definition of shape and spatial relations are given by combining shape algebra and symbolic logic. This logic has been used in applications like floor planning, identification of spaces and geographical information systems. Some developers on contextfreeart.org took this idea of context free grammar and applied it to generative computer graphics. Giving a starting state and a set of simple drawing rules, astonishingly beautiful graphics can be generated automatically.

Feature structure (FS) is closely related to design/shape grammar. Instead of using design rules, Feature structure uses special data structures to represent and store variability. A feature structure is essentially a structured set of attribute value pairs. The value of an attribute can be atomic or more complicated values like a sub feature structure, a set, or a list. There are many ways to represent a feature structure such as matrix, tree, or directed acyclic graph (DAG). Feature structures have been used extensively in the field of linguistics to model phrase structure grammars such as generalized phrase structure grammar, lexical functional grammar, and head-driven phrase structure grammar (HPSG) to generate formal languages.

Typed feature structure (TFS) [52] combines the ideas of unification-based grammar formalisms (feature structure), knowledge representation languages (inheritance) and logic programming (logical variables and declarativity). In typed feature structure, the feature structures are used to define types and build knowledge networks with the inheritance relationships among nodes. Typed feature structure has been successfully used in architecture design as an efficient structure to hold design knowledge and shape the design space [53-55]. The design space contains all possible ways to construct and combine the design elements and TFS constraints expressed as rules can validate the design choices and filter out invalid combinations. Designers can use design requirements to query this structure and efficient algorithms such as π resolution (used in Woodbury's work) can calculate all possible matching solutions that are relevant. Note that these solutions can be partial solutions, meaning there can still be unresolved variability, and further design decisions may be required to achieve a concrete design specification.

Most of the design grammar and feature structure work have their applications in linguistics and natural language processing. Head-Driven Phrase Structure Grammar (HPSG) [56], one of the well known generative grammar frameworks, employs TFS to represent the structure of grammatical categories. Natural language grammars such as those for English, German, and Japanese have been made available using HPSG. Carpenter’s book [52] presents three linguistic application examples: unification-based phrase structure grammars, definite clause programming, and recursive type constraints. Although we are not natural language processing or linguistics experts, we believe it is quite possible that one of these generative grammar frameworks such as HPSG can be used to model a software application domain, since programming language syntax can be viewed as a set of formal rules and source code as chunk of text conforming to these rules. Textual descriptions of feature models, instantiation of the variability model, can be automatically generated according to the grammar model.

After all, what we need to represent a software application domain is a way to capture domain knowledge and express variability and constraints among design elements and choices. So, our variability modeling language is quite similar in purpose to these grammar frameworks, although the actual syntax can differ. Our variability modeling language is quite complex already. As we found out, the linguistic grammar frameworks are even more complicated. The graphical representations of TFS can grow to a large size and are not as intuitive as our XML representations, which can be self-explanatory.

3.1.5 Hamilton 001 Tool Suite

001 Tool Suite [57, 58] is an automatic software design and development toolset based on a paradigm called Development before the Fact (DBTF). It is a very successful commercial system that has been deployed in many large US military and government projects. The tool suite is a modeling environment to build abstract system models that represents the structure, operation and behavior of real systems. The modeling elements collectively form an abstract layer on top of specific programming language, computing platform and coding libraries. Therefore, as a designer, you just use these abstract model elements without worrying about specific underlying technology stacks. The open architecture also allows expanding these model elements when new technologies are needed.

The suite uses a language called 001 AXES that creates structures, relationships and interactions among objects known as Function Maps (FMaps) and Type Maps (TMaps). FMaps define operations associated with object types using three primitive control structures: join, include and or. These structures help construct parent operations using child operations. TMaps define type properties and structures using primitive structures such as TupleOf, one of, OSetOf, etc. Static analysis on preventative properties and dynamic analysis on user intent properties are performed on FMaps and TMaps to make sure the model is consistent and logically valid. Then, in the next implementation phase, the model is either being transformed into source code via Resource Allocation Tool (RAT) or converted as simulation studies for Xecutor. Finally, at the executing stage, the generated code is compiled and linked into executables.

The 001 AXES language and the system oriented objects (SOO) modeled using this language are independent of any programming language and computing platform. The SOOs are created throughout the whole development life cycle from model definition, model analysis, to code generation and execution. The 001 Tool Suite provides a rich set of GUI editors for model composition and resource management at each development phase. Code representing the same semantic model can be generated for different programming language such as APL, Lisp, Pascal, FORTRAN, ADA, COBOL, C, C++, Java, etc. and various platforms such as UNIX, NT, IBM mainframe, AS 400, etc. The open architecture of the generation tool RAT also allows extensions to any language and platform as long as corresponding RAT specifications can be provided. Documentation, test cases and English representation (like pseudo code) are also automatically generated with the source code. The tool suite also includes features like version control and automatic requirement acquisition from plain English documents or other requirement formats.

Compared with our approach, this tool suite is very powerful, advanced and complete. The notation of static entity variation point vp_entity and dynamic behavior variation point vp_application in our system are quite similar to the notation of TMap and FMap in 001 Tool Suite. In the early design stage of our system, VP_APPLICATION element is used to store input and output parameters and various control flow structures like while_loop, if_elseif_else, etc. This is essentially what TMap and FMap are used for in 001 Tool Suite. However, we choose a simpler way to implement the code generator. In the current version of our system, vp_application only hold textual description and dependency information and there is no abstract layer of primitive operations in our system. We just rely on the type system for the programming language used in templates and directly implement these vp_application elements as source code in the templates. The code generation in 001 Tool Suite is based on a mathematical and formal language, which is a general model-based code generation approach. In our system, human developers are responsible to capture implementation knowledge in templates. This is a simpler and more flexible code generation approach. New code generation patterns can be discovered and integrated into generator easily. The templates are the single point of control for code generation instead of spread generation semantics and instructions across different models. Our Rhizome system focuses on the feature structure of a product family, dependency structure of the feature variability and impact of design choices. It is not intended to a comprehensive solution for automatic software development for different platforms and programming languages.

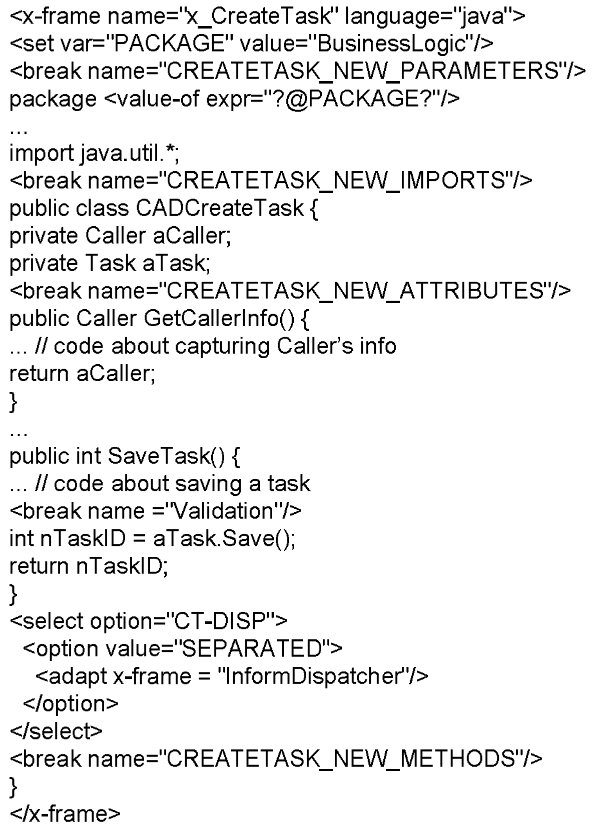
3.1.6 XML-based Variant Configuration Language

The XML-based Variant Configuration Language (XVCL) [13, 59, 60] is another work that is similar to our variability modeling framework. The authors based their work on the concept of frames. Originally, frames were a concept to represent knowledge and perform queries and assertions on it using predicate calculus [61]. Later, Bassett [62] first introduced the frame idea into the software reuse field and observed significant achievements in cutting large software project cost and development cycle. A frame is the smallest information unit, and multiple frames can be combined to construct a hierarchical structure where frames “link” to each other using embedded frame commands like COPY, INSERT, BREAK, SELECT, and REPLACE. Source code is associated with the frames for component reuse. The entire frame structure represents the variability model; the higher in the structure the more variability a frame contains. The root frame contains all the variation options. The process to derive a concrete product model from this frame structure is to use a specification as input and do a depth-first walk until all variability is resolved. The XVCL project developed their frame implementation called x-frame using XML. Product line assets are represented as x-frames that describe both commonality and variability in a product family. A specific product can then be built by composing and adapting x-frames.

The XML modeling language XVCL is like a template language that expresses variability in software components including source code, requirements and documents. For example, software components can be modeled as x-frame templates using XVCL to express dependency on external x-frames (using ADAPT, INSERT, REPLACE commands), branching conditions (using SELECT command for if-else loop and WHILE command for while-loop), place holders for unexpected variability to be resolved by ancestor x-frames (using BREAK command), single value or value list definition (using SET command), and expression evaluation (using VALUE-OF command). To get a customized model (meaning to remove all the variability and generate valid code), a specification x-frame called SPC is defined as input to the XVCL processor. SPC provides fixed variable values and defines adaptation or composition of lower level x-frames. Using SPC as an input, variability in the x-frame structure is removed and a customized system is created, that is, valid source code for software components is selected and then knit together to form a running software system. Figure 3-1 is an x-frame for CreateTask activity and Figure 3-2 is an SPC x-frame where this CreateTask x-frame is referenced.

```
<x-frame name="x_CreateTask" language="java">
<set var="PACKAGE" value="BusinessLogic"/>
<break name="CREATETASK_NEW_PARAMETERS"/>
package <value-of expr="?@PACKAGE?"/>
...
import java.util.*;
<break name="CREATETASK_NEW_IMPORTS"/>
public class CADCreateTask {
private Caller aCaller;
private Task aTask;
<break name="CREATETASK_NEW_ATTRIBUTES"/>
public Caller GetCallerInfo() {
... // code about capturing Caller's info
return aCaller;
}
...
public int SaveTask() {
... // code about saving a task
<break name ="Validation"/>
int nTaskID = aTask.Save();
return nTaskID;
}
<select option="CT-DISP">
  <option value="SEPARATED">
    <adapt x-frame = "InformDispatcher"/>
  </option>
</select>
<break name="CREATETASK_NEW_METHODS"/>
}
</x-frame>
```

Figure 3-1 The x_CreateTask x-frame [13].



```
<x-frame name="x_CreateTask" language="java">
<set var="PACKAGE" value="BusinessLogic"/>
<break name="CREATETASK_NEW_PARAMETERS"/>
package <value-of expr="?@PACKAGE?"/>
...
import java.util.*;
<break name="CREATETASK_NEW_IMPORTS"/>
public class CADCreateTask {
private Caller aCaller;
private Task aTask;
<break name="CREATETASK_NEW_ATTRIBUTES"/>
public Caller GetCallerInfo() {
... // code about capturing Caller's info
return aCaller;
}
...
public int SaveTask() {
... // code about saving a task
<break name ="Validation"/>
int nTaskID = aTask.Save();
return nTaskID;
}
<select option="CT-DISP">
  <option value="SEPARATED">
    <adapt x-frame = "InformDispatcher"/>
  </option>
</select>
<break name="CREATETASK_NEW_METHODS"/>
}
</x-frame>
```

Figure 3-2. A partial SPC for a CAD system [13].

XVCL is an XML approach to model variability with frames being used as a format to store knowledge. The source code components are all prepared beforehand during the process of domain analysis and product family source code construction. The X-frame format is used to add markups in use cases, source code and design choice SPC documents. XVCL framework knits together source code pieces according to these x-frame descriptions instead of generating source code. Our template-based code generation approach has a more powerful template language than XVCL, and it can generate different patterns of code snippets on the fly instead of using prepared source code. This template approach saves substantial effort as compared to the XVCL approach because XVCL needs to prepare beforehand all possible combinations of source code snippets. For example, one template might cover multiple similar entity source code files or many code snippet scenarios. Whereas using XVCL, those cases need to be prepared literally one by one.

Rhizome also supports a hierarchical code generation by using the snippet_join code generation unit type, where final source code can be generated by aggregating code snippets generated by other templates, similar to how INSERT and BREAK commands work in XVCL.

3.1.7 Feature Oriented Programming and AHEAD Tools

Feature Oriented Programming (FOP) [63] [64] is based on the idea of “stepwise development”: complex programs can be built using simple programs by adding features incrementally. The AHEAD tool suite is a set of FOP tools that implement hierarchical algebraic composition operators to build complex systems from simple program fragments. Figure 3-3 shows the workflow of the AHEAD tool suite. Features are first described using Jak, a language that extends Java. Then, these jak feature files are aggregated together into a single jak file using a composer. In the next step, a translation tool called jak2java converts the jak code into Java source code, which is then compiled into executables. There are two types of composers: jampack and mixin. Jampack is simply a macro expansion type of composer—it flattens inheritance structures into a single class simply by appending all inherited fields and methods to rewrite the original child class definition. A mixin directly translates the refinement hierarchy in jak files into an inheritance hierarchy in Java classes. Besides jak files, other resource types like equation files, grammar files, design rules can also be composed in a similar way using AHEAD.

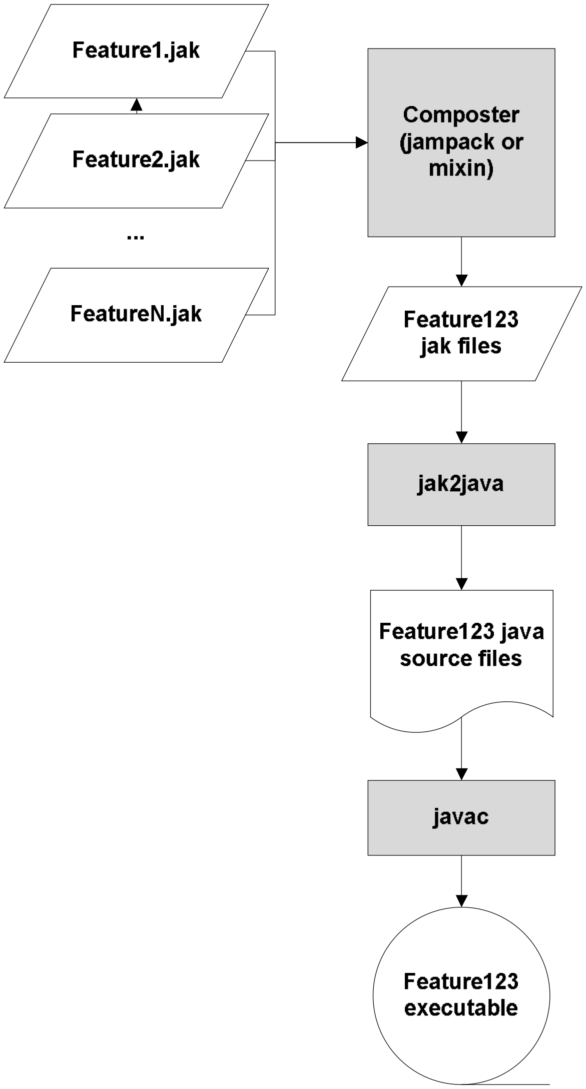


Figure 3-3. AHEAD tool suite workflow overview [63].

Feature dependency and constraints in FOP are expressed using design rules. These rules take the form of a FODA feature diagram, which can then be mapped into an interactive grammar and finally into propositional formulae [65]. Thus, AHEAD can use a Satisfiability (SAT) solver to check propositional formulae and validate the design rules. Dependency types supported in AHEAD are required, optional, AND, OR, and XOR. So, inputs to the SAT solver are a list of feature variables $\{v_1 \ldots v_n\}$ and a set of propositional formulas $\{f_1 \ldots f_m\}$. Each feature variable marks the existence of a particular feature or feature option. Possible values include True, False and Unknown. If feature variable values can satisfy all the formulas, then the feature model is valid.

FOP has a quite long history and significant impact on the field of feature-based software product lines. Besides the basic ideas introduced above, Batory's group also tries to integrate Aspect Oriented Programming with FOP to model features in terms of aspects [66]. Compared to Rhizome, the aspect of data modeling is pushed up into the feature model and therefore allows code generation using CGU with a rich set of code generation patterns. FOP just knits existing code pieces together and there is not much code generation.

3.1.8 Eclipse Modeling Project

The Eclipse Modeling Project [39] is a collection of model driven technologies for software development. Under its umbrella are many well-known projects such as the

Eclipse Modeling Framework (EMF), Graphical Modeling Framework (GMF), Model to Model Transformation (M2M), Model to Text Transformation (M2T), etc.

EMF is the core part of the Eclipse modeling project. It provides a modeling framework and code generation tools based on a structured data model. The input model specification given to the EMF generator is described in standard XML Metadata Interchange format (XMI). The output from the generator includes Java entity classes and adapter classes for the model and a basic tree-based graphical editor to create model instances.

GMF is a generator to produce domain specific graphical editors based on EMF models and a graphical framework Graphical Editor Framework (GEF). For example, you can define a domain model and graphical model for electronic component design. Then, GMF can use these inputs to generate a graphical editor, which allows users to drag and drop components to design layouts for PCB boards, make connections, enter parameters, etc.

M2M and M2T are model transformation frameworks in the Eclipse modeling project. M2M defines model-to-model languages and tools and M2T defines model-to-text tools such as generating source code from an UML model.

The Eclipse modeling project has seen wide adoption both in industry and in the open source community. It provides a common platform, modeling languages and tools for model driven software development. This project conforms to OMG standards and provides reference implementations for many of these standards. Because this is a general-purpose model driven technology stack, its emphasis is not on modeling and generating features, feature variability and feature interactions. But with such a solid and rich technology stack, it is possible to develop frameworks that fulfill such functionalities. This approach can solve many existing problems with our framework: difficulty and ambiguity in translating domain semantics into code can be solved using UML models that act as an abstract visual model for a software application. Manual work in updating templates and feature modeling language can be saved by using model transformation languages and automatically updating a feature model. A graphical feature model editor tool may be easily generated using GMF and OCL to help visualize the design process and enforce constraints.

On the other hand, Rhizome is specifically designed for automatic code generation based on feature models and feature modeling elements are primary focus of the FeatureML language. Eclipse modeling is one implementation of Model Driven Development and lacks the richness for feature modeling support. Eclipse modeling also lacks sufficient domain knowledge to generate complete applications in many cases—MDD is best suited for data-oriented computing, such as data access layer for a Web application.

3.2 Related Work Comparison

In the previous section, we briefly introduced research projects related to our work. They are divided into two categories: work that focuses on variability modeling methodology and does not provide code generation; and work that has both modeling and software generation capabilities. In this section, we use several criteria to compare these systems side by side so that readers have a better idea where our work is positioned.

3.2.1 Variability Modeling Language

a) **Language Form**: There are three types of expressions in general: textual, graphical, and both. The Rhizome modeling language is a textual language.

- **FODA and its extension work**: They use a graphical form called feature diagram. This is a tree-like structure in which a parent feature consists of child features. Dependency relationships such as ***Required***, *Optional*, *AND*, *OR*, *XOR*, etc. are expressed using special visual expressions on the arc ends. The original FODA feature diagram only had notations for Required and Optional constraints. Later, in FORM and its extension work, group notations were added to express OR and XOR in required groups, also added was a notation for optional groups. Czarnecki's work simplified these feature diagram notations using cardinality on both ends of an arc. More complex dependencies are described as textual annotations on the feature diagram.

- **XFeature**: This is a pure XML solution that uses XML, XSD, XSL, and Schematron text formats to express meta-model, feature model, display model, model transformation and constraint rules. Tool support is a suite of Eclipse plugins with nice graphical wizards and editors. The graphical elements used in the GUI editor for the feature model are quite similar to FODA-style feature diagrams with cardinality support.
- **Software Product Line**: The typical approach is to extend UML diagrams to express variability modeling. One way is to use UML extension mechanisms like stereotype, tagged value, and the OCL language. Another way is to add two UML modeling elements VP (Variation Point) and V (Variant) as first-class elements in UML.
- **Design/Shape Grammar and Typed Feature Structure**: A Design/Shape grammar can be described using a combination of shape algebra and symbolic logic. Typed Feature Structure can take many forms such as a matrix, a tree or a directed acyclic graph. One popular matrix form is called Attribute Value

Matrix (AVM) where attribute-value pairs can be defined using a matrix. Each attribute value is either a primitive type or a reference to another matrix structure.

- **001 Tool Suite**: Every model in 001 is constructed using Function Maps (FMaps) that express time characteristics and Type Maps (TMaps) that define space characteristics. FMaps are function graphs showing functional interactions between object states. TMaps are type graphs that define type hierarchy and relationships between types of objects. 001 Tool Suite has GUI editors to construct feature models using these map data structures.
- **XVCL**: Uses an XML language called XVCL to build a variability model consisting of many x-frames. These x-frames are knit together using embedded XML tags coded using XVCL to express dependency relationship, branching conditions, expression evaluation, etc. A specification called SPC contains variable values and design choices to solve variability. Using this SPC as input, XVCL creates a feature model configuration and source code that implements this feature configuration.
- **FOP and AHEAD**: Jak, an extension to the Java language to describe features.
- **Eclipse Modeling Project**: There is no direct variability modeling support, but many existing feature modeling and variability modeling projects rely on

Eclipse modeling technologies such as EMF. EMF provides a powerful platform to convert existing UML models, parse and retrieve model information, and generate model manipulation code with a GUI editor.

- **Rhizome**: FeatureML, an XML-based feature modeling language. Each feature model is represented as an XML document that conforms to the language schema. FeatureML also contains definitions for code generation units, which connect a feature model with the code generator to produce source code.

b) **Modeling Granularity**: Some related work models features at the conceptual level, which means that they can define feature structures to any extent, but they do not involve variability concerning implementation issues. Some work uses abstractions such as UML diagram elements, which are a step closer to implementation details. However, a UML model is still an abstracted model for the actual software system—we can implement the same UML model with different programming languages, and application frameworks on different platforms. The third granularity is to the source code level, where variability is directly expressed in terms of source code content. Rhizome covers variability and feature modeling both at the conceptual level and at the source code level using templates to translate semantic meanings between these two levels.

- **Conceptual Level**: FODA and its extension work, Design/Shape Grammar, Typed Feature Structure and Rhizome.
- **Abstraction Level**: XFeature, Software Product Line, 001 Tool Suite, Eclipse Modeling Project.
- **Source Code Level**: XVCL, FOP, AHEAD and Rhizome.

c) **Relationship Modeling**: Relationships between feature structures and feature values is one of the core components in a variability model. Feature models need to be checked using these relationships to see if they are valid. The term has different names in various related work: constraints, dependencies, rules, etc.

- **FODA and its extension work**: Simple relationships involving a single feature or features inside a group can be expressed using graphical notations. More complicated relationships are expressed in text and attached to a feature diagram as composition rules.
- **XFeature**: There are two types of relationships: local constraints and global constraints. Local constraints are like group relationships in FODA—they define relationships among sub-features of a parent feature. Local constraints

are defined using the XML schema language XSD, which can define many constraints on property values such as value type, value pattern, etc., and element structures like minimum and maximum occurrences, element type choices, etc. Global constraints cover feature structures at any level and are defined in a separate XML document using a special constraint XML schema known as a constraint meta-model. There are four types of global constraints: require, exclude, if-then, and custom type.

- **Software Product Line**: UML provides many extension mechanisms to define relationships such as stereotypes, tags and OCL. OCL is more useful for building code generation frameworks because it is a formal language and computer programs understand it well. Extension mechanisms like stereotypes and tags are more intended to be manipulated by human designers.
- **Design/Shape Grammar and Typed Feature Structure**: Logic programs define constraints among feature structures and there are special algorithms to check if feature structures are consistent using operations like unification.
- **Hamilton 001 Tool Suite**: The 001 AXES language is used to define constraints in Function Maps and Type Maps. Static analysis on preventative properties and dynamic analysis on user intent properties are performed on FMaps and TMaps to make sure the model is consistent and logically valid.

- **XVCL Work**: The XVCL language defines dependencies among tagged code pieces. These tags cover all possible design branches and all scenarios.
- **FOP and AHEAD**: Constraints are expressed using FODA feature diagrams. They are mapped to an interactive grammar and into propositional formula. This approach enables AHEAD to validate design rules using a SAT solver to check the propositional formulas.
- **Eclipse Modeling Project**: Sub-projects conform to OMG standards and use OCL to express feature model constraints. A sub-project OCL defines APIs for parsing and evaluating OCL constraints and queries on EMF models.
- **Rhizome**: Dependencies are expressed using XML elements and attributes in the FeatureML language. The ***Required*** dependency is further classified into Required dependency among entity variation points, Required dependency among application variation points, and Required dependency between entity variation points and application variation points. Beside Required dependency, FeatureML also allows a whole-part relationship similar to the composition relationship in UML. FeatureML also supports inheritance where a child variation point can inherit a set of feature design decisions from a parent variation point. Another dependency type in FeatureML is indirectly inplied by code generation unit depdencies: two feature variation points are connected by their corresponding code generation units.

d) **Application Domain**: Design/Shape Grammar and Typed Feature Structure has its unique application domain in linguistics and natural language processing since it is closer to what humans interpret as the variability in the design process. Other works including Rhizome can be applied to general application domains such as telecommunications, control systems, etc., since they create models suitable for computers to understand.

3.2.2 Code Generation

a) **Generation Mechanism**: There are many different mechanisms to perform code generation and three types are discussed here. The first type is to use a formal abstract modeling language to describe a generated application. The language elements are interfaces based on a virtual machine layer that hides programming language or computing platform details underneath it. The second type uses a template language to embed special tags in source code template files. These tags are expanded based on parameter values and context information at code generation time. The third type is an assembler that selectively picks source code snippets based on a given feature model and merge these snippets into final source code files.

- **Abstract Modeling Language**: The Eclipse Modeling Project uses the UML modeling language to build an abstract model for a software application. Code generation facilities are provided by the EMF sub-project. The Hamilton 001 Tool Suite has its own modeling language 001 AXES to create FMaps and TMaps. The Code generation tool RAT does the job of converting these abstract models into actual source code files.
- **Snippet Assembler**: XVCL and FOP/AHEAD fall into this category.
- **Template Language**: In Rhizome, both a snippet assembler and a template language are used. The template language is used to embed tags in a template file. The tags are processed by the generator to create code snippets and replace the tags. Other powerful template engines include the Java annotation framework [67] and the Tapestry template engine [68] for code generation.

b) **Model and Code Synchronization**: Synchronization is quite important since both a model and source code need constant updates. Most of the code generation process is model-to-code only, but there are also bidirectional generation approaches.

- **Model-to-code only**: Rhizome, XVCL, FOP/AHEAD and the Hamilton 001 Tool Suite.
- **Bidirectional**: Eclipse Modeling Project.

c) **Variability Tracing**: Variability tracing means the ability to tell how a feature design choice impacts the generated source code. For example, we can trace how and where a design parameter is used in the source code. Another example is during the process of upgrading an existing feature model, if we can trace the design choices in the source code, we can know where potential bugs might reside and pay special attention to those locations. None of the related work discussed in this chapter except AHEAD provides tracing capability. In Rhizome, it is possible to locate where design parameters by doing a global search on the parameter names. But variability tracing function is not yet available in Rhizome.

4 Design Requirements

The Rhizome system was highly influenced by strengths and drawbacks in our previous code generation system called Bamboo. Bamboo is domain-specific code generation framework for content management systems. It is based on a modeling approach called the Containment Modeling Framework (CMF) and has applications in three areas: automatic generation of content management system repositories, customized domain feature generation, and combination of features from different domains to create new hybrid features. Bamboo has successfully generated source code for content management systems, hypertext systems and version control systems based on feature structure and relationship descriptions.

Bamboo has several limitations that are difficult to resolve due to its initial design. Bamboo was first designed to test applications of a data modeling framework called Containment Modeling Framework (CMF). This framework has graphical and textual elements for data model structure modeling and code can be automatically generated based on CMF models. Unfortunately, domain-specific application activity descriptions not related to data models are difficult to express using CMF. A pattern-based code assembler approach was used to address this issue. It worked for simple domains with a small set of application activities, but could not scale to more complicated domains. The code generation approach is code assembler and suffers from scalability problems and lacks customization flexibility.

These problems motivated the design and implementation of a brand new feature modeling and code generation framework from scratch, with static and dynamic feature description support as the core modeling elements. Essentially, the new approach was genuine code generation rather than the simple code assembler approach. It is necessary that we review the history and lessons learned from the Bamboo project and explicitly specify design requirements for the Rhizome framework. In this chapter, we first introduce the Bamboo project, and then provide an example of CMF modeling and code generation using Bamboo. Finally, based on the Bamboo experience, design requirements are presented for the Rhizome platform.

4.1 Bamboo Project Review

4.1.1 Containment Modeling Framework Overview

Content management systems have similar repository structure—a set of primitive data operations such as create, retrieve, update and delete (CRUD) on top of a data repository usually implemented using a database. This set of operations is repeatedly implemented in different content management systems and the data repository schemas are largely the same as well. But there is a lack of standard graphical notations to describe the operations and data repository structure. This was the original problem the content modeling framework was trying to solve. Later, we discovered that by using a textual translation of the graphical containment models and exploting the structural similarities between different systems, it is also possible to generate content management system repositories automatically.

Containment is one of the most common relationships among entities in the real world. For example, a backpack contains textbooks, a lunch box contains food items, and a library catalog contains reference cards for books. A containment model is a specialized form of E-R model [69] that describes containment relationships among a set of given entities. CMF is a specialized modeling framework for constructing containment models. The actual data stored in a repository, where structure matches a containment model, is called an instance of the containment model.

CMF only has two basic modeling primitives: entities and containment relationships. Yet it is descriptive enough to model the repository structure for content management systems in any application domain. Entities represent data abstractions and relationships describe connections and constraints between entities. There are two types of entities: *atom* and *container*. Atoms cannot contain other entities and they often represent things like file content, attributes, or metadata items like author, last modified time, etc. Containers are used to form containment structures consisting of atoms or other containers. Relationships can be either *inclusive* or *referential*. For an inclusive containment relationship, if the container is deleted the containees must be removed too; but for a referential containment relationship, the container only stores pointers to the containees and containees are not removed when the container is deleted.

Relationships have properties called *ordering* and *multiplicity* to define data model constraints. The ordering property for the relationship identifies whether there is a partial order on the relationship instances in the containment model instance. For example, suppose a containment model defines a referential relationship from a project folder to several meeting memos. If the ordering property has the value “true”, all the meeting memo instances will have a persistently maintained partial order for any project folder instance that contains them. The multiplicity property defines the upper bound (called *membership*) and lower bound (called *cardinality*) for the number of occurrences of entity instances on both ends of a relationship. Membership restricts the number of container instances (of a particular type defined on the relationship) that one containee instance can belong to; cardinality restricts the number of containee instances (of a particular type defined on the relationship) that one container can have. Both membership and cardinality are represented in the form of ***m..n***, where ***m*** and ***n*** are integers and ***m*** represents the lower bound and ***n*** represents the upper bound. Using the same project folder example, if the membership on the relationship from project folder to memo is ***1..1*** and cardinality on the relationship is ***0..n***, then a project folder instance can have ***0*** or unlimited number of memo instances and a memo instance can only belong to ***1*** project folder instance.

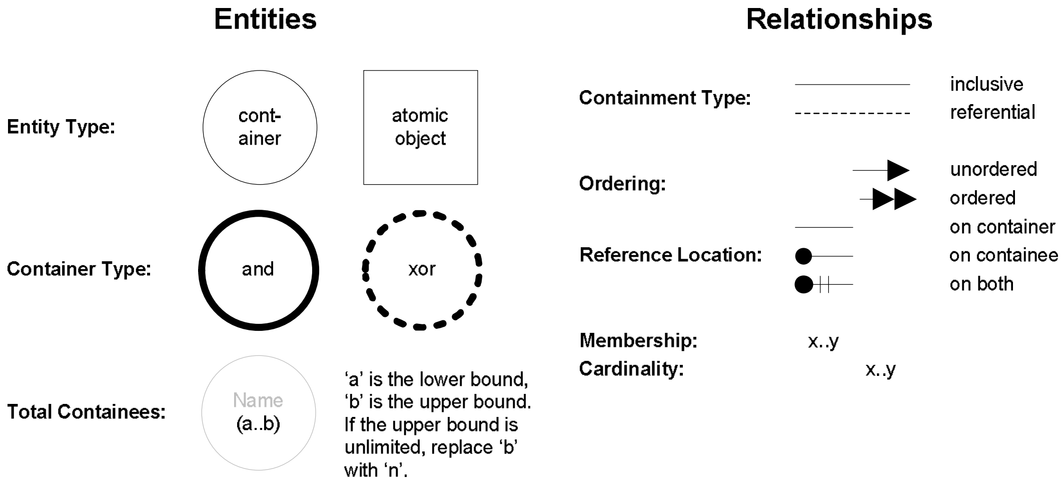


Figure 4-1. Containment modeling framework graphical language notations.

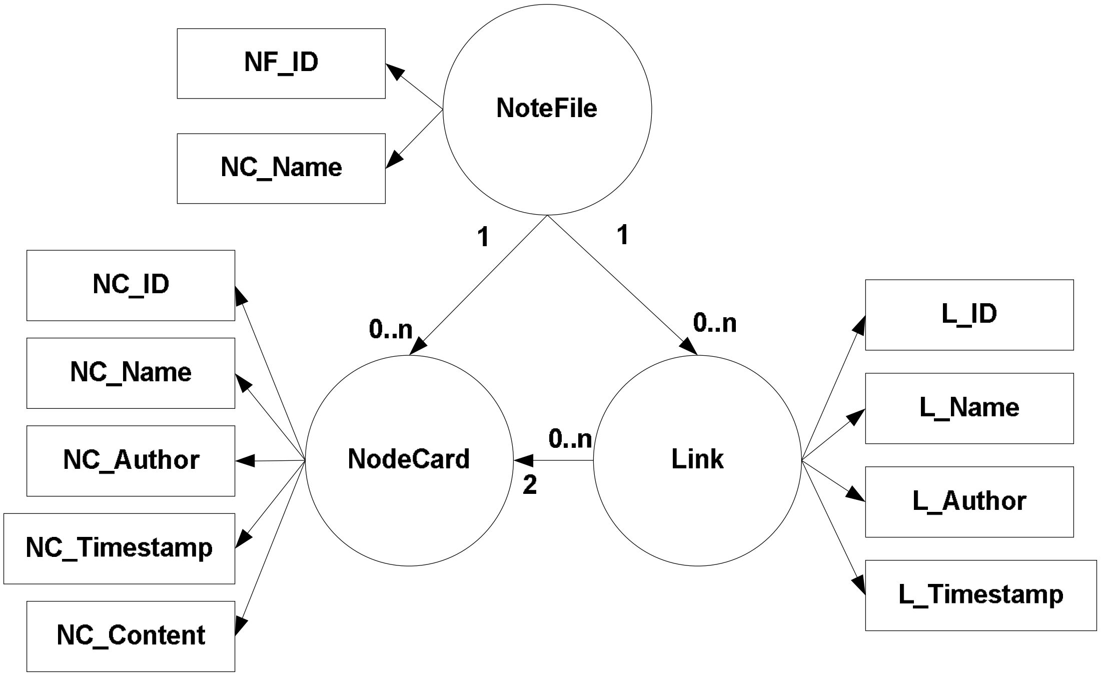


Figure 4-2. NoteCards modeled using CMF graphical language.

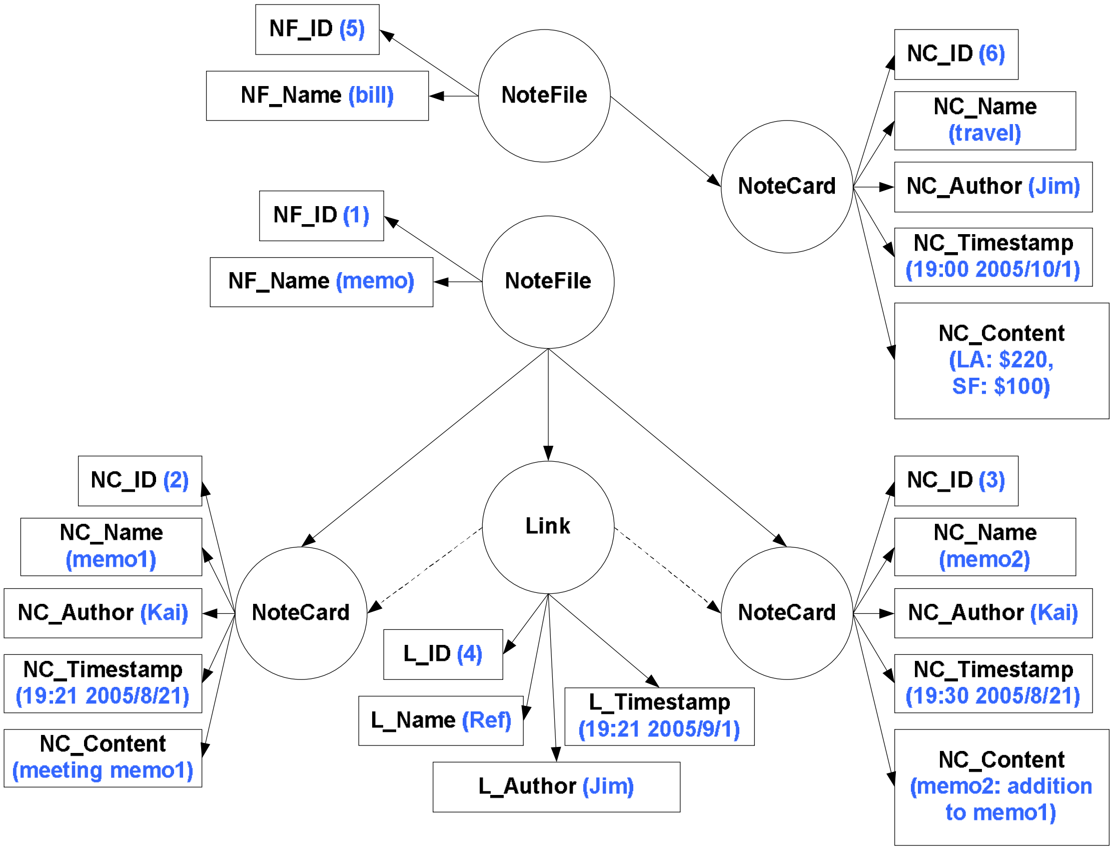


Figure 4-3. An instance of the NoteCards containment model. Values in parenthesis are atom entity values stored in the repository.

There are two formats of CMF modeling language: one is a graphical format and the other is an XML format. These two representations are semantically equivalent. The only difference is that the graphical format is intended for human users and the XML format is for computer programs. Figure 4-1 shows a collection of graphical notations used to describe a graphical CMF model. The meanings of these notations have been introduced in the previous paragraphs.

A simple example demonstrates these graphical notations in action. In this example, we use NoteCards [70], a well-known early hypertext system. The NoteCards system help users with daily information gathering and management tasks: a user writes small note cards to capture useful information, and then organizes them into relevant categories. In NoteCards, both textual and graphical information are recorded in small windows known as NoteCards. Typed links connect NoteCards, permitting hypertext navigation. NoteCards are organized into NoteFiles (groups of NoteCards) or Browsers (structures of linked NoteCards). Figure 4-2 shows the graphical containment model for a slightly modified NoteCards system using the CMF graphical language. In the model, a NoteFile inclusively contains multiple NoteCards and those NoteCards are connected using Links (each Link referentially contains two NoteCards). Simple atom entities record metadata information like ID, name, author, creation timestamp, etc. To simplify the model, the Browser and typed links in the original NoteCards system are not represented.

A containment model is analogous to a database schema—it defines the structure and relationship for entities. Just like a database schema can have multiple instances, each populated with real data, a containment model has model instances. The CMF language can describe not only a containment model, but also containment model instances. Figure 4-3 shows an instance of the NoteCards containment model, showing one possible repository structure with real data. There are two NoteFiles, one for meeting memo, the other for bill management. The meeting memo NoteFile contains two meeting memos NoteCards created by Kai and these two memos can be traversed using a Link created by Jim. The bill management NoteFile only has one NoteCard representing travel bills. Values in parenthesis are sample values stored in the repository. Note that CMF is independent of repository formats: flat files, XML files, or database systems are all candidates as long as the entities and relationships are preserved in the repository.

The graphical language is suitable for human users to manipulate in a graphical model editor and the graphical models are easier to understand. However, storing and editing graphical models are more difficult for computer programs. Thus, an XML schema is designed as the textual representation for the graphical language. The textual language describes entity and relationship definitions with their properties. APPENDIX A: Containment Modeling Framework XML Schema (Graphical and Textual) shows the XML schema for the CMF modeling language (scm.xsd) and Figure 4-4 shows the NoteCards system described using this XML modeling language (notecards_scm.xml), which is just the textual representation of the graphical containment model in Figure 4-2.

4.1.2 Containment Modeling Framework Applications

In addition to abstract modeling of content management repositories, CMF has three distinct system building applications: automatic generation of repositories, customization of system features, and combining features from different domains.

In the first case, a containment model is designed for a content management system, and then code generation tools produce repository code consisting of a repository management module, a reusable repository API, an entity and relationship control module and a command line repository client. The repository management module manages requests and responses for the repository and performs routine tasks like repository schema initialization. The repository API implements a standard repository interface, through which data can be created, retrieved, updated and destroyed (CRUD operations). These two components are independent of a specific containment model definition. The entity and relationship control module realizes entity and relationship definitions in the containment model and communicates with the repository. The repository client is an interactive command line shell for users to manipulate entities and relationships: creating, deleting, updating entities and relationships, exploring the containment model, viewing or changing atom entity values, etc. This command line client is generated for testing and demo purposes. It is also possible to implement other types of client applications such as GUI clients and Web applications using the generated repository modules.

To generate repository code, an additional repository model is required to map CMF data types to those physical repository data types such as the ones used in MySQL database. Appendix B shows the XML schema for the CMF repository modeling language (repos.xsd). Figure 4-4 shows the repository model for NoteCards system (notecards_repos.xml). The input to the Bamboo repository generator includes a containment model and a repository model. The output is a generated content management system repository with a command line shell client.

The second type of CMF application is feature customization. Automatic repository generation allows rapid development at the repository level. However, it does not create semantic operations on top of the generated repository. To add semantic operations, domain knowledge needs to be captured and modeled to apply this knowledge in a way the code generator can understand.

```
<model xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"xsi:noNamespaceSchemaLocation="scm.xsd">
 <entities>
    <container name="NoteFile" hasOrderedChildren="false" ttlContainerLower="0" ttlContainerUpper="0"
ttlContaineeLower="2" ttlContaineeUpper="INF" type="generic"/>
    <container name="Link" hasOrderedChildren="false" ttlContainerLower="1" ttlContainerUpper="1"
ttlContaineeLower="4" ttlContaineeUpper="4" type="generic"/>
    <container name="NoteCard" hasOrderedChildren="false" ttlContainerLower="1" ttlContainerUpper="INF"
ttlContaineeLower="3" ttlContaineeUpper="3" type="generic"/>
    <atom name="NF_ID" ttlContainerLower="1" ttlContainerUpper="1" logicalType="Integer"/>
    <atom name="NF_Name" ttlContainerLower="1" ttlContainerUpper="1" logicalType="String"/>
    <atom name="L_ID" ttlContainerLower="1" ttlContainerUpper="1" logicalType="Integer"/>
    <atom name="L_Name" ttlContainerLower="1" ttlContainerUpper="1" logicalType="String"/>
    <atom name="L_Author" ttlContainerLower="1" ttlContainerUpper="1" logicalType="String" />
    <atom name="L_Timestamp" ttlContainerLower="1" ttlContainerUpper="1" logicalType="String" />
...
 </entities>
 <relationships>
    <relationship name="Unordered_Referential" isOrdered="false" type="referential"/>
    <relationship name="Unordered_Inclusive" isOrdered="false" type="inclusive"/>
 </relationships>
 <er_model>
    <arc refRelName="Unordered_Referential" from="NoteFile" to="Link" membershipLower="1"
membershipUpper="1" cardinalityLower="0" cardinalityUpper="INF" location="on_container"/>
    <arc refRelName="Unordered_Referential" from="NoteFile" to="NoteCard" membershipLower="1"
membershipUpper="1" cardinalityLower="0" cardinalityUpper="INF" location="on_container"/>
    <arc refRelName="Unordered_Inclusive" from="NoteFile" to="NF_Name" membershipLower="1"
membershipUpper="1" cardinalityLower="1" cardinalityUpper="1" location="on_container"/>
    <arc refRelName="Unordered_Inclusive" from="NoteFile" to="NF_ID" membershipLower="1"
membershipUpper="1" cardinalityLower="1" cardinalityUpper="1" location="on_container"/>
    <arc refRelName="Unordered_Referential" from="Link" to="NoteCard" membershipLower="0"
membershipUpper="INF" cardinalityLower="2" cardinalityUpper="2" location="on_container"/>
    <arc refRelName="Unordered_Inclusive" from="Link" to="L_ID" membershipLower="1"
membershipUpper="1" cardinalityLower="1" cardinalityUpper="1" location="on_container"/>
...
 </er_model>
</model>
```

notecards_scm.xml

```
<repos_mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=" repos.xsd">
 <value_type logicalType="Integer" physicalType="INT"/>
 <value_type logicalType="String" physicalType="VARCHAR(50)"/>
 <value_type logicalType="Datetime" physicalType="DATETIME"/>
 <value_type logicalType="Text" physicalType="TEXT"/>
</repos_mapping>
```

notecards_repos.xml

Figure 4-4. Textual containment model representation for NoteCards.

Domain roles and feature implementation patterns are the key to injecting semantic knowledge into the feature model. A domain role is an abstraction for a CMF entity that has domain-specific properties and operations. For example, s_revision is a domain role for the version control domain, representing an atomic revision entity such as a revision for one text file. Similarly, s_col is a domain role that represents a collection revision entity, such as a revision for a directory full of files. A feature implementation pattern consists of these domain roles modeled using the CMF language in the same way that a content repository is modeled. For example, we have identified five different implementation patterns to implement the version history feature in the version control system domain. Hence, a feature implementation pattern is an abstract semantic model that describes how a feature should be implemented and what entity structure and behaviors should be expected.

In the Bamboo project, a variability model captures domain knowledge and defines all possible choices for a feature implementation. These design choices are defined using domain roles and feature implementation patterns. In this way, when the code generator “sees” the design choice “OrderedRevision” for the version history feature, it instantly understands the entity structures and operations associated with these entities because this semantic information is captured in the version history feature implementation pattern.

A variability model also defines various relationships such as existence (AND), optional (OR), exclusive (XOR), option-value-selector (it can be used for cases such as “if option1 is selected for feature1, then option2 must be selected for feature2”), etc. A user makes design choices to select options and derive a feature model from the variability choices.

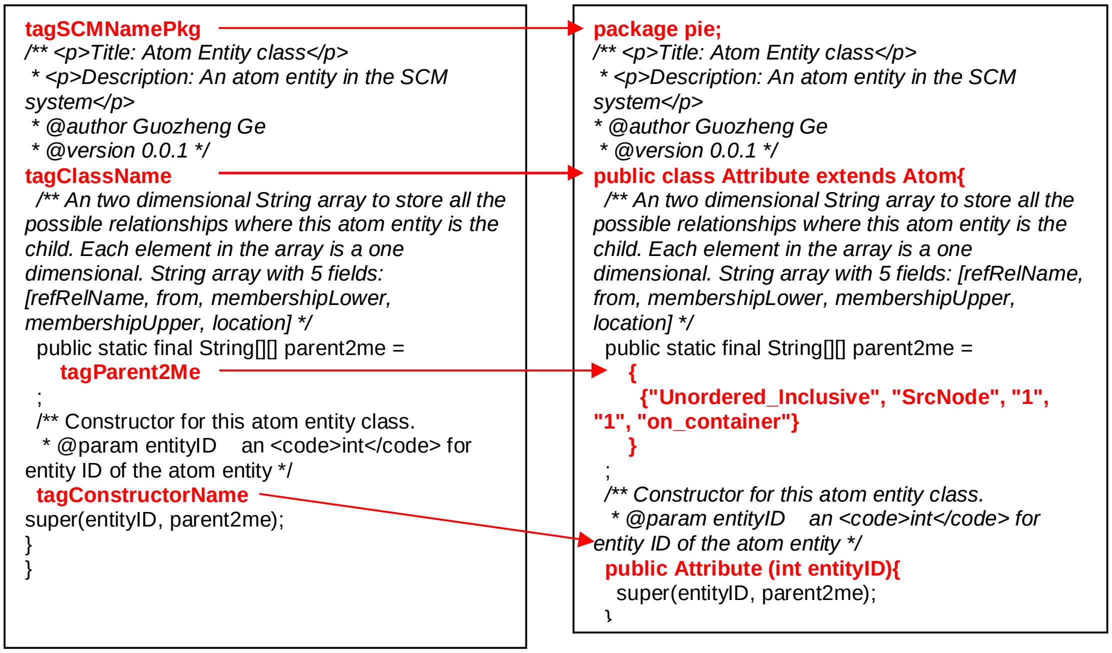
A repository containment model is also needed to generate the repository layer. To do this, we first need an extra repository-to-feature mapping file. This file defines mappings from entity names in the repository model to domain roles in the feature model.

Given a feature model, a repository model and a mapping file, it is possible to generate customized feature configurations and create a running application. As a demonstration of this approach, two version control systems were generated: a simple linear versioning system and a branching versioning system. This application opens the possibility for rapid prototyping of different feature combinations at the software design phase.

The third CMF application type involves the creation of hybrid features by combining features from different domains. In many cases, we have individual applications from different domains and what we really want is one application that combines features from each domain. The Bamboo project has model transformation tools that can merge repository models and feature models to generate an integrated repository and merged semantic operations for the new hybrid features. As a demonstration, linear version control capabilities (from the version control domain) were added to NoteCards (from hypertext domain), so that the updated NoteCards can perform both hypertext and version control operations. One of the most challenging problems in creating hybrid features is how to detect and fix discrepancies or irrelevant semantics brought in by different domains. Substantial manual work was required to fix problems in the code generator to make it generate code for such hybrid features.

4.2 Bamboo Project Discussion

The Bamboo project uses template-based code generation to produce source code. A template file contains several tags, each with a unique tag name in the whole generator namespace. To generate a source code file, the code generator simply replaces a tag with a block of customized code. Figure 4-5 explains this idea of template-based code generation. Specially formatted tags are embedded in source code and they are replaced with blocks of source code created or retrieved at code generation runtime. For example, in Figure 4-5, *tagSCMNamePkg* stands for the package name for a Java class which the code generator replaces with the actual package name associated with the class.



```
tagSCMNamePkg
/** <p>Title: Atom Entity class</p>
 * <p>Description: An atom entity in the SCM
system</p>
 * @author Guozheng Ge
 * @version 0.0.1 */
tagClassName
  /** An two dimensional String array to store all the
possible relationships where this atom entity is the
child. Each element in the array is a one
dimensional. String array with 5 fields:
[refRelName, from, membershipLower,
membershipUpper, location] */
  public static final String[][] parent2me =
      tagParent2Me
  ;
  /** Constructor for this atom entity class.
   * @param entityID    an <code>int</code> for
entity ID of the atom entity */
  tagConstructorName
super(entityID, parent2me);
}
}
```

```
package pie;
/** <p>Title: Atom Entity class</p>
 * <p>Description: An atom entity in the SCM
system</p>
* @author Guozheng Ge
 * @version 0.0.1 */
public class Attribute extends Atom{
  /** An two dimensional String array to store all the
possible relationships where this atom entity is the
child. Each element in the array is a one
dimensional. String array with 5 fields: [refRelName,
from, membershipLower, membershipUpper,
location] */
  public static final String[][] parent2me =
      {
        {"Unordered_Inclusive", "SrcNode", "1",
"1", "on_container"}
      }
  ;
  /** Constructor for this atom entity class.
   * @param entityID    an <code>int</code> for
entity ID of the atom entity */
  public Attribute (int entityID){
    super(entityID, parent2me);
  }
```

Figure 4-5. Template-based code generation concept demonstration.

There are two sources for customized code blocks that replace tags in the template. One is code generated on-the-fly based on model information and the other is code snippets with fixed content. Code for the data model has a deterministic structure and represents well-understood variability in the source code. These variable parts include items such as attribute name, types, value constraints, etc. Tags related to repository code can be replaced with code blocks generated on-the-fly. However, design choices related to semantic features are not predictable since you cannot predetermine what choices a user will make. As a result, on-the-fly code generation for semantic features is challenging and we had difficulty to support it in the code generator.

Instead of guessing user design choices, all possible design choices are exhaustively collected beforehand. Then, these design choices are described using the concepts of domain roles and feature implementation patterns. Feature implementation patterns are abstract forms that represent all possible ways to implement a semantic feature. Each feature implementation pattern contains parameters. Each permutation of parameter value combination is associated with a set of source code snippets prepared beforehand so that when a particular implementation pattern is selected and has its parameter values all set in a feature model, the code generator can directly use these pre-defined code snippets to replace the tags in a template file. This is an example of the type of code generation called code snippets with fixed content.

The code generation part for feature models can be viewed as more of a code assembler, rather than a genuine code generator. The code snippets are prepared beforehand for a particular feature model. When a user’s design choices match this feature model, these code snippets are selected to replace tags in the template files. This approach worked fine for the sample domain of version control systems because there are fewer than ten variability factors in this feature model. A directory tree structure stores the snippet files, feature model and role mapping file. Each directory level represents one feature and the directory name is the design choice for its parent feature. The directory at the leaf node level stores the matching code snippets.

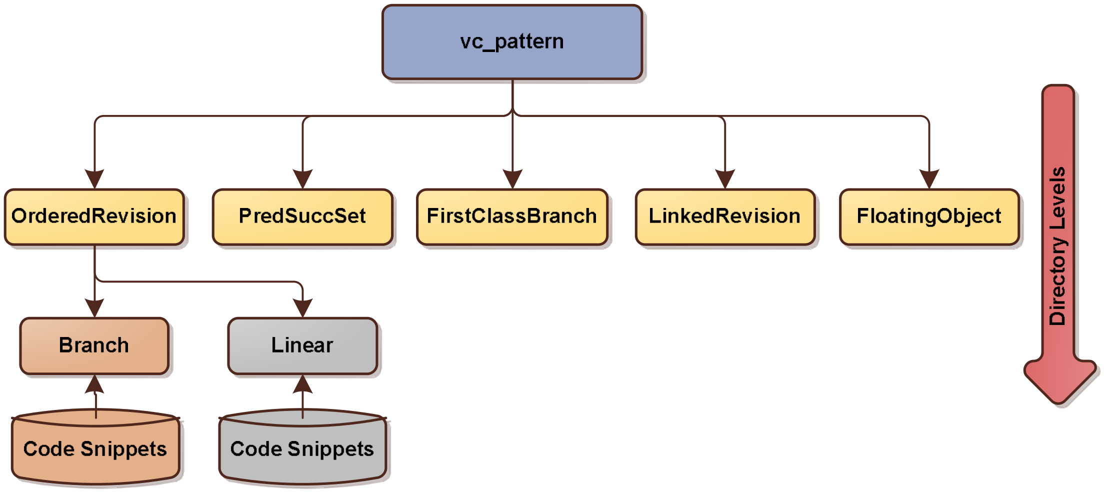


Figure 4-6. Directory-based code snippets search example.

Figure 4-6 shows the directory-based code snippet search method used in the Bamboo generator. In this example, a directory called “vc_pattern” represents the version control implementation pattern. It has five sub-directories: “OrderedRevision”, “PredSuccSet”, “FirstClassBranch”, “LinkedRevision”, and “FloatingObject”, each representing a design choice for the version control pattern. The “OrderedRevision” pattern itself can have two variants based on version history thread type: linear or branching. Linear means there is only one version history sequence for all revisions;

branching means the version history can have multiple sequences and these sequences can merge and split. So, two sub-directories are created in the OrderedRevision directory: “linear” and “branch”. This structure can extend to deeper directory level so long as there are still variability choices. When there is no other variablity at this directory level, code snippets are put into sub-directories at this level. When a code generator tries to search for snippet files, it follows each feature design choice through the directory structure until it finds a snippet file directory.

There are potential problems with this code assembler approach. This snippet assembler generation approach does not scale well when there are a large number of variability dimensions. There are fewer than ten variability dimensions in our sample domain of version control systems, yet the directory structure grew substantially, and was difficult to manage. When there are hundreds of variability dimensions and each dimension has many variants, it is no longer possible to implement all beforehand to create snippet files. It is better not to use a code generator in the first place, because it is much easier to pick a set of feature design choices and manually implement this one combination.

Bamboo’s sample domain is version control systems, which has a simple feature model and has the property that several independent feature implementation patterns can cover the semantic features. For large systems, feature implementation patterns have a complex structure and there are dependencies between these patterns. The Bamboo generator has built-in assumptions based on the implementation patterns. When there are many pattern structures and dependencies, the complexity of the code generator grows at an exponential rate, eventually making implementation infeasible.

In Bamboo, feature implementation patterns are described using the CMF language, a static data modeling language. A drawback of this approach is that CMF cannot be directly used to model dynamic activities in the feature model. This leads to the use of two containment models to describe an activity in the Bamboo project—one to describe the data model state before an activity, and the other to describe the data model state afterwards. When there are conditional branches mixed with dynamic activities, it is very difficult to explicitly model these features, if possible at all.

The Bamboo project can be viewed as an effort to explore model-based code generation in the content management systems domain. Instead of trying to build domain knowledge into a variability model, Bamboo was designed to explore the possiblity of using the Containment Modeling Framework to represent feature models and generating code base on feature models. Unfortunately, CMF cannot represent complex dynamic activities without making assumptions in the code generator, and CMF models are difficult to manage when the sample domain has a large scale. In the Bamboo project, we studied feature models, but did not really explore how to describe variability models, what process to follow to derive a feature model out of a variability model, modeling dependencies among features, and the relationships among feature design choices. Bamboo did, however, demonstrate the importance of these issues, leading to their consideration in the follow-on system—Rhizome.

4.3 Rhizome Design Requirements

The experience gained from Bamboo informed several design requirements for the new framework called Rhizome. First, Rhizome needs an explicit variability modeling language and feature modeling language. The variability modeling language is used to build a variability model that includes dependencies among feature model design choices and design options for individual features. It represents all possible feature models in a domain. A feature model is a blueprint for application design. It includes design choices made by a user that should be implemented in the final system. The following is a list of requirements for both languages:

- The variability modeling language should be independent of platform factors such as programming languages and operating systems.
- It should be extensible so that new language components can be conveniently added in the future to enhance modeling capabilities.

- It should be machine processable. Variability models and feature models should be easy to parse, transform and edit by programs. Human users may also perform similar tasks through GUI clients such as a design wizard or by editing model text directly.

Second, the snippet assembler type of code generation must be eliminated because it is impractical to anticipate all possible feature model scenarios beforehand and prepare sufficient code snippets. The code should be generated on-the-fly using only information from a feature model that captures user inputs. The code generator should be kept simple without being coupled with variability and feature modeling languages and specific domain knowledge. The feature modeling and code generation aspects of the system should be largely decoupled except for explicitly modeled code generation instructions that connect both parts. This decoupled structure ensures that changes to either part will be isolated. Code generation instructions should be externalized from the code generator so that users can customize and control the code generation process. These instructions can be integrated with a feature model or they can have their own instruction modeling language and be separated from a feature model.

The Rhizome framework was designed to meet these requirements. For the variability modeling part, a variability modeling language describes a variability model for a domain of interest. A variability model serves as a design knowledge base or a design space, where feature models can be derived. A feature model is described using a feature modeling language and stores the design choices a user makes during the design process. Both modeling languages are implemented using the XML schema language and models are represented as XML documents conforming to the language schemas.

Code generation adopts a template-based generation approach. Each feature model is associated with a set of template files, where a template file is a source code file embedded with XML tags. The XML tags are defined using a template language called MarkerML. Each tag contains instructions and parameters telling the code generator how to produce a block of source code that replaces the tag. When all tags in a template file are processed, the result is a complete source code file. Note that a template file might create more than one source code files in cases when a template file name contains a parameter that has multiple values.

Code generation instructions connect to a feature model definition via code generation units (CGUs). A CGU describes code generation types, input parameters and it is represented as an XML element in an XML feature model document. The code generator is simply a text file scanner and tag processor—it scans template files and replaces tags with code blocks generated on-the-fly. The online exam systems are selected as our sample domain to test Rhizome and the code generator produces one

J2EE application for online exam systems based on different feature models. Details about this Rhizome framework are revealed in the next several chapters.

5 Sample Domain: Online Exam Systems

An online exam system is a Web application that brings the traditional classroom exam experience to the online world. Users of online exam systems have specific roles, such as student, teacher, teaching assistant (TA), and administrator (admin). Each role has designated activities: for example, a student can take an exam by answering questions and submitting answers. A teacher or TA can then start grading the answered exams and prepare grade reports. A student can also perform other activities such as communicating with a teacher or a TA, paying exam fees, etc. Besides these user role entities, there are also data entities like questions, question answers, exams and exam answers, which also have designated activities that can act on them. The major difference between a user entity and a data entity is that the user entity is the actor who performs various activities and the data entity holds data to be processed. A detailed discussion about these entities and their activities will be presented in the domain analysis section. First, we give an overview of several online exam systems that form the base of our domain analysis.

Blackboard [71] is the biggest online education system after the company acquired another popular e-learning system WebCT. Blackboard has an intuitive user interface and rich features. Blackboard system is implemented using ASP and other .NET technologies. Online exam support is one of the many core capabilities provided by Blackboard. Other related capabilities include online teaching and reading materials, surveys, forums, e-commerce payments, etc. It has an open architecture consisting of Building Blocks and PowerLinks, which allow feature customization and integration with existing applications through a set of APIs. Blackboard is quite expensive—its enterprise version costs more than 200K USD per license. The studied Blackboard system was studied by reading its online documentation since it was not possible to try out the system directly.

Moodle [72] is the most well-known open source online education system. It has a huge user community and the second largest market share. The system is implemented using PHP with a plugin architecture. The online exam feature is just a small portion of the entire system—it supports online communities, polls, blogs, wiki, various communication methods including email, instant messaging, and content management. Users can write their own plugins or modify existing plugins to customize system features. A nice feature is that Moodle supports a wide variety of question formats such as GIFT [73], IMS Question and Test Interoperability Specification (QTI) [74], XML, XHTML, etc. Advantages of Moodle compared to other systems include its open source development model, flexible architecture, support for multimedia content, and a rich set of tools including glossary, polls, lessons, journals, and support for collaborative learning processes.

Hot Potatoes [75] is another commercial e-learning system. It has many unique question types not found in other systems, such as crossword, mix and match for letters and words, pair matching, etc. The system is very popular for online language education programs. Although it is a commercial system, it provides free license for state-funded educational institutes and costs less than the Blackboard system. On the Hot Potatoes website, many demo exams are available, which we studied to explore the capabilities of the system.

Sakai CLE [76] is an open source collaboration and learning environment developed by a group of 150 academic institutions and academic institutions. It is developed using Java and service-oriented architecture. Sakai CLE has a long list of features with an emphasis of communication tools: testing, assignments, grade book, syllabus, online presentation, discussion, email, chat room, poll, calendar, wiki, mailing list, RSS reader, etc.

Claroline [77] is a lightweight open source eLearning and eWorking platform. It is built on PHP and MySQL platform and has a rich set of tools including announcement, agenda, exercise, assignment, learning path, chat, wiki, etc. Claroline originated from a teamed effort in several European universities, but it has nice internationalization support and is being used in 95 countries and 30 languages.

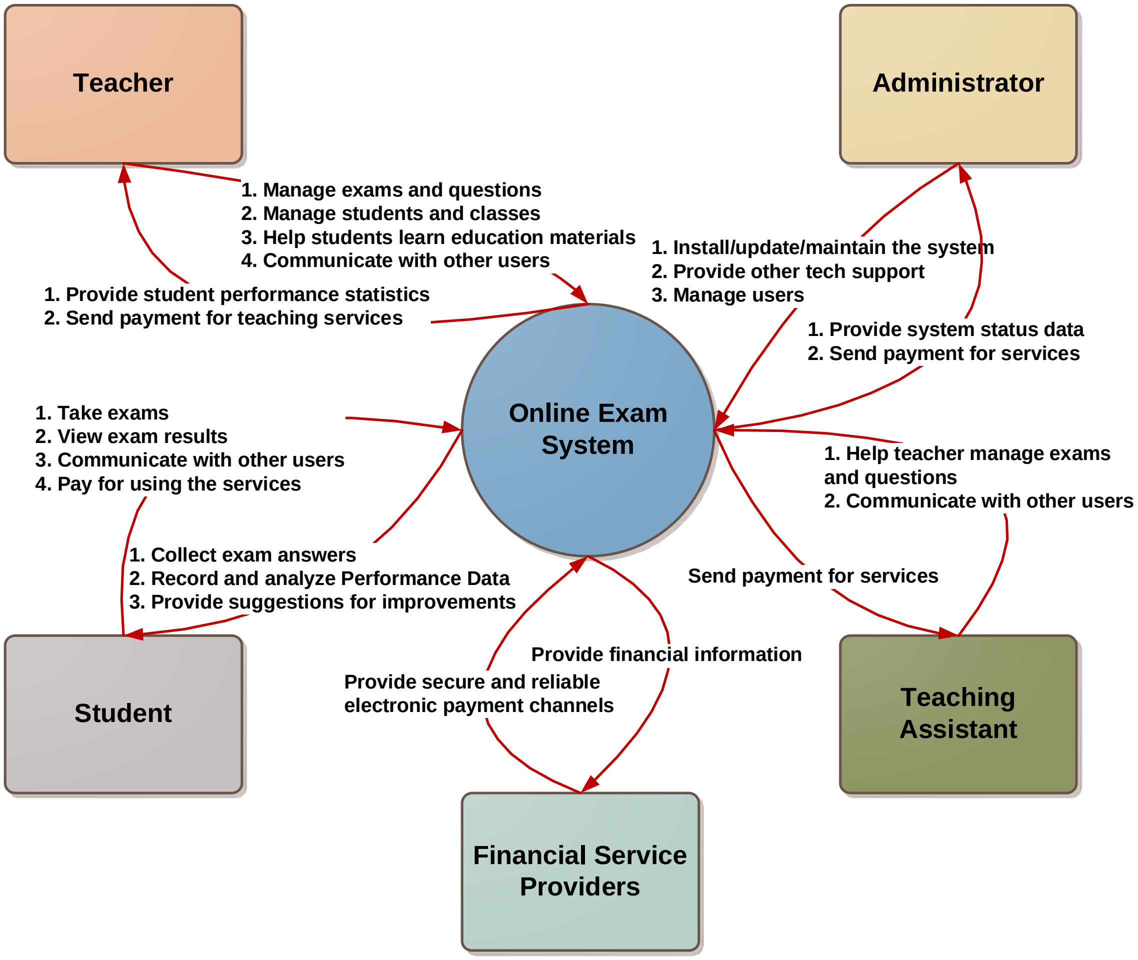

Teacher
Administrator
1. Manage exams and questions
2. Manage students and classes
3. Help students learn education materials
4. Communicate with other users
1. Install/update/maintain the system
2. Provide other tech support
3. Manage users
1. Provide student performance statistics
2. Send payment for teaching services
1. Provide system status data
2. Send payment for services
Online Exam System
1. Take exams
2. View exam results
3. Communicate with other users
4. Pay for using the services
1. Help teacher manage exams and questions
2. Communicate with other users
1. Collect exam answers
2. Record and analyze Performance Data
3. Provide suggestions for improvements
Send payment for services
Provide financial information
Provide secure and reliable electronic payment channels
Student
Teaching Assistant
Financial Service Providers

Figure 5-1. Online exam system context diagram with major activities.

5.1 Context Analysis

Figure 5-1 is a context analysis diagram for the online exam system domain. It shows five major parties participating in the system: teachers, students, teaching assistants, administrators, and financial service providers. A teacher can perform activities such as managing exams and questions, managing students and classes, helping students learn various educational materials and communicating with other users. A teacher also receives statistical data regarding student performance, system status, etc. A student can take exams, review exam results, and interact with other users. A teaching assistant can offer help to share responsibilities with the teacher, such as help manage exams and questions and answer questions for students. An administrator's major responsibilities include system maintenance, user management and technical support. An interesting feature we found in some commercial systems is on demand services—students pay to use the online exam system while teachers, teaching assistants and administrators are paid for their contributions to the exam system. In such a fee-based system, there need to be financial service providers such as credit card companies, banks, or other third party financial service providers like PayPal [78], or Google Checkout [79]. These providers handle financial transactions for payments among senders and receivers. The exact amount for payments is calculated in the exam system based on pre-defined algorithms.

5.2 Variability Analysis

5.2.1 Network Architecture Styles

Online exam systems are Web applications that use a client/server[80] network architecture, where users connect to the Web applications hosted on a server and interact with these applications using a Web Browser. The Web client and server applications fulfill user operations by exchanging requests and responses.

One common architecture variant allows certain activities to be carried out offline first, synchronizing offline data with the server later. For example, a teacher can compose questions and exams in offline mode and upload them later when a network connection becomes available. Similarly, a student can download an exam and answer questions in the offline mode and submit answers to the server when his laptop connects to the network again. This offline architecture is a variant of client-server architecture and requires extra complexity for synchronization and conflict resolution. For instance, conflicts might happen when multiple teachers collaboratively edit the same exam, or when students submit their answers to find out that the questions are updated during their offline sessions.

To distinguish these two architectural styles, the first style is referred to as “online client/server” and the second style as “offline client/server”. A comparison of the two styles is summarized in the following table (Table 5-1). In the Rhizome implementation, we used the online client-server architecture style since it is much simpler than the offline client-server style to implement and test.

Table 5-1. Comparison of two online exam system network architecture types.

	Online client/server	Offline client/server	Discussion
Accessibility	Less flexible	More flexible	Network is not yet available anywhere and anytime
Security and Controllability	User behaviors and data can be designed to be held under strict security policies and monitored with tight control polices	Sensitive data may leak or intentionally hacked; offline data may also get lost completely	Special content encryption algorithms are required for offline style. User identification and monitoring is much easier for online style.
Content Modification	More convenient	Less convenient	Content changes are easier at the server side. It is difficult and troublesome to propagate changes from client to server in offline style due to potential conflict problems.
Academic Dishonesty	Less possible to cheat	More possible to cheat	Hacking a well-maintained server is more difficult than analyzing readily available offline materials.

5.2.2 Variability in Repository Layer

In addition to the network architecture, an online exam system has typical three-tier system architecture: repository layer for persistent data storage, middleware layer for business logic, and Web client layer for user interface. These three layers are examined one by one below.

The repository layer provides persistent storage services such as creation, retrieval, update and deletion of system data. There are many types of system data in an online exam system. We summarize these data types in the following itemized list.

a) **Question**: It represents the content structure for a question consists of correct and incorrect answers, question type, difficulty level, grading policy, and question constraints such as time constraints, number of trials, prerequisites, etc. There are many types of questions and each type has a slightly different content structure. In Table 5-2, all the existing question types we found through domain analysis of existing online exam systems are listed.

Table 5-2. Question types in the online exam systems.

Question Type	Description
Multiple Choice (MC)	Each question contains at least one correct answer and several incorrect answers. Only when a student selects all correct answers will he get full credit for the question.
Single Choice (SC)	Each question contains exactly one correct answer and several incorrect answers. Only when the correct answer is selected will the student get full credit.
True or False (TF)	This is a special type of Single Choice question. It only always contains two choices: True or False. A student gets the credit if he picks the correct Boolean answer.
Short Answer (SA)	This is a question and answer type. The user inputs text as the answer to a question consisting of a paragraph of text. Reading comprehension is one most frequently used short answer types.
Short Answer Choice (SAC)	This type is found in Hot Potatoes system. A combination of Short Answer and Multiple Choice: initially, it is a Short Answer question asking a user to input text; when a student makes several failed attempts, it turns itself into an Multiple Choice question to lower the difficulty level.
Cloze Answer (CA)	A paragraph of text contains several holes and the user needs to fill these holes with correct text.
Cloze Answer	This is a variant of CA type where a drop-down list of selections is provided at each hole and

Selection (CAS)	the user picks the answers from the list.
Matching (M)	The question is presented with two columns of items and the student tries to match one item in left column to another item in the right column based on some semantic connection. For example, one column can be fruit names and the other column contains pictures of these fruits.

Questions can be manually created or generated automatically by algorithms. For example, if a teacher wants to test students' knowledge about Einstein's mass-energy equivalence ($E = mc^2$), he can give different values for m and c, and then let the computer calculate the correct answer for the value of E using the equation. This gives students different question bodies while testing the same knowledge. It adds variability into the questions and helps avoid using the same question repeatedly. Academic dishonesty is also more preventable when the question body changes on the fly.

b) **Question Answer**: It represents a student's answer to a question. The actual answer content varies depending on the question type: it can be a String value that represents the answer to an SA question, or it can be a True or False Boolean value as the answer to a TF question. A question answer also includes properties like student information, start and end timestamp, network access information, etc. For automatically generated questions, the correct answer content will be the algorithm script to calculate the correct answer. Each time the generated question is graded, the script will be executed to check if the student answer is correct.

c) **Exam**: It is a structure that holds references, e.g. question ids, to the questions. It also contains additional information such as question ordering algorithms, grading policies, exam constraints like time constraints, number of trials, prerequisite classes, etc.

d) **Exam Answer**: This is a list of references to question answers contained in a given exam. It also stores additional metadata such as student information, start and end timestamp, question answering order, network access information, etc.

e) **User**: There are multiple user types including student, teacher, teaching assistant, administrator, etc. Each role shares common properties with other roles and has unique properties as well. Common properties include user ID, name, email, address, phone number, etc. Examples for unique user properties include office location and phone number for a teacher, since the teacher holds office hours and receives phone calls related to classes. In our implementation of demo online exam systems, common user properties are captured in a User superclass and unique properties for individual user types are defined in subclasses that extend the User class.

f) **User Group**: It organizes users according to a specific quality. For example, one commonly used user group is “class”, where students taking the same class are grouped together. There are also user groups based on the same teacher or the same exam subject.

g) **Analytical Data**: It includes various statistical analysis data for student performance such as grade distribution, exam answer analysis, performance improvement over a period of time, etc. These data help a student understand their performance with different subjects, and their improvement between different learning stages. It also tells a teacher how to wisely distribute teaching effort and effectively design teaching strategies.

h) **Communication Message Archive**: This records communication messages among users. The content of this data type depends on the communication mechanism used in a system such as email, instant messages, online forum, wiki, etc.

i) **Payment Information and Transaction Data**: The data contain financial information for each user and keeps transaction records such as timestamp, amount, sender, receiver, network access information, etc.

Now, we examine the variation points (VPs) related to the repository layer. One important VP is the **question type VP**, which determines many other VPs. For example, question type affects how incorrect and correct answers are prepared and stored. For SA (Short Answer) type, the answers are quite subjective and there is no way to store correct answers or use a formula to calculate correct answers beforehand. For M (Matching) type, it is best to store correct matching pairs beforehand—this makes much easier. For other types, either preparing incorrect and correct answers beforehand or using a formula to generate correct answers works equally fine. Question type VP also affects VPs in other application layers: different GUI presentations for viewing, editing and answering pages, availability for automatic grading (question types like SA cannot be automatically graded since the answer is subjective), etc.

Question content and properties VP is a VP that is dependent on question type as well. In general, these properties are required for each question type: GUID, question body, incorrect and correct answers, author (authors if multiple authors are supported), creation date, grading policy, and difficulty level. There are also optional properties like category name or keywords (used when question search feature is available) and access control bit (used to protect an author's work so that only the author can use a question or an exam when he sets the control bit to private).

Question type VP also determines **question answer properties VP**. The question answer records a student's answer to a question. Different question types need to record answers in unique formats. Required properties include student username, submission timestamp, student answer, score, grader username, etc.

Another VP is **question storage format VP—**it can be a customized text format like Moodle GIFT, or XML, or a standard format like Sharable Content Object Reference Model (SCORM). The storage format does not have much impact on other VPs except for the import and export module features. Many online exam systems support multiple import and export modules for different question storage formats.

Similar to question types, another import VP related to an exam is the **exam types**. An exam is a structural container that holds references to questions and metadata properties such as ordering, grading policy, expiration time and number of attempts, etc. The domain analysis identified four exam types: static, shuffle, semi-automatic and adaptive. A static exam is the most frequently used exam type: an exam author manually selects exam properties such as questions, correct and incorrect answers for each question, and a fixed question ordering. When a static exam is presented to students, they will see exactly the same exam content and layout. A shuffle exam is a variant of static exam designed to prevent academic dishonesty such as questions being leaked from a student who has previously taken the same exam. In a shuffle exam, both question ordering and answer ordering are randomized every time the exam is presented to a student. A shuffle exam still uses a fixed set of questions pre-selected from the question bank and the same parameter settings as static exam.

Although ordering is changed, a shuffle exam still uses a fixed pool of questions. To be more effective in the battle against academic dishonesty, this question pool can be randomized too. First, a teacher sets search criteria for questions by properties such as topics, difficulty levels, amount of questions, etc. Then, the exam composition engine automatically picks a set of questions from the question pool that matches the specified search criteria. If the question pool is big enough compared to the number of questions in an exam, each student will be able to see a completely different set of questions and thus this approach works effectively against cheating. However, just like static and shuffle exams, the set of questions are fixed before a student starts taking the exam and won't change over the exam session. The only difference is that the question pool is much larger than a single student sees. This exam type is called semi-automatic, since questions are automatically assigned to a student but questions in an exam are determined at the beginning of the exam and cannot change during the exam-taking process.

The last exam type known as an adaptive exam is the most flexible and dynamic. The idea is that the exam composition engine will pick questions dynamically from the question pool based on student behavior during the exam taking process. For example, if a student continuously makes mistakes, the system should lower difficulty level to pick easier questions. On the other hand, if a student gets many correct answers, more difficult questions will be presented. The grading policy should be adaptive as well to give easy questions fewer credits and difficult questions more credits. Adaptive exams are the best type for academic dishonesty problems. Many international computer-based exams are implemented using adaptive exams, such as the Graduate Record Examination (GRE) [81]. This question type also supports a more effective learning process: the questions can be set to display hints when a student repeatedly submits wrong answers until he finally makes a correct answer. However, adaptive exams require more complicated control and monitoring of student behavior. Grading policies are also more complex since they need to accommodate the dynamic exam process and question properties. Thus, only a few high-end commercial exam systems provide adaptive exams.

Each exam system also has a workflow model to manage different stages of an exam's lifecycle—this variation point is called **exam lifecycle management VP**. One typical workflow model can be described in the following way. During the composition phase, an exam has "draft" status, which means it is still a work-in-progress and further modifications are expected before it can be released. After exam content is fixed, the author can release this exam by changing its status changed from “draft” to “open”. After this release action, students with appropriate permissions, e.g. students that subscribe to the class related to this exam, are able to see the exam and start answering questions. Usually, each exam has an expiration date. After this expiration date, the exam status is changed to “closed” and students cannot view or answer questions anymore. Then, the teacher or TAs grade students’ answers and prepare grade reports. Finally, when all answers are graded, the exam status changes to “graded” and students are able to see the exam again by viewing their grade reports. After all students have seen their grade reports, the exam author changes the exam status to “draft” again for further modification or future reuse. This workflow model is mostly used by teachers to evaluate student comprehension and their skills to solve problems.

Another workflow model is typically seen in self-evaluating exams. The purpose of these exams is not to help a teacher quantify a student’s knowledge or capability, but to help a student evaluate his learning process and locate areas for improvement. Students can take exams as many times as they want and there is no expiration date for exams. After a student submits the answers, they are automatically graded and the student receives a report instantly with detailed explanations. The exam grade is mostly ignored by the exam system in this workflow model since a student can take the same exam multiple times at anytime. This model is simple and only requires two exam status “draft” and “released”. When the exam is being edited, its status is “draft” and later changed to “released” when it is published.

Question source VP defines information about where to look for questions and how to select questions for an exam. Questions can either be handpicked from the question pool, or automatically selected using search algorithms. Search algorithms can be keyword-based using tags or categorization-based search that finds results by following a given path expression. For example, an exam author can specify an integer value of 10 to indicate there are 10 questions and a tag of “mathematics”. When an exam is created, the exam composition engine first performs a search to find a pool of question candidates marked with the tag “mathematics”. Then, the composition engine randomly selects 10 questions out of that pool. Categorization search works in a similar way, except for the process of finding question candidates. When a new question is stored in the question repository, it is categorized using keywords starting from general categories down to specific categories. To locate question candidates, an exam author provides a categorization path such as mathematics->discrete mathematics->information theory->entropy. The exam composition engine follows this categorization path and refines the question pool at each step. The final questions to use are selected from the categorization-based search results.

Exam termination mode VP specifies the way to terminate an exam session. There are two different choices: *stop-by-timer* or *stop-by-completion*. Stop-by-timer mode terminates an exam when the timer runs out. When an exam is defined, there is a duration property to specify the maximum time duration for an exam session. If a student cannot finish all the questions before the timer runs out, the exam session will submit the existing answers and close automatically. Warning messages pop up before the session terminates asking students to speed up the answering process.

The other mode, stop-by-completion, means there is no time limit on the exam session—as long as a student finishes all the questions before the exam expiration date (an exam expiration date is defined as a timestamp when the exam status changes from "open" to "closed"). Both stop modes are found in many online exam systems and this VP directly affects **exam grading VP**. The exam grading VP takes care of grading policies. Exam termination mode is one factor to consider for the grading policies. The actual time it takes for a student to complete the exam should be used as a grading policy factor in stop-by-completion mode. For stop-by-time mode, the students who finish early should get extra credit for completing the exam more quickly than others.

Exam question amount VP is another variation point closely related to the exam type, exam termination mode and grading policy variation points. For the stop-by-timer exam termination mode, there can either be a fixed question number or a dynamic question number. In the case of fixed question number, either a student finishes the given set of questions or runs out of time without completing all questions. If the student completes the exam ahead of time, we might want to use the actual time a student spends as one factor for the grading policy so that faster students are awarded with more credit.

In the case where an exam has a dynamic number of questions, a student is provided with a set of required questions first. If he finishes these questions and the timer has not run out, extra credit questions appear and the student can answer these questions to earn extra credit before the timer stops. The grading policy should include this extra credit as a factor.

For stop-by-completion mode, there is just one choice: to finish a predefined set of questions as quickly as possible. There is no time limit except for the exam expiration date. But the actual time it takes for a student to complete the exam will be one of the grading policy factors.

Another variation point related to the repository layer is the **question and exam editor and description language VP**. Most online exam systems have a GUI editor for teachers to prepare questions and exams. A teacher fills out input boxes with question properties using the GUI editor. Different question types require different

GUI elements for its properties. Besides the GUI editor approach, we can also use a plain textual description language to compose questions literally. The advantage of a textual question description language is that it allows offline editing and convenient exchange of questions between different exam systems. For example, Moodle has its own question description language called GIFT [73] and also allows import and export of questions in other format such as IMS QTI [74], XML and XHTML.

User type VP defines the various types of users in the exam system, such as student, teacher, TA, administrator, etc. They share some common properties such as username, password, ID, first name, last name, email, etc. If the user group feature is available, the user group ID should be added to user properties. If fee-based exams are available, personal financial information such as account number, credit card number, billing address, expiration date, etc. should be added to user properties. Besides these common properties, different user types require additional properties. For example, a student needs properties such as major, department, etc. A teacher has properties such as department, office, phone number, etc.

Statistical data type VP includes various statistical data including each student’s performance data, question and exam usage data, question and exam answers and statistical analysis, etc. **Communication method VP** specifies various ways to communicate between users. This can be point-to-point communication like email and instant messenger or text messages on mobile phones, this can be one-to-many approaches like twitter, blogs, podcast, or this can be many-to-many communication like online forum, mailing list or wiki.

5.2.3 Variability in the Middleware Layer

The middleware layer implements the business logic of an online exam system. It has modules for user management, question and exam management, communication channels, statistical analysis, and financial services. These modules have integrated security capabilities to prevent malicious attacks and cheating behavior. Security features also guarantee proper authentication and authorization to various resources and activities associated with these resources. In this section, we discuss different ways to implement these core modules in the middle layer: user management, question and exam management, exam answer and submission, user communication, data analysis and logging, financial service, and security. For each module, we also discuss how different design choices in the repository layer affect the design choices in the middleware layer.

User Management

The user management module is responsible for user registration, updating user information, deleting user accounts, etc. Note that different user types are associated with different routines. For example, a student can register a new student account, subscribe to user groups and classes, update his personal profile, search for other students, etc. A teacher can register a new teacher account, update his profile, search for students, manage classes, supervise students in his classes, assign TAs to classes, etc. A system admin has permission to access all user management data and operations. The system admin is also responsible for monitoring system status, providing technical support, etc. Also notice that the same routine can have quite distinct semantics for different user types. For example, both students and teachers may register online by themselves, but to be eligible to register as a teacher, security checking must be in place to make sure the request is from a real teacher.

We now examine variation points related to user management routines. **Registration VP** covers all possibilities of user registration. There are two ways to register a new user: individual mode and batch mode. Individual users can go to the registration page and input all the fields on the registration form, where some are required and some are optional. For security reasons more and more systems provide captcha (a distorted image only readable to a human user to prevent malicious form filling using robots) on the registration form. Each user type has special properties determined by the user type variation point. For example, a teacher is a core user type with access to sensitive data and its registration process should include extra safety mechanisms, e.g., some systems may ask for extra security screening to make sure the user is really a teacher. This can be done by a face-to-face meeting or a phone call with an administrator to obtain special credentials for teacher registration. Alternatively, the registration system can look up in the faculty database directory to match the email address and send a confirmation email to activate the user. Besides individual registration mode, all administrators can also perform a batch registration for a list of users. Note that a TA is first registered as a student and then promoted by a teacher or an administrator. An administrator is created when the system is first built and there is no registration for an administrator user type.

Question and Exam Management

The question and exam management module allows teachers and TAs to perform work related to questions and exams. New questions are created by providing question content, correct and incorrect answers and other question properties. Question content and answers may also be generated on-the-fly: in this case, teachers and TAs need to provide input parameter value ranges and algorithms to generate dynamic content and answers using these input parameters. We examine several VPs related to question and exam management below.

An important VP related to question and exam management is the **grading VP**. Each question answer made by a student has a grade, and an exam grade is calculated using algorithms that take into account many factors such as question grades contained in the exam, question difficulty levels, time consumption on each question, etc. The grading VP consists of child VPs such as grading policy and answer points. The **question answer points VP** defines points for correct and incorrect answers. The most common case is to have a question author assign points to correct answers and zero points for incorrect answers. Again, this choice might vary depending on question type. For example, for a MC question, say there are two correct answers but the student only selects one of them, we can either give half points or give zero points. Another choice for answer points VP is to let the exam composition engine distribute the points automatically. For example, if there are 10 questions in the exam and the total number of points is 100, the composition engine can evenly distribute 10 points to each correct answer. The **question grading policy VP** represents algorithms to calculate a grade for a question answer. In addition to the question answer points VP, the grading policy VP can be affected by other VPs such as difficulty level, time consumption, extra credit, etc. Difficulty level can add additional weight for question answer points—in an exam, a more difficult question can have its answer points multiplied by a weight to reflect the fact that it is a tough question. We can also add a time consumption factor so that the more quickly a student finishes a question (statistically speaking with comparison of other students) the more extra credit a student earns.

The **exam grading policy VP** is based on the question grading policy VP. An exam grade is computed by simply adding up each individual question grade. Extra credit may also be added based on the exam stop mode and exam question number VPs. For example, when the exam is in stop-by-timer mode and has a dynamic question number, a student finishes required questions early and goes on to answer extra credit questions. This extra credit should be added to the final exam grade. When the exam is in stop-by-work mode, if a student finishes all questions quickly, he might receive extra credits based on the statistical time consumption of all the students.

Another tricky VP related to question and exam management is about archiving questions and exams. This is called **exam and question version history VP**. The most common practice is to overwrite the last version with the latest version and always uses the latest version. The only problem is that old content is permanently lost and authors may never know how and why the previous authors did that change. Another less common way is to make questions and exams read-only after they are created. This choice is easy to implement and maintain since there is no modification problem to worry about at all. But the problem is sometimes there are bugs in the questions and exams that are not caught during the creation phase. Read-only questions and exams make it impossible to fix these bugs and the only way is to disable the old questions and exams and create new questions and exams that fix the bugs. However, references to the old questions and exams still exist throughout the system and thus they need to be updated carefully, which requires a lot of effort. Another choice is to use simple linear version control. When a question or an exam is used, the default behavior is to use the latest version in the version history. This would be an interesting approach for modification property VP although none of the systems studied in the domain analysis actually implement this idea.

Exam Answer and Submission

So far, this section has presented VPs related to questions and exams in the repository layer. Here are some additional VPs in the middleware business logic layer related to the way a student answers questions and submits these answers. The first one is the **number of exam attempts VP**. Most exams only allow a student to take an exam once. But in systems focusing on educational and learning goals, answering questions is just a way of learning. In these systems, a student can take on exam as many times he would like. Based on other exam VP settings, the student might see exactly the same exam repeatedly, or different exams each time. Another VP related to how a student takes an exam is the **exam submission mode VP**. A student can either submit a question immediately after he completes its answer (single question submission mode) or wait until all questions are submitted (batch submission mode). Single question submission mode can prevent accidental data loss caused by power loss or hardware failure, but it is more expensive in terms of network communication cost.

Batch submission mode is good for a student to go back to review finished questions and make changes before final submission.

User Communication

The most common way to communicate among users is email. The benefits of email is that it is non-interruptive. If a teacher many student emails in a short period of time such as right after the final exam, he can choose to ignore it until he has finished work at hand. Privacy is another benefit of using emails—a student can perform one-on-one conversation with a teacher and keep it only between them. Email is also easy to use because it emulates the way people exchange snail mails: compose, send and receive. The drawbacks of using emails include difficulty to track and organize related emails, and the possibility of redundant replies. For example, if ten students ask related questions, a teacher has to compose similar but not exactly the same answers to each of them if there is email content cannot be shared among students.

Another communication method is online instant messaging. The good thing about instant messages is that it enables real time access to another person. The down side is that instant messages are very interruptive and unmanageable if there are too many conversations. It is challenging to keep track of each topic when a user still has work at hand.

Online forums and wikis are two good ways to build community-based interaction. They both offer collaborative features such as topic-oriented discussion, FAQs, polls, file sharing in the message attachment, etc. They are effective ways to broadcast messages, provide first-hand information. For example, sometimes a student may understand another student's question better than a teacher or a TA because they have shared learning experience.

Blogs and podcasts are new ways for one-to-many communication. For example, a teacher can post presentation slides, podcasts and reading materials on his blog and the students uses an RSS reader to keep track of latest updates. In some online exam systems, Twitter is integrated to help quickly broadcast a user's status to others.

Statistical Analysis and System Logging

The statistical analysis module is responsible for collecting and aggregating performance data to better serve students and teachers. It provides an overview of a student's learning performance, a teacher's instruction effectiveness, and the quality of teaching materials. For example, besides reading each exam grade report, a student can see his standing among the class using a grade distribution graph. If questions are associated with topics, a student can check to see in what topics he did best and what topics he did worst. A teacher can also obtain first-hand analytical data for his class and understand where to allocate resources that could best help improve student performance. These statistical data include distributions for exam grades and exam duration, commonly occurring mistakes, performance data related to specific topics, and the performance curve for each student.

The system administrator also requires logging information about system usage, such as connection IP address, user log in and log out timestamp, question and exam answer submission timestamp, etc. By studying the system log, a system administrator can identify suspicious user activity or perform optimization like load balancing.

Financial Service

The financial service module executes of transactions such as charging students for using the exam system, and paying teachers, TAs and administrators for their contributions to maintain a healthy learning system. This module is mostly seen in commercial systems [71, 82]. Each user type involved in financial transactions needs to provide personal financial identity information to the financial service provider. And these third party service providers perform secure and reliable transactions such as deducting a student's account and paying a teacher for his contribution. Special algorithms are implemented in the financial service module to calculate contributions and convert them into financial compensation. There are many factors to take into consideration: number of questions and exams made, number of times a question and an exam is used, user ratings and feedback for a question and an exam, etc.

Security Issues

The security aspect not only guards personal information and sensitive data such as exam grades, user profile and financial information from unauthorized access, but also provides anti-cheating features. Below is a list of features related to online exam system security.

- Authentication using username and password
- Resource access permissions based on user profiles
- Secure HTTP connections (SSL/TLS)
- Fine-grain security settings for questions and exams. For example, a teacher can set exams and questions that he composed to be public (can be reused in other teacher's classes and exams) or private (only available to himself). An access password can be set for an exam so that only those who know the password can take the exam. Other restrictions can also be set on an exam, such as limiting user IP addresses, bind each student with his static IP address, etc.

- Present the exam in a way such that exam materials cannot be copied or printed easily, for example, disable printing from Web browser, use non-text format to avoid copy-n-paste.
- For a static exam (those exams have their question pools fixed), try to randomize the question ordering and answer ordering. Try to use a dynamic exam (those have their questions dynamically selected from the question bank) as much as possible.
- Regularly update the question bank by adding new questions and removing overly used and outdated questions.
- Monitor system status and keep logs for user access and run background daemons to alarm when abnormal access behavior patterns are observed.
- Perform regular backups of the system repository.

5.2.4 Variability in the User Interface Layer

The third layer is the user interface, the part that has direct contact with users. Online exam systems have UI components similar to other Web applications. The most frequently used widgets are the form components that collect user inputs and submit them to the server side for further processing. Some of these commonly used widgets include Label, Text Field, Text Area, Checkbox, Multiple Choice Checkbox, Dropdown Choice, Radio Choice and Button. Capabilities of these UI components also depend on the Web application framework and Webpage authoring technologies used to implement the online exam system.

Each business logic module has a set of Web pages associated with it. So, we have a login page as the starting point for each user type. For new users, there are links to the registration pages. Once an existing user logs in, he will be redirected to his homepage based on his user type. Each homepage contains links to activities associated with this user type. For example, a student can see activities like take open exams, view graded exams, change personal settings, email tools, online forum, etc. To perform a specific activity, the user just clicks on the link widget, which takes him to the activity page. Once a user finishes his intended activities, he can just log out of the system or close the Web browser to end the session.

6 Feature modeling language: FeatureML

Chapter 6 begins, and the next several chapters continue, a detailed presentation of the Rhizome feature modeling and code generation platform. Chapter 6 presents the feature modeling language. Chapter 7 introduces the code generation patterns, code templates and how the template-based code generation works. Chapter 8 reveals details for the code generation process and the code generator architecture. Throughout these chapters, an online exam system is used as a concrete example to demonstrate how the feature model to code generation process works and highlight many advantages of using the Rhizome platform for rapid software product line development.

6.1 Variability model and feature model

A variability model contains all design choices and dependencies among design choices for a specific domain. It captures existing design knowledge for a domain and guides a designer in making design choices and ruling out invalid design choice combinations until a valid feature model is found that meets requirements. Figure 6-1 is a simplified variability model for the online exam system domain. It only shows one feature variation point of user types, and provides four design choices for user types: Admin, TA, Student and Teacher. Each choice has its own design choice substructure. For example, the Student design choice consists of username and password, representing data modeling decisions for a student. Similarly, the Teacher design choice contains username, password, department and office, which also represents data modeling decisions. The activities sub-feature of Teacher is different and it represents behavior design choices.

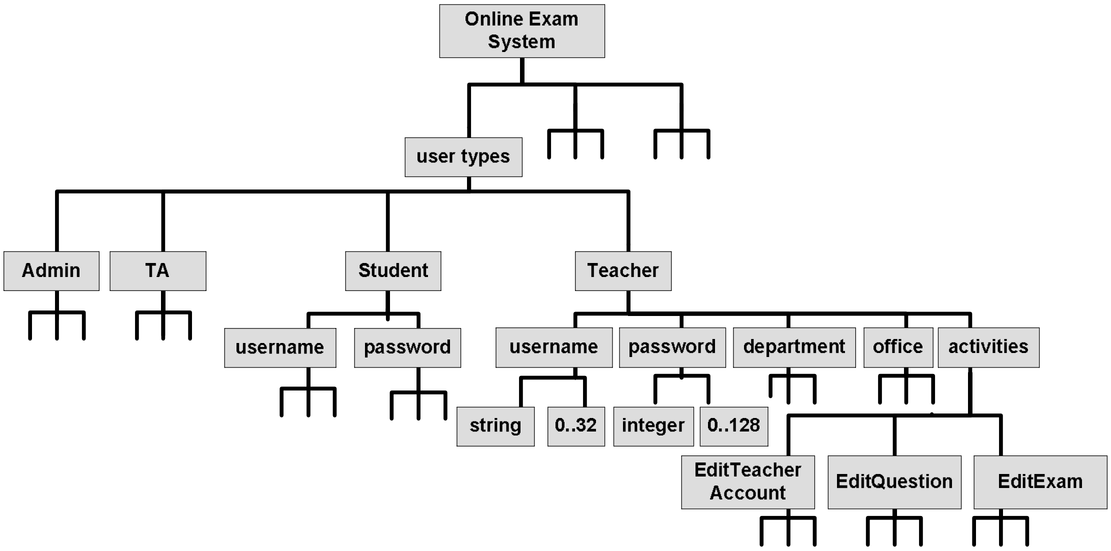


Figure 6-1. A portion of the variability model for an online exam system.

A feature model captures a set of valid design decisions and represents one of many possible ways to design a system. Figure 6-2 shows a feature model obtained by making the design decisions and selecting only two user types, Student and Teacher, out of the four possible user types available in Figure 6-1. For the Teacher user type, we only selected EditTeacherAccount activity and hence EditQuestion and EditExam are not present in Figure 6-2.

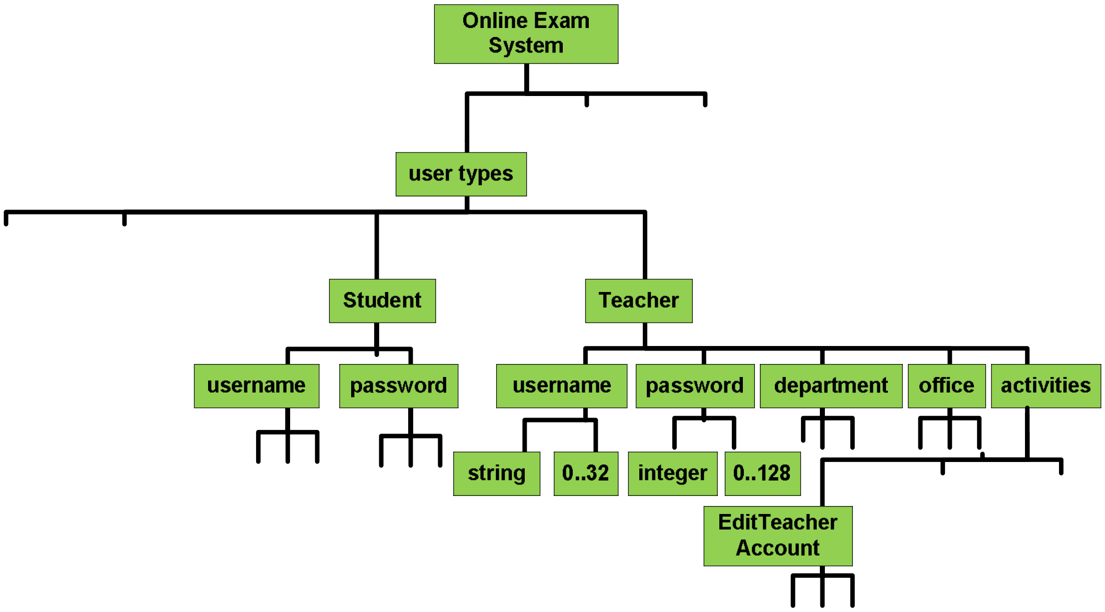


Figure 6-2. A feature model derived from the variability model in Figure 6-1.

Based on their natures, features are divided into two categories: data features and behavior features. For example, *user types* is a data feature that defines property structures and data content for user types. The leaf nodes define content for the property structure. For example, a username property has a string value type and minimum of 0 and maximum of 32 for value length.

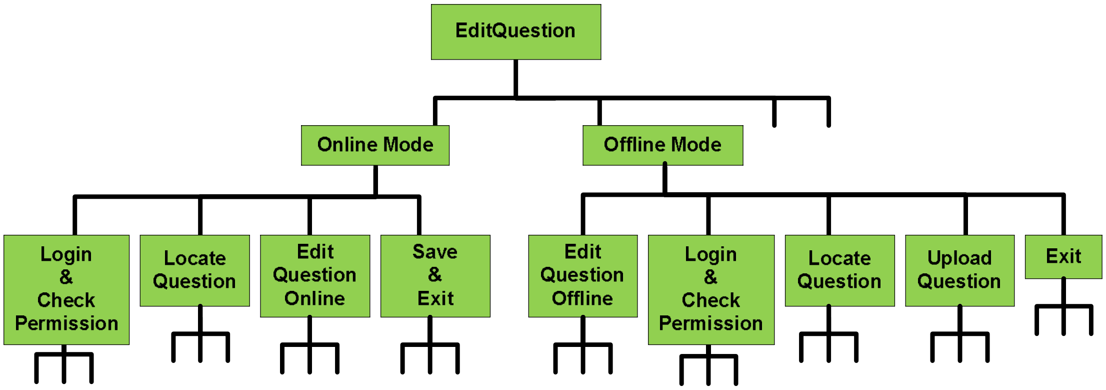


Figure 6-3. The expanded view of the variability model for the EditQuestion activity.

Behavior features describe the dynamic aspects of a feature model. They capture various ways to design state changes and thus they can model dynamic behaviors for a system. For example, the feature called activities defines operations associated with each user type. As shown in Figure 6-1, a teacher user type has activities such as EditTeacherAccount, EditQuestion and EditExam. Each activity has a substructure in a similar way like a data feature does. Figure 6-3 shows an expanded view for the EditQuestion activity variability model. For the EditQuestion activity, there can be multiple ways to implement it. One choice is the Online Mode, where a user needs to connect to the online exam system and edit the question using online editor. The other choice is the Offline Mode, where a user can edit the question offline and upload it later to the exam system. Notice their substructure contains basic workflow, which can be further refined into their own substructures.

A behavior feature is usually affected by many other feature variation points—the more influencing variation points, the bigger the number of possible design choices for a behavior feature. For example, in Figure 6-1, the Login & Check Permission sub-feature can be designed in two ways: allowing only the original question owner to edit the question, or allowing all users from an authorized group to edit the question (e.g. all Math teachers can edit all the Math questions). The Edit Question Online sub-feature also contains multiple design choices depending on the question type. For example, the online editor for a Cloze question is dramatically different from a True or False question. Multiplying the number of design choices for each sub-feature, the total number of design choices for EditQuestion feature can be calculated, which could be a huge number. But not all these combinations are valid and variability repository validator is the tool to enforce feature dependencies and rule out invalid combinations.

Besides recording various feature design choices, a feature model also contains additional elements such as code generation instructions and traceability information between design choices and code generation. There are different types of code generation instructions and each one corresponds to a unique generation pattern. Instructions also contain parameters to be used by the code generator, as well as queries to find values for these parameters. The Rhizome code generator uses a template-based generation approach and these templates are associated with feature design choices and specified in the feature model.

6.2 FeatureML overview

The FeatureML language is the feature modeling language used in Rhizome. The language syntax is implemented as an XML schema and thus each feature model is an XML document conforming to the language XML schema. Appendix C shows the graphical and textual Schema for FeatureML language. There are three basic element types in FeatureML: vp_entity, vp_application, and code_generation_unit. The vp_entity type represents user features and data features, which cover static aspects of a feature model. The vp_application type defines behavior features. The code_generation_unit type stores instructions for code generation. Both vp_entity and vp_application types have a positive integer attribute called vp_id and elements of these two types share the same id space. The code_generation_unit type has its own id space and its elements use the attribute unit_id to uniquely identify themselves.

Both vp_entity and vp_application elements can hold multiple option sub-elements: each option sub-element represents a design decision the designer makes for that feature. Note that all these options are design decisions made by a designer during the feature modeling process. Different options are just there to serve different circumstances because the same feature may be used for various purposes. The feature needs to be designed differently for these circumstances. One example is the user_props feature—it defines properties for a user, such as name, username, email, address, etc. However, the user may exhibit different sets of properties for different applications: almost all user properties, except those automatically generated by the system such as integer userid, should appear on the user registration page, but on the user management page that lists properties of all users, sensitive properties like password and SSN should be hidden. To model this scenario, we can store these two different property sets as two options for user_props feature, one for user registration and the other for user management. Otherwise, there is no way we will know which set of user properties to use for which application.

This option sub-element in a feature model is a unique contribution. Most existing feature modeling work does not need to generate application source code, so there is no need to match a feature design choice with the final application functionality. But for the Rhizome work, this is definitely needed to be clearly defined in the feature model so that the code generator knows which option to use for which application functionality.

The vp_entity element stores information about user feature and data feature including choices for feature structures and feature values. A vp_entity element needs to contain at least one option element. An option element can contain one of four possible sub-elements: enum_value, scope_value, sub_option, and extended_option (see Appendix C for a graphical structure of the vp_entity element). Figure 6-4. UML analogy of the FeatureML option element.shows an analogy of these four option types in the UML class diagram with aggregation and extends relationships.

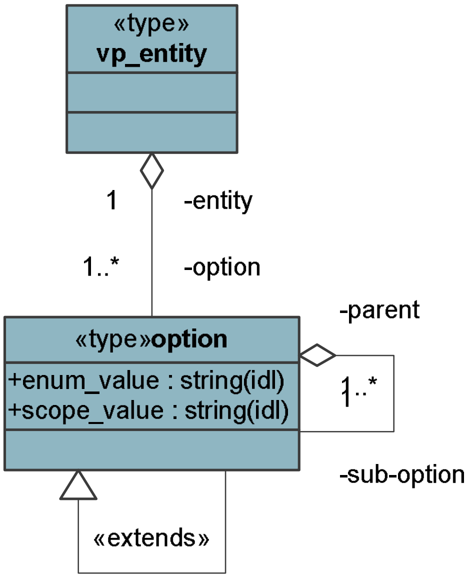


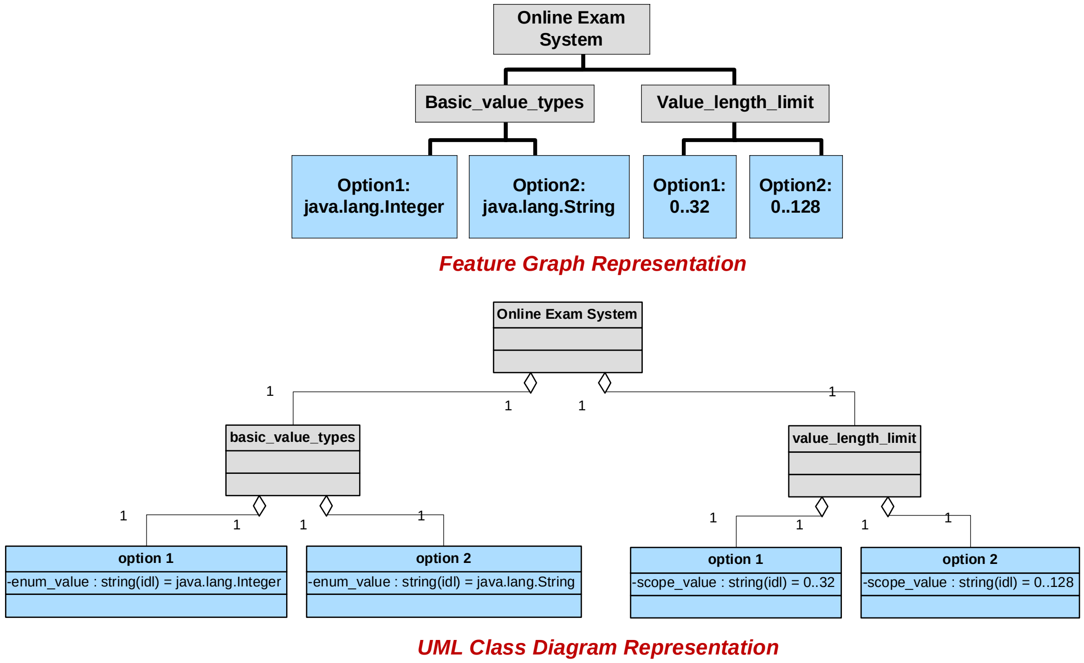
Figure 6-4. UML analogy of the FeatureML option element.

enum_value: Each enum_value defines a possible value for the option. For example, a vp_entity for color contains two options: one option contains an enum_value for RED, the other option contains an enum_value for BLUE. This means this feature model contains two color options RED and BLUE.

scope_value: Each scope_value defines an integer range value. For example, a vp_entity represents the user input length: on a Web form, we want to limit the length of a password input box to be within 0 and 16, and the length of a text area to be within 0 and 512. So, this user input length vp_entity contains two options: one option has a scope_value of 0..16 and the other one has a value of 0..512.

sub_option: Each sub_option contains a reference to another feature design option. Using this element, we reuse existing feature design options and build a graph-like feature structure. UML aggregation for the class diagram is an analogy for this reference relationship between an option element and its sub_option elements.

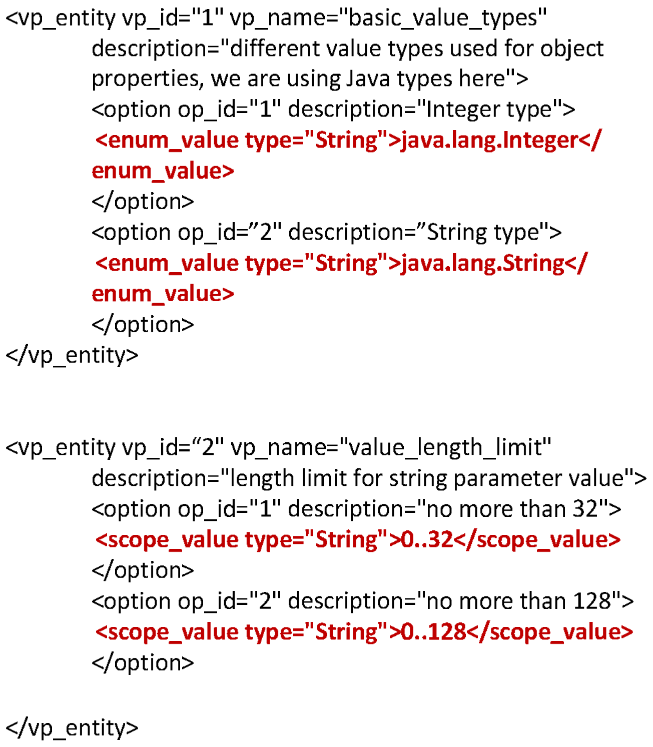
extended_option: Each extended_option defines an inheritance relationship with its parent option. This option type is an analogy of the UML extends relationship for the class diagram. It allows a common set of options to be shared with a group of options easily. For example, a user_props vp_entity is used to define common user properties such as username and email. Then student_props can be defined, which contain username, email and class call number, and teacher_props, which contain username, email, office and phone number. An extended_option is specified to extend user_props, which contains common user properties shared by student_props and teacher_props. Unique properties are defined using additional_sub_option element defined within an extended_option.


Online Exam System
Basic_value_types
Value_length_limit
Option1: java.lang.Integer
Option2: java.lang.String
Option1: 0..32
Option2: 0..128
Online Exam System
basic_value_types
value_length_limit
option 1
-enum_value : string(idl) = java.lang.Integer
option 2
-enum_value : string(idl) = java.lang.String
option 1
-scope_value : string(idl) = 0..32
option 2
-scope_value : string(idl) = 0..128
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1

Feature Graph Representation

UML Class Diagram Representation

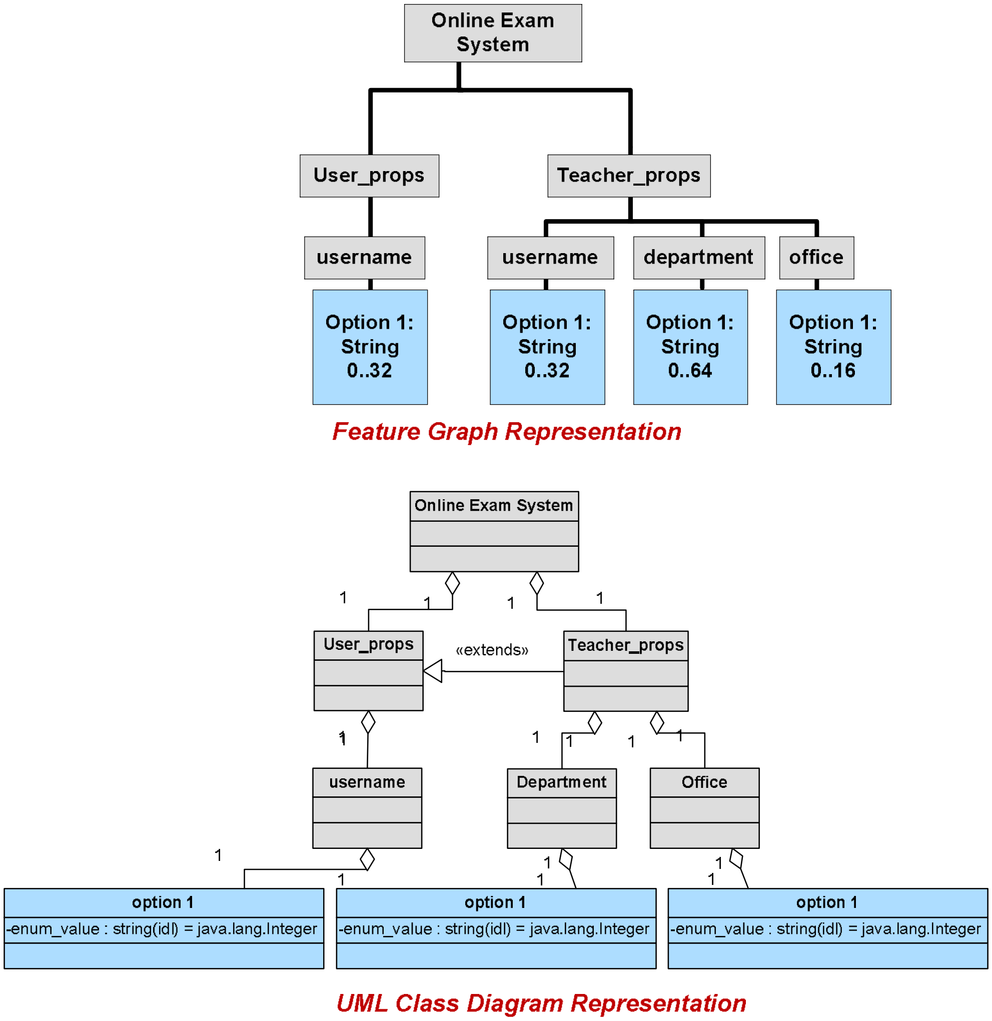
Figure 6-5. vp_entity example for enum_value and scope_value option types represented graphically using feature graph and UML.



```
<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>


<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 32">
           <scope_value type="String">0..32</scope_value>
          </option>
          <option op_id="2" description="no more than 128">
           <scope_value type="String">0..128</scope_value>
          </option>

</vp_entity>
```

Figure 6-6. vp_entity example (same example as in Figure 6-5) for enum_value and scope_value option types represented in FeatureML language.

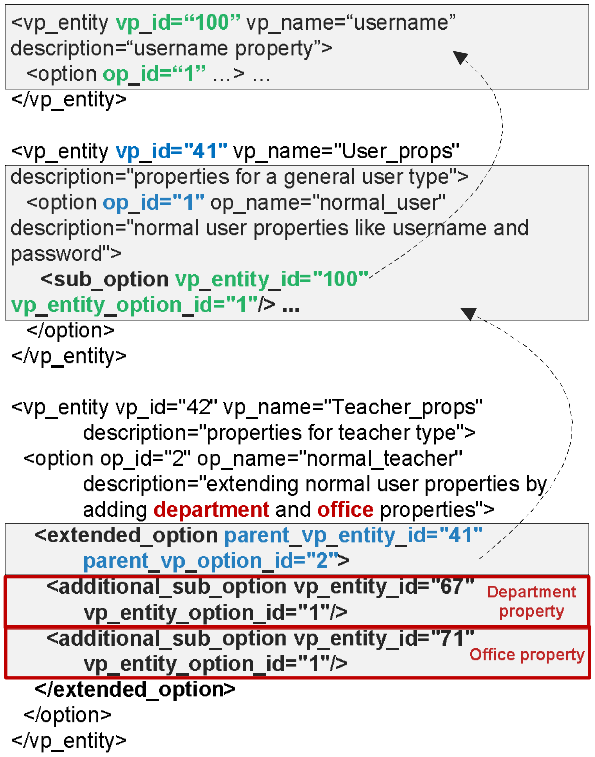
Figure 6-5 and Figure 6-6 show the same example for using enum_value and scope_value option types. A vp_entity element called basic_value_type is defined, which contains data types to be used in the system. Since the generated source code is implemented in Java, two basic types are included: java.lang.Integer for integer values and java.lang.String for string values. Another vp_entity value_length_limit uses scope_value options: it defines two length scopes, one between 0 and 32 and the other between 0 and 128. Note that FeatureML does not have a graphical representation yet, we borrow graphical notations from feature graph and UML to present these two vp_entity elements graphically. The original feature graph represents all design choices in the same way—it does not differentiate the existence of a feature and how many options the feature has. The UML class diagram has similar problems since these options are actually instances of the option class and they are not classes by themselves. So both graphical representations make it easier to understand the example described in FeatureML and they do not match precisely with the FeatureML semantics.



Feature Graph Representation

UML Class Diagram Representation

Figure 6-7. vp_entity example for sub_option and extended_option presented using feature graph and UML.



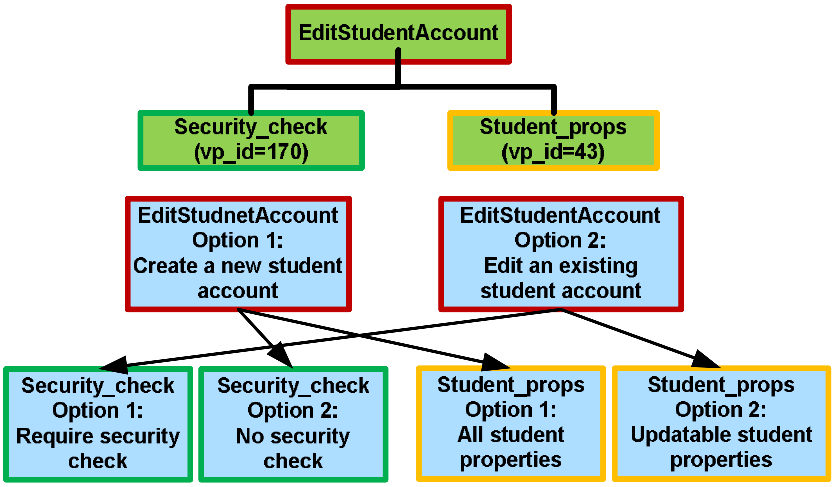
```
<vp_entity vp_id="100" vp_name="username"
description="username property">
  <option op_id="1" ...> ...
</vp_entity>

<vp_entity vp_id="41" vp_name="User_props"
description="properties for a general user type">
  <option op_id="1" op_name="normal_user"
description="normal user properties like username and
password">
    <sub_option vp_entity_id="100"
vp_entity_option_id="1"/> ...
  </option>
</vp_entity>

<vp_entity vp_id="42" vp_name="Teacher_props"
          description="properties for teacher type">
  <option op_id="2" op_name="normal_teacher"
          description="extending normal user properties by
          adding department and office 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>
```

Figure 6-8. vp_entity examples (same example as in Figure 6-7) for sub_option and extended_option types. The dotted arrows in the figure are references to other vp_entity options.

Figure 6-7 and Figure 6-8 is an example demonstrating how to use sub_option and extended_option types. In this example, User_props vp_entity element defines a set of user properties and one of them is username property. The username property is modeled as a separate vp_entity element and we use a sub_option to reference the option chosen for the username vp_entity element. The sub_option type allows construction of a vp_entity using references to other feature options and thus forms a hierarchical feature design structure.

The extended_option type implements a basic inheritance structure in FeatureML. Our example has two vp_entity elements: User_props that includes common properties for all users such as username; The Teacher_props “extends” the User_props by adding its unique properties of department and office. This option type allows sharing common design options at the parent vp_entity element level.

The vp_application element contains behavior feature information, including textual description, dependencies to other vp_entity, vp_application elements and dependencies to code generation instructions. It also has reference information for code generation instructions defined in code_generation_unit element.



```
<vp_application vp_id="2100" vp_name="EditStudentAccount" description="input
      property values to create a new Student or edit an existing Student">
  <option op_id="1" op_name="newStudent" description="create a new student
      account from scratch">
    <dependency dep_name="self_registration" dep_explanation="authorization not
      required since the user does not have a student yet" vp_id="170"
      vp_option_id="2">
      <ref_CGU ref_CGU_id="49" ref_CGU_name="RegistrationPage_java"
      description="generate java class for registration page"/>
    </dependency>
    <dependency dep_name="student_props" dep_explanation="for each student
      property, create its widget on the edit page" vp_id="43" vp_option_id="1">
      <ref_CGU ref_CGU_id="10" ref_CGU_name="Student_java"
      description="generate java class for Student"/>
    </dependency>
  </option>
  <option op_id="2" op_name="existingStudent" description="edit an existing
      student account">
    <dependency dep_name="authorization_check" dep_explanation="need to
      check authorization before account editing action" vp_id="170"
      vp_option_id="1">
      <ref_CGU ref_CGU_id="49" ref_CGU_name="RegistrationPage_java"
      description="generate java class for registration page"/>
    </dependency>
    <dependency dep_name="student_props" dep_explanation="for each student
      property, create its widget on the edit page" vp_id="43" vp_option_id="2">
      <ref_CGU ref_CGU_id="10" ref_CGU_name="Student_java"
      description="generate java class for Student"/>
    </dependency>
  </option>
</vp_application>
```

Figure 6-9. vp_application example for behavior feature and feature dependencies.

Figure 6-9 is a behavior feature for editing a student account. There are two different design choices: one for creating a brand new student account, the other for editing an existing student account. Each choice depends on choices of other features. For the feature to create a new student account, it is specified that no authorization is required and self-registration is allowed. It is also defined that all student properties are required for a new student registration. The other option is for editing an existing account. The system requires an authorization check. We also specify that only a subset of account properties appear because properties like username can only be set once at creation time. Each dependency element also contains ref_CGU sub-elements to connect with code generation instructions. These code generation instruction elements are referenced by their ids (ref_CGU_id in Figure 6-9).

A code_generation_unit (CGU) element specifies code generation instructions. It includes the following items: code generation pattern, template file location, template parameter queries. Many code generation patterns are identified based on the study of software systems in the online exam systems domain. These patterns define code generation activities, e.g. create multiple files based on a file template, replace tags embedded in a template with dynamically generated code blocks, etc. Template files capture commonality and variability for the system source code. Variability is modeled as tags to be replaced with dynamically generated code blocks. Commonality is represented as source code blocks that wrap around these tags. To dynamically generate code blocks, user design decisions in the feature model need to be translated into parameter values so that these values can be used to produce the complete source code. The template parameter queries are used to find parameter values using a specially designed query mechanism based on the in-memory parameter data structure.

Figure 6-10 provides a graphical overview of how vp_entity, vp_application and CGU elements are connected in a feature model. Each vp_application option is dependent on several CGU elements to connect to the code generator. In this example, we only show one CGU element that does a global string replacement code generation. This CGU element is in turn dependent on another two vp_entity options: username takes the option of 0 to 32 character string and password takes the option of 0 to 128 digit integer. The entity dependency defined on a CGU is usually used for parameter queries to find proper parameter values to be used in code generation. In the next chapter, we will discuss details about how CGU works.

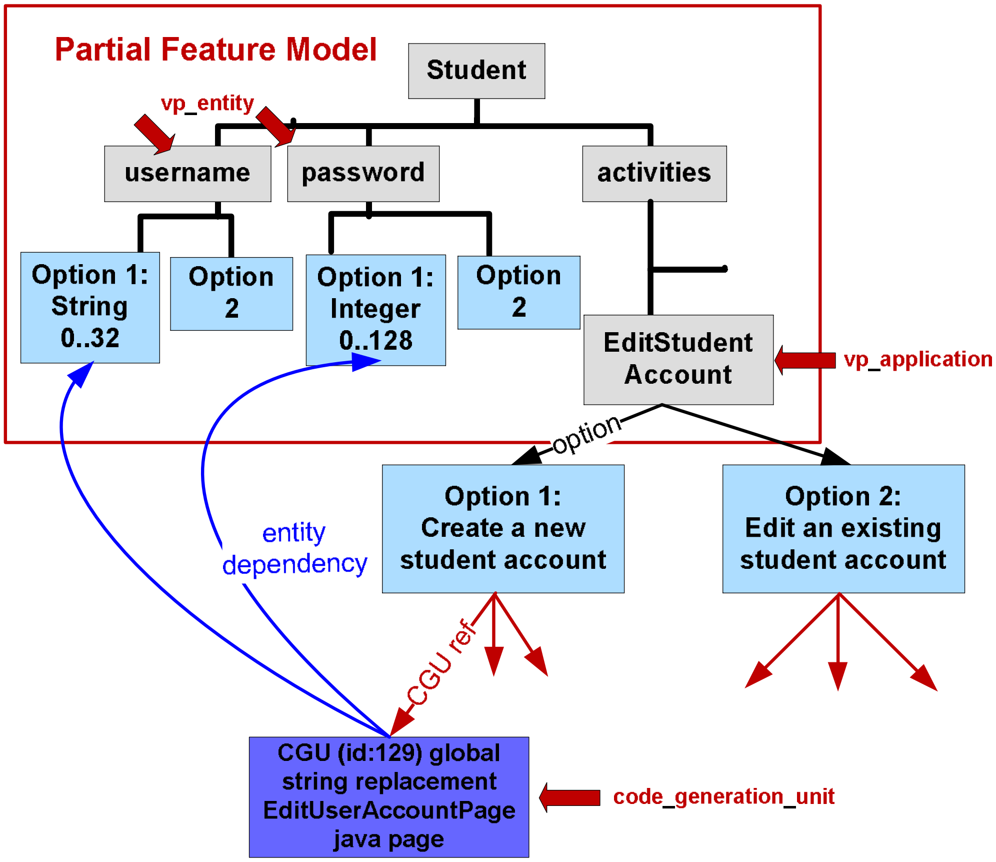


Figure 6-10. Graphical explanation about how CGU is connected with vp_application and vp_entity.

APPENDIX E: Online Exam System Feature Model Example in FeatureML shows a complete feature model of an online exam system represented using FeatureML. A running online exam Web application is generated using this feature model together with the associated template files. A feature model organizes user design choices in a standard format and gave them specific semantic meanings and relationships. In the next chapter, we discuss details of our template-based code generation approach using CGU and templates.

7 Code generation unit and template language

In the previous chapter, we focused on the feature modeling side, where a feature model designer makes various design decisions about what user entities and data entities to include in a feature model, what specific design options for each entity, and what dynamic behaviors each entity will demonstrate. All these design decisions are captured in the feature model. However, these decisions are in textual form and contain domain-specific semantic meanings. In this chapter, the connector element between a feature model and the code generator—Code Generation Unit (CGU) element is discussed in detail.

Figure 7-1 gives a brief overview about the role that CGU plays in the code generation process. CGU element is defined as part of the feature model. It translates feature design decisions about vp_entity and vp_application elements into parameter values stored in a parameter lookup table. Based on these design decisions, a set of templates are selected from the code template repository. Each CGU element references one template and defines the queries to retrieve parameter values for the template. Then, the code generator processes all CGU elements and produces the source code that implements the feature model.

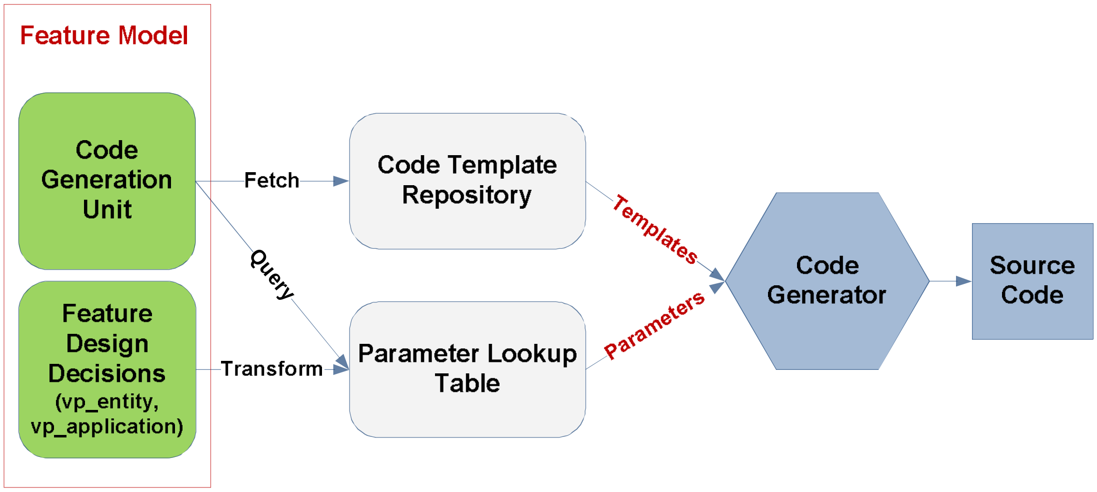


Figure 7-1. The role of CGU in the code generation process.

The Rhizome template-based code generation is specifically designed so that variability in a feature model can be reflected and connected with tags in code templates. Chapter 3 explains that most existing work either focuses on methods to represent a feature model and design variability in a domain or focuses on mechanisms to implement a code generator. Rhizome is among the first several systems that offer automated code generation based on feature models.

Compared with other possible alternatives, this template-based code generation is more flexible, easier to upgrade and much simpler to implement. Model-driven code generation is still not mature and mostly used for specific domains such as content repository, Web applications. In addition, variability modeling is not supported in the models for model-driven code generators. Extending a model-driven code generator such as Eclipse Modeling Framework (EMF) requires tremendous work including modifying existing modeling languages and the JET code generation engine (JET is also a template-based generator, by the way). The snippet assembler code generators use snippets that have been prepared beforehand and select a subset of these snippets based on parameters. These snippets are specific to domains and the assembler generators do not scale well. The Rhizome generator has a simple architecture—it is a text processor to generate code based on different directives and no domain knowledge is built into the generation engine. The programming language and architecture design for the generated code is decided by the code templates. If the templates are using the Java language, the generated system will be implemented in Java. If the templates are using specific frameworks like Spring, the generated code will be using the same Spring framework. Rhizome also makes it easier to accommodate new code generation requirements such as adding a new generation pattern. We just need to add a new pattern processing function in the generation engine and update the template language to insert a new element representing this new pattern. As for the implementation, Rhizome does not require either complicated model-to-model transformation or abstract logic thermo proving—everything is based on text processing with parameters and code templates.

7.1 Parameter Value Lookup Mechanisms in the Code Generator

7.1.1 Transform Feature Design Decisions into Parameters

The feature design decisions are stored in the feature model but the code generator needs to understand the semantic meaning for these choices and obtain properly typed values in order to produce valid source code. CGUs define code generation instructions, provide queries to fetch parameter values and match parameters with code templates. The code generator then processes CGUs and produces the final source code.

Feature design decisions are first converted into an in-memory data structure called parameter lookup tree. It is a multi-level hash table and Figure 7-2 shows the detailed structure of its content used for our demo online exam systems. Note that this lookup tree is domain-specific and it contains entities for our sample online exam system domain such as users, questions, exams and applications associated with these entities.

For a different domain, this lookup tree structure might be quite different. By converting an XML feature model document into a parameter lookup tree, we achieve two goals: first, domain-specific design decisions are translated into parameter values; second, a better performance can be achieved by storing parameter values in the memory.

The code generator is also independent of domains—it is simply a template processor and knows nothing about semantics for the code that it produces. The code generator executes the code generation instructions by processing a code template according to a code generation pattern and replace parameters embedded inside a template. However, each feature model is targeted for a particular domain—a design choice on the properties list for the question entity has particular meaning and implication for an online exam system. So, design decisions must be parameterized so that the code generator has the input values to generate source code. The actual domain semantics are implemented manually in the code templates, which we will talk about later in this chapter.

The second reason to have this parameter lookup tree is for performance. Most of the parameters are used in multiple locations in multiple templates, so their values are stored in the memory to save the result of repeated queries. We also want to separate the parameter value lookup component from other part of the code generator to have a clean architecture design. Other XML query technologies such as XQuery and XPath are also explored, but we decided not to use them because they add extra complexity and provide no query results caching.

Figure 7-2 presents this customized lookup tree structure. The lookup tree consists of two parts: entity definitions and application definitions. Entity definitions cover all the user and data entities (vp_entity elements) and their properties. Entity names are obtained by searching for vp_entity elements with the name pattern of “EntityName_props”. Properties are classified into two groups, single property and collection property, based on the property data type. For example, a *name* property is a single property with the String data type; a *questionAnswers* property is a collection property that contains a list of question answers (List and Set are collection data types). Property definitions can be found by following references in the entity definitions. For example, a Student user entity contains multiple sub_option elements, each referring to a property definition element. To differentiate a single property and a collection property, specific name patterns are used—a collection property has a name of “CollectionPropertyName_collection” and a single property uses a name of “SinglePropertyName”.

Application definitions cover all applications (vp_application elements). Each application definition includes a list of application properties including property name, property type and GUI widget used in the application.

You can also insert additional hash tables based on your needs. For example, we used Spring framework, Hibernate Object Relational Mapping, and MySQL relational database in our demo application of online exam systems. So, extra hash tables are needed to define data type mappings used in configuration files and database schema definition.

In summary, feature design decisions are converted into parameters and values of these parameters are stored in the parameter lookup tree stored in the memory. Each CGU definition specifies how to look up values for each parameter used in a code template. When the code generator processes the CGU, it will find parameter values and perform text transformation using the template and its parameter values to generate the final source code.

7.1.2 Three Parameter Value Lookup Approaches

Once the parameter lookup tree is built in the memory during the initialization phase of the code generation, queries can be launched to fetch values from this lookup tree.

There are three different ways to search for parameter values in the current system: *String*, *Dictionary,* and *Function*.

1) **String**: A literal string value provided by the designer. For example, in a CGU, a parameter can be defined with the String lookup type and provide a literal string value. The code generator simply uses this value at all occurrences of the parameter.
2) **Dictionary**: In this approach, CGU defines a path expression to search for parameter values in the parameter lookup tree. For example, the path expression "questionDefs.MC.props.propName" will find all the property names for the Multiple Choice question type. The code generator understands the grammar of path expressions and explores this multi-level hash table to find the parameter values.

```
////////////////////////////////////////////////////////////entity definitions data structure////////////////////////////////////////////////////////////
//userDefs is a list of entityDefs, each for a user type such as Teacher, Student
userDefs = [entityDef, …]

//questionDefs is a list of entityDefs, each for a question type such as MC, TF
questionDefs = [entityDef, …]

//examDefs is a list of entityDefs, each for an exam type such as Exam
examDefs = [entityDef, …]

//questionAnswerDefs is a list of entityDefs, each for a question answer type such as QuestionAnswerPerStudent
questionAnswerDefs = [entityDef, …]

//examAnswerDefs is a list of entityDefs, each for an exam answer type such as ExamAnswerPerStudent
examAnswerDefs = [entityDef, …]

//entityDef is a dictionary containing a list of key:value pairs for entity definition information
//name is the entity name, parent is the parent entity name if there is one, props is a list of simple entity properties
//collections is a list of collection entity properties, e.g. MC entity has collection property questionAnswers to record student answers
//mappings is a list of Hibernate mapping definition information
entityDef = {'name':name, 'parent':parent, 'props':props, 'collections':collections, 'mappings':mappings}

//props is a list of simple entity property definitions
props = [prop, …]

//prop is a dictionary for an entity property definition, it includes information for propName, propType, lengthLimitLower and
lengthLimitUpper (for value types like String), valuePattern (for properties like phone number), propSQLType (a matching SQL type for
current property type), skippedTemplates (a property may not appear in certain templates)
prop = {'propName':propName, 'propType':propType, 'lengthLimitLower':lengthLimitLower,
'lengthLimitUpper':lengthLimitUpper, 'valuePattern':valuePattern, 'propSQLType':propSQLType,
'skippedTemplates':skippedTemplates}

//a property may not appear in certain templates, this list includes paths to those templates that current property cannot appear in
skippedTemplates = [templatePath, …]

//collections is a list of collection entity property definitions
collections = [collection, …]

//a collection property is one that has a Collection type such as Set, List, etc., colName is the name for the collection property,
//colType is the interface type such as Set, List, etc., colTypeImpl is the implementation class for the interface type such as HashSet,
//ArrayList, etc., colMemberType is the class type for the members in the collection property.
collection = {'colName':colName, 'colType':colType, 'colTypeImpl':colTypeImpl, 'colMemberType':colMemberType}

//mappings is a list of Hibernate mapping information related to a collection property (a collection mapping in Hibernate needs these
//mapping information)
mappings = [mapping, …]

//mapping definition includes type (Hibernate mapping types: one-to-many, many-to-one, many-to-many, etc.), name (name of the
//collection mapping), targetClass (class on the other end of Hibernate mapping), table (table name in the RDBMS referenced in this
//mapping), column (column name in the table that is referenced or used as key)
mapping = {'type':type, 'name':name, 'targetClass':targetClass, 'table':table, 'column':column}

////////////////////////////////////////////////////////////application definitions data structure////////////////////////////////////////////////////////////
//appDefs is a dictionary, each item uses entityName as key and apps dictionary as value,
//e.g. Exam entity is related to applications like DeleteExamConfirm and EditExam
appDefs = {'entityName':apps, …}

//apps is a dictionary, each item uses appName as key and a list of application property definitions as value
apps = {'appName':props, …}

//props is a list for application property definitions
props = [prop, …]

//prop is a dictionary for an application property definition, it includes information for propName, propType, and GUIWidget
//e.g. tags property in EditExam has the propName of tags, propType of String and GUIWidget of RequiredTextField
prop = {'propName':propName, 'propType':propType, 'GUIWidget':GUIWidget}
```

Figure 7-2. Parameter lookup tree data structure for online exam systems.

3) **Function**: When neither String nor Dictionary can serve our purpose, for example, when we need to filter out certain parameter values based on specific conditions, a third query format can be used by calling customized query functions. The basic idea is to use a query function in the code generator and find a list of values for a given parameter. For example, the userDefs hash table stores all user entities together without differentiating parent entity (such as User_props) and child entities (such as Student_props and Teacher_props) that extend their parent entity. When we only want to query for these child entities, path expression will not work. In this case, a customized query function can be written to check which user entities contain an extended_option element, which means it is a child entity. In our demo case, a function called "findSubclassUserEntityNames" will be called by the code generator and it returns a list of subclass question entity names like "Student" and "Teacher".

   The Function query is the most flexible approach—it allows the code generator developers to write utility functions as part of the generator code that implements special processing requirements.

For the Dictionary parameter lookup type, a special path expression is used to explore the multi-level hash tables in the parameter lookup tree. In general, the path expression takes the following form: **HashTable.(Key)+**. For example, the path expression "userDefs.Student.props.propName" will find all property names for the Student user entity. Here is the explanation how it works. The userDefs is a hash table containing all user entity definitions. First, the key named "Student" is used to find the Student hash table containing Student definition details such as name, parent, single properties, collection properties, and mappings (see Figure 7-2). Then, the key named "props" is used to find the hash table for student single property definitions. Each property definition is also a hash table containing property name, property type, etc. (see Figure 7-2). Finally, a key named "propName" is used to retrieve student entity property names.

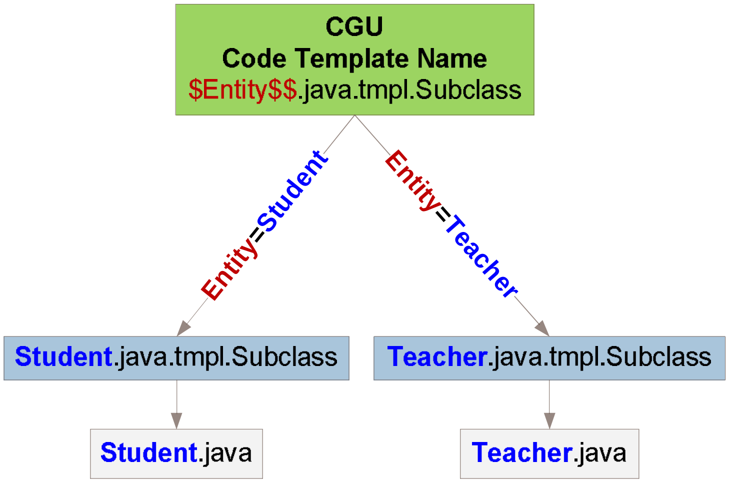
7.1.3 Applications of CGU Parameters

There are two major applications for CGU parameters: provide values for code template parameters and simplify CGU definition. We have discussed the first application in the last section. Each CGU usually refers to one code template, which contains multiple parameters. CGU parameters match the names of template parameters in one-to-one matches. Thus, a CGU collects values and supply these values to the parameters in the templates. This CGU parameter application has been widely used in many code generation patterns that we will discuss in the next chapter.

The other application of CGU parameters is to simplify the definition of CGU elements. Using CGU parameters, multiple code templates can be processed all at once. For example, there are two user entities Student and Teacher and the source code generated to implement the class of these two entities has similar structure: a list of property definitions with getter and setter accessor methods. If one CGU only refers to one template, two CGUs should generate both classes and the content of both CGUs are almost the same except for the entity name part. This problem of redundancy in the CGU definitions is more apparent when there are tens of similar CGUs.

Our solution to this problem is to use parameters in the template name and path expressions. Figure 7-3 gives an example showing how to use parameter in the template name. In this example, the same CGU is used to generate different entity class source code for Student and Teacher user entity. The template name is $Entity$$.java.tmpl.Sublcass, which contains a parameter named Entity (enclosed between $ and $$). To look up values for this Entity parameter, the function lookup method is used and the function findAllUserEntityNames is called (see the code snippet on the right side of Figure 7-3). This function returns two values for the Entity parameter: Student and Teacher. By replacing $Entity$$ with these two values, two templates, Student.java.tmpl.Subclass and Teacher.java.tmpl.Subclass, can be found—each template corresponds to one parameter value.

Notice in the code snippet on the right side of Figure 7-3, there are two parameters used in the template: propType and propName. The propType has a path expression of userDefs.$Entity$$.props.propType and propName has a path expression of userDefs.$Entity$$.props.propName. Since the Entity parameter has two values (Student and Teacher), two path expressions are obtained, one for the Student template and the other for the Teacher template. For example, the two path expressions for the Student template are: userDefs.Student.props.propType and userDefs.Student.props.propName. Using the Dictionary lookup method, we can find values for these two path expressions and provide the values as input to the Student.java.tmpl.Subclass code template. Then, the final code file Student.java, which implements the Student class, is generated. Similarly, the code for Teacher class, Teacher.java, is generated in the same way. From this example, we can see that by using the parameters in the template name and path expression, CGU definitions are greatly simplified.


CGU
Code Template Name
$Entity$$.java.tmpl.Subclass
Entity=Student
Entity=Teacher
Student.java.tmpl.Subclass
Teacher.java.tmpl.Subclass
Student.java
Teacher.java

```
<code_generation_unit unit_id="2"
unit_name="User_Entity_tag_expansion" description="generate user
entity class" template_location="$Entity$$.java.tmpl.Subclass"
template_param_lookup_type="Function"
template_param_lookup_value="findAllUserEntityNames">
    <tag_expansion tag_id="field_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="propType">userDefs.$Entity$$.props.propType</
parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">userDefs.$Entity$$.props.propName</
parameter_content>
    </tag_expansion>
</code_generation_unit>
```

Figure 7-3. Use parameters in code template name and path expression.

CGU parameters can also have “value operators” such as CamelCase, UpperCase, LowerCase, which add extra processing to parameter values before they are used in the code generation. For example, the parameter expression $UserEntity.CamelCase$$ means to convert the parameter values of UserEntity parameter to camel case (capitalizing the first character of each word in a string, e.g. “CamelCaseIsUseful”) before using it.

Transforming design decisions into parameter lookup tree and use parameters in the CGUs is the preparation for the code generation. CGU contains instructions how to use these parameters in the code templates and the code generator finishes the job of CGU batch processing and produce the final source code. We will discuss details about the CGU element structure in the next section. It is a core component of the Rhizome platform and it bridges the gap between the feature model at the semantic level and the code generator at the textual source code level.

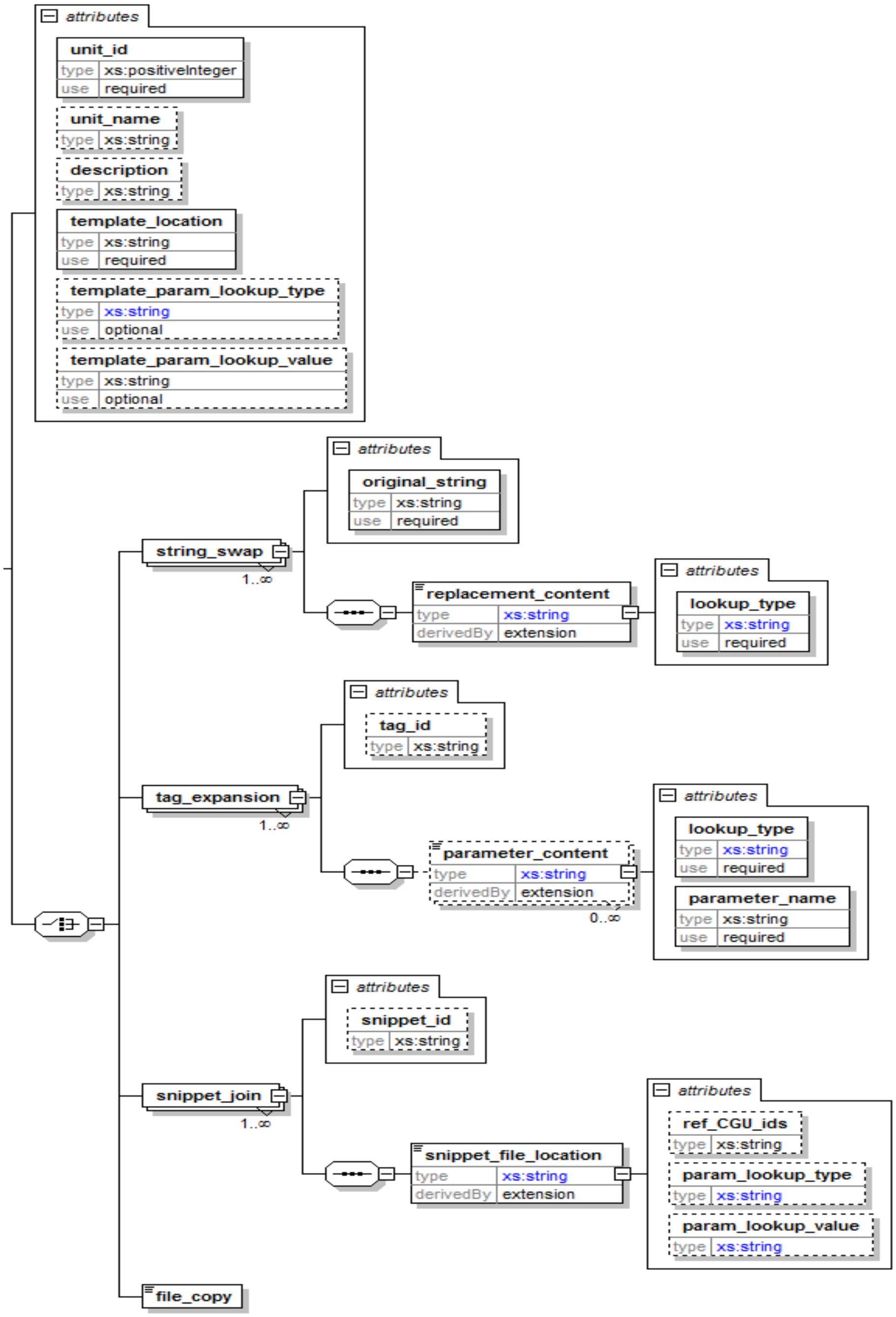


Figure 7-4. Code generation unit structure in FeatureML schema.

7.2 Introduction to Code Generation Unit

The Code Generation Unit (CGU) is an element in the feature model that fills the gap between design decisions in the feature model and the code generator. Figure 7-4 presents the XML schema for code_generation_unit element that represents a CGU. CGU definitions are included as part of the feature model. In the following sections, we first introduce basic attributes for a CGU element, then describe the four types of CGUs with examples illustrating how and when to use each type.

7.2.1 CGU Attributes

Each CGU element has a list of attributes:

- **unit_id** (required): an integer value to uniquely identify a CGU.
- **template_location** (required): a string value to specify the path to a code template. Note that this template location may contain a parameter. In the last section, an example is discussed for a template location: $Entity$$.java.tmpl.Subclass, which contains a parameter Entity enclosed by $ and $$. This template location points to multiple templates, e.g. Student.java.tmpl.Subclass and Teacher.java.tmpl.Subclass if the Entity parameter has two values Student and Teacher.

The complete template path is obtained by appending the value of template_location to the root directory (default root directory is the code generator root directory). For example, given a code generator root directory E:\OnlineExamGenerator, and a template_location /datamodel/Student.java.tmpl.Subclass, a template can be found at E:\OnlineExamGenerator\datamodel\Student.java.tmpl.Subclass. The path separator is automatically adjusted based on the operating system platform.

- **template_param_lookup_type** and **template_param_lookup_value** (optional): this attribute pair must exist when the template_location attribute contains a parameter. The lookup type attribute has three possible values: String, Dictionary and Function. The lookup value attribute provides a corresponding string value, query, or function to provide parameter values. Table 7-1 gives the three combinations for the two attributes.
- **unit_name** (optional): a string value as the literal name for a CGU. This can be a descriptive name so that the CGU can be meaningfully referenced in textual documents.
- **description** (optional): a string value to represent a longer description about a CGU, including textual descriptions about what code this CGU generates, under what conditions should this CGU be selected, etc.

Table 7-1. Explanation of the template parameter lookup type and value.

template_param_lookup_type	template_param_lookup_value	Example
String	A single string value for the parameter	“Student”, “Teacher”
Dictionary	Path expression to fetch parameter values	userDefs.Student.props.propName
Function	Custom function to fetch parameter values	findAllUserEntityNames

7.2.2 Code Generation Activities

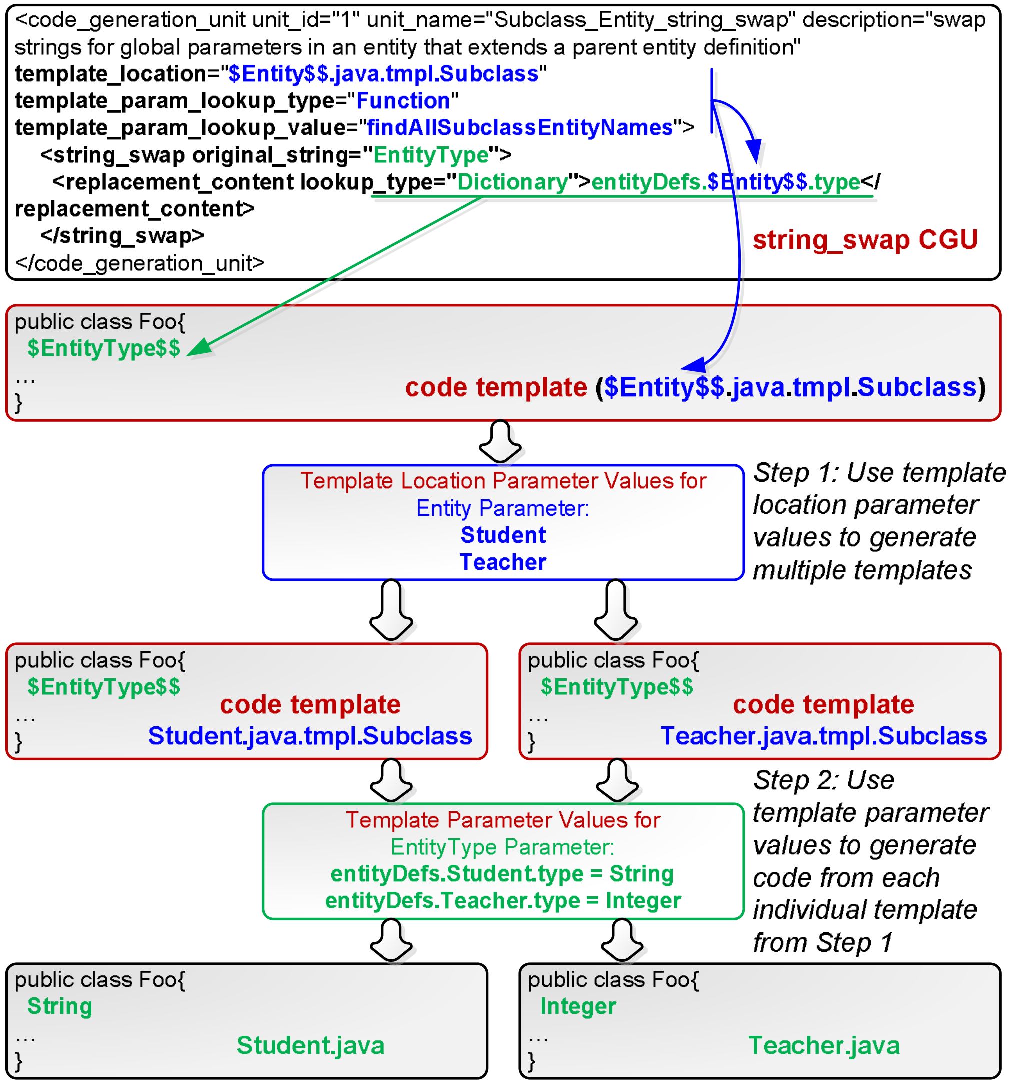
By studying existing source code and architecture design for online exam systems, many code generation patterns have been identified. A code generation pattern captures recurring code generation activities, such as copying the exact file content to multiple new files, or replacing all occurrences of a string with a new string. More complicated generation pattern involves using a template file with embedded tags, where these tags will be replaced by dynamically generated code snippets. These code generation patterns are independent of particular feature modeling language and the code generator implementation. In the Rhizome platform, we choose to model these code generation activity elements using the FeatureML modeling language.

In addition to its attributes, a CGU has four child elements that represent code generation activities (see Figure 7-4): *file_copy*, *string_swap*, *tag_expansion*, and *snippet_join*. Each CGU can only contain one of the aforementioned activity elements. This permits the code generator to process a list of activity elements sequentially without worrying about their ordering problem. This design also allows new activity elements to be added in future to expand the code generation capabilities without affecting existing activity elements. The following list gives a conceptual overview about these four activity elements and technical details are covered in later paragraphs.

- **file_copy**: Generates source code filenames and copies template file content into the generated source code files.
- **string_swap**: Replace an original string in the template with a replacement string.
- **tag_expansion**: Expand embedded tags in the template and replace the tag using dynamically generated code snippets.
- **snippet_join**: Combines snippet file contents into a template file, thereby producing a final source code file. Snippet files may need to be generated before they are combined together.

The file copy activity creates source files that have the same content as the template file, but with different filenames. This means the generator need only to generate appropriate filenames for target source files. There are two possible situations: (1) the *template_location* attribute contains no parameters, (2) *template_location* attribute contains a parameter. In the first case, the generator just creates an empty file, names it using the same template filename without the ".tmpl" extension, and copies the template content into the new file.

The second case is more complicated since it requires parameter value lookup that we mentioned in 7.1.2. A template location containing parameters represents a list of template filenames and this list can be found by replacing the parameter with its values. The generator uses attributes named *template_param_lookup_type* and *template_param_lookup_value* attributes to do the parameter value lookup and find this list of template files, as we just discussed in 7.1.2. In this way, the second case is reduced to the first case. For each resolved template file, the template content is copied over to each new source file.



```
<code_generation_unit unit_id="1" unit_name="Subclass_Entity_string_swap" description="swap
strings for global parameters in an entity that extends a parent entity definition"
template_location="$Entity$$.java.tmpl.Subclass"
template_param_lookup_type="Function"
template_param_lookup_value="findAllSubclassEntityNames">
   <string_swap original_string="EntityType">
     <replacement_content lookup_type="Dictionary">entityDefs.$Entity$$.type</
replacement_content>
   </string_swap>
</code_generation_unit>
```

```
public class Foo{
  $EntityType$$
…
}
```

Template Location Parameter Values for
Entity Parameter:
Student
Teacher

Step 1: Use template location parameter values to generate multiple templates

```
public class Foo{
  $EntityType$$
…
}
```

```
public class Foo{
  $EntityType$$
…
}
```

Template Parameter Values for
EntityType Parameter:
entityDefs.Student.type = String
entityDefs.Teacher.type = Integer

Step 2: Use template parameter values to generate code from each individual template from Step 1

```
public class Foo{
  String
…
}
```

```
public class Foo{
  Integer
…
}
```

Figure 7-5. An example for string_swap that contains a template location parameter and a template parameter.

String replacement is a common activity in template-based code generation. In the template, string replacement place holders, such as $EntityName$$, are inserted and during the code generation process, each occurrence of the place holder will be replaced by a given String value. The *string_swap* CGU activity element is designed for this global string replacement task. Its *original_string* attribute stores the original string. This value matches the place holder name, i.e. the part between $ and $$ (for example, the place holder $EntityName$$ has the name EntityName). To obtain the value to replace this place holder, the CGU parameter value lookup mechanism is used again: textual content of the *replacement_content* child element provides the lookup value and the *lookup_type* attribute on the child element provides the lookup type. For a single CGU, there can be multiple string_swap child elements and the code generator uses regular expression to find and replace strings in the code template.

Figure 7-5 gives an example for string_swap CGU activity element. It contains a template location parameter $Entity$$ and a template parameter $EntityType$$. The $Entity$$ parameter has two values Student and Teacher using the lookup function findAllSubclassEntityNames. The $EntityType$$ parameter has two values String and Integer using the dictionary lookup method. The path expression for the dictionary lookup contains the $Entity$$ parameter, so this CGU will be processed in two steps. In the step 1, the code template $Entity$$.java.tmpl.Subclass will be copied into two templates using the template location parameter values:

Student.java.tmpl.Subclass and Teacher.java.tmpl.Subclass. In the step 2, the template parameter values are used to do the actual string replacement for each template and generate the final code. This two-step parameter value consumption is very typical for all activity elements.

The third CGU activity element is *tag_expansion*. As we know, a code template is a source code file with place holders and tags. A place holder is a string in the format of $name$$, which defines the position where global string replacement should happen. A tag is a piece of XML code that defines how and what the code generator should do to produce a block of source code. The generated code block then replaces the XML tag. Once all the place holders and tags in a template are processed, the result is the final generated source code file. The tag expansion mechanism is not discussed in great details here and later sections describe this topic more completely.

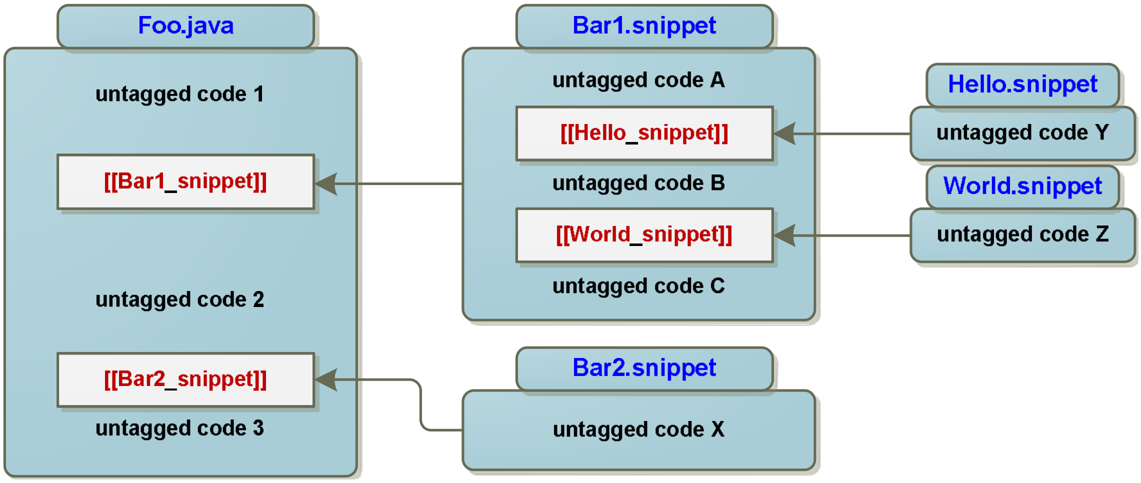


Figure 7-6. An example for the snippet_join CGU.

The last CGU activity element is *snippet_join*. This element performs code generation by assembling small pieces of code snippets. Figure 7-6 shows a simple example about the way snippet_join type CGU works. The red strings in the form of "[[SnippetTagName]]" are snippet tags embedded in the template. These snippet tags are replaced by code snippets. These code snippets have a file extension of ".snippet" and they are either prepared manually or dynamically generated by other CGUs. So, snippet_join activity element can form a hierarchical code generation structure where the generation of one code file depends on the generation of other code files (code snippets). This ripple effect may take place over multiple levels in the code generation hierarchy. For example, consider file Foo.java, which is assembled from

Bar1.snippet and Bar2.snippet (see Figure 7-6). In turn, Bar1.snippet is assembled from Hello.snippet and World.snippet. This requires Hello.snippet and World.snippet to be generated first. Bar2.snippet is prepared manually beforehand.

In Figure 7-4, each snippet_join element has a *snippet_id* attribute that has to match the snippet tag in the template file. A template file is located using the same mechanism as in other CGU types as we have discussed above in *string_swap* CGU activity element. Locations of snippet files are stored as the text content of the *snippet_file_location* element. Its value can be found in a similar way as when we search for the value of the template location attribute. The attributes *param_lookup_type* and *param_lookup_value* are needed when the snippet location contains parameters.

Another attribute, *ref_CGU_ids*, is a list of CGUs that generate snippets used to assemble the current source code file (may be a snippet itself). This attribute represents dependencies between the current snippet_join CGU and other CGUs—all of the referenced CGUs must be processed before snippets can be generated to assemble current source code file. In the code generator, a depth-first ordering is implemented to handle these CGU dependencies.

In this section, the schema for the CGU elements in the FeatureML language is presented. CGU is part of a feature model configuration and it is the connection between the feature model and the code generator. There are four CGU activity elements: file_copy, string_swap, tag_expansion, and snippet_join. These activity elements are the implementation of the four abstract code generation patterns we observe during the domain analysis and source code study process. These patterns can be implemented differently in other systems and still be valid and useful.

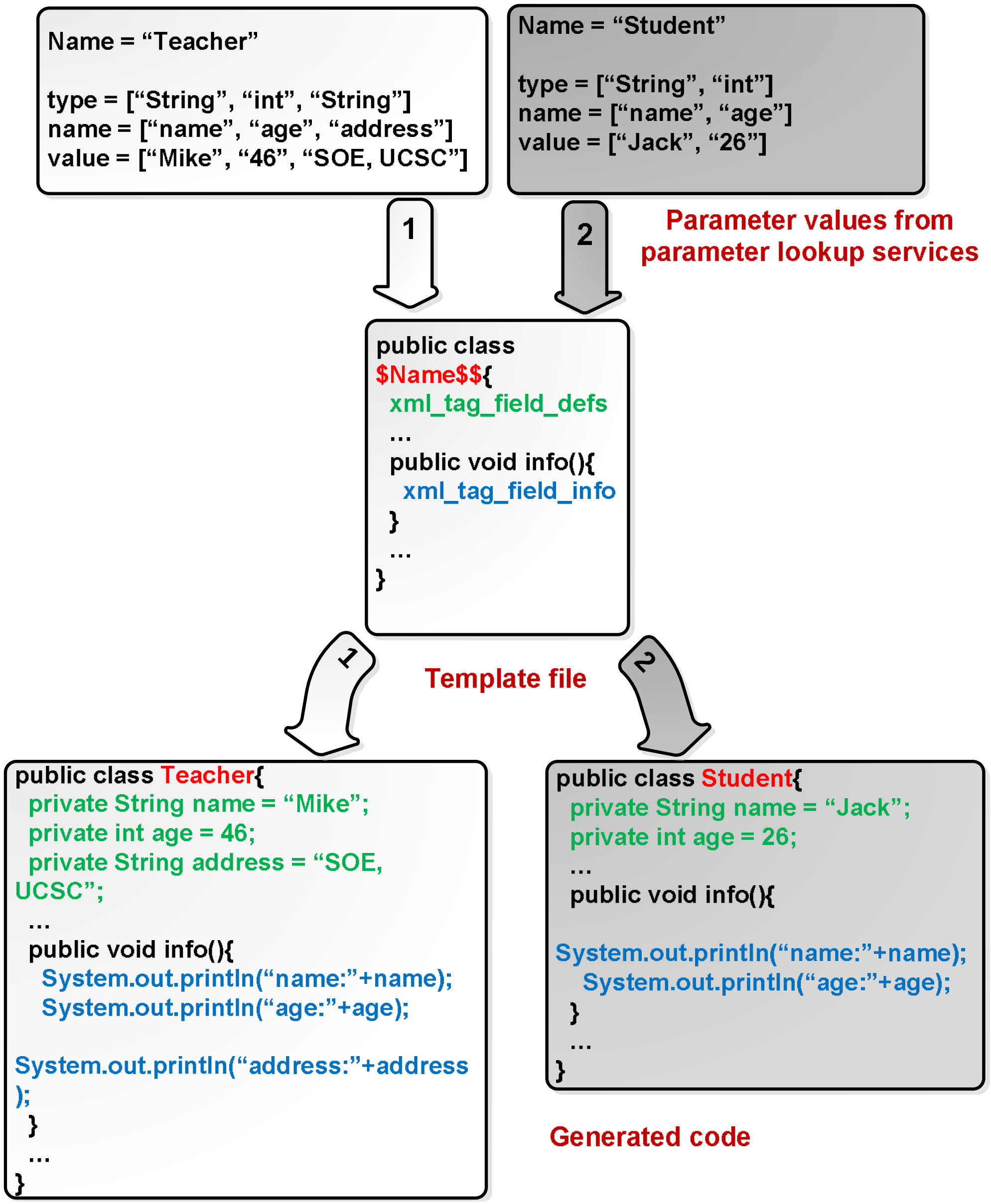
Among these four patterns, tag_expansion is the most complicated one. It covers a general category for generating code using embedded tags in the code templates. There are different types of these embedded tags. Each type represents a different way of generating a block of code using input parameter values provided by the CGUs. The next section describes this tag_expansion CGU activity and template-based code generation in greater details.

7.3 Template-based Code Generation

Template-based code generation has a long history and it is a practical and easy-to-use code generation mechanism. Popular template processing engines [83] include PHP, JSP, ASP.NET, Apache Velocity, Tapestry, etc. The template-based code generation uses template files to capture commonality and variability in a set of source code files. Commonality is retained as original untagged source code and variability is captured using tags and place holders. So, essentially, a template is a mixture of untagged code, tags and place holders.

A place holder is in the format of $place_holder$$ or $place_holder.operation$$ and it is used for global string replacement. In the case of $place_holder$$ format, every occurrence in a template file is replaced with the value of a given parameter called “place_holder” defined in CGU. For the $place_holder.operation$$ case, a string operation, such as CamelCase (Capitalize the first character of each word), FirstLowerCase (change first character of the string to lower case), UpperCase (change all characters to upper case) and LowerCase (change all characters to lower case), to the parameter value before performing global string replacement in the template.

In Rhizome code templates, a tag is an XML document containing instructions for code generation. The code generator processes these instructions and generates a source code block that replaces the XML tag in the template. Tags are described using an XML-based template language described in details in the following sections.



```
Name = “Teacher”

type = [“String”, “int”, “String”]
name = [“name”, “age”, “address”]
value = [“Mike”, “46”, “SOE, UCSC”]
```

```
Name = “Student”

type = [“String”, “int”]
name = [“name”, “age”]
value = [“Jack”, “26”]
```

```
public class
$Name$${
  xml_tag_field_defs
  …
  public void info(){
    xml_tag_field_info
  }
  …
}
```

```
public class Teacher{
  private String name = “Mike”;
  private int age = 46;
  private String address = “SOE,
UCSC”;
  …
  public void info(){
    System.out.println(“name:”+name);
    System.out.println(“age:”+age);

System.out.println(“address:”+address
);
  }
  …
}
```

```
public class Student{
  private String name = “Jack”;
  private int age = 26;
  …
  public void info(){

System.out.println(“name:”+name);
    System.out.println(“age:”+age);
  }
  …
}
```

Figure 7-7. Template-based Code Generation.

Figure 7-7 is a simplified illustration of how source files are generated using a template. Inputs to the template processing engine in the code generator are parameter values obtained from parameter value lookup using one of String, Dictionary and Function types. These parameter values either replace place holders or provide inputs for XML tag processing. Different feature models contain different sets of parameter values and eventually create different source code files using the same template file.

7.3.1 Template Language Overview

The template language describes XML tags that are embedded in the template files. After being processed by the code generator, these tags expand to blocks of source code which replace the tags to produce final source code files. The syntax of this template language is described using a standard XML schema. We introduce the language syntax with examples from our online exam system generation case study.

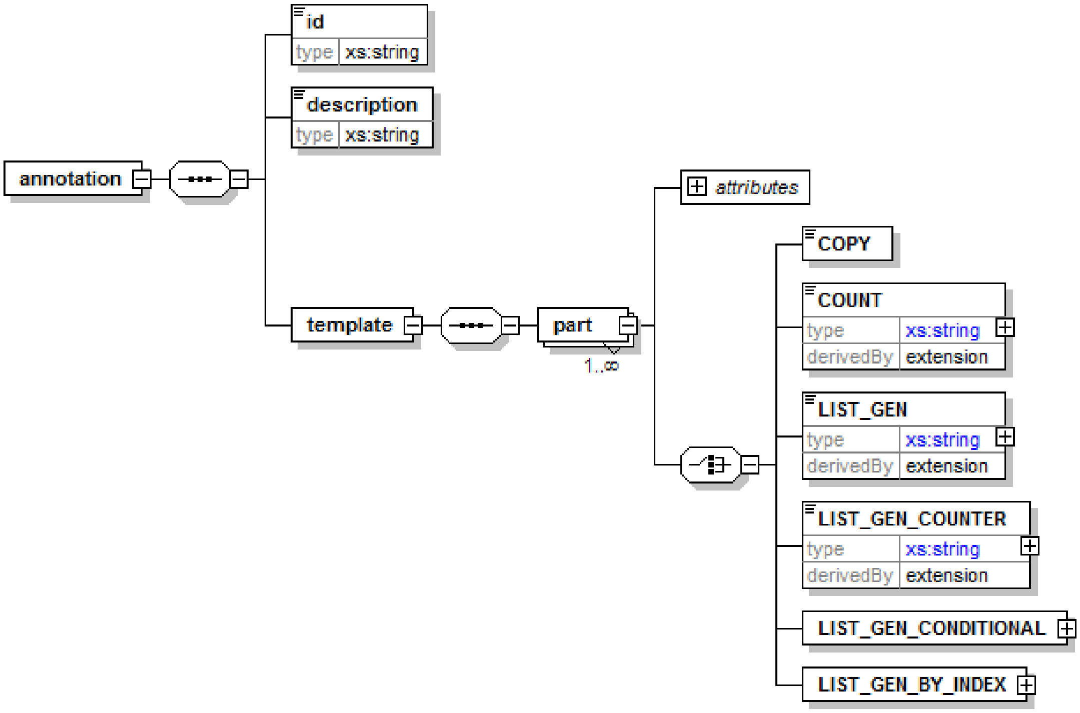


Figure 7-8. Template language XML schema overview.

Figure 7-8 presents the complete XML schema definition for the template language. Each tag has a root element called annotation, which has a String *id* attribute that uniquely identifies the tag and a String *description* attribute containing a human-readable description. The *id* attribute matches the *tag_id* attribute defined on *tag_expansion* activity element in CGU definitions.

Under the root element there is a child element called *template*. Each template element has at least one child element called *part*. A part element contains a code generation directive that tells the code generator how to expand this tag to produce source code.

Because there can be multiple parts in a template element and we want the final generated code to be human readable, a string is specified as the separator after a code block for a part element is generated. For example, the "\n" string will insert a return plus new line after the generated code block. The attribute *separator_to_next_part* stores this string separator. If this attribute is skipped in a part element definition, a default space character will be used as the separator.

There are four types of code generation directives that are represented in a part element:

- **COPY**: copy the text content of this element to the generated code for this part element.
- **LIST_GEN**: generate a list of similar code items using an item template consisting of parameters.
- **LIST_GEN_CONDITIONAL**: generate a list of code items using different item templates. The item template to use for a particular code item is chosen

based on whether a condition attached on the item template is satisfied. One scenario for this type is to control if an attribute should appear in the generated code based on its attribute name, e.g. Class A contains all of Entity1's attributes, but Class B only contains a subset of Entity1's attributes.

- **LIST_GEN_BY_INDEX**: generate a list of code items using different item templates based on the item index value. Item template to use for a particular code item is chosen based on the index of the code item. One common use case for this type is to generate "IF-ELSE IF-ELSE" code structure, i.e. the IF clause has an index of 0 and uses an item template, the ELSE clause has an index of -1 and uses a different item template, the remaining clauses has an index of 1 to -2 and uses another different item template. Negative index values are used here because we do not know beforehand when we prepare a template how many ELSE IF clauses will be generated. The exact count of ELSE IF clauses is determined by design decisions in the feature model.

The detailed structure for these child elements is presented in the following sections with code examples that illustrate how the tag expansion mechanism works.

7.3.2 COPY and LIST_GEN Directives

Figure 7-9 shows the structure for the COPY and LIST_GEN generator directives. Whenever the code generator sees a COPY element, it simply appends the content of COPY element text content to the newly generated text stream. This element is very handy in the cases to copy a fixed code block, such as opening and closing brackets, between the code generated by two ***part*** elements that dynamically generate code blocks inside the same XML tag. For example, consider two parts elements where part1 generates code_block1 and part2 generates code_block2. However, there needs to be a closing curly bracket between code_block1 and code_block2. The bracket remains unchanged and code_block1 and code_block2 can change based on different input parameter values. In this case, a COPY directive is added between part1 and part2. The text content of this COPY element is a closing curly bracket.

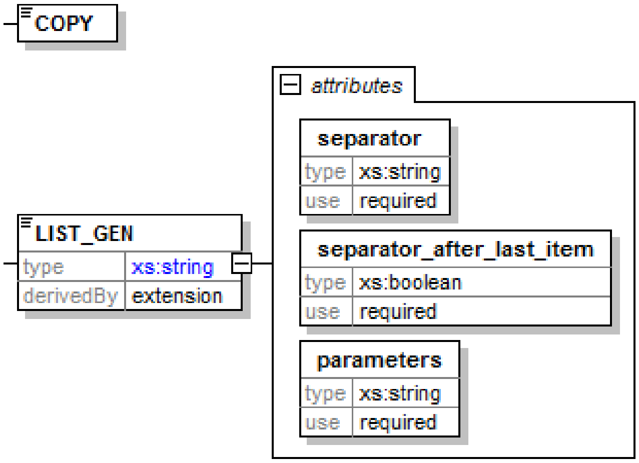
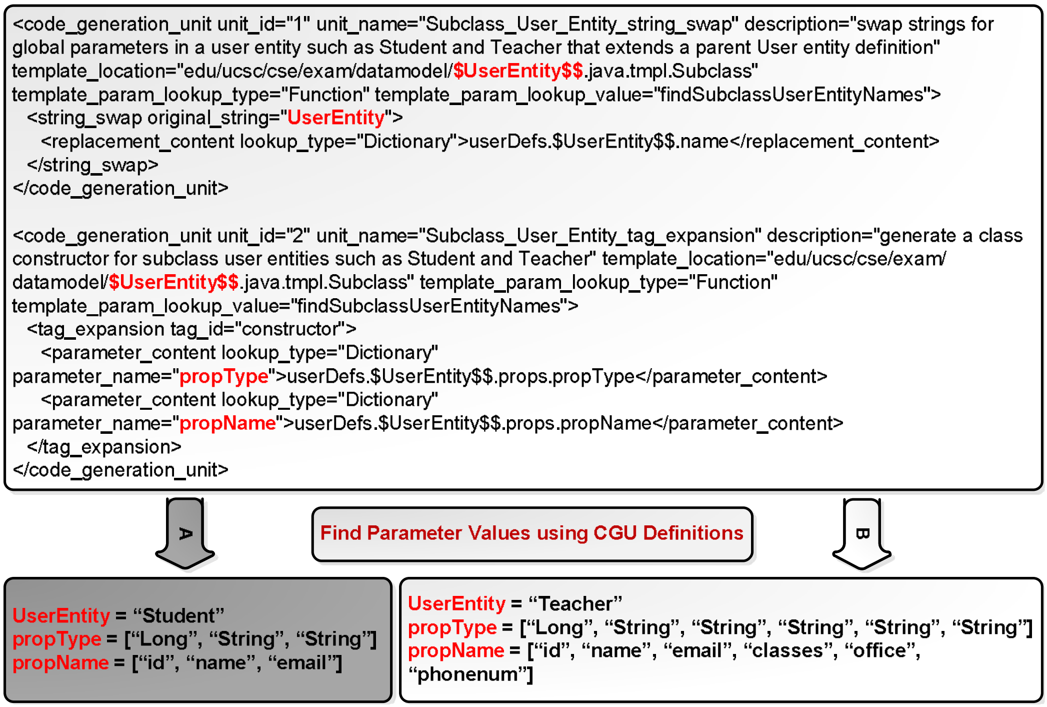


Figure 7-9. XML schema structures for the COPY and LIST_GEN elements.

An alternative approach that has the same effect as the COPY directive is to define two separate XML tags, one for each part element, and treat the curly brackets as unchanged code in between these two XML tags. These two approaches to represent unchanged code blocks are quite similar. The trade-offs to consider when deciding which approach to use are as follows. The COPY element saves the overhead of composing and processing a new XML tag just for a COPY element in the template and the overhead of creating and process an extra tag_expansion element in the CGU. The downside of using COPY is that it will copy the content twice: the first time occurs when the template content is copied into a temporary file during initialization; the second time occurs when the content of the COPY element is copied again during the tag processing stage. This alternative way only copies the content once in initialization stage because the untagged code does not need tag processing. In the template files for the online exam system domain, both approaches can be found. The alternative approach is employed to handle bigger unchanged code blocks, i.e. more than a few lines, and the COPY directive is used for smaller unchanged code blocks such bracket separators, new line characters, quotation marks, etc.

The LIST_GEN directive generates a list of similar code items based one item template. It is one of the most common elements used in the online exam system templates. This directive can be used to generate a list of Java libraries to import, a list of class field definitions, input parameters for methods, etc. As shown in Figure 7-9, this directive has three attributes: ***separator***, ***separator_after_last_item***, and ***parameters***. The separator attribute is a string, such as the new line or bracket character, which will be inserted between generated items. The most frequently used separator is “\n” representing a new line. The separator_after_last_item attribute is a Boolean value. Only if it is true, will a separator be appended after last generated item. For example, for a method “foo(param1, param2, param3)”, the generator creates the input parameters “param1, param2, param3” part and use the “,” character as the separator. But the last generated item “param3” should not have the “,” character after it. This situation can be solved by setting the separator attribute to “, ” and the separator_after_last_item attribute to “false”.

The *parameters* attribute of the LIST_GEN directive defines a list of parameter names. Each parameter name maps to a parameter definition in the CGU parameter_content element found in feature document. The code generator first retrieves the parameter definition from a CGU, and then performs a parameter value lookup to gather a list of parameter values. The generator then begins executing a loop. In each of the loop iterations, the generator generates one code item by pulling a parameter value and replacing the parameter in the item template using this value. When the loop completes, a list of code items are generated.

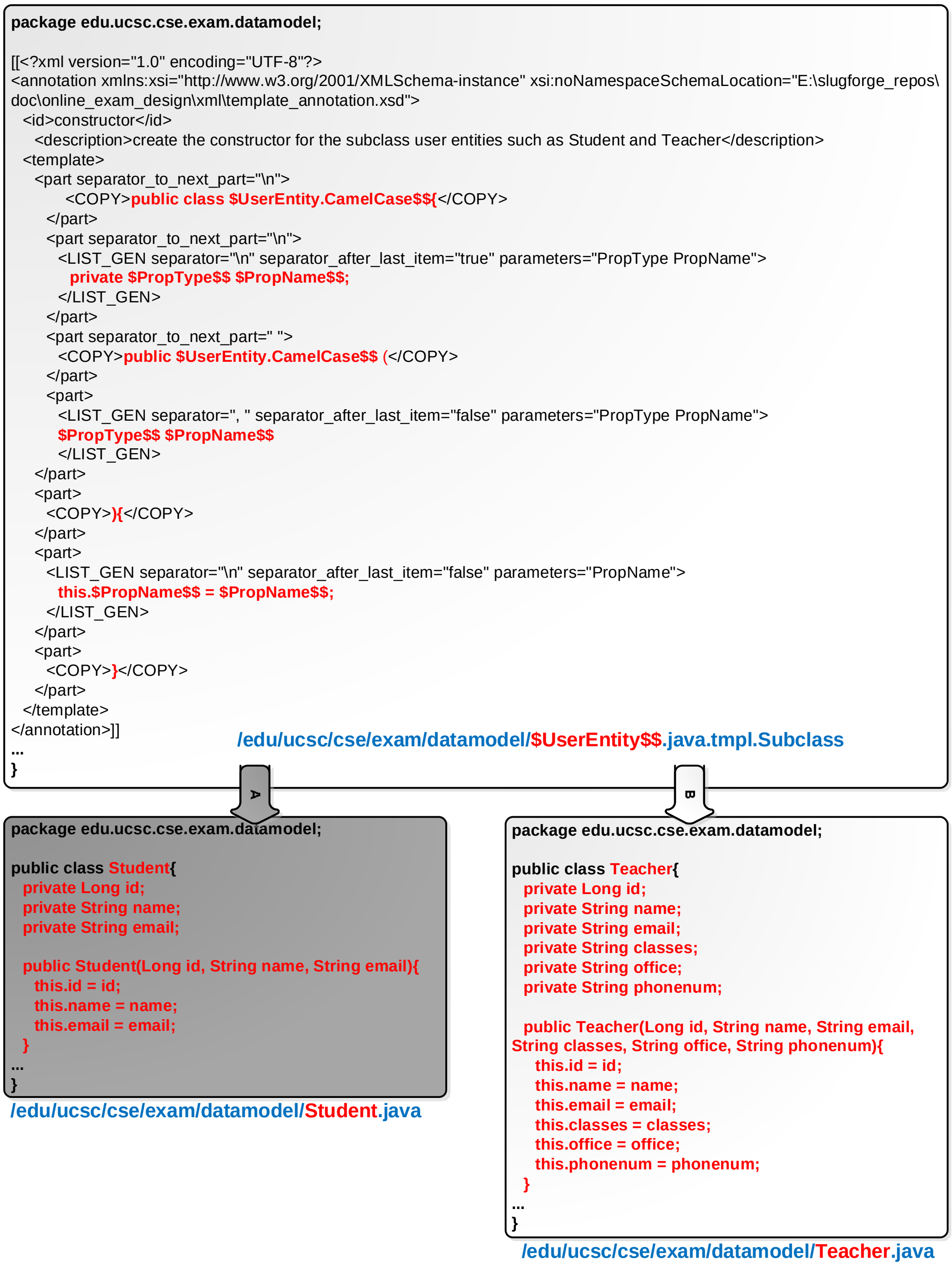


```
<code_generation_unit unit_id="1" unit_name="Subclass_User_Entity_string_swap" description="swap strings for
global parameters in a user entity such as Student and Teacher that extends a parent User entity definition"
template_location="edu/ucsc/cse/exam/datamodel/$UserEntity$$.java.tmpl.Subclass"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

UserEntity = “Student”
propType = [“Long”, “String”, “String”]
propName = [“id”, “name”, “email”]

UserEntity = “Teacher”
propType = [“Long”, “String”, “String”, “String”, “String”, “String”]
propName = [“id”, “name”, “email”, “classes”, “office”, “phonenum”]

Figure 7-10. Example for global string replacement and COPY, LIST_GEN tag expansion (Step 1).



```
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$${</COPY>
      </part>
      <part separator_to_next_part="\n">
        <LIST_GEN separator="\n" separator_after_last_item="true" parameters="PropType PropName">
          private $PropType$$ $PropName$$;
        </LIST_GEN>
      </part>
      <part separator_to_next_part=" ">
        <COPY>public $UserEntity.CamelCase$$ (</COPY>
      </part>
      <part>
        <LIST_GEN separator=", " separator_after_last_item="false" parameters="PropType PropName">
        $PropType$$ $PropName$$
        </LIST_GEN>
    </part>
    <part>
      <COPY>){</COPY>
    </part>
    <part>
      <LIST_GEN separator="\n" separator_after_last_item="false" parameters="PropName">
        this.$PropName$$ = $PropName$$;
      </LIST_GEN>
    </part>
    <part>
      <COPY>}</COPY>
    </part>
  </template>
</annotation>]]
...
}
```

/edu/ucsc/cse/exam/datamodel/$UserEntity$$.java.tmpl.Subclass
A
B

```
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;
  }
...
}
```

```
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;
  }
...
}
```

Figure 7-11. Example for global string replacement and COPY, LIST_GEN tag expansion (Step 2).

A simple example demonstrates global string replacement for place holders and tag expansion for COPY and LIST_GEN directives. Figure 7-10 and Figure 7-11 show the code generation process for Student.java and Teacher.java. Figure 7-10 shows the first step where the code generator processes CGUs and retrieves parameter values using one of the three aforementioned lookup services (String, Dictionary, or Function). It uses a function (param_lookup_type=”Function”) called “findSubclassUserEntityNames” to find values for parameter “UserEntity” used in the template filename. In this example, the FeatureML document has two subclass user types, Student and Teacher. As a result, this function will find two strings “Student” and “Teacher”, and thus two source code files will be generated: one for Student and the other for Teacher. Arrows labeled A and B in the Figure 7-10 separate the two code generation iterations. Two other parameters “propType” and “propName” also have their values retrieved using the Dictionary service.

Figure 7-11 shows the step 2, in which place holders and tags are processed to generate code. Place holders are in the form of $name$$ and their names correspond to the replacement_content element’s attributes called “original_string” in a string swap CGU definition. In our example, there is only one place holder defined in the string swap CGU: $UserEntity$$ or $UserEntity.CamelCase$$, where CamelCase is one of the operations (CamelCase, FirstLowerCase, LowerCase and UpperCase). The code generator just replaces all occurrences of $UserEntity$$ or

$UserEntity.CamelCase$$ with the values or CamelCased values found in step 1, i.e. use “Student” to replace $UserEntity$$ in Student.java and “Teacher” to replace $UserEntity.CamelCase$$ in Teacher.java (the CamelCased string “Teacher” is the same as “Teacher” string).

After place holders are processed, the code generator starts working on XML tags enclosed using a pair of double square brackets ([[and]]). If it sees a COPY element, it simply appends the element’s content to the generated source code. Note that in the example, two COPY elements contain the place holder $UserEntity.CamelCase$$. By the time the generator performs COPY element processing, these place holders will have their final value since place holder replacement does not distinguish if the place holder is in or outside a tag.

Processing next continues to the LIST_GEN directive. In the example, the LIST_GEN element is used to generate class data field definitions. Data definitions are found in two parameters *PropertyType* and *PropertyName*. In the case of the Student user type in Figure 7-11, PropertyType has three values: Long, String and String, and correspondingly, PropertyName has three values: id, name and email. LIST_GEN processing proceeds through three iterations to generate three code items. In each iteration, one value is chosen from each of the PropertyType and PropertyName value lists and use this value pair to replace parameters in the code item template “private $PropertyType$$ $PropertyName$$;”. In this way, three field definitions will be generated: “private Long id; \n private String name; \n private String email;”. The “\n” is the specified list item separator and it will appear as a new line in the generated code.

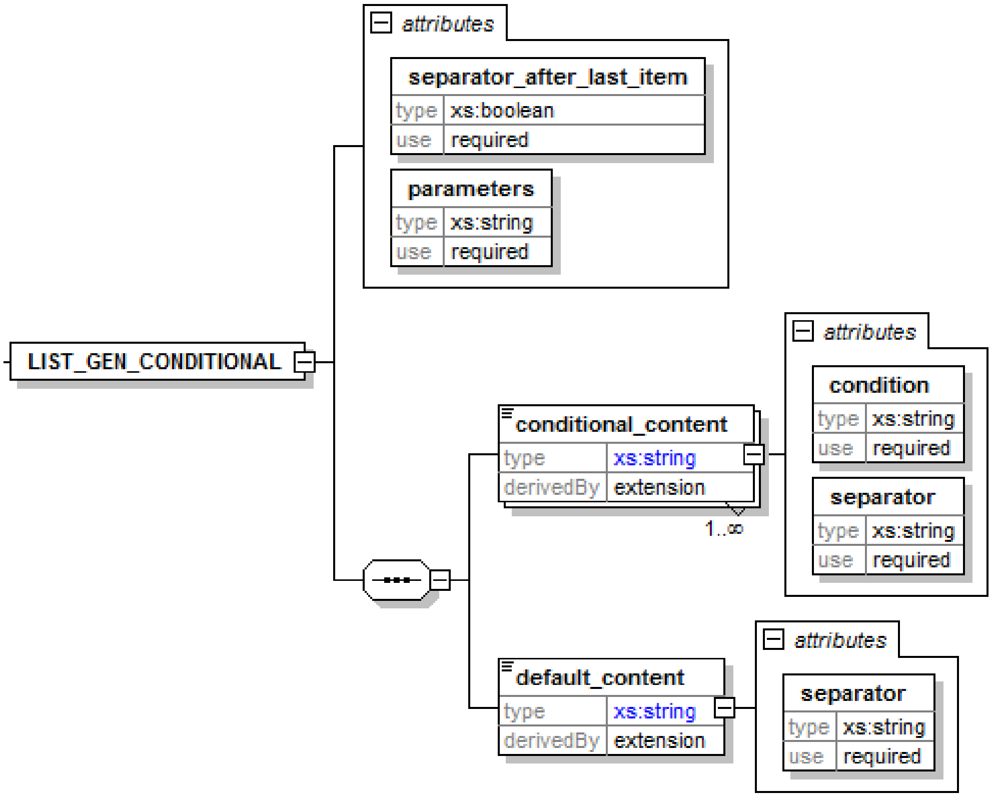


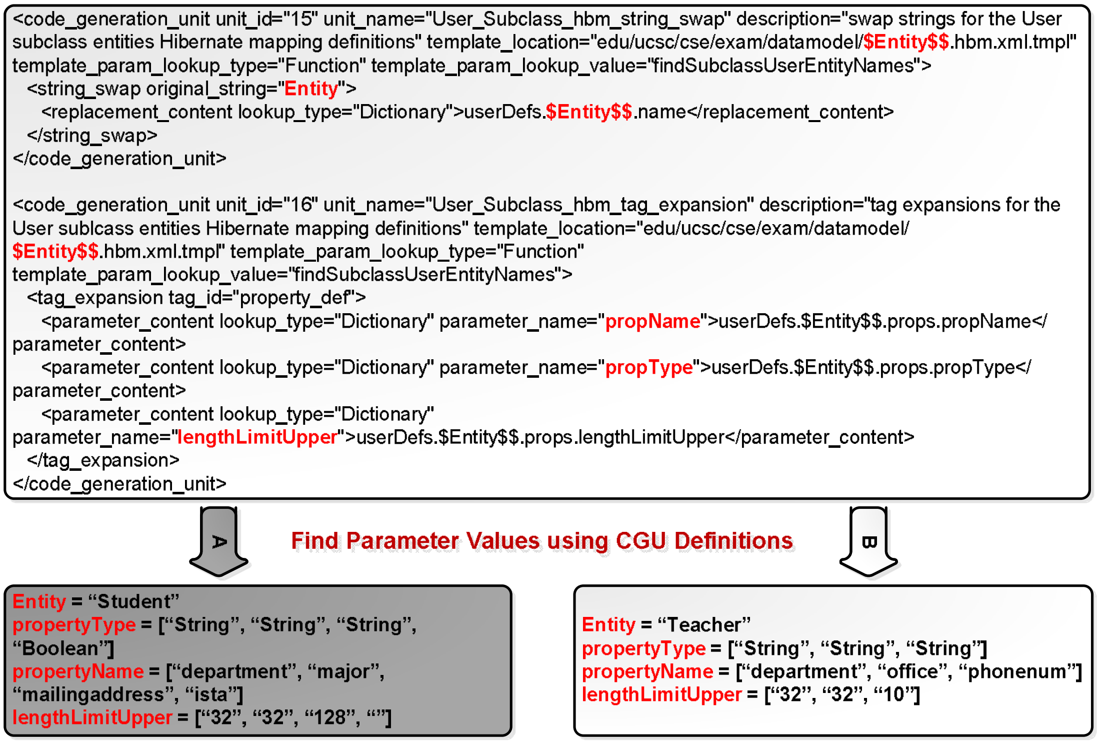
Figure 7-12. XML schema structure for the LIST_GEN_CONDITIONAL element.

7.3.3 LIST_GEN_CONDITIONAL Directive

The LIST_GEN directive is useful in cases when the code generator needs to produce a list of code items using a single list item template. However, there are some cases where a list of code items needs to be generated from a set of different list item templates based on preset conditions. For example, for a string property on the Student entity, we need to define the maximum length of the string; but not for an int property. This kind of requirement drove the design of the LIST_GEN_CONDITIONAL element.

This element can contain different item template branches, where each branch has a condition defined on it. During a loop iteration to generate one code item, the code generator first picks parameter values from parameter value lists in the same fashion as for the LIST_GEN element. For example, if there are two parameters and each parameter has three values, then the code generator will pick the first parameter value from both value lists for the first iteration, and so on. Second, the code generator evaluates each branch one by one using the correct parameter values. The first branch whose condition evaluates to be true is chosen. The code generator uses this branch's template to create the code item for this iteration. After processing all the parameter values, the code generator stops processing the LIST_GEN_CONDITIONAL directive, yielding the final created code.

Figure 7-12 illustrates the XML schema structure for the LIST_GEN_CONDITIONAL directive. It has two attributes: *separator_after_last_item* and *parameters*. The former defines the separator string to use after the last generated code item and the latter defines parameters to use for all conditional branches. There are two types of child elements: *conditional_content* and *default_content*. A conditional_content element defines a Boolean *condition* attribute for the code generator to evaluate. Currently, only conditions in the format of "parameter==value" are supported since this is the only condition type we found during code analysis. Implementation of other compare operators such as >, <, !=, or multiple conditions combined with AND, OR, XOR won't be difficult in the code generator—we just need to implement a conditional handling function for these more complicated condition structures. Another attribute on the conditional_content element is *separator*, which defines the separator to be appended to this generated code item. The textual content for the *conditional_content* element is the code item template. A *default_content* element is the safety net when all other conditional_content elements have not been chosen. It also has a *separator* attribute similar to the conditional_content element.

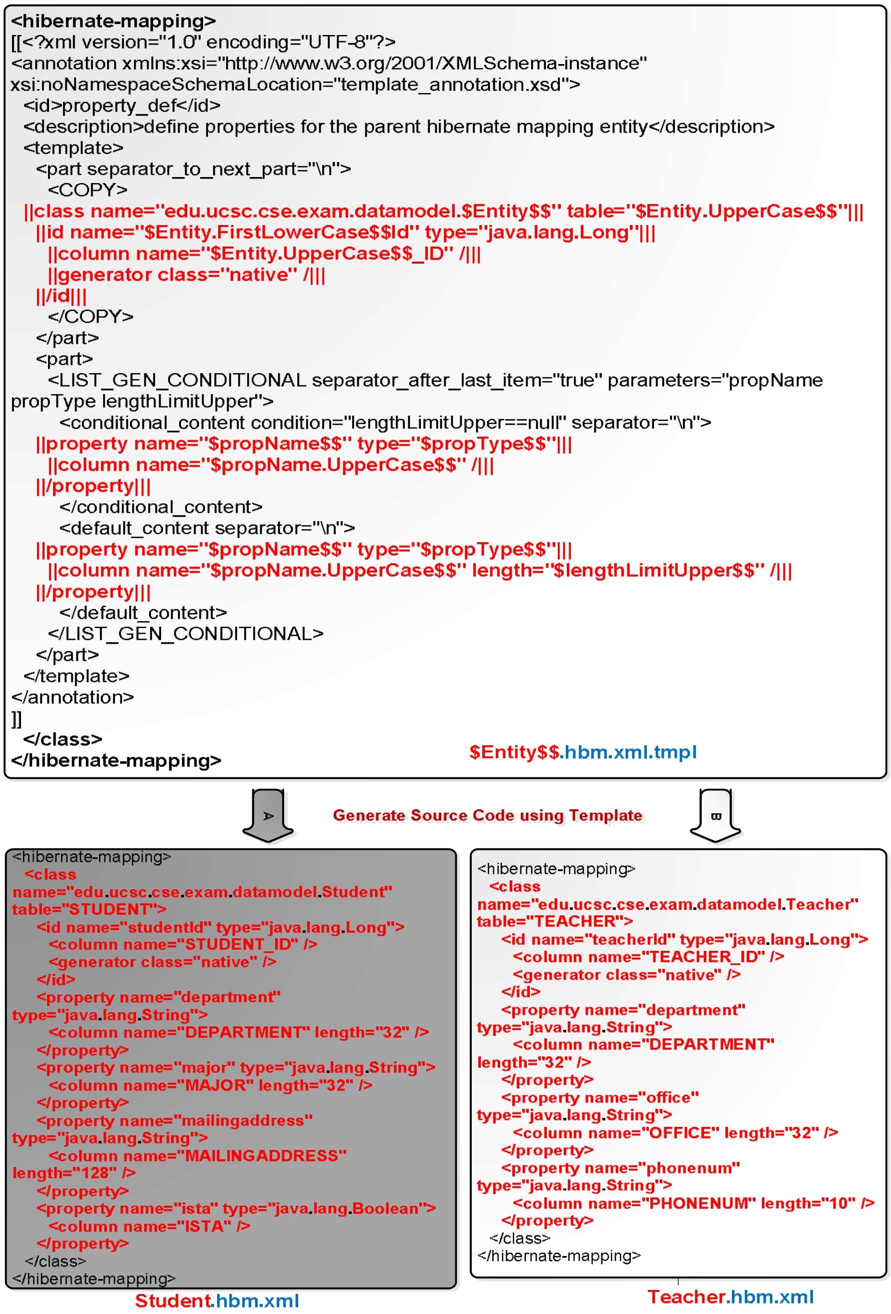


```
<code_generation_unit unit_id="15" unit_name="User_Subclass_hbm_string_swap" description="swap strings for the User
subclass entities Hibernate mapping definitions" template_location="edu/ucsc/cse/exam/datamodel/$Entity$$.hbm.xml.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
  <string_swap original_string="Entity">
    <replacement_content lookup_type="Dictionary">userDefs.$Entity$$.name</replacement_content>
  </string_swap>
</code_generation_unit>

<code_generation_unit unit_id="16" unit_name="User_Subclass_hbm_tag_expansion" description="tag expansions for the
User sublcass entities Hibernate mapping definitions" template_location="edu/ucsc/cse/exam/datamodel/
$Entity$$.hbm.xml.tmpl" template_param_lookup_type="Function"
template_param_lookup_value="findSubclassUserEntityNames">
  <tag_expansion tag_id="property_def">
    <parameter_content lookup_type="Dictionary" parameter_name="propName">userDefs.$Entity$$.props.propName</
parameter_content>
    <parameter_content lookup_type="Dictionary" parameter_name="propType">userDefs.$Entity$$.props.propType</
parameter_content>
    <parameter_content lookup_type="Dictionary"
parameter_name="lengthLimitUpper">userDefs.$Entity$$.props.lengthLimitUpper</parameter_content>
  </tag_expansion>
</code_generation_unit>
```

Entity = “Student”
propertyType = [“String”, “String”, “String”, “Boolean”]
propertyName = [“department”, “major”, “mailingaddress”, “ista”]
lengthLimitUpper = [“32”, “32”, “128”, “”]

Entity = “Teacher”
propertyType = [“String”, “String”, “String”]
propertyName = [“department”, “office”, “phonenum”]
lengthLimitUpper = [“32”, “32”, “10”]

Figure 7-13. Example for LIST_GEN_CONDITIONAL tag expansion (Step 1).



```
<hibernate-mapping>
[[<?xml version="1.0" encoding="UTF-8"?>
<annotation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="template_annotation.xsd">
  <id>property_def</id>
  <description>define properties for the parent hibernate mapping entity</description>
  <template>
    <part separator_to_next_part="\n">
      <COPY>
  ||class name="edu.ucsc.cse.exam.datamodel.$Entity$$" table="$Entity.UpperCase$$"|||
    ||id name="$Entity.FirstLowerCase$$Id" type="java.lang.Long"|||
      ||column name="$Entity.UpperCase$$_ID" /|||
      ||generator class="native" /|||
    ||/id|||
      </COPY>
    </part>
    <part>
      <LIST_GEN_CONDITIONAL separator_after_last_item="true" parameters="propName
propType lengthLimitUpper">
        <conditional_content condition="lengthLimitUpper==null" separator="\n">
    ||property name="$propName$$" type="$propType$$"|||
      ||column name="$propName.UpperCase$$" /|||
    ||/property|||
        </conditional_content>
        <default_content separator="\n">
    ||property name="$propName$$" type="$propType$$"|||
      ||column name="$propName.UpperCase$$" length="$lengthLimitUpper$$" /|||
    ||/property|||
        </default_content>
      </LIST_GEN_CONDITIONAL>
    </part>
  </template>
</annotation>
]]
  </class>
</hibernate-mapping>
```

$Entity$$.hbm.xml.tmpl

```
<hibernate-mapping>
  <class
name="edu.ucsc.cse.exam.datamodel.Student"
table="STUDENT">
    <id name="studentId" type="java.lang.Long">
      <column name="STUDENT_ID" />
      <generator class="native" />
    </id>
    <property name="department"
type="java.lang.String">
      <column name="DEPARTMENT" length="32" />
    </property>
    <property name="major" type="java.lang.String">
      <column name="MAJOR" length="32" />
    </property>
    <property name="mailingaddress"
type="java.lang.String">
      <column name="MAILINGADDRESS"
length="128" />
    </property>
    <property name="ista" type="java.lang.Boolean">
      <column name="ISTA" />
    </property>
  </class>
</hibernate-mapping>
```

Student.hbm.xml

```
<hibernate-mapping>
  <class
name="edu.ucsc.cse.exam.datamodel.Teacher"
table="TEACHER">
    <id name="teacherId" type="java.lang.Long">
      <column name="TEACHER_ID" />
      <generator class="native" />
    </id>
    <property name="department"
type="java.lang.String">
      <column name="DEPARTMENT"
length="32" />
    </property>
    <property name="office"
type="java.lang.String">
      <column name="OFFICE" length="32" />
    </property>
    <property name="phonenum"
type="java.lang.String">
      <column name="PHONENUM" length="10" />
    </property>
  </class>
</hibernate-mapping>
```

Teacher.hbm.xml

Figure 7-14. Example for LIST_GEN_CONDITIONAL tag expansion (Step 2).

Now, let’s see an example showing LIST_GEN_CONDITIONAL element in action. This example shows how the generator creates Hibernate mapping files for Student and Teacher user entity. Figure 7-13 illustrates two CGU definitions: one for string swap and the other for tag expansion. The template filename contains a parameter “Entity” and the function “findSubclassUserEntityNames” will find two values for this parameter: “Student” and “Teacher”. So, two Hibernate mapping files will be generated for these two user entities. String swap CGU will replace every occurrence of “$Entity$$” place holder with either “Student” or “Teacher” string.

There is only one tag expansion element in the second CGU. It defines three parameters: *propertyType*, *propertyValue* and *lengthLimitUpper*. Code generator uses dictionary lookup service to find values for these parameters as shown in Figure 7-13. For Student user entity, it has four properties: *department* and *major* are strings with maximum length of 32, *mailingaddress* is a string with maximum length of 128, and *ista* is a Boolean which does not need a maximum length. Different property types like String and Boolean cause the need for different item templates for code generation.

Figure 7-14 shows the template and the generated Hibernate mapping files for Student and Teacher. The COPY element works in a same way as we have talked about in the last section. The LIST_GEN_CONDITIONAL element defines a branch with the condition “lengthLimitUpper==null” to test if the lengthLimitUpper property value is equal to an empty string (it is more readable to compare with “null” than an empty string “”). If the condition is evaluated to be true, then the property is a non-String property type, the template that does not have a length attribute on the column element is selected. The default branch is chosen only when the property is a String type, the template associated with the default branch has a *length* attribute on the *column* element. We can see the difference among the column elements generated using these two different item templates in the Student.hbm.xml mapping file. Since all Teacher’s properties are of the String type, the default branch is always chosen and all the generated column elements have a length attribute.

One last thing is about these “|” characters. Because an XML element does not allow “<” and “>” characters to appear in its textual content, “||” characters are used to represent “<” and “|||” to represent “>” in the XML element content in tags. XML has the CDATA language element to prevent the text content to be parsed by XML parser. But we feel that using “<![CDATA[“ and “]]>” as the separator pair is more complicated for the parser than simply using the separator pair of “||” and “|||”. At the generation time, code generator will automatically detect these “|” characters and restore them into “<” and “>” characters so that the output remains to be a valid XML document.

7.3.4 LIST_GEN_BY_INDEX Element

LIST_GEN_CONDITIONAL element solves the problem of conditionally selecting item templates based on the parameter values. Another frequently used code pattern is identified: IF-ELSE IF-ELSE statements. In this pattern, item templates are selected based on the index of a parameter value in the parameter value list. For example, suppose there is a parameter and it has a list of three values, for the first parameter value (index = 0..0), the “IF” item template should be picked; for the last parameter value (index = -1..-1), the “ELSE” item template should be used; for the rest of parameter values (index = 1..-2), the “ELSE IF” item template is used by default. Note that “m..n” is a range format to express the index range: m is the lower index bound, n is the upper index bound and both m and n are included for the range.

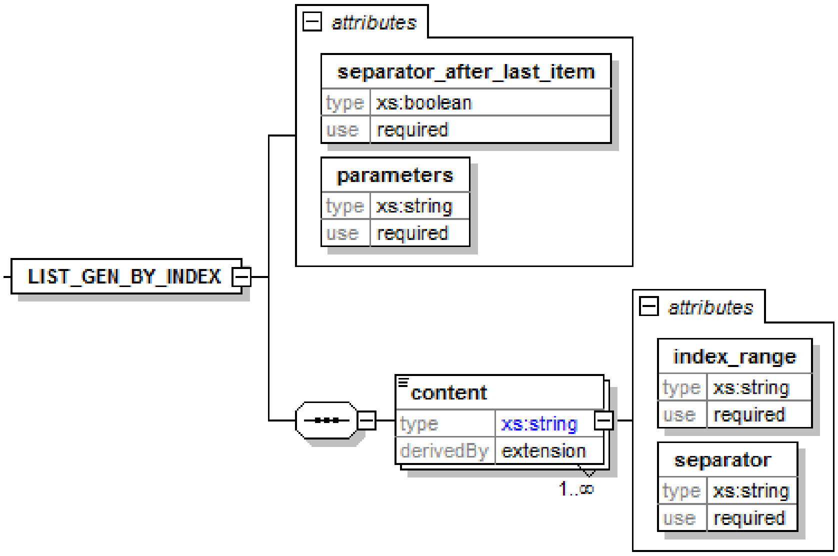
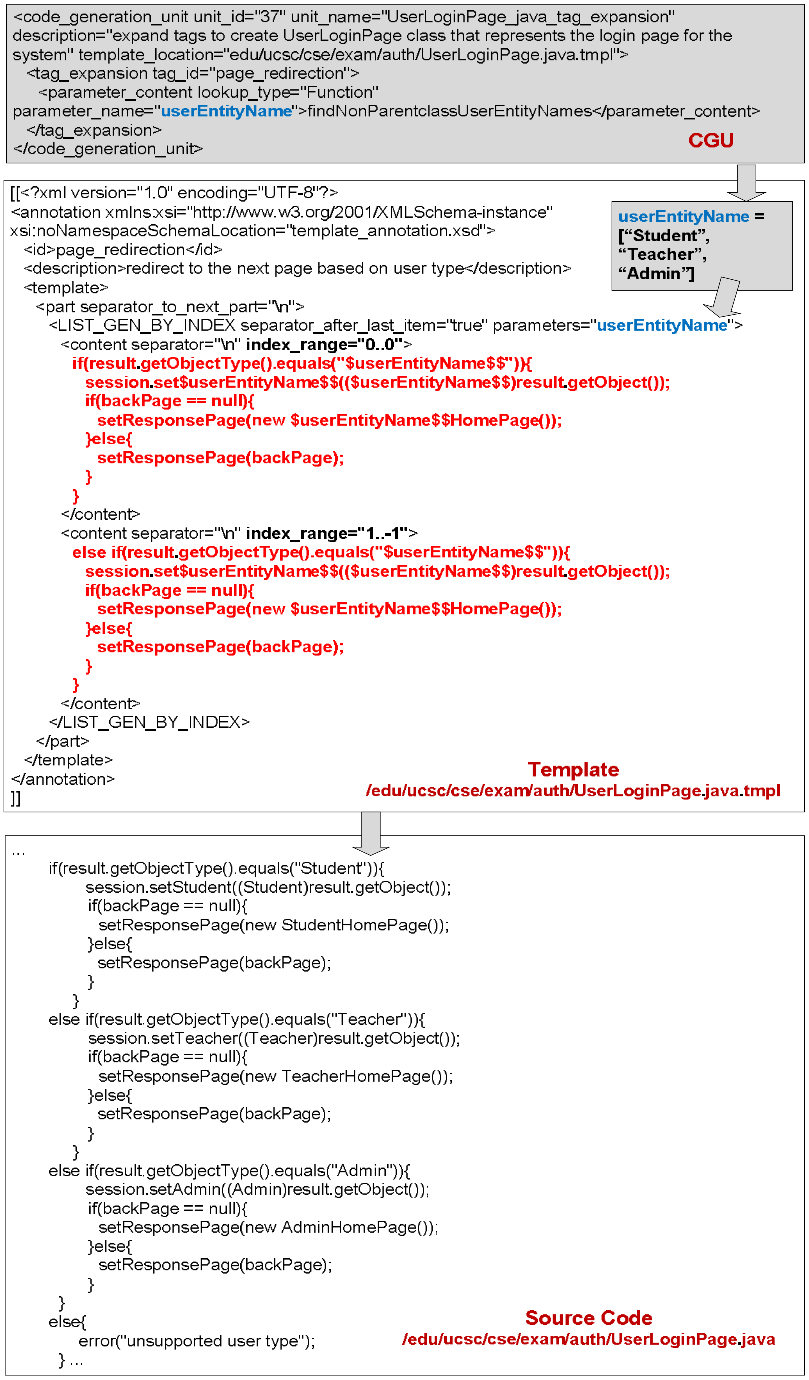


Figure 7-15. XML schema structure for LIST_GEN_BY_INDEX element.

The directive to satisfy this code generation pattern is LIST_GEN_BY_INDEX. Figure 7-15 illustrates the XML schema for this element, which has a quite similar structure to LIST_GEN_CONDITIONAL. The only difference is that we pick an item template, which is stored as *content* element's text, based on index range stored in *index_range* attribute. The *separator_after_last_item* attribute defines a string separator after the last generated code item. The *parameter* attribute is a list of parameter names used in the code item templates. The *separator* attribute on the content element defines a string separator to be inserted after each code item.

Figure 7-16 is an example to generate user login page based on the list item generation using LIST_GEN_BY_INDEX directive. Due to the space limit, only the tag containing LIST_GEN_BY_INDEX element is shown in the figure. The generated user login page is a page for existing users to log in or for new users to register for a new account. Based on its user type, a user should be directed to an appropriate Home page or Registration page. For example, Tom is an existing student and when he logs in, he sees a home page prepared for the Student type users. Mike is a new teacher and since he wants to register for a new account, he is directed to a new teacher account page because of his Teacher user type.

In this LIST_GEN_BY_INDEX element, there is only one parameter "userEntityName" which has three values: Student, Teacher and Admin in our feature model configuration, according to parameter value lookup service. Code generator will use an "IF" template for the first value "Student" with index range (0..0) and an "ELSE IF" template for the rest of the values with index range (1..-1). Since the final "ELSE" clause does not contain parameter, we just leave it as untagged code in the template.



```
<code_generation_unit unit_id="37" unit_name="UserLoginPage_java_tag_expansion"
description="expand tags to create UserLoginPage class that represents the login page for the
system" template_location="edu/ucsc/cse/exam/auth/UserLoginPage.java.tmpl">
  <tag_expansion tag_id="page_redirection">
    <parameter_content lookup_type="Function"
parameter_name="userEntityName">findNonParentclassUserEntityNames</parameter_content>
  </tag_expansion>
</code_generation_unit>
```

userEntityName = ["Student", "Teacher", "Admin"]

```
[[<?xml version="1.0" encoding="UTF-8"?>
<annotation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="template_annotation.xsd">
  <id>page_redirection</id>
  <description>redirect to the next page 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(result.getObjectType().equals("$userEntityName$$")){
            session.set$userEntityName$$(($userEntityName$$)result.getObject());
            if(backPage == null){
              setResponsePage(new $userEntityName$$HomePage());
            }else{
              setResponsePage(backPage);
            }
          }
        </content>
        <content separator="\n" index_range="1..-1">
          else if(result.getObjectType().equals("$userEntityName$$")){
            session.set$userEntityName$$(($userEntityName$$)result.getObject());
            if(backPage == null){
              setResponsePage(new $userEntityName$$HomePage());
            }else{
              setResponsePage(backPage);
            }
          }
        </content>
      </LIST_GEN_BY_INDEX>
    </part>
  </template>
</annotation>
]]
```

```
...
      if(result.getObjectType().equals("Student")){
            session.setStudent((Student)result.getObject());
            if(backPage == null){
              setResponsePage(new StudentHomePage());
            }else{
              setResponsePage(backPage);
            }
          }
      else if(result.getObjectType().equals("Teacher")){
            session.setTeacher((Teacher)result.getObject());
            if(backPage == null){
              setResponsePage(new TeacherHomePage());
            }else{
              setResponsePage(backPage);
            }
          }
      else if(result.getObjectType().equals("Admin")){
            session.setAdmin((Admin)result.getObject());
            if(backPage == null){
              setResponsePage(new AdminHomePage());
            }else{
              setResponsePage(backPage);
            }
        }
      else{
           error("unsupported user type");
        } ...
```

Figure 7-16. Example for LIST_GEN_BY_INDEX tag expansion.

7.3.5 COUNT and LIST_GEN_COUNTER Elements

COUNT and LIST_GEN_COUNTER elements are designed for array and array element definitions whose length depends on the length of parameter value lists, which is dependent on the design decisions in the feature model. For example, a String parameter called paramName has a list of five values. The source code statements define a String array to hold these values and print out each array element afterwards. A template developer does not know beforehand how many values this paramName parameter might have—this depends on how the parameter values are defined in the feature model. To solve this problem, we introduce two directives, COUNT and LIST_GEN_COUNTER, which are mostly used together. COUNT element is used to generate array definition statements and LIST_GEN_COUNTER is used to generate statements specific to each array element. The basic idea is that the code generator keeps an internal counter that increments by one each time we find a new parameter value until the parameter value list is exhausted. The code generator then uses the current counter value as the current array element index (LIST_GEN_COUNTER directive) or the final counter value as the array length (COUNT directive).

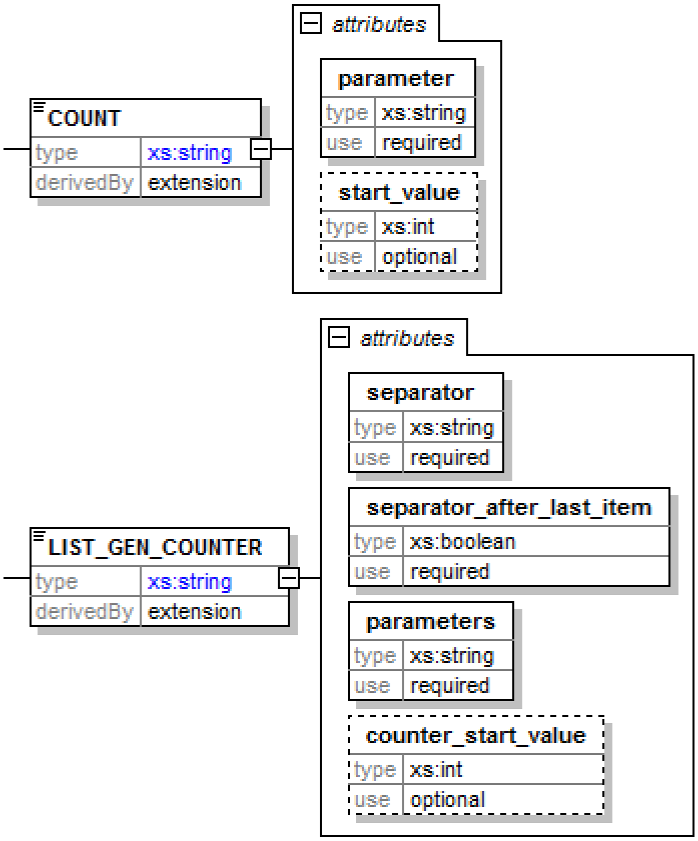
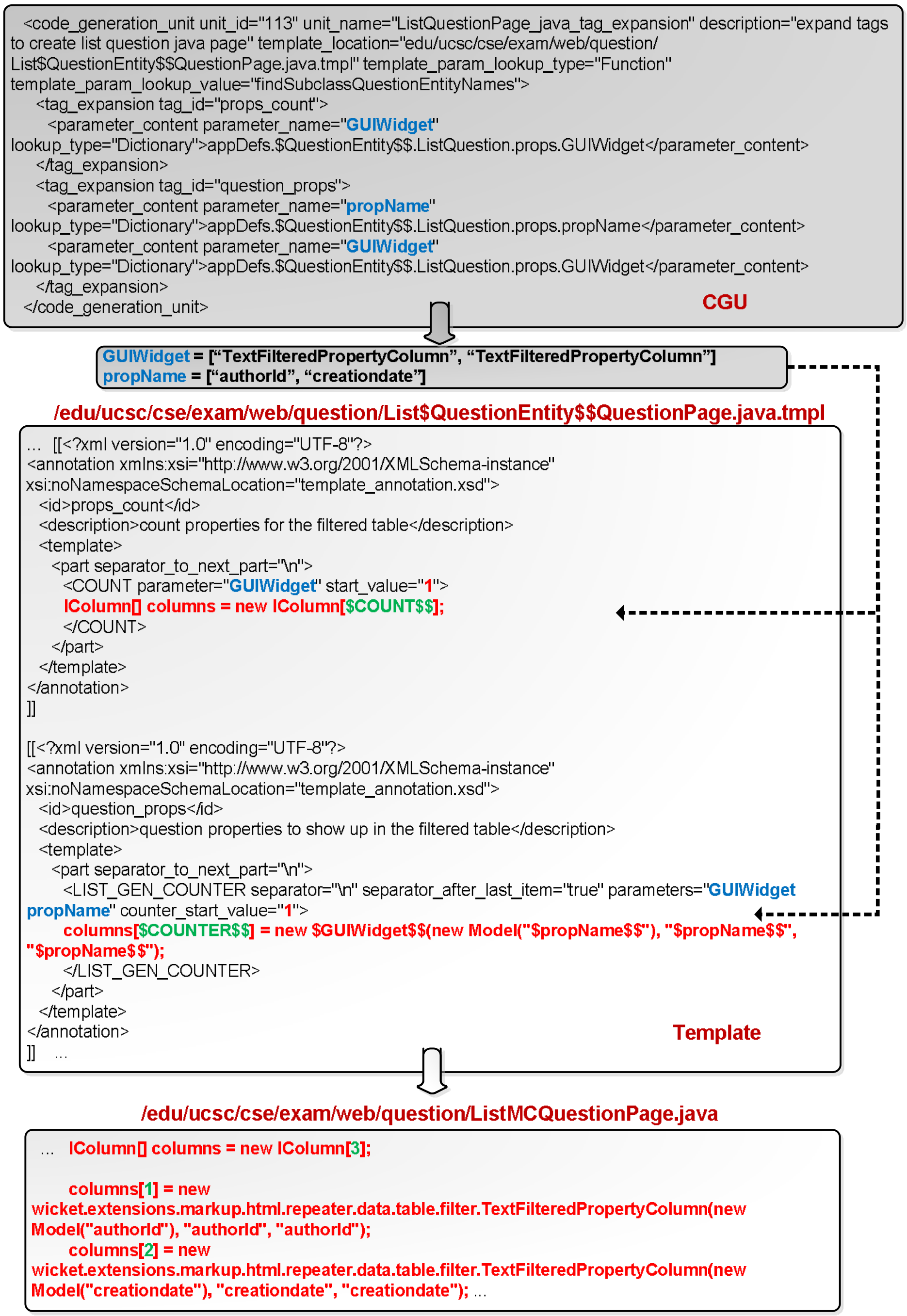


Figure 7-17. XML schema structure for COUNT and LIST_GEN_COUNTER elements.

Figure 7-17 shows XML Schema definition for these two elements. The COUNT element has two attributes: *parameter* and *start_value*. The parameter attribute specifies the name of the template parameter whose value list will be used for the COUNT element. The start_value attribute defines a initial integer value for the counter. Note that this start_value attribute is optional—when the start_value is not specified, code generator uses 0 as the default start value. The text body of the COUNT element is the item template to be used to generate code. A parameter $COUNT$$ is embedded in the template to represent the final value of our counter when the parameter value list is fully explored. The value of the $COUNT$$ can be computed using this formula: **$COUNT$$ = start_value + len(param_value_list) – 1**.

LIST_GEN_COUNTER element is used to generate list items related to each array element. The attributes *separator* and *separator_after_last_item* are similar to those in other list item generation elements: separator defines the String to separate each list item and separator_after_last_item determines whether an extra String separator is required after the last generated list item. The attribute *parameters* lists parameter names to be used in the list item template. The *counter_start_value* attribute is similar to start_value attribute for COUNT element—it defines the starting value for counter. At each tick, the counter value can be accessed using an internal parameter $COUNTER$$, e.g. at first tick, the value of $COUNTER$$ is counter_start_value and it keeps increasing until the parameter value list exhausts.



```
<code_generation_unit unit_id="113" unit_name="ListQuestionPage_java_tag_expansion" description="expand tags
to create list question java page" template_location="edu/ucsc/cse/exam/web/question/
List$QuestionEntity$$QuestionPage.java.tmpl" template_param_lookup_type="Function"
template_param_lookup_value="findSubclassQuestionEntityNames">
    <tag_expansion tag_id="props_count">
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.$QuestionEntity$$.ListQuestion.props.GUIWidget</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="question_props">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.$QuestionEntity$$.ListQuestion.props.propName</parameter_content>
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.$QuestionEntity$$.ListQuestion.props.GUIWidget</parameter_content>
    </tag_expansion>
</code_generation_unit>
```

```
GUIWidget = [“TextFilteredPropertyColumn”, “TextFilteredPropertyColumn”]
propName = [“authorId”, “creationdate”]
```

/edu/ucsc/cse/exam/web/question/List$QuestionEntity$$QuestionPage.java.tmpl

```
…  [[<?xml version="1.0" encoding="UTF-8"?>
<annotation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="template_annotation.xsd">
  <id>props_count</id>
  <description>count properties for the filtered table</description>
  <template>
    <part separator_to_next_part="\n">
      <COUNT parameter="GUIWidget" start_value="1">
      IColumn[] columns = new IColumn[$COUNT$$];
      </COUNT>
    </part>
  </template>
</annotation>
]]

[[<?xml version="1.0" encoding="UTF-8"?>
<annotation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="template_annotation.xsd">
  <id>question_props</id>
  <description>question properties to show up in the filtered table</description>
  <template>
    <part separator_to_next_part="\n">
      <LIST_GEN_COUNTER separator="\n" separator_after_last_item="true" parameters="GUIWidget
propName" counter_start_value="1">
      columns[$COUNTER$$] = new $GUIWidget$$(new Model("$propName$$"), "$propName$$",
"$propName$$");
      </LIST_GEN_COUNTER>
    </part>
  </template>
</annotation>
]]   …
```


/edu/ucsc/cse/exam/web/question/ListMCQuestionPage.java

```
…   IColumn[] columns = new IColumn[3];

    columns[1] = new
wicket.extensions.markup.html.repeater.data.table.filter.TextFilteredPropertyColumn(new
Model("authorId"), "authorId", "authorId");
    columns[2] = new
wicket.extensions.markup.html.repeater.data.table.filter.TextFilteredPropertyColumn(new
Model("creationdate"), "creationdate", "creationdate"); …
```

Figure 7-18. Example for COUNT and LIST_GEN_COUNTER tag expansion.

Figure 7-18 shows an example to generate source code for the page to list MC questions. The CGU contains two tag expansion elements: one for a tag with id “props_count”, the other for a tag with id “question_props”. The first tag is used to generate the array definition for the columns array. Notice that because we cannot predict the length of parameter value list, $COUNT$$ parameter is used as the array length in the definition. Later, this $COUNT$$ parameter will be computed using the formula we discussed in the previous sections based on the actual parameter value list length. In this example, the computed $COUNT$$ parameter value is 3. Since there is always a default first element in this array, the start_value attribute is set to 1 instead of using the default value of 0.

The example for LIST_GEN_COUNTER is also quite obvious: list items are generated one by one using list item template. The only difference from other list generation patterns is that array index needs to be inserted in each generated item. So, we designed an internal counter that increases by one when a new list item is generated. Using the counter, array index is computed using the current counter value and replace a $COUNTER$$ parameter in the item template. Since the counter starts from 1 in our case, the first generated array element has index 1 and the second array element has index 2.

8 Code Generation

The Rhizome code generator is quite simple in structure: it builds the parameter lookup tree, sequentially processes CGUs defined in a feature model, searches for parameter values and performs text transformation to produce the final source code. This chapter describes the basic code generation process. It next details each module in the code generator and illustrates how these modules fit together to produce final source code. Finally, the chapter discusses how to make changes to the template language, the CGU XML schema and the code generator.

Figure 8-1 shows the code generator architecture and overall code generation process. Code generation starts with a FeatureML document. This document contains feature model design choices and CGU definitions. This document is parsed and transformed into a parameter lookup tree for convenient and high performance access. Note that this parameter lookup tree content is specific to each FeatureML document. Then, all CGUs are processed one-by-one. The result after processing a single CGU might be a work-in-progress template file or completed source code, depending on whether all place holders and tags have been processed. After all CGUs have been processed, the code generator wraps up and cleans up memory spaces allocated for the parameter lookup tree and FeatureML document. Below, each step (marked with numbers in Figure 8-1) is described in detail.

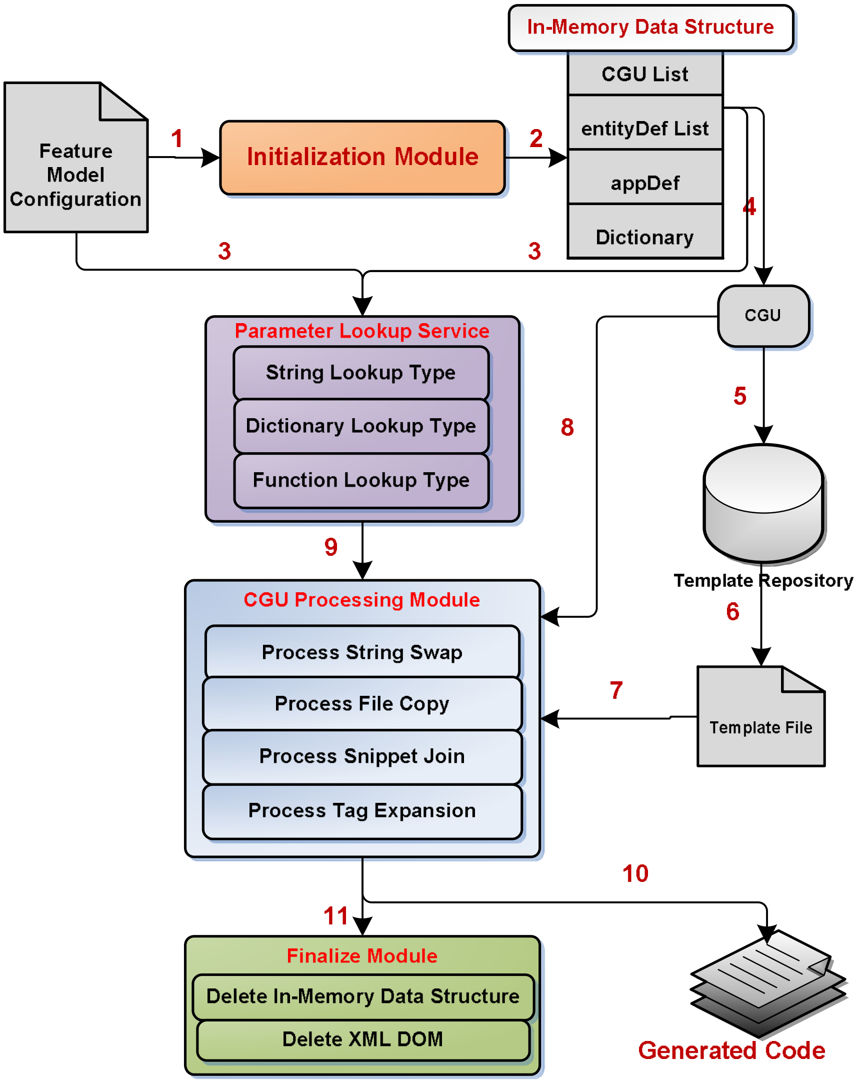


Figure 8-1. Code generator architecture and code generation process.

After the feature model design phase completes, a FeatureML document is created. This document has design decisions for every aspect of the entities and applications. It also contains all of the CGU definitions the code generator needs to begin processing. Step 1 is to feed this FeatureML document into the Initialization Module. This module parses the FeatureML document into a DOM model and sets up a parameter lookup tree that stores entity and application design choices using Dictionaries and Lists.

This lookup tree is domain-specific because it is essentially another form of the feature models in a particular domain. A generic feature modeling process can give a standard set of structures in this lookup tree, such as user entities, data entities and applications. However, the exact meaning of an entity and an application is specific to a domain. For example, the question and exam entities are unique to the online exam system domain, thus their properties like question contents and exam answers are domain-specific too. For the same reason, the parameter queries that use this lookup tree are defined particularly for a specific domain.

Another factor that affects the lookup tree is how the code templates are implemented. Programming language has an impact on the primate and collection data types used in the templates. Implementation libraries and frameworks might also affect the content for the lookup tree. For example, Spring framework, Hibernate object relational mapping and MySQL relational database are used in the demonstration online exam systems. They have configuration files, the contents of which are affected by the design decisions in the feature model. To correctly generate these configuration files, there are specific structures in the lookup tree. For example, we store all design choices related to Hibernate mapping in a vp_entity XML element called "hibernate_mapping_settings". Its sub-elements store information such as mapping name, type, table column, table and table class.

To transform a FeatureML document into this lookup tree, the code generator uses a set of naming conventions for vp_entity and vp_application names. This name convention scheme varies by many factors such as application domain, programming language, platform, toolkit and framework. This naming convention makes it possible to identify and store user entities, data entities, application entities and their properties correctly. It also helps to store containment and inheritance relationships.

We list several existing naming conventions we used to translate the FeatureML document for online exam systems below and explain their meanings. Note that some of these naming patterns are applicable for other domains, such as use *EntityType_types* to find all the names for entities. But some naming patterns are for our demo online exam domain only, such as ***hibernate_mapping_settings***, because it is highly unlikely that systems in another domain still use Hibernate.

- **EntityType_types**: An entity type defines a set of related entities. For example, we have an EntityType called User, which includes various entities such as Teacher, Student, TA, Admin, etc. When the code generator sees a vp_name in the form of "ABC_types", then it will create a list ABCDef in the memory to store entity definitions for this "ABC" entity type.
- **EntityName_props**: An entity name defines the name for an entity. It is used in many places in the generated code: entity class, entity DAO class, Hibernate mapping file, database table definition, Web pages, etc.
- **basic_value_type**: This is used to define basic value types such as Integer, Long, String, etc. Since our online exam systems are implemented in Java, we use values like java.lang.Integer, java.lang.Long, java.lang.String, etc. We can also use types specific to a generated system, such as edu.ucsc.cse.exam.datamodel.Question for the Question type in our online exam systems.
- **value_length_limit**: The option value for this vp_entity takes a form of m..n
- **PropName_collection**: There are two property types: basic value types and collection types. A collection type holds a collection of objects such as a list, a map, a hash table, etc. A vp_name in the form of "ABC_collection" defines a

collection property with the name “ABC”. This vp_name also defines an actual implementation class for this collection property such as java.util.ArrayList and collection content type such as java.lang.String.

- **collection_type_CollectionInterface**: The CollectionInterface substring of this format defines a collection interface such as Map, List, Set, etc. Its option defines specific classes such as java.util.HashMap, java.util.TreeMap, java.util.ArrayList, java.util.LinkedList, java.util.HashSet, java.util.TreeSet, etc.
- **hibernate_mapping_settings**: We use Hibernate ORM framework in the code generator implementation. This element and its sub-elements store Hibernate settings and table mapping information.
    - **op_name**: This attribute on each sub_option element has a format of “EntityName_MappingName”, from which the code generator gets the Hibernate mapping name.
    - **hibernate_mapping_type**: This defines Hibernate mapping types such as one-to-many, many-to-one, and many-to-many.
    - **hibernate_mapping_column**: This defines table columns used as Hibernate mapping IDs.
    - **hibernate_mapping_table**: This defines database table names used for Hibernate mappings.

- **hibernate_mapping_target_class**: This defines target classes used in Hibernate mappings.

The result of Step 2 is a quite complicated data structure and we have covered its working mechanisms, including parameter value lookup methods, in previous chapters. Step 3 represents the data inputs for the parameter lookup service: either from the entityDef list or appDef dictionary (the Dictionary lookup), or directly from the FeatureML document (the String and Function lookups). These three types of lookup service have been covered in previous sections already.

The CGU definitions are processed one by one. Some CGUs have dependencies on other CGUs, e.g. the snippet_join CGU requires that its referenced CGUs (identified by the ref_CGU_ids attribute) to be processed and generate snippets first. First, according to the specific details of each CGU definition, the code generator retrieves the template file path from the template repository (Step 5, 6, and 7). If the template path contains a parameter, the parameter needs to be resolved using the parameter lookup service possibly creating several template paths. Each path is processed individually to produce one source code file. Besides the template file path, other CGU definition information is also passed to the CGU processing module (Step 8). Each template file is processed according to its CGU type. The string swap CGU performs a global string replacement in the template file content. The file copy CGU just copies a template file to a new file. The snippet join CGU assembles output from other CGUs. Tag expansion is more complex: it contains a list of tag expansion elements. Based on the tag expansion element type, the code generator performs generation activities and replaces the tags in the template files. During this CGU handling process, the required parameters values are provided through the lookup service, as depicted in Step 9.

Finally, once all CGUs have completed their processing, the result is the generated code, as shown in Step 10. A finalize module performs cleanup jobs in Step 11, such as deleting the XML DOM and releasing the in-memory parameter lookup tree.

8.1 Code Generator Development

8.1.1 Code Generator Development Process Overview

Our process to create a code generation infrastructure for a new domain is a gradual and iterative process. First, a detailed domain analysis is performed to classify feature variability. The two-step iterative method for domain analysis centers on the entities in a system. First, entities need to be identified. For example, an online exam system requires entities like users, questions, exams, question answers, and exam answers. These entities are either objects or subjects of domain-specific operations. The second step of the domain analysis is to determine the operations associated with these entities. For example, a student user can take an open exam or view their score on a graded exam. A question is the subject of update and delete operations. These two steps might run several iterations, since new operations may require new entities and vice versa. Entities and operations represent entity variation points and application variation points correspondingly.

After the domain analysis completes, the domain boundary and feature variability are known, and this drives the implementation in the code generator. There are as yet no template files, the core pieces of our template-based code generation framework. To create these templates, several sample applications for the domain are developed, which explores a most flexible architecture for easy implementation of different feature design decisions. The development knowledge to design template files is obtained from this trial implementation effort.

With the template files in place, it is possible to start writing customized FeatureML documents, which contain design choices for each feature variation point. Each domain has a specific in-memory data structure called parameter lookup tree, which converts the FeatureML document into a more efficient representation for parameter queries. This lookup tree provides a better performance than retrieving parameter values by directly querying the FeatureML document. Code generation units are defined to connect feature model with the code generator. A CGU defines where to find a template file and how to find parameter values using parameter value lookup services. CGU also has different types and each type will be processed differently in the code generator to produce code.

When all CGUs are turned into source code files, a system that implements a given feature model configuration is generated automatically. In a similar manner, a different system can be generated given a different feature model configuration.

When new domain features appear in an existing domain, new features in terms of source code by changes to existing template files or adding new template files. Entity and application variation points are then updated, along with the addition of new items to the parameter lookup tree and the addition of new CGU definitions. However, the CGU processing module does not require modification since the code generation is based on text processing and thus is not affected by changes in the semantic layer. The code generator's job is to scan file content and replace tags with generated code according to predefined rules.

8.1.2 Extending the Code Generator

The CGU types and the template language are products of a careful examination of the source code for existing domain applications. CGU types represent code generation activity between and within templates, such as globally replacing a string in a template file, the replication of template file content to create a new source file, the assembly of generated source code snippets into a bigger source code file, and the expansion of tags in a template file. The template language is used to describe tag expansion elements at the intra-template level. The language defines small blocks of XML code inside a template file, which are replaced by actual generated code.

During analysis of the code generation requirements for a new domain, it may happen that new code generation requirements are uncovered. Adding template level code generation activities requires modification of the CGU XML schema. Currently, there are four types of sub-elements for code_generation_unit element (see Figure 7-4). A new CGU type is appended as a new sub-element. In the generator code, there are CGU processing functions in the form of processXYZCGU, where XYZ is the name of the CGU type. For example, processStringSwapCGU perform global string replacement operations. Another function called processCGU works as a switch to redirect to proper CGU processing function based on the type of an input CGU. To add a new template level code generation activity, a new CGU processing function is written and registered with the processCGU function.

The addition of intra-template level code generation activities, such as generating a new code block pattern, requires modification of the template language XML schema.

Tag expansion element types are defined under a part element (see Figure 7-8). There are four existing types: COPY, LIST_GEN, LIST_GEN_CONDITIONAL, and LIST_GEN_BY_INDEX. New types can be appended to existing ones in the schema. In the generator code, tag expansion element handlers have the form of processXYZNode, where XYZ stands for the tag expansion element type. For example, the processListGenByIndexNode function handles LIST_GEN_BY_INDEX tag expansion elements. We can create a new handler and register it with processXMLTag function, a switch that redirects to the proper tag expansion element handler.

The Rhizome feature modeling language, template language, and the code generator have a modular architecture. Adding new code generation capabilities is relatively straightforward with minimal interference with existing code generation activities. Code generator developers using our framework can customize and extend the existing code generator to fit their specific requirements.

8.2 Implementation Details

The original Bamboo generator for content management systems based on containment modeling framework was implemented using Java. Since a large part of the code generation is the plain text matching, replacement and XML processing, a scripting language is more suitable for this purpose. Based on the lessons learned from Bamboo, we used Python to implement the code generator in Rhizome. It has convenient libraries for text processing, file system operations, regular expression matching and XML processing. In general, any programming language is suitable to implement the code generator, but a scripting language could be much easier and the code size is significantly smaller.

Wicket Web application framework [84] was used to implement the online exam system code templates. It is a modular Java Web application framework that provides a clean separation of HTML presentation layer and Java business logic layer. This provides great benefits for the design of code templates: each function, as defined in a vp_application entity in FeatureML document, is implemented using a pair of HTML file and Java file. This proves the choice of programming language, code libraries and frameworks do have an impact on the code templates and code generation process.

For the demo online exam systems, it took two weeks to implement code templates using Spring, Hibernate, Wicket and MySQL. It is about two days of work to compose a FeatureML document (around 3000 lines of XML code) assuming a user is familiar with the FeatureML modeling language. The actual code generation takes about 8 seconds. The environment configuration includes the following items: an

IBM Thinkpad R32 with an Intel Pentium 4 1.60GHz processor, 768MB DDR 266MHz memory, Windows XP with SP2, SUN J2SE 1.6.0_01-b06 and Python 2.5.2 (XML processing is implemented using xml.dom.minidom).

9 Future Work

9.1 Variability Repository

This dissertation focused on the feature modeling language and the code generator. Not much effort was put on building a variability repository due to time and resource limitations. However, a variability repository is another key component of an automated software product line. It models the whole design space for a product line—instances of existing feature models, feature design choices, feature documentation, design rationale and impact, and links between feature design choices and code generation (CGUs implement these links in our Rhizome platform). Here, we present some initial design ideas for the content and implementation of this variability repository.

Feature design choices list feature definitions and structures found in existing feature models. A designer can either reuse existing design choices or create new choices and save them in the repository. Each design choice contains a description of the circumstances during which this choice should be selected—this is called a design rationale.

A variability repository also stores dependencies between feature design choices. Simple dependencies, including required, optional and exclusive relationships, can be expressed using logic operators *AND*, *OR* and *XOR*. Cardinality-based group dependency can be defined using a scope notation (*minimum..maximum*) to specify the number of choices a particular feature could possibly have. Combining simple dependencies and group dependencies, a large array of dependency relationship types in feature modeling can be covered. For other customized dependency relationships, we need a formal language to express more complex conditions, such as an IF-ELSE IF-ELSE style condition. For example, we can define a customized dependency like "if the security level is high, then a teacher must contact system admin in person to obtain passphrase for new account registration, and exam content should be generated randomly for each student".

When a feature model is derived from the variability repository, a feature model validation tool checks all three types of dependency relationships to confirm that the feature model satisfies these dependencies. The variability repository also maintains links between feature design choices and CGUs so that when we have a valid feature model, these corresponding CGUs can be associated and loaded automatically into the code generator. So, given a feature model and these dependency relationships, the code generator can create source code using the templates without extra information.

We also present some initial thoughts on the implementation of the variability repository. One approach is to use the XML technology and implement the variability repository for a specific domain as an XML document with a set of design tools built on it. This XML document conforms to a variability modeling XML schema, which is quite similar to feature modeling schema in structure. It also contains entity variation points, application variation points and CGU definitions. The difference is that the variability repository contains extra attributes and elements to describe feature dependency relationships. The relationships define rules to validate feature models—each valid feature model derived from the variability repository must conform to these rules. Figure 9-1 shows the relationship between the variability repository and feature models derived from it. It also shows the level of modeling: the variability repository and feature models are on the model layer and their schemas are on the meta-model layer.

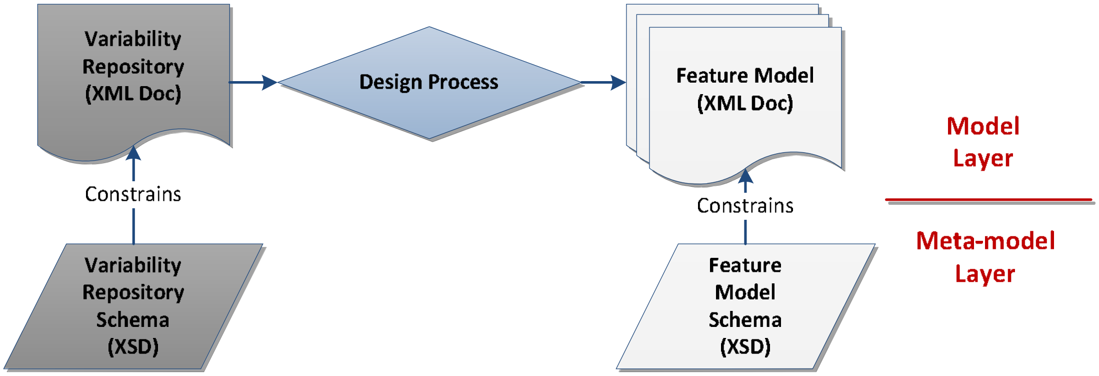

Variability Repository (XML Doc)
Design Process
Feature Model (XML Doc)
Constrains
Variability Repository Schema (XSD)
Constrains
Feature Model Schema (XSD)
Model Layer
Meta-model Layer

Figure 9-1. Relationship between variability repository and feature models.

In the variability repository, feature design choices can be defined in a similar way as in the feature model. Dependencies are defined differently. Simple dependencies like required or, optional relationships and feature group dependencies can be defined on the feature design choice elements. For example, consider feature P with three child features C1, C2, and C3. We can specify a dependency on feature P saying that only one of C1, C2, and C3 can be selected as a child feature for P (an XOR relationship). Customized dependencies can involve arbitrary features and feature design choices. A new element type such as "customized dependency" is required at the global level in the variability repository. This element defines condition expressions that must evaluate to be true when a feature model is derived. These global conditions use variables about feature existence, feature structure pattern, and feature design choice values. For example, we can define a global condition saying “if feature A is selected, then feature B must have a design choice value of val1 or val2”.

Some initial investigation has been done regarding what tools to use for the implementation of customized dependencies. The XML schema language called Schematron [49] fits our requirements best. It allows us to use path expressions to define rules and assertions about an XML document. These Schematron rules and assertions can be evaluated during the feature model design process to validate a feature model and provide useful information about how to fix an invalid feature model.

Another optional new element in the variability repository is a feature model instance element. This element records all the existing feature models derived from the variability model. Each feature model corresponds to one of such instance element. In each feature model instance element, feature options are referenced by their entity ids and option ids. The benefits of this element is that we can quickly reuse an existing feature model to create a new feature model, making only a few necessary changes.

9.2 Design Wizard

The creation of a GUI-based feature model design wizard is another direction of future work. Currently, a feature model designer needs be familiar with the FeatureML feature modeling language and manually editing a feature model XML document. This is non-trivial work: our demo application is an online exam system with relatively simple features, yet the sample FeatureML document is around 3000 lines of XML. During the editing process, it is easy to make errors such as typos or referencing a non-existent feature option. It is challenging to locate and correct these errors since they are only visible in the code generator—some hidden errors have to be identified when we run the generated system.

A design wizard will ease the pain of manually editing the FeatureML document. The design wizard presents feature design choices to a designer through a GUI. Instead of directly editing XML documents, a designer interacts with GUI controls such as dropdown lists, options, and input boxes to make design choices about the new feature model and the actual XML document is generated by the wizard.

The dependency relationships in the variability repository are checked automatically during each design step. This not only helps eliminate invalid feature design choices in the following design steps, but also provides assistance information to that a designer knows the impact of each design choice. The wizard also allows a designer to scroll back and forth through the steps to make adjustments about his design choices. The validator running at the background immediately ensures validity of current design choices.

The design wizard can also enforce our user-centered feature model design process, in which we first define user entities in a system, and then define data entities, and finally associate dynamic applications with user and data entities.

Besides using existing knowledge in the variability model, the wizard can also assist a designer in creating brand new feature options and updating existing variability model. However, the feature model output with new feature options does not contain CGUs for these newly created feature options. Thus, it cannot be used to generate code immediately until platform developers prepare corresponding templates and CGUs and fix the feature model.

9.3 Change Impact Analysis

Design impact analysis studies how changes in the feature model affect the source code. Code change impact analysis studies the problem in the other direction: it tries to identify what features will be affected if we modify source code in a particular position. In the current code generator implementation, feature model design choices are stored in an in-memory data structure and CGUs employ special path expression and query mechanisms to retrieve these design choices and pass them to the template as parameter values. Since all these elements—the feature, feature option, CGU, template, and template tags all have identifiers, tracking down how a feature design choice is translated into code blocks in the code base is possible. This tracking information can be stored in a separate impact analysis repository, also with query tools to communicate with this repository.

One thing to note is that impact analysis data depends on the template and CGU definitions, which are developed by product line platform developers. They manually identify similarity and variability in the code base and define CGUs based on their findings. So, if the source code change is not identified as a tag in the template or if a feature option is never used to generate code, it is not possible to study the change impact analysis for these two cases. In this sense, our change impact analysis is semi-automatic and based on templates.

9.4 Improvement on the Code Generator

There are several areas where the code generator can be further optimized: improving code generator performance, extending the CGU modeling element and template language to include more code generation patterns, applying the Rhizome platform and methodology on software product lines with a larger number of product variants at grander scale. Currently, the input feature models for our demo online exam systems are include 3000 lines of XML code and the performance is quite acceptable—it takes about 10 seconds to process all the templates. During the code generation process, much of the processing power is spent on queries to retrieve parameter value lists based on path expressions. Each tag in the template might have multiple parameters and each one costs a separate lookup. But in reality, many of the queries are for the same parameter and hence caching query results is helpful to avoid repeating the same query over and over again. Query results can be cached in a hash table using the path expression or function call expression as the key. Currently, if an error occurs during the code generation process, the generator will halt and abandon the rest of the CGUs. Once errors are fixed, processed CGUs need to be reprocessed even though they are not necessarily affected by the error correction changes. This part of the work can be saved if state tracking is supported for CGUs so that generator can "remember" those processed CGUs. If changes to correct these errors do not affect CGUs that have been processed already, the generated source code can be reused. This new design also allows the code generator to skip only those problematic CGUs and complete a full pass of all CGUs even when errors occur during the generation process.

Although several code generation patterns have been identified, there will likely be new patterns to emerge. The extensible generator architecture allows for such upgrades to the CGU modeling elements, the template language for tags and in the code generator code. Code generator developers can follow the discussion in section 8.2.2 to upgrade these language elements and the code generator. Ideally, one could build a code generation pattern library similar to software design pattern library, using the Rhizome feature modeling and generation framework so that the code generator developers can focus on the code generation patterns without worrying about platform implementation issues.

Currently, the feature modeling language, template language and the code generator work well for the sample domain of online application systems. This Web application domain is of a medium size and its functionality and code complexity is limited compared to real world applications. Effectiveness of template-based code generation depends largely on the quality of the templates. If the online exam systems can be tested in a production environment to generate software systems with a larger scale, it will provide better evaluation about Rhizome's strength and weakness. This will provide valuable guidelines in our further work to improve various modeling elements and the code generator.

10 Conclusion

Rhizome is a feature modeling and generation platform for automated software product line development. Software designers have the design expertise and understanding of user requirements, but maybe lack the detailed coding skills. Platform developers are experts on programming languages, application frameworks and libraries, but they may not have complete knowledge about the software domain and user requirements. Using our platform, these two types of software engineers can focus on high-level design and low-level implementation issues respectively. A software designer can use our feature modeling language to do high-level design work by selecting feature design choices that have been implemented by platform developers. Platform developers examine existing software applications in the product line and summarize similarity and variability into code templates. These templates have tags at locations where variability needs to be revealed to feature model designer. Each tag corresponds to a CGU element in the feature model, which collects parameter values for the tag. Hence, a feature model contains information to resolve variability in the template. Then, after a software designer finishes a feature model, he can provide the feature model to the code generator which processes each CGU and turns templates into actual source code.

The Rhizome approach is different from model-based code generation that allows any degree of variability using MDA models. Model-based code generation is based on multi-stage model transformation. Many MDA generators are domain-specific—they have well-defined data structures and application behavior, such as GUI [16], database repository and data management tools, Web application for tables, charts and reports [85], etc. In Rhizome, although the code templates and feature models are specific to the domain, the platform elements such as the feature modeling language, the template language, the template processing engine in the code generator are all general purpose and can be applied for different domains.

Rhizome is also different from code assembler generators where source code for all possible scenarios is prepared beforehand. A software designer selects which scenario to use and the corresponding set of source code files are picked from all candidates. In Rhizome, platform developers control what variability is exposed to the feature model designer without knowing information beforehand about possible design decisions a designer will make. It is only during the actual design process that these design decisions are fixed and eventually passed to the templates for code generation. Feature model designers do not need to understand the template language or edit any template files. Platform developers do not prepare templates based on feature design choice combinations, but rather on design choices for each feature, with the code generator creating the code for all combinations.

Another common comparison case to Rhizome is a big software package that contains all possible variants, which uses configuration files to turn on and off design choices to yield a customized system. A configuration file is a popular approach for exposing variable functionality to the end user. The Apache Web server, UNIX packages, PHP runtime and J2EE application servers all use this approach for customization. These configuration files have simple content format, usually just variable name-value pairs. Dependencies among these configuration items have to be managed manually by the user. There is also little explanation about impact of different configuration variable values. In a broad sense, the FeatureML document is a more advanced configuration file that includes dependencies among design choices. This allows automatic validation of this “feature configuration” file. Using a code generator instead of including all possible variants has the advantage of smaller distribution package size. It is also more convenient for software upgrades—instead of having the risk of breaking existing code, we can update a FeatureML document, validate it and generate code that is guaranteed to work.

A framework is an abstract object-oriented design [86]. It defines abstract classes and interfaces for components, but leaves the actual implementation to the user. The user can tailor the framework by subclassing and implementing the component designs in the framework. Another characteristic about a framework is the inversion of the control—user code implements what to do under what circumstances, but the framework controls the actual process flow. This is known as the Hollywood Principle, "Don't call us, we'll call you" [87]. The framework is a software design technique for software reuse.

There are several key differences between feature-oriented code generation and using a framework. First, they are at different levels. Rhizome is a rapid development platform and methodology for automated software product line development. The framework represents an abstract design for an application architecture and application components. So, in a framework, design decisions have been made, leaving only the actual implementation details to the user code. Second, Rhizome has many different sources of variability: the existence of a feature, the structure and behavior of a feature, dependencies between feature design choices, template implementation variability, etc. In a framework, the focus is on component implementation variability, e.g. using different algorithms and process flow, etc. Third, feature-oriented generation and the framework serve different goals: feature-oriented generation aims to automate product development by reusing implementation knowledge and generating code; the framework is a reusable object-oriented design methodology, the code still needs to be manually implemented. Current implementation of the code generator is a sequential process. We might apply the framework concept to the generator implementation, especially the Inversion of Control (IoC) technique, which might help eliminate the domain dependency in the parameter lookup tree.

This dissertation has yielded improved understanding about various problems related to template-based code generation. First, making a template is a “brittle” process—it depends on many factors: the programming language, coding style, software architecture, and design patterns. One example is the use of XML-based template language to embed XML tags in a template file. However, some template files are XML documents by themselves and the tags are not allowed because <> signs are not legal inside an XML element content. This led to the use of || and ||| to represent < and > respectively in the code templates. Then, an XML normalization changes || and ||| back to < and > after a tag is replaced and before the code is finally produced. Since we compose template files are composed by analyzing similarities and differences between source code files, when source code is implemented by different developers, the structures might differ and the template files could vary as well. Similarly, different application frameworks and libraries might affect the template file content too. If architecture and design patterns are enforced in building the product family, then we might find it easier to compose template files because those patterns already abstract similarities and recurring best practices.

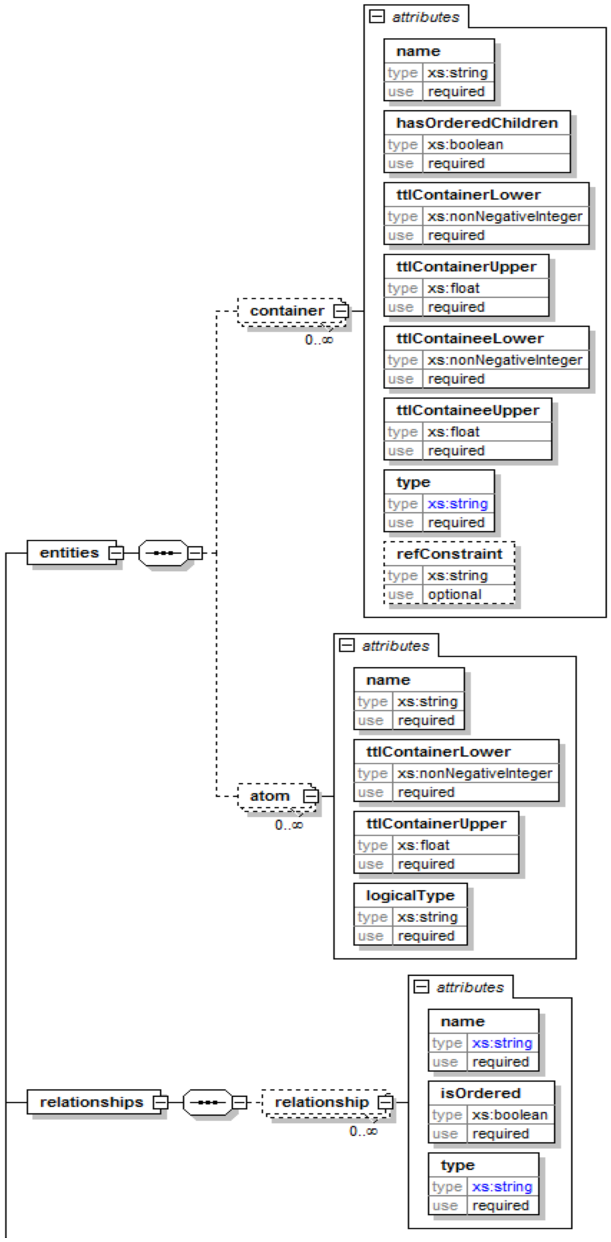
Second, template composition is a controlled process that relies on how much variability platform developers want to expose to the feature model designers. The template files play a bridging role between semantic level (feature model) and code level (source code). Platform developers control how much variability in the source code can be manipulated in the feature model. The more variability in the feature model, the more complex is the code structure that must be put into the code template and the code generator. For variability that is not exposed in the feature model, it is possible to just use default code pieces. One simple example is the data types for data fields in the Java class file. String can be used as the default type for all the text fields, and double can be used as the default type for all the numeric values, if we choose not to expose the data type option in the feature model. If this variability needs to be exposed as a design choice, the code generator has to be modified so that it can recognize features related to data type definitions in the FeatureML document. In the templates where those data types are used, tags must also be added to handle data type processing.

Third, template composition is a dynamic and evolving process. In an existing product line implemented using the Rhizome platform, when new features are added changes must be made to the templates and the code generator, sometimes even requiring changes in the template language. This happens several times during the development of Rhizome code generator when a new code generation pattern was discovered and it could not be handled by the existing code generator.

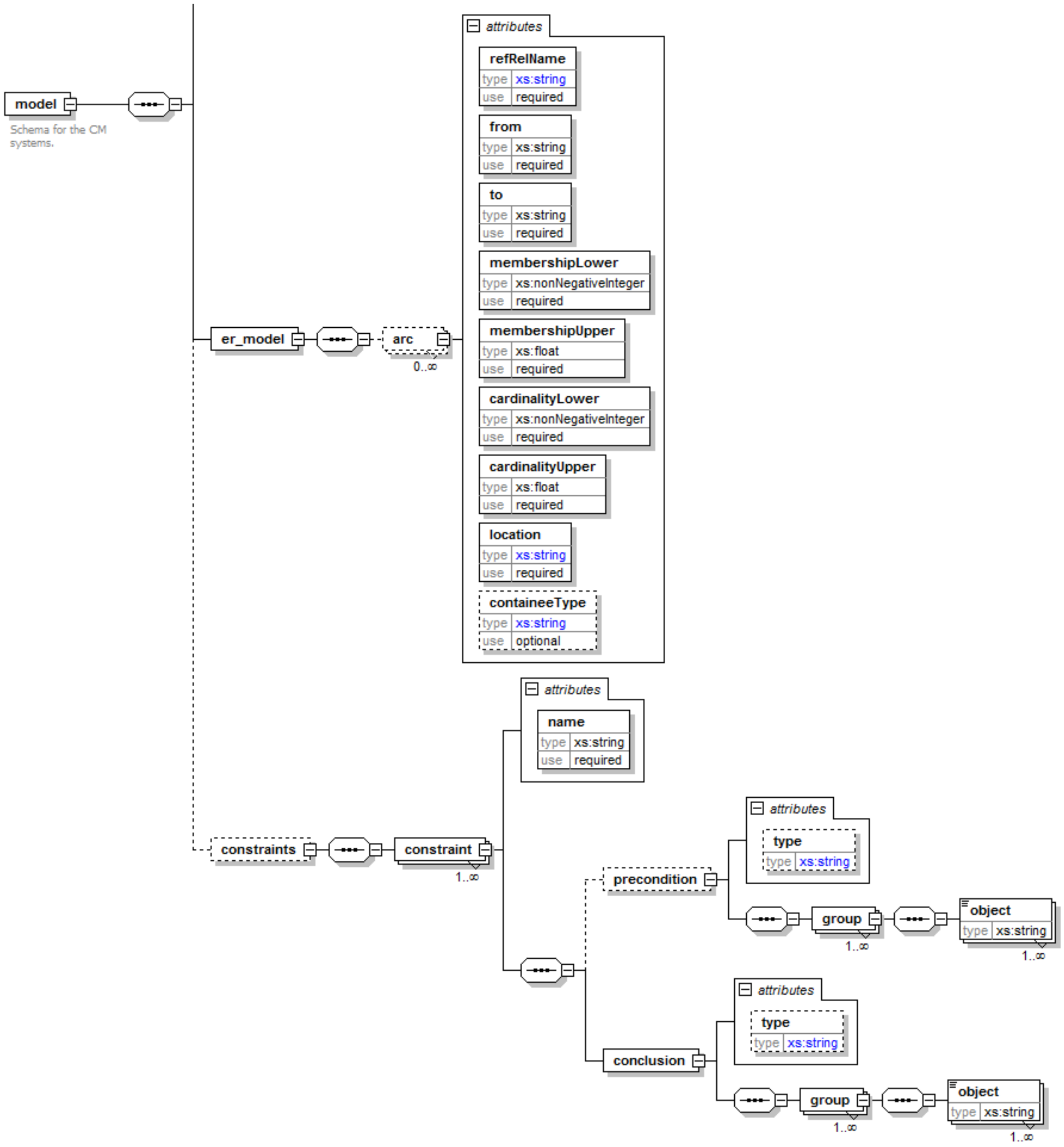
Rhizome's feature modeling language can be used separately from the code generator to model variability in feature structure and feature options down to any level in a feature diagram. Rhizome classified feature variability into entity variability and application variability, with two separated modeling elements vp_entity and vp_application. A graphical editor was developed for an early version of feature modeling language, but later we decided a GUI-based feature model designer will be more useful and intuitive (although the current implementation does not include a design wizard). The CGU definition element is integrated into the feature modeling language, but it can be separated out to have its own XML schema language if needed.

This work only finishes the core platform for automatic code generation based on feature modeling. There are many research directions that can be performed based on our platform, such as a GUI-based feature model wizard that uses a variability model as its design knowledge base and guides a designer to find a path across the design space, change impact analysis both on the feature model side and source code side, and automatic design documentation and reasoning that captures how and why a designer makes particular design choices. We also wish that Rhizome can be adopted and tested on a commercial software product line development, so feedback can be collected for further improvement. Lessons and experiences learned during the Rhizome project can contribute to the research field of automatic application development based on feature modeling. The Rhizome platform will finally promote a new paradigm of software development based on high-level modeling and automated development technologies.

APPENDIX A: Containment Modeling Framework XML Schema (Graphical and Textual)



Containment Modeling Language Schema Part 1.

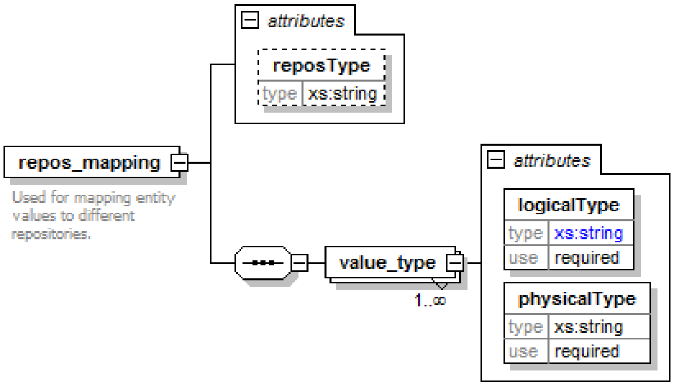


Containment Modeling Language Schema Part 2.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="model">
        <xs:annotation>
            <xs:documentation>Schema for the CM systems.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="entities">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="container" minOccurs="0" maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:attribute name="name" type="xs:string" use="required"/>
                                    <xs:attribute name="hasOrderedChildren" type="xs:boolean" use="required"/>
                                    <xs:attribute name="ttlContainerLower" type="xs:nonNegativeInteger" use="required"/>
                                    <xs:attribute name="ttlContainerUpper" type="xs:float" use="required"/>
                                    <xs:attribute name="ttlContaineeLower" type="xs:nonNegativeInteger" use="required"/>
                                    <xs:attribute name="ttlContaineeUpper" type="xs:float" use="required"/>
                                    <xs:attribute name="type" use="required">
                                        <xs:simpleType>
                                            <xs:restriction base="xs:string">
                                                <xs:enumeration value="generic"/>
                                                <xs:enumeration value="and"/>
                                                <xs:enumeration value="xor"/>
                                            </xs:restriction>
                                        </xs:simpleType>
                                    </xs:attribute>
                                    <xs:attribute name="refConstraint" type="xs:string" use="optional"/>
<!-- "hasCycle" attribute indicates if the container instance and contain other container instances of the same type -->
 <!-- "hasOrderedChildren" attribute indicates if there is ordering of different containee types -->
 <!-- "ttlContainersLower" is the LOWER bound for the number of containers to which this entity may belong -->
 <!-- "ttlContainerUpper" is the UPPER bound for the number of containers to which this entity may belong -->
 <!-- "ttlContaineeLower" is the LOWER bound for the nubmer of containees that may belong to this container -->
 <!-- "ttlContainerUpper" is the UPPER bound for the number of containees that may belong to this container -->
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="atom" minOccurs="0" maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:attribute name="name" type="xs:string" use="required"/>
                                    <xs:attribute name="ttlContainerLower" type="xs:nonNegativeInteger" use="required"/>
                                    <xs:attribute name="ttlContainerUpper" type="xs:float" use="required"/>
                                    <xs:attribute name="logicalType" type="xs:string" use="required"/>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="relationships">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="relationship" minOccurs="0" maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:attribute name="name" use="required">
                                        <xs:simpleType>
                                            <xs:restriction base="xs:string">
```

```
                                                <xs:enumeration value="Ordered_Referential"/>
                                                <xs:enumeration value="Ordered_Inclusive"/>
                                                <xs:enumeration value="Unordered_Referential"/>
                                                <xs:enumeration value="Unordered_Inclusive"/>
                                            </xs:restriction>
                                        </xs:simpleType>
                                    </xs:attribute>
                                    <xs:attribute name="isOrdered" type="xs:boolean" use="required"/>
                                    <xs:attribute name="type" use="required">
                                        <xs:simpleType>
                                            <xs:restriction base="xs:string">
                                                <xs:enumeration value="referential"/>
                                                <xs:enumeration value="inclusive"/>
                                            </xs:restriction>
                                        </xs:simpleType>
                                    </xs:attribute>
 <!-- "isOrdered" attribute shows if there is an ordering for multiple instances of this relationship. In another words, it
shows if there is an ordering for multiple containees of the same type. -->
 <!-- "type" attribute indicates how the containee is stored. It has two selections: inclusive, which physically stores the
containee with the container, and referential, which stores only the reference within the container. -->
 <!-- "location" attribute indicates where to store the reference(relationship). -->
 <!-- "membershipUpper" attribute indicates the UPPER bound for the number of instances of a particular container
type indicated by this relationship. -->
 <!-- "membershipLower" attrtibute is the LOWER bound for the nubmer of instances of a particular container type
indicated by this relationship. -->
 <!-- "cardinalityUpper" attribute indicates UPPER bound for the number of instances of a particular containee type
indicated by this relationship. -->
 <!-- "cardinalityLower" attribute indicates LOWER bound for the number of instances of a particular conatinee type
indicated by this relationship. -->
 <!-- "containeeType" attribute is only for versioning systems.  It is employed to distinguish three types of
relationships in a versioning system: multiple versions of a single containee (1+nv), single versions of multiple
containees (n+1v), and multiple versions of multiple containees (n+nv). -->
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="er_model">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="arc" minOccurs="0" maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:attribute name="refRelName" use="required">
                                        <xs:simpleType>
                                            <xs:restriction base="xs:string">
                                                <xs:enumeration value="Ordered_Referential"/>
                                                <xs:enumeration value="Ordered_Inclusive"/>
                                                <xs:enumeration value="Unordered_Referential"/>
                                                <xs:enumeration value="Unordered_Inclusive"/>
                                            </xs:restriction>
                                        </xs:simpleType>
                                    </xs:attribute>
                                    <xs:attribute name="from" type="xs:string" use="required"/>
                                    <xs:attribute name="to" type="xs:string" use="required"/>
                                    <xs:attribute name="membershipLower" type="xs:nonNegativeInteger" use="required"/>
                                    <xs:attribute name="membershipUpper" type="xs:float" use="required"/>
                                    <xs:attribute name="cardinalityLower" type="xs:nonNegativeInteger" use="required"/>
                                    <xs:attribute name="cardinalityUpper" type="xs:float" use="required"/>
                                    <xs:attribute name="location" use="required">
                                        <xs:simpleType>
                                            <xs:restriction base="xs:string">
                                                <xs:enumeration value="on_container"/>
                                                <xs:enumeration value="on_containee"/>
                                                <xs:enumeration value="on_both"/>
                                            </xs:restriction>
                                        </xs:simpleType>
                                    </xs:attribute>
                                    <xs:attribute name="containeeType" use="optional">
                                        <xs:simpleType>
                                            <xs:restriction base="xs:string">
                                                <xs:enumeration value="1+nv"/>
                                                <xs:enumeration value="n+1v"/>
                                                <xs:enumeration value="n+nv"/>
                                            </xs:restriction>
                                        </xs:simpleType>
                                    </xs:attribute>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="constraints" minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="constraint" maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="precondition" minOccurs="0">
                                            <xs:complexType>
                                                <xs:sequence>
                                                    <xs:element name="group" maxOccurs="unbounded">
                                                        <xs:complexType>
                                                            <xs:sequence>
                                                                <xs:element name="object" type="xs:string" maxOccurs="unbounded"/>
                                                            </xs:sequence>
                                                        </xs:complexType>
                                                    </xs:element>
                                                </xs:sequence>
                                                <xs:attribute name="type">
                                                    <xs:simpleType>
                                                        <xs:restriction base="xs:string">
                                                            <xs:enumeration value="and"/>
                                                            <xs:enumeration value="or"/>
                                                            <xs:enumeration value="xor"/>
                                                            <xs:enumeration value="not"/>
                                                        </xs:restriction>
                                                    </xs:simpleType>
                                                </xs:attribute>
                                            </xs:complexType>
                                        </xs:element>
                                        <xs:element name="conclusion">
                                            <xs:complexType>
                                                <xs:sequence>
                                                    <xs:element name="group" maxOccurs="unbounded">
                                                        <xs:complexType>
                                                            <xs:sequence>
                                                                <xs:element name="object" type="xs:string" maxOccurs="unbounded"/>
                                                            </xs:sequence>
                                                        </xs:complexType>
                                                    </xs:element>
                                        </xs:sequence>
                                        <xs:attribute name="type">
                                          <xs:simpleType>
                                            <xs:restriction base="xs:string">
                                              <xs:enumeration value="and"/>
                                              <xs:enumeration value="or"/>
                                              <xs:enumeration value="xor"/>
                                              <xs:enumeration value="not"/>
                                            </xs:restriction>
                                          </xs:simpleType>
                                        </xs:attribute>
                                      </xs:complexType>
                                    </xs:element>
                                  </xs:sequence>
                                  <xs:attribute name="name" type="xs:string" use="required"/>
                                </xs:complexType>
                              </xs:element>
                            </xs:sequence>
                          </xs:complexType>
                        </xs:element>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:schema>
```

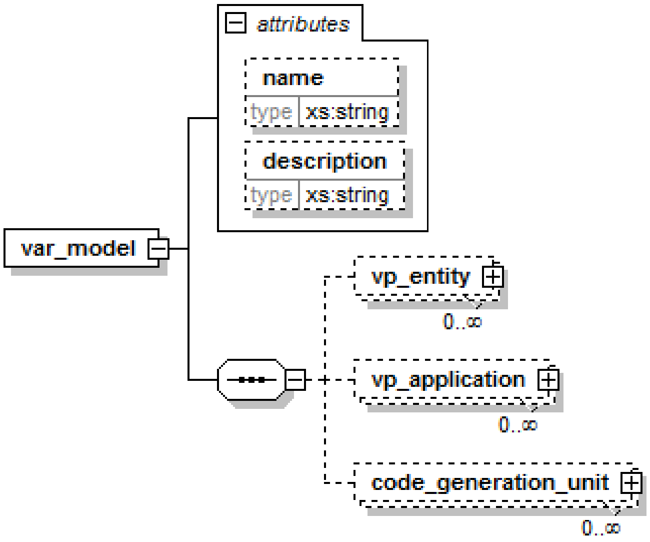
APPENDIX B: Containment Modeling Framework Repository XML Schema (Graphical and Textual)



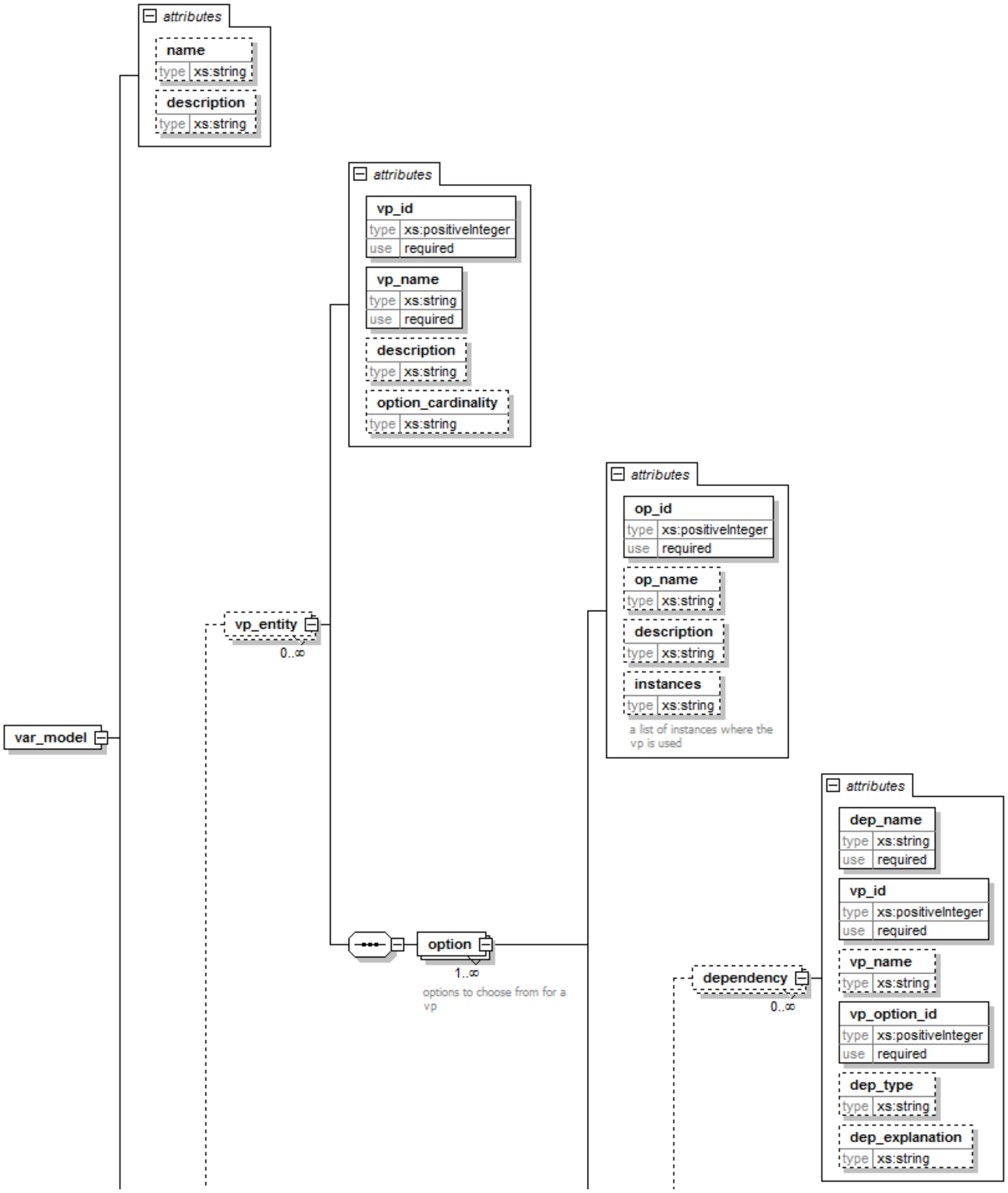
Containment Modeling Framework Repository XML Schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="repos_mapping">
        <xs:annotation>
            <xs:documentation>Used for mapping entity values to different repositories.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="value_type" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:attribute name="logicalType" use="required">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:enumeration value="String"/>
                                    <xs:enumeration value="Integer"/>
                                    <xs:enumeration value="Float"/>
                                    <xs:enumeration value="Datetime"/>
                                    <xs:enumeration value="Text"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:attribute>
                        <xs:attribute name="physicalType" type="xs:string" use="required"/>
                        <!-- currently the schema supports four value types: String, Integer, Float, and Timestamp -->
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="reposType" type="xs:string"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

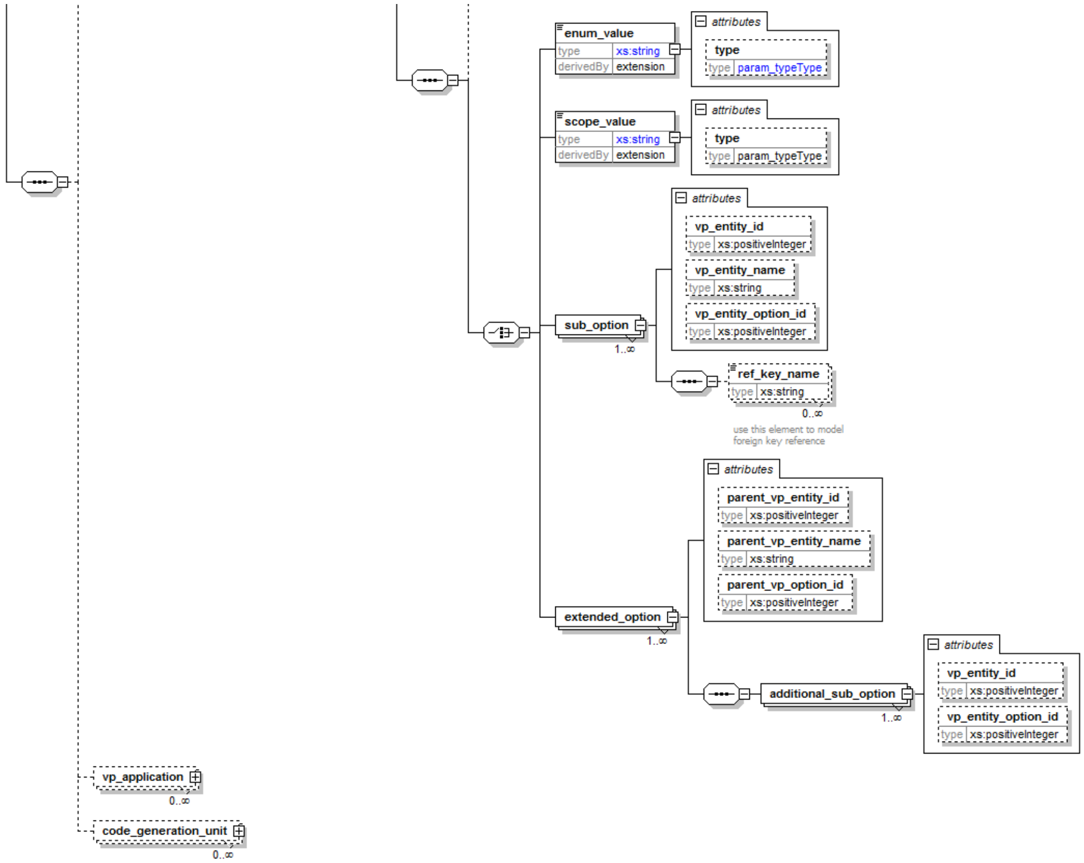
APPENDIX C: FeatureML Language XML Schema (Graphical and Textual)



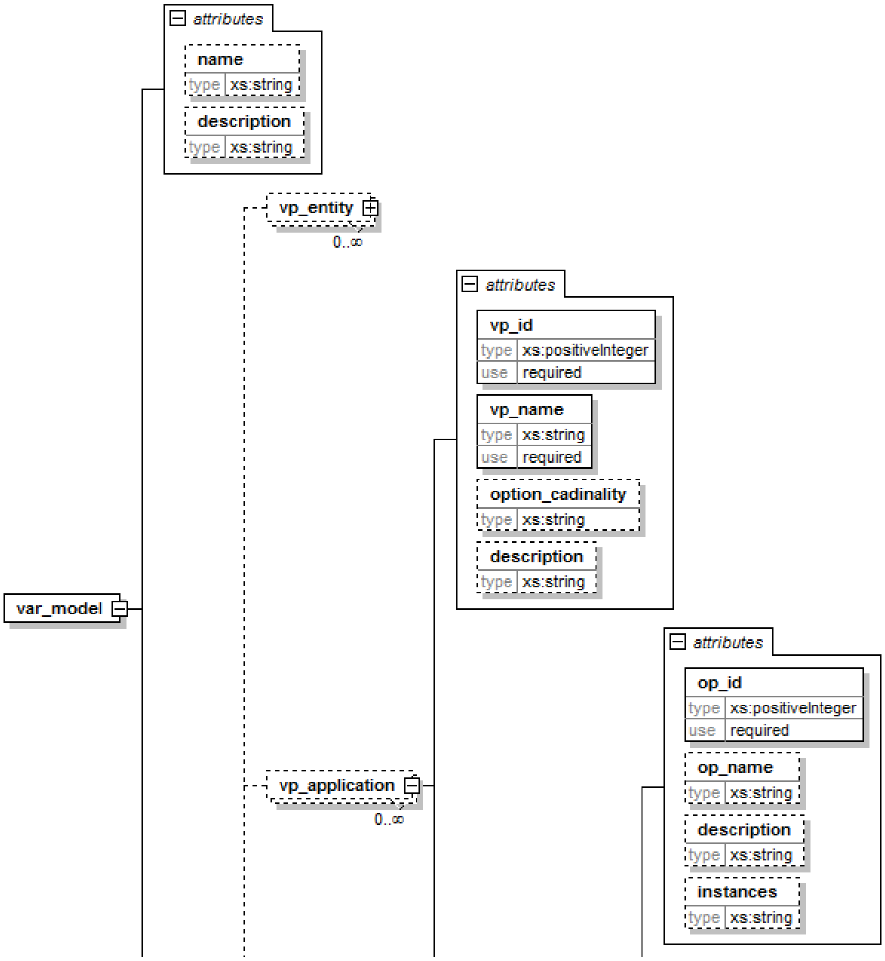
FeatureML Schema Overview.



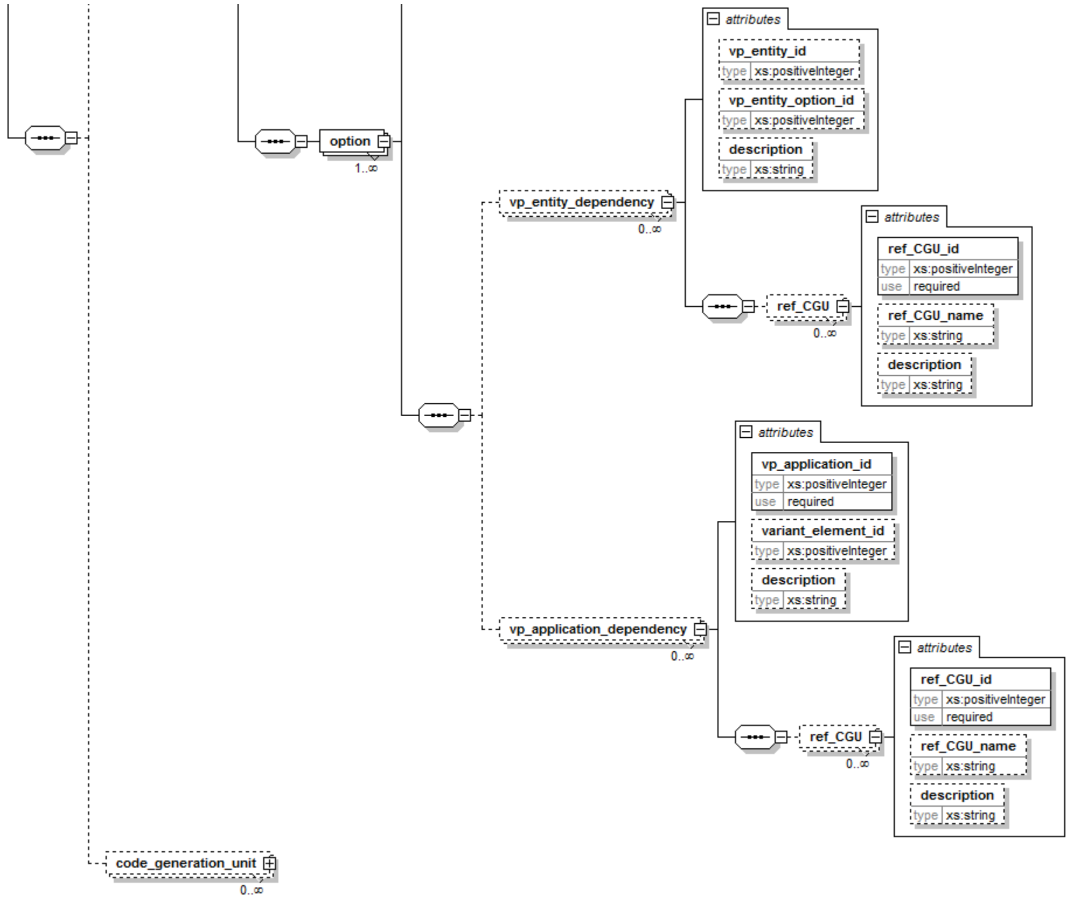
FeatureML Schema vp_entity Element Part 1.



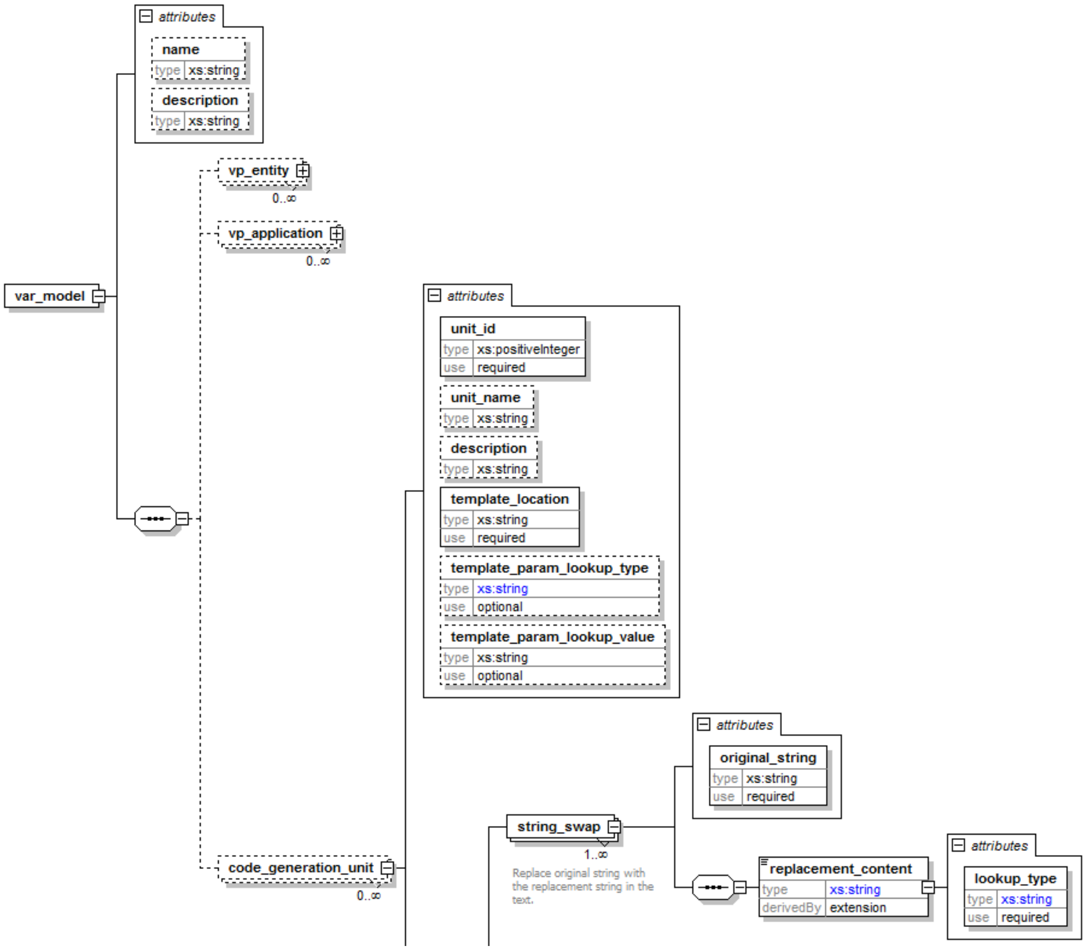
FeatureML Schema vp_entity Element Part 2.



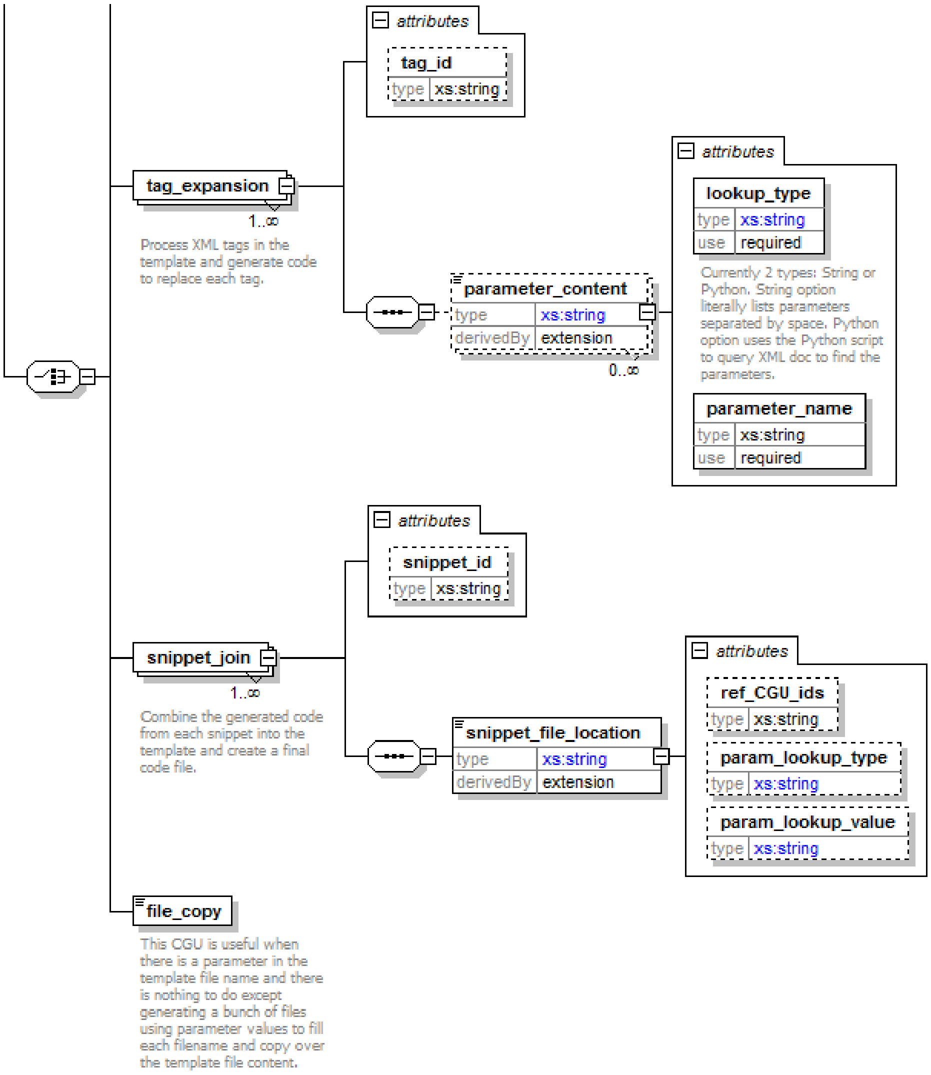
FeatureML Schema vp_application Element Part 1.



FeatureML Schema vp_application Element Part 2.



FeatureML Schema code_generation_unit Element Part 1.



Process XML tags in the template and generate code to replace each tag.

Currently 2 types: String or Python. String option literally lists parameters separated by space. Python option uses the Python script to query XML doc to find the parameters.

Combine the generated code from each snippet into the template and create a final code file.

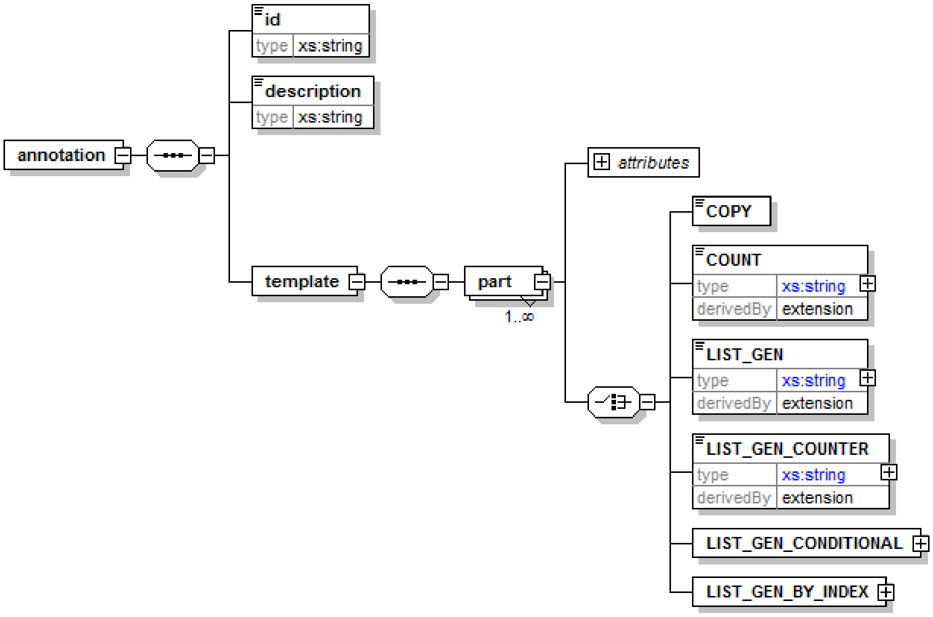
This CGU is useful when there is a parameter in the template file name and there is nothing to do except generating a bunch of files using parameter values to fill each filename and copy over the template file content.

FeatureML Schema code_generation_unit Element Part 2.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2008 sp1 (http://www.altova.com) by SOHO (SOHO) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:element name="var_model">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="vp_entity" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="option" maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>options to choose from for a vp</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="dependency" minOccurs="0" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:attribute name="dep_name" type="xs:string" use="required"/>
                        <xs:attribute name="vp_id" type="xs:positiveInteger" use="required"/>
                        <xs:attribute name="vp_name" type="xs:string"/>
                        <xs:attribute name="vp_option_id" type="xs:positiveInteger" use="required"/>
                        <xs:attribute name="dep_type" type="xs:string"/>
                        <xs:attribute name="dep_explanation" type="xs:string"/>
                      </xs:complexType>
                    </xs:element>
                    <xs:choice>
                      <xs:element name="enum_value">
                        <xs:complexType>
                          <xs:simpleContent>
                            <xs:extension base="xs:string">
                              <xs:attribute name="type">
                                <xs:simpleType>
                                  <xs:restriction base="param_typeType"/>
                                </xs:simpleType>
                              </xs:attribute>
                            </xs:extension>
                          </xs:simpleContent>
                        </xs:complexType>
                      </xs:element>
                      <xs:element name="scope_value">
                        <xs:complexType>
                          <xs:simpleContent>
                            <xs:extension base="xs:string">
                              <xs:attribute name="type" type="param_typeType"/>
                            </xs:extension>
                          </xs:simpleContent>
                        </xs:complexType>
                      </xs:element>
                      <xs:element name="sub_option" maxOccurs="unbounded">
                        <xs:complexType>
                          <xs:sequence>
                            <xs:element name="ref_key_name" type="xs:string" minOccurs="0" maxOccurs="unbounded">
                              <xs:annotation>
                                <xs:documentation>use this element to model foreign key reference</xs:documentation>
                              </xs:annotation>
                            </xs:element>
                  </xs:sequence>
                  <xs:attribute name="vp_entity_id" type="xs:positiveInteger"/>
                  <xs:attribute name="vp_entity_name" type="xs:string"/>
                  <xs:attribute name="vp_entity_option_id" type="xs:positiveInteger"/>
                </xs:complexType>
              </xs:element>
              <xs:element name="extended_option" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="additional_sub_option" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:attribute name="vp_entity_id" type="xs:positiveInteger"/>
                        <xs:attribute name="vp_entity_option_id" type="xs:positiveInteger"/>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="parent_vp_entity_id" type="xs:positiveInteger"/>
                  <xs:attribute name="parent_vp_entity_name" type="xs:string"/>
                  <xs:attribute name="parent_vp_option_id" type="xs:positiveInteger"/>
                </xs:complexType>
              </xs:element>
            </xs:choice>
          </xs:sequence>
          <xs:attribute name="op_id" type="xs:positiveInteger" use="required"/>
          <xs:attribute name="op_name" type="xs:string"/>
          <xs:attribute name="description" type="xs:string"/>
          <xs:attribute name="instances" type="xs:string">
            <xs:annotation>
              <xs:documentation>a list of instances where the vp is used</xs:documentation>
            </xs:annotation>
          </xs:attribute>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="vp_id" type="xs:positiveInteger" use="required"/>
    <xs:attribute name="vp_name" type="xs:string" use="required"/>
    <xs:attribute name="description" type="xs:string"/>
    <xs:attribute name="option_cardinality" type="xs:string"/>
  </xs:complexType>
</xs:element>
<xs:element name="vp_application" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="option" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="vp_entity_dependency" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="ref_CGU" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:attribute name="ref_CGU_id" type="xs:positiveInteger" use="required"/>
                      <xs:attribute name="ref_CGU_name" type="xs:string"/>
                      <xs:attribute name="description" type="xs:string"/>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
                <xs:attribute name="vp_entity_id" type="xs:positiveInteger"/>
                <xs:attribute name="vp_entity_option_id" type="xs:positiveInteger"/>
                <xs:attribute name="description" type="xs:string"/>
              </xs:complexType>
            </xs:element>
            <xs:element name="vp_application_dependency" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="ref_CGU" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:attribute name="ref_CGU_id" type="xs:positiveInteger" use="required"/>
                      <xs:attribute name="ref_CGU_name" type="xs:string"/>
                      <xs:attribute name="description" type="xs:string"/>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
                <xs:attribute name="vp_application_id" type="xs:positiveInteger" use="required"/>
                <xs:attribute name="variant_element_id" type="xs:positiveInteger"/>
                <xs:attribute name="description" type="xs:string"/>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
          <xs:attribute name="op_id" type="xs:positiveInteger" use="required"/>
          <xs:attribute name="op_name" type="xs:string"/>
          <xs:attribute name="description" type="xs:string"/>
          <xs:attribute name="instances" type="xs:string"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="vp_id" type="xs:positiveInteger" use="required"/>
    <xs:attribute name="vp_name" type="xs:string" use="required"/>
    <xs:attribute name="option_cadinality" type="xs:string"/>
    <xs:attribute name="description" type="xs:string"/>
  </xs:complexType>
</xs:element>
<xs:element name="code_generation_unit" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:choice>
      <xs:element name="string_swap" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>Replace original string with the replacement string in the text.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="replacement_content">
              <xs:complexType>
                <xs:simpleContent>
                  <xs:extension base="xs:string">
                    <xs:attribute name="lookup_type" use="required">
                      <xs:simpleType>
                        <xs:restriction base="xs:string">
                          <xs:enumeration value="String"/>
                          <xs:enumeration value="Function"/>
                          <xs:enumeration value="Dictionary"/>
                        </xs:restriction>
                      </xs:simpleType>
                    </xs:attribute>
                  </xs:extension>
                </xs:simpleContent>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
          <xs:attribute name="original_string" type="xs:string" use="required"/>
        </xs:complexType>
       </xs:element>
       <xs:element name="tag_expansion" maxOccurs="unbounded">
        <xs:annotation>
         <xs:documentation>Process XML tags in the template and generate code to replace each
tag.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
         <xs:sequence>
          <xs:element name="parameter_content" minOccurs="0" maxOccurs="unbounded">
           <xs:complexType>
            <xs:simpleContent>
             <xs:extension base="xs:string">
              <xs:attribute name="lookup_type" use="required">
               <xs:annotation>
                <xs:documentation>Currently 2 types: String or Python. String option literally lists parameters
separated by space. Python option uses the Python script to query XML doc to find the
parameters.</xs:documentation>
               </xs:annotation>
               <xs:simpleType>
                <xs:restriction base="xs:string">
                 <xs:enumeration value="Function"/>
                 <xs:enumeration value="Dictionary"/>
                 <xs:enumeration value="String"/>
                </xs:restriction>
               </xs:simpleType>
              </xs:attribute>
              <xs:attribute name="parameter_name" type="xs:string" use="required"/>
             </xs:extension>
            </xs:simpleContent>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
         <xs:attribute name="tag_id" type="xs:string"/>
        </xs:complexType>
       </xs:element>
       <xs:element name="snippet_join" maxOccurs="unbounded">
        <xs:annotation>
         <xs:documentation>Combine the generated code from each snippet into the template and create a final
code file.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
         <xs:sequence>
          <xs:element name="snippet_file_location">
           <xs:complexType>
            <xs:simpleContent>
             <xs:extension base="xs:string">
              <xs:attribute name="ref_CGU_ids" type="xs:string"/>
              <xs:attribute name="param_lookup_type">
               <xs:simpleType>
                <xs:restriction base="xs:string">
                 <xs:enumeration value="Function"/>
                 <xs:enumeration value="Dictionary"/>
                 <xs:enumeration value="String"/>
                </xs:restriction>
               </xs:simpleType>
              </xs:attribute>
              <xs:attribute name="param_lookup_value">
               <xs:simpleType>
                <xs:restriction base="xs:string"/>
```

```
                              </xs:simpleType>
                            </xs:attribute>
                          </xs:extension>
                        </xs:simpleContent>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="snippet_id" type="xs:string"/>
                </xs:complexType>
              </xs:element>
              <xs:element name="file_copy">
                <xs:annotation>
                  <xs:documentation>This CGU is useful when there is a parameter in the template file name and there is
nothing to do except generating a bunch of files using parameter values to fill each filename and copy over the
template file content.</xs:documentation>
                </xs:annotation>
              </xs:element>
            </xs:choice>
            <xs:attribute name="unit_id" type="xs:positiveInteger" use="required"/>
            <xs:attribute name="unit_name" type="xs:string"/>
            <xs:attribute name="description" type="xs:string"/>
            <xs:attribute name="template_location" type="xs:string" use="required"/>
            <xs:attribute name="template_param_lookup_type" use="optional">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="Function"/>
                  <xs:enumeration value="Dictionary"/>
                  <xs:enumeration value="String"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="template_param_lookup_value" type="xs:string" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string"/>
      <xs:attribute name="description" type="xs:string"/>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="parmType">
    <xs:attribute name="param_id" type="xs:nonNegativeInteger"/>
    <xs:attribute name="param_name" type="xs:string"/>
    <xs:attribute name="param_type" type="param_typeType"/>
    <xs:attribute name="param_value" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="single_step_activityType">
    <xs:sequence>
      <xs:element name="input_parm" type="parmType" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="output_parm" type="parmType" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="next_step" minOccurs="0">
        <xs:complexType>
          <xs:attribute name="vp_activity_id" type="xs:positiveInteger"/>
          <xs:attribute name="vp_activity_option_id" type="xs:positiveInteger"/>
          <xs:attribute name="step_id" type="xs:positiveInteger">
            <xs:annotation>
              <xs:documentation>if no vp_activity_id and vp_activity_option_id are specified, the step_id refers to
activity_step in current option</xs:documentation>
            </xs:annotation>
          </xs:attribute>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="content" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="activity_refType">
    <xs:attribute name="vp_activity_id" type="xs:positiveInteger"/>
    <xs:attribute name="vp_activity_option_id" type="xs:positiveInteger"/>
    <xs:attribute name="ref_description" type="xs:string"/>
    <xs:attribute name="input_param_match" type="xs:string">
      <xs:annotation>
        <xs:documentation>this attribute matches the current activity's input parameters with the referenced activity
input parameters</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="output_param_match" type="xs:string">
      <xs:annotation>
        <xs:documentation>this attribute matches the current activity's output parameters with the referenced activity
output parameters</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:simpleType name="param_typeType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="short"/>
      <xs:enumeration value="int"/>
      <xs:enumeration value="long"/>
      <xs:enumeration value="float"/>
      <xs:enumeration value="double"/>
      <xs:enumeration value="boolean"/>
      <xs:enumeration value="char"/>
      <xs:enumeration value="String"/>
      <xs:enumeration value="Object"/>
      <xs:enumeration value="BigDecimal"/>
      <xs:enumeration value="Date"/>
      <xs:enumeration value="Blob"/>
      <xs:enumeration value="Clob"/>
      <xs:enumeration value="Set"/>
      <xs:enumeration value="List"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

APPENDIX D: Template Language XML Schema (Graphical and Textual)



Appendix 1. Template Language Schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xs:element name="annotation">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="id" type="xs:string"/>
    <xs:element name="description" type="xs:string"/>
    <xs:element name="template">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="part" maxOccurs="unbounded">
        <xs:complexType>
         <xs:choice>
          <xs:element name="COPY"/>
          <xs:element name="COUNT">
           <xs:complexType>
            <xs:simpleContent>
             <xs:extension base="xs:string">
              <xs:attribute name="parameter" type="xs:string" use="required"/>
              <xs:attribute name="start_value" type="xs:int" use="optional"/>
             </xs:extension>
            </xs:simpleContent>
           </xs:complexType>
          </xs:element>
          <xs:element name="LIST_GEN">
           <xs:complexType>
            <xs:simpleContent>
             <xs:extension base="xs:string">
              <xs:attribute name="separator" type="xs:string" use="required"/>
              <xs:attribute name="separator_after_last_item" type="xs:boolean" use="required"/>
              <xs:attribute name="parameters" type="xs:string" use="required"/>
             </xs:extension>
            </xs:simpleContent>
           </xs:complexType>
          </xs:element>
          <xs:element name="LIST_GEN_COUNTER">
           <xs:complexType>
            <xs:simpleContent>
             <xs:extension base="xs:string">
              <xs:attribute name="separator" type="xs:string" use="required"/>
              <xs:attribute name="separator_after_last_item" type="xs:boolean" use="required"/>
              <xs:attribute name="parameters" type="xs:string" use="required"/>
              <xs:attribute name="counter_start_value" type="xs:int" use="optional"/>
             </xs:extension>
            </xs:simpleContent>
           </xs:complexType>
          </xs:element>
          <xs:element name="LIST_GEN_CONDITIONAL">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="conditional_content" maxOccurs="unbounded">
              <xs:complexType>
               <xs:simpleContent>
                <xs:extension base="xs:string">
                 <xs:attribute name="condition" type="xs:string" use="required"/>
                 <xs:attribute name="separator" type="xs:string" use="required"/>
```

```
                                </xs:extension>
                              </xs:simpleContent>
                            </xs:complexType>
                          </xs:element>
                          <xs:element name="default_content">
                            <xs:complexType>
                              <xs:simpleContent>
                                <xs:extension base="xs:string">
                                  <xs:attribute name="separator" type="xs:string" use="required"/>
                                </xs:extension>
                              </xs:simpleContent>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                        <xs:attribute name="separator_after_last_item" type="xs:boolean" use="required"/>
                        <xs:attribute name="parameters" type="xs:string" use="required"/>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="LIST_GEN_BY_INDEX">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="content" maxOccurs="unbounded">
                            <xs:complexType>
                              <xs:simpleContent>
                                <xs:extension base="xs:string">
                                  <xs:attribute name="index_range" type="xs:string" use="required"/>
                                  <xs:attribute name="separator" type="xs:string" use="required"/>
                                </xs:extension>
                              </xs:simpleContent>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                        <xs:attribute name="separator_after_last_item" type="xs:boolean" use="required"/>
                        <xs:attribute name="parameters" type="xs:string" use="required"/>
                      </xs:complexType>
                    </xs:element>
                  </xs:choice>
                  <xs:attribute name="separator_to_next_part" type="xs:string"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

APPENDIX E: Online Exam System Feature Model Example in FeatureML

```
<?xml version="1.0" encoding="UTF-8"?>

<var_model xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="E:\slugforge_repos\doc\online_exam_design\xml\var_model_v5.xsd"
name="online_exam" description="A feature model description for the online exam system">
  <!-- START OF VP ENTITIES -->

  <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="Long type">
      <enum_value type="String">java.lang.Long</enum_value>
    </option>
    <option op_id="3" description="Short type">
      <enum_value type="String">java.lang.Short</enum_value>
    </option>
    <option op_id="4" description="Character type">
      <enum_value type="String">java.lang.Character</enum_value>
    </option>
    <option op_id="5" description="BigDecimal type">
      <enum_value type="String">java.math.BigDecimal</enum_value>
    </option>
    <option op_id="6" description="Float type">
      <enum_value type="String">java.lang.Float</enum_value>
    </option>
    <option op_id="7" description="Double type">
      <enum_value type="String">java.lang.Double</enum_value>
    </option>
    <option op_id="8" description="Boolean type">
      <enum_value type="String">java.lang.Boolean</enum_value>
    </option>
    <option op_id="9" description="String type">
      <enum_value type="String">java.lang.String</enum_value>
    </option>
    <option op_id="10" description="Date type">
      <enum_value type="String">java.util.Date</enum_value>
    </option>
    <option op_id="13" description="Object type">
      <enum_value type="String">java.lang.Object</enum_value>
    </option>
    <option op_id="14" description="Question type">
      <enum_value type="String">edu.ucsc.cse.exam.datamodel.Question</enum_value>
    </option>
    <option op_id="15" description="ExamAnswerPerStudent type">
      <enum_value type="String">edu.ucsc.cse.exam.datamodel.ExamAnswerPerStudent</enum_value>
    </option>
    <option op_id="16" description="Exam type">
      <enum_value type="String">edu.ucsc.cse.exam.datamodel.Exam</enum_value>
    </option>
  </vp_entity>
  <vp_entity vp_id="2" vp_name="value_length_limit" description="length limit for user input parameters">
    <option op_id="1" description="no more than 16">
      <enum_value type="String">0..16</enum_value>
    </option>
    <option op_id="2" description="no more than 32">
      <enum_value type="String">0..32</enum_value>
    </option>
    <option op_id="3" description="no more than 64">
      <enum_value type="String">0..64</enum_value>
    </option>
    <option op_id="4" description="no more than 128">
      <enum_value type="String">0..128</enum_value>
    </option>
    <option op_id="5" description="no more than 12">
      <enum_value type="String">0..12</enum_value>
    </option>
    <option op_id="6" description="no more than 9">
      <enum_value type="String">0..9</enum_value>
    </option>
    <option op_id="7" description="no more than 256">
      <enum_value type="String">0..256</enum_value>
    </option>
    <option op_id="8" description="no more than 512">
      <enum_value type="String">0..512</enum_value>
    </option>
    <option op_id="9" description="no more than 4">
      <enum_value type="String">0..4</enum_value>
    </option>
    <option op_id="10" description="no more than 8">
      <enum_value type="String">0..8</enum_value>
    </option>
    <option op_id="11" description="exactly 16">
      <enum_value type="int">16</enum_value>
    </option>
    <option op_id="12" description="exactly 32">
      <enum_value type="int">32</enum_value>
    </option>
    <option op_id="13" description="exactly 64">
      <enum_value type="int">64</enum_value>
    </option>
    <option op_id="14" description="exactly 128">
      <enum_value type="int">128</enum_value>
    </option>
    <option op_id="15" description="between 8 and 16, including 8, but not 16">
      <enum_value type="String">8..16</enum_value>
    </option>
  </vp_entity>
  <vp_entity vp_id="3" vp_name="GUI_widget" description="available GUI widgets">
    <option op_id="1" op_name="Label" description="Label displays some given information">
      <enum_value type="String">wicket.markup.html.basic.Label</enum_value>
    </option>
    <option op_id="2" op_name="RequiredTextField" description="RequiredTextField reports errors if user does not
enter anything">
      <enum_value type="String">wicket.markup.html.form.RequiredTextField</enum_value>
    </option>
    <option op_id="3" op_name="PasswordTextField" description="PasswordTextField shows * instead of clear text
user enters">
      <enum_value type="String">wicket.markup.html.form.PasswordTextField</enum_value>
    </option>
    <option op_id="4" op_name="PasswordConfirmTextField" description="PasswordConfirmTextField matches 2nd
time input with the first time password input. It uses the same class as PasswordTextField, but it has an extra
EqualPasswordInputValidator">
      <enum_value type="String">wicket.markup.html.form.PasswordTextField</enum_value>
    </option>
    <option op_id="5" op_name="DropDownChoice" description="DropDownChoice shows a list of selectable items">
      <enum_value type="String">wicket.markup.html.form.DropDownChoice</enum_value>
    </option>
    <option op_id="6" op_name="CheckBox" description="CheckBox records boolean selections">
      <enum_value type="String">wicket.markup.html.form.CheckBox</enum_value>
    </option>
    <option op_id="7" op_name="EmailTextField" description="EmailTextField allows user to enter email address. It
uses RequiredTextField with an additional EmailAddressPatternValidator">
      <enum_value type="String">wicket.markup.html.form.RequiredTextField</enum_value>
    </option>
    <option op_id="8" op_name="TextFilteredPropertyColumn" description="TextFilteredPropertyColumn is used in
tables that support filters on each column">
      <enum_value
type="String">wicket.extensions.markup.html.repeater.data.table.filter.TextFilteredPropertyColumn</enum_value>
    </option>
    <option op_id="9" op_name="TextArea" description="TextArea allows input of a big chunk of text">
      <enum_value type="String">wicket.markup.html.form.TextArea</enum_value>
    </option>
    <option op_id="20" op_name="MultipageSortableFilterTable" description="multipage sortable filter table for large
data set display">
      <enum_value type="String">MultipageSortableFilterTable</enum_value>
    </option>
    <option op_id="21" op_name="MultipageNormalTable" description="multipage normal table">
      <enum_value type="String">MultipageNormalTable</enum_value>
    </option>
  </vp_entity>
  <vp_entity vp_id="4" vp_name="collection_type_Set" description="implementation for the Set type">
    <option op_id="1" description="HashSet">
      <enum_value type="String">HashSet</enum_value>
    </option>
  </vp_entity>
  <vp_entity vp_id="5" vp_name="collection_type_List" description="implementation for the List type">
    <option op_id="1" description="ArrayList">
      <enum_value type="String">ArrayList</enum_value>
    </option>
  </vp_entity>
  <vp_entity vp_id="6" vp_name="special_value_pattern" description="a pattern that a user input has to conform to
">
    <option op_id="1" description="US phone number pattern">
      <enum_value type="String">USPhoneNumber</enum_value>
    </option>
    <option op_id="2" description="email address pattern">
      <enum_value type="String">EmailAddress</enum_value>
    </option>
  </vp_entity>
  <vp_entity vp_id="7" vp_name="sql_value_types" description="sql value types used in the database definition">
    <option op_id="1" description="VARCHAR type">
      <enum_value type="String">VARCHAR</enum_value>
    </option>
    <option op_id="2" description="TIMESTAMP type">
      <enum_value type="String">TIMESTAMP</enum_value>
    </option>
    <option op_id="3" description="BIGINT type">
      <enum_value type="String">BIGINT</enum_value>
    </option>
    <option op_id="4" description="TINYINT type">
      <enum_value type="String">TINYINT</enum_value>
    </option>
    <option op_id="5" description="BIT type">
      <enum_value type="String">BIT</enum_value>
    </option>
    <option op_id="6" description="INT type">
      <enum_value type="String">INT</enum_value>
    </option>
  </vp_entity>
  <vp_entity vp_id="8" vp_name="is_property_required" description=
  "boolean value TRUE or FALSE to tell if a property is required or not, this affects the GUI required input validator">
    <option op_id="1" description="required property">
      <enum_value type="String">true</enum_value>
    </option>
    <option op_id="2" description="not required property">
      <enum_value type="String">false</enum_value>
    </option>
  </vp_entity>

  <!-- add more primitive value related entities here -->

  <vp_entity vp_id="30" vp_name="User_types" description="user types supported in the system">
    <option op_id="1" description="system1 has two user types: teacher and student">
      <sub_option vp_entity_id="42" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="43" vp_entity_option_id="1"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="41" vp_name="User_props" description="properties for a general user type">
    <option op_id="2" op_name="commericial_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"/>
      <sub_option vp_entity_id="61" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="62" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="63" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="64" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="65" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="66" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="95" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="96" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="97" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="98" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="42" vp_name="Teacher_props" description="properties for teacher type">
    <option op_id="2" op_name="commerical_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"/>
        <additional_sub_option vp_entity_id="72" vp_entity_option_id="1"/>
        <additional_sub_option vp_entity_id="109" vp_entity_option_id="1"/>
        <additional_sub_option vp_entity_id="110" vp_entity_option_id="1"/>
      </extended_option>
    </option>
  </vp_entity>
  <vp_entity vp_id="43" vp_name="Student_props" description="properties for student type">
```

```
    <option op_id="2" op_name="commercial_system_creditcard_extend" description="extending the
commercial_system_creditcard user entity by adding department, major, mailingaddress and ista">
      <extended_option parent_vp_entity_id="41" parent_vp_option_id="1">
        <additional_sub_option vp_entity_id="67" vp_entity_option_id="1"/>
        <additional_sub_option vp_entity_id="68" vp_entity_option_id="1"/>
        <additional_sub_option vp_entity_id="69" vp_entity_option_id="1"/>
        <additional_sub_option vp_entity_id="70" vp_entity_option_id="1"/>
        <additional_sub_option vp_entity_id="109" vp_entity_option_id="1"/>
        <additional_sub_option vp_entity_id="110" vp_entity_option_id="1"/>
      </extended_option>
    </option>
  </vp_entity>
  <!-- add more user props types here -->
  <vp_entity vp_id="46" vp_name="Question_types" description="question types supported in the system">
    <option op_id="1" op_name="base_system" description="question types supported in the base system">
      <sub_option vp_entity_id="48" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="49" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="47" vp_name="Question_props" description="properties for question super type">
    <option op_id="1" op_name="base_system" description="properties for question type in the base system">
      <sub_option vp_entity_id="101" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="103" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="73" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="74" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="75" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="76" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="77" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="121" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="123" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="127" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="127" vp_entity_option_id="2"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="48" vp_name="MC_props" description="properties for MC, multiple choice, questions">
    <option op_id="1" op_name="base_system" description="properties for MC question type in the base system
    ">
      <extended_option parent_vp_entity_id="47" parent_vp_option_id="1">
        <additional_sub_option vp_entity_id="78" vp_entity_option_id="1"/>
        <additional_sub_option vp_entity_id="79" vp_entity_option_id="1"/>
        <additional_sub_option vp_entity_id="80" vp_entity_option_id="1"/>
      </extended_option>
    </option>
  </vp_entity>
  <vp_entity vp_id="49" vp_name="TF_props" description="properties for TF, true or false, questions">
    <option op_id="1" op_name="base_system" description="properties for TF question type in the base system">
      <extended_option parent_vp_entity_id="47" parent_vp_option_id="1">
        <additional_sub_option vp_entity_id="78" vp_entity_option_id="1"/>
        <additional_sub_option vp_entity_id="81" vp_entity_option_id="1"/>
      </extended_option>
    </option>
  </vp_entity>
  <!-- add more question props types here -->
  <vp_entity vp_id="54" vp_name="Exam_types" description="exam types supported in the system">
    <option op_id="1" op_name="base" description="exam types supported in the base system">
      <sub_option vp_entity_id="55" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="55" vp_name="Exam_props" description="properties for the exam">
    <option op_id="1" op_name="default" description="default exam properties">
```

```
      <sub_option vp_entity_id="102" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="103" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="66" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="73" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="74" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="75" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="77" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="99" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="120" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="122" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="127" vp_entity_option_id="3"/>
      <sub_option vp_entity_id="127" vp_entity_option_id="4"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="61" vp_name="username" description="user name for system login">
    <option op_id="1" op_name="default" description="use string to store, max length 16, VARCHAR SQL type, Label
widget to present">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
    <option op_id="2" op_name="input" description="use string to store, max length 16, VARCHAR SQL type,
RequiredTextField widget to collect input value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
    </option>
    <option op_id="3" op_name="filteredtable" description="use string to store, max length 16, VARCHAR SQL type,
filtered table column widget to present">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="62" vp_name="password" description="password for a user">
    <option op_id="1" op_name="default" description="use string to store, max length 16, use Label widget to present
value if needed">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
    <option op_id="2" op_name="input" description="user PasswordTextField widget to collect input value, string to
store, max length 16">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="3"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="63" vp_name="firstname" description="first name for a person">
    <option op_id="1" op_name="default" description="use string to store, max length 32, VARCHAR SQL type, Label
widget to present value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
    <option op_id="2" op_name="input" description="use string to store, max length 32, VARCHAR SQL type, use
RequiredTextField to collect input value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
    </option>
    <option op_id="3" op_name="filteredtable" description="use string to store, max length 32, VARCHAR SQL type,
filtered table column to present value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="64" vp_name="lastname" description="last name for a person">
    <option op_id="1" op_name="default" description="use string to store, max length 32, VARCHAR SQL type, Label
widget to present value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
    <option op_id="2" op_name="input" description="use string to store, max length 32, VARCHAR SQL type,
RequiredTextField to collect input value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
    </option>
    <option op_id="3" op_name="filteredtable" description="use string to store, max length 32, VARCHAR SQL type,
filtered table column widget to present value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="65" vp_name="email" description="email address">
    <option op_id="1" op_name="default" description="use string to store, max length 64, VARCHAR SQL type, email
string pattern, use Label widget to present">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="3"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="6" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
    <option op_id="2" op_name="input" description="use string to store, max length 64, use RequiredTextField to
collect input value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="3"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
    </option>
    <option op_id="3" op_name="filteredtable" description="use string to store, max length 64, VARCHAR SQL type,
email string pattern, use filtered table column widget to present">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="3"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
      <sub_option vp_entity_id="6" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="66" vp_name="classcode" description="class code">
    <option op_id="1" op_name="default" description="use string to store, max length 16, VARCHAR SQL type, use
Label for presentation">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
    <option op_id="2" op_name="input" description="use String to store, max length 16, use DropDownChoice widget
to collect input">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="5"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
    </option>
    <option op_id="3" op_name="filteredTableOutput" description="properties to show up in a filtered table">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="67" vp_name="department" description="department name">
    <option op_id="1" op_name="default" description="use string to store, max length 32, VARCHAR SQL type and
Label widget to prsent value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
    <option op_id="2" op_name="input" description="use string to store, max length 32, use DropDownChoice to
collect input value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="5"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
    </option>
    <option op_id="3" op_name="filteredtable" description="use string to store, max length 32, VARCHAR SQL type
and filtered table column widget to prsent value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="68" vp_name="major" description="majors for the students">
    <option op_id="1" op_name="default" description="use string to store, max length 32, use Label to present value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
    <option op_id="2" op_name="input" description="use string to store, max length 32, use DropDownChoice to collect input value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="5"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
    </option>
    <option op_id="3" op_name="filteredtable" description="use string to store, max length 32, use filtered table column to present value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="69" vp_name="mailingaddress" description="mailing address">
    <option op_id="1" op_name="default" description="use string to store, max length 128, use Label to present value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="4"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
    <option op_id="2" op_name="input" description="use string to store, max length 128, use RequiredTextField to collect input value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="4"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="70" vp_name="ista" description="marker for the status of a student, to tell if it is a TA or not">
    <option op_id="1" op_name="default" description="a boolean value, BIT SQL type, Label to present value">
      <sub_option vp_entity_id="1" vp_entity_option_id="8"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="5"/>
    </option>
    <option op_id="2" op_name="input" description="a boolean value, BIT SQL type, CheckBox widget to collect input">
      <sub_option vp_entity_id="1" vp_entity_option_id="8"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="6"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="5"/>
      <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
    </option>
    <option op_id="3" op_name="filteredtable" description="a boolean value, BIT SQL type, filtered table column to present value">
      <sub_option vp_entity_id="1" vp_entity_option_id="8"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="5"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="71" vp_name="office" description="office address, e.g. building number and office number">
    <option op_id="1" op_name="default" description="use string to store, max length 32">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
    <option op_id="2" op_name="input" description="use string to store, max length 32, use RequiredTextField to
collect input value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
    </option>
    <option op_id="3" op_name="filteredtable" description="use string to store, max length 32, use filtered table
column to present">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="72" vp_name="phonenum" description="phone number">
    <option op_id="1" op_name="default" description="use string to store, max length 12, use the phone number
pattern">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="5"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="6" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
    <option op_id="2" op_name="input" description="use string to store, max length 12, use RequiredTextField to
collect input value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="5"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
    </option>
    <option op_id="3" op_name="filteredtable" description="use string to store, max length 12, use the phone number
pattern, use filtered table column to present">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="5"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
      <sub_option vp_entity_id="6" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="73" vp_name="creationdate" description="creation date">
    <option op_id="1" op_name="default" description="use date to store, TIMESTAMP SQL type, Label widget to
present">
      <sub_option vp_entity_id="1" vp_entity_option_id="10"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
    </option>
    <option op_id="2" op_name="filteredTableOuput" description="used in a filted table">
      <sub_option vp_entity_id="1" vp_entity_option_id="10"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="74" vp_name="difficulty" description="difficulty level">
    <option op_id="1" op_name="default" description="use short to store, use Label to present">
```

```
   <sub_option vp_entity_id="1" vp_entity_option_id="3"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="4"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
  </option>
  <option op_id="2" op_name="input" description="settings to get input, use short to store and DropDownChoice to
collect">
   <sub_option vp_entity_id="1" vp_entity_option_id="3"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="4"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="5"/>
   <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
  </option>
  <option op_id="3" op_name="filteredTable" description="used in a filtered table">
   <sub_option vp_entity_id="1" vp_entity_option_id="3"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="4"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
  </option>
 </vp_entity>
 <vp_entity vp_id="75" vp_name="ispublic" description="whether a resource is public to use or only available to its
creator">
  <option op_id="1" op_name="default" description="a boolean value, use Label to present value">
   <sub_option vp_entity_id="1" vp_entity_option_id="8"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="5"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
  </option>
  <option op_id="2" op_name="input" description="a boolean value, use CheckBox to collect value">
   <sub_option vp_entity_id="1" vp_entity_option_id="8"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="5"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="6"/>
   <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
  </option>
  <option op_id="3" op_name="filteredTable" description="used in a filtered table">
   <sub_option vp_entity_id="1" vp_entity_option_id="8"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="5"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
  </option>
 </vp_entity>
 <vp_entity vp_id="76" vp_name="isactive" description="whether a resource is active">
  <option op_id="1" op_name="default" description="a boolean value, use Label to present its value">
   <sub_option vp_entity_id="1" vp_entity_option_id="8"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="5"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
  </option>
  <option op_id="2" op_name="filteredTable" description="a boolean value, used in a filtered table">
   <sub_option vp_entity_id="1" vp_entity_option_id="8"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="5"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
  </option>
 </vp_entity>
 <vp_entity vp_id="77" vp_name="tags" description="tags to classify a resource">
  <option op_id="1" op_name="default" description="use string to store, max length 64, use Label to present value
  ">
   <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
   <sub_option vp_entity_id="2" vp_entity_option_id="3"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
  </option>
  <option op_id="2" op_name="input" description="use string to store, max length 64, use RequiredTextField to
collect value">
   <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
   <sub_option vp_entity_id="2" vp_entity_option_id="3"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="2"/>
   <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
  </option>
  <option op_id="3" op_name="filteredTable" description="used in a filtered table">
   <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
   <sub_option vp_entity_id="2" vp_entity_option_id="3"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
  </option>
 </vp_entity>
 <vp_entity vp_id="78" vp_name="body" description="question body content">
  <option op_id="1" op_name="default" description="use string to store, max length 512, use Label to display">
   <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
   <sub_option vp_entity_id="2" vp_entity_option_id="8"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
  </option>
  <option op_id="2" op_name="input" description="use string to store, max length 512, use TextArea to collect
input">
   <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
   <sub_option vp_entity_id="2" vp_entity_option_id="8"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="9"/>
   <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
  </option>
  <option op_id="3" op_name="filteredTable" description="use string to store, max length 512, used in a filtered
table column">
   <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
   <sub_option vp_entity_id="2" vp_entity_option_id="8"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
  </option>
 </vp_entity>
 <vp_entity vp_id="79" vp_name="correctanswers" description="correct answers to the question">
  <option op_id="1" op_name="default" description="use string to store, max length 256, Label to display content
  ">
   <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
   <sub_option vp_entity_id="2" vp_entity_option_id="7"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
  </option>
  <option op_id="2" op_name="input" description="use string to store, max length 256, TextArea to collect input">
   <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
   <sub_option vp_entity_id="2" vp_entity_option_id="7"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="9"/>
   <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
  </option>
  <option op_id="3" op_name="filteredTable" description="use string to store, max length 256, used in a filtered
table column">
   <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
   <sub_option vp_entity_id="2" vp_entity_option_id="7"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
  </option>
 </vp_entity>
 <vp_entity vp_id="80" vp_name="wronganswers" description="wrong answers to the question">
  <option op_id="1" op_name="default" description="use string to store, max length 512">
   <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
   <sub_option vp_entity_id="2" vp_entity_option_id="8"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
  </option>
  <option op_id="2" op_name="input" description="use string to store, max length 256, TextArea to collect input">
   <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
   <sub_option vp_entity_id="2" vp_entity_option_id="7"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="9"/>
   <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
  </option>
  <option op_id="3" op_name="filteredTable" description="use string to store, max length 512, used in a filtered
table column">
   <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
   <sub_option vp_entity_id="2" vp_entity_option_id="8"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
  </option>
 </vp_entity>
 <vp_entity vp_id="81" vp_name="correctanswer" description="correct answer, TRUE or FALSE, used by True-
False question type">
  <option op_id="1" op_name="default" description="use boolean to store, Label to present the value">
   <sub_option vp_entity_id="1" vp_entity_option_id="8"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="5"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
  </option>
  <option op_id="2" op_name="input" description="use boolean to store, DropDownChoice to collect input value">
   <sub_option vp_entity_id="1" vp_entity_option_id="8"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="5"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="5"/>
   <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
  </option>
  <option op_id="3" op_name="filteredTable" description="use boolean to store, used in the filtered table column
  ">
   <sub_option vp_entity_id="1" vp_entity_option_id="8"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="5"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
  </option>
 </vp_entity>
 <vp_entity vp_id="91" vp_name="score" description="score information">
  <option op_id="1" op_name="default" description="use string to store, max length 4, display using Label widget
  ">
   <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
   <sub_option vp_entity_id="2" vp_entity_option_id="9"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
  </option>
 </vp_entity>
 <vp_entity vp_id="92" vp_name="duration" description="answering process duration for the question or exam">
  <option op_id="1" op_name="default" description="use int to store, use Label widget to display">
   <sub_option vp_entity_id="1" vp_entity_option_id="1"/>
   <sub_option vp_entity_id="7" vp_entity_option_id="6"/>
   <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
  </option>
 </vp_entity>
 <vp_entity vp_id="93" vp_name="question_types" description="question type">
  <option op_id="1" op_name="default" description="use string to store, max length 8, Label widget to present
value">
   <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
   <sub_option vp_entity_id="2" vp_entity_option_id="10"/>
```

```
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="94" vp_name="answer" description="question answer information">
    <option op_id="1" op_name="default" description="use string to store, max length 512, Label widget to present
value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="8"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="95" vp_name="creditcardnum" description="credit card number">
    <option op_id="1" op_name="default" description="use string to store, max length 16, Label widget to present
value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
    <option op_id="2" op_name="input" description="use string to store, max length 16, use RequiredTextField to
collect input value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="96" vp_name="creditcardtype" description="credit card type, e.g. AmericanExpress, Master,
Visa">
    <option op_id="1" op_name="default" description="use string to store, max length 16, Label widget to present
value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
    <option op_id="2" op_name="input" description="use string to store, max length 16, use DropDownChoice to
collect input value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="5"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="97" vp_name="creditcardexpdate" description="credit card expiration date">
    <option op_id="1" op_name="default" description="use date to store, TIMESTAMP SQL type, Label widget to
present value">
      <sub_option vp_entity_id="1" vp_entity_option_id="10"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="2"/>
    </option>
    <option op_id="2" op_name="input" description="use date to store, use RequiredTextField together with
DatePicker widget to collect input value">
      <sub_option vp_entity_id="1" vp_entity_option_id="10"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="98" vp_name="billingaddress" description="billing address">
    <option op_id="1" op_name="default" description="use string to store, max length 128, Label widget to present
value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="4"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
    <option op_id="2" op_name="input" description="use string to store, max length 128, use RequiredTextField to
collect input value">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="4"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="8" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="99" vp_name="state" description="state information for the exam, such as draft, released, closed,
etc.">
    <option op_id="1" op_name="default" description="use string to store, max length 16">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
    <option op_id="2" op_name="filteredTable" description="used in a filtered table">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="100" vp_name="userId" description="user id">
    <option op_id="1" description="use long type to store id">
      <sub_option vp_entity_id="1" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="6"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="11"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="12"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="3"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="101" vp_name="questionId" description="question id">
    <option op_id="1" description="use long type to store id">
      <sub_option vp_entity_id="1" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="7"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="13"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="14"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="3"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="102" vp_name="examId" description="exam id">
    <option op_id="1" description="use long to store id">
```

```
      <sub_option vp_entity_id="1" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="3" />
      <sub_option vp_entity_id="200" vp_entity_option_id="8"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="3"/>
    </option>
    <option op_id="2" op_name="filteredTableOupt" description="properties to show up in a filtered table">
      <sub_option vp_entity_id="1" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="3" />
      <sub_option vp_entity_id="200" vp_entity_option_id="8"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="3"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
    </option>
    <option op_id="3" op_name="normalTableOutput" description="props to show up in a normal table">
      <sub_option vp_entity_id="1" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="3" />
      <sub_option vp_entity_id="200" vp_entity_option_id="8"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="3"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="103" vp_name="authorId" description="author id">
    <option op_id="1" description="use long to store id">
      <sub_option vp_entity_id="1" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="3"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="1"/>
    </option>
    <option op_id="2" op_name="filteredTableOutput" description="properties to show up in a filtered table">
      <sub_option vp_entity_id="1" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="3"/>
      <sub_option vp_entity_id="3" vp_entity_option_id="8"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="104" vp_name="examAnswerPerStudentId" description="exam answer per student id">
    <option op_id="1" description="use long to store id">
      <sub_option vp_entity_id="1" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="4"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="3"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="105" vp_name="questionAnswerPerStudentId" description="question answer per student id
">
    <option op_id="1" description="use long to store id">
      <sub_option vp_entity_id="1" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="5"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="10"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="3"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="106" vp_name="studentId" description="student id">
    <option op_id="1" description="use long to store id">
      <sub_option vp_entity_id="1" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="3"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="107" vp_name="questiontype" description="question type like MC, TF, etc.">
    <option op_id="1" description="use string of length 8 to store question type">
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="10"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="108" vp_name="passwordconfirm" description="password confirmation stores the 2nd time
password input to match the 1st time password input">
    <option op_id="1" op_name="default" description="use string to store, max length 16" >
      <sub_option vp_entity_id="1" vp_entity_option_id="9"/>
      <sub_option vp_entity_id="2" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="6"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="11"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="12"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="13"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="14"/>
      <sub_option vp_entity_id="7" vp_entity_option_id="1"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="109" vp_name="editable_entity" description="this is a boolean value that tells whether an entity
is editable or not, which affects the existence of various system operations like create a new entity and edit an
existing entity">
    <option op_id="1" op_name="True" description="entity is editable">
      <enum_value type="String">True</enum_value>
    </option>
    <option op_id="2" op_name="False" description="entity is not editable">
      <enum_value type="String">False</enum_value>
    </option>
  </vp_entity>

  <vp_entity vp_id="110" vp_name="deletable_entity" description="this is a boolean value that tells whether an entity
can be deleted, its value affects the existence of system operations like delete entity">
    <option op_id="1" op_name="True" description="entity is deletable">
      <enum_value type="String">True</enum_value>
    </option>
    <option op_id="2" op_name="False" description="entity is not deletable">
      <enum_value type="String">False</enum_value>
    </option>
  </vp_entity>

  <vp_entity vp_id="120" vp_name="examAnswers_collection" description="a collection of exam answers">
    <option op_id="1" op_name="default">
      <sub_option vp_entity_id="4" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="151" vp_entity_option_id="1"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="121" vp_name="questionAnswers_collection" description="a collection of question answers">
    <option op_id="1" op_name="default">
      <sub_option vp_entity_id="4" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="152" vp_entity_option_id="1"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="122" vp_name="childQuestions_collection"  description="a collection of child questions for the
exam">
    <option op_id="1">
```

```
      <sub_option vp_entity_id="4" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="47" vp_entity_option_id="1"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="123" vp_name="parentExams_collection" description="a collection of parent exams for a
question">
    <option op_id="1">
      <sub_option vp_entity_id="4" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="55" vp_entity_option_id="1"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="124" vp_name="question" description="an Question object">
    <option op_id="1">
      <sub_option vp_entity_id="1" vp_entity_option_id="14"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="5"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="10"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="125" vp_name="examAnswer" description="an ExamAnswerPerStudent object">
    <option op_id="1">
      <sub_option vp_entity_id="1" vp_entity_option_id="15"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="5"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="10"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="126" vp_name="exam" description="an Exam object">
    <option op_id="1">
      <sub_option vp_entity_id="1" vp_entity_option_id="16"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="4"/>
      <sub_option vp_entity_id="200" vp_entity_option_id="9"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="127" vp_name="hibernate_mapping_settings" description="hibernate mapping settings for
association between entities">
    <option op_id="1" op_name="Question_questionAnswers" description="settings for the question answers set in
Question entity">
      <sub_option vp_entity_id="128" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="130" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="131" vp_entity_option_id="1"/>
    </option>
    <option op_id="2" op_name="Question_parentExams" description="settings for the exam set in Question entity">
      <sub_option vp_entity_id="128" vp_entity_option_id="3"/>
      <sub_option vp_entity_id="129" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="130" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="131" vp_entity_option_id="2"/>
    </option>
    <option op_id="3" op_name="Exam_examAnswers" description="settings for the exam answer set in Exam entity
    ">
      <sub_option vp_entity_id="128" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="130" vp_entity_option_id="3"/>
      <sub_option vp_entity_id="131" vp_entity_option_id="3"/>
    </option>
    <option op_id="4" op_name="Exam_childQuestions" description="settings for the child question set in Exam
entity">
      <sub_option vp_entity_id="128" vp_entity_option_id="3"/>
      <sub_option vp_entity_id="129" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="130" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="131" vp_entity_option_id="4"/>
    </option>
    <option op_id="5" op_name="ExamAnswerPerStudent_questionAnswers" description="settings for the question answers set in ExamAnswerPerStudent entity">
      <sub_option vp_entity_id="128" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="130" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="131" vp_entity_option_id="1"/>
    </option>
    <option op_id="6" op_name="ExamAnswerPerStudent_exam" description="settings for the exam in ExamAnswerPerStudent entity">
      <sub_option vp_entity_id="128" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="129" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="131" vp_entity_option_id="2"/>
    </option>
    <option op_id="7" op_name="QuestionAnswerPerStudent_question" description="settings for the question in QuestionAnswerPerStudent entity">
      <sub_option vp_entity_id="128" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="129" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="131" vp_entity_option_id="4"/>
    </option>
    <option op_id="8" op_name="QuestionAnswerPerStudent_examAnswer" description="settings for the exam answer in QuestionAnswerPerStudent entity">
      <sub_option vp_entity_id="128" vp_entity_option_id="2"/>
      <sub_option vp_entity_id="129" vp_entity_option_id="3"/>
      <sub_option vp_entity_id="131" vp_entity_option_id="3"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="128" vp_name="hibernate_mapping_type" description="different hibernate mapping types such as one-to-many, many-to-one, many-to-many">
    <option op_id="1" op_name="one-to-many">
      <enum_value type="String">one-to-many</enum_value>
    </option>
    <option op_id="2" op_name="many-to-one">
      <enum_value type="String">many-to-one</enum_value>
    </option>
    <option op_id="3" op_name="many-to-many">
      <enum_value type="String">many-to-many</enum_value>
    </option>
  </vp_entity>

  <vp_entity vp_id="129" vp_name="hibernate_mapping_column" description="column name used for the hibernate mapping ID">
    <option op_id="1">
      <enum_value type="String">QUESTION_ID</enum_value>
    </option>
    <option op_id="2">
      <enum_value type="String">EXAM_ID</enum_value>
    </option>
    <option op_id="3">
      <enum_value type="String">EXAMANSWERPERSTUDENT_ID</enum_value>
    </option>
  </vp_entity>

  <vp_entity vp_id="130" vp_name="hibernate_mapping_table" description="table names for the hibernate mapping and database">
    <option op_id="1">
      <enum_value type="String">QUESTION_ANSWER_PER_STUDENT</enum_value>
    </option>
    <option op_id="2">
      <enum_value type="String">EXAM_QUESTION</enum_value>
    </option>
    <option op_id="3">
      <enum_value type="String">EXAM_ANSWER_PER_STUDENT</enum_value>
    </option>
    <option op_id="4">
      <enum_value type="String">QUESTION_ANSWER_PER_STUDENT</enum_value>
    </option>
  </vp_entity>

  <vp_entity vp_id="131" vp_name="hibernate_mapping_target_class" description="target class for hibernate
mapping">
    <option op_id="1">
      <enum_value type="String">edu.ucsc.cse.exam.datamodel.QuestionAnswerPerStudent</enum_value>
    </option>
    <option op_id="2">
      <enum_value type="String">edu.ucsc.cse.exam.datamodel.Exam</enum_value>
    </option>
    <option op_id="3">
      <enum_value type="String">edu.ucsc.cse.exam.datamodel.ExamAnswerPerStudent</enum_value>
    </option>
    <option op_id="4">
      <enum_value type="String">edu.ucsc.cse.exam.datamodel.Question</enum_value>
    </option>
  </vp_entity>


  <vp_entity  vp_id="140" vp_name="ExamAnswerPerStudent_types" description="different types of exam answers
in the system">
    <option op_id="1" op_name="default" description="includes only one simple exam answer type that records the
studentId, creationdate, final score and duration">
      <sub_option vp_entity_id="151" vp_entity_option_id="1"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="141" vp_name="QuestionAnswerPerStudent_types" description="different types of question
answers in the system">
    <option op_id="1" op_name="default" description="includes only one simple question answer type that records
studentId, creationdate, answer, score, etc.">
      <sub_option vp_entity_id="152" vp_entity_option_id="1"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="151" vp_name="ExamAnswerPerStudent_props" description="stores exam answer and score
information for each student in each exam">
    <option op_id="1" op_name="default" description="default properties for ExamAnswerPerStudent entity">
      <sub_option vp_entity_id="104" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="106" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="102" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="73" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="91" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="92" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="121" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="126" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="127" vp_entity_option_id="5"/>
      <sub_option vp_entity_id="127" vp_entity_option_id="6"/>
    </option>
  </vp_entity>
  <vp_entity vp_id="152" vp_name="QuestionAnswerPerStudent_props" description="stores question answer and
score information for each student in an exam">
    <option op_id="1" op_name="default" description="default properties for QuestionAnswerPerStudent entity">
      <sub_option vp_entity_id="105" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="104" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="101" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="106" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="73" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="107" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="94" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="91" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="124" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="125" vp_entity_option_id="1"/>
      <sub_option vp_entity_id="127" vp_entity_option_id="7"/>
      <sub_option vp_entity_id="127" vp_entity_option_id="8"/>
    </option>
  </vp_entity>



  <vp_entity vp_id="180" vp_name="appear_in_hibernate" description="whether a property should be stored in the
Hibernate mapping file">
    <option op_id="1" op_name="False" description="no, the property only resides in the memory and should not be
stored in the Hibernate mapping file, one example is password confirmation property">
      <enum_value type="String">False</enum_value>
    </option>
    <option op_id="2" op_name="True" description="yes, store a property in the Hibernate mapping file">
      <enum_value type="String">True</enum_value>
    </option>
  </vp_entity>

  <vp_entity vp_id="181" vp_name="appear_in_sql" description="whether a property should appear in the SQL
definition file">
    <option op_id="1" op_name="False" description="no, the property should not be stored in the SQL definition file
    ">
      <enum_value type="String">False</enum_value>
    </option>
    <option op_id="2" op_name="True" description="yes, store the property in the SQL definition file">
      <enum_value type="String">True</enum_value>
    </option>
  </vp_entity>


  <vp_entity vp_id="200" vp_name="skipped_template" description="This variation point is used in the cases where a
template should be hidden from code generation and return an empty string as the generated code. It is useful for
conditional code generaiton.">
    <option op_id="1" op_name="User_hbm" description="template for User hibernate mapping file">
      <enum_value type="String">edu/ucsc/cse/exam/datamodel/User.hbm.snippet.tmpl</enum_value>
    </option>
    <option op_id="2" op_name="Question_hbm" description="template for Question hibernate mapping file">
      <enum_value type="String">edu/ucsc/cse/exam/datamodel/Question.hbm.snippet.tmpl</enum_value>
    </option>
    <option op_id="3" op_name="Exam_hbm" description="template for Exam hibernate mapping file">
      <enum_value type="String">edu/ucsc/cse/exam/datamodel/Exam.hbm.xml.tmpl</enum_value>
    </option>
    <option op_id="4" op_name="ExamAnswerPerStudent_hbm" description="template for ExamAnswerPerStudent
hibernate mapping file">
      <enum_value type="String">edu/ucsc/cse/exam/datamodel/ExamAnswerPerStudent.hbm.xml.tmpl</enum_value
>
```

```
    </option>
    <option op_id="5" op_name="QuestionAnswerPerStudent_hbm" description="template for
QuestionAnswerPerStudent hibernate mapping file">
      <enum_value
type="String">edu/ucsc/cse/exam/datamodel/QuestionAnswerPerStudent.hbm.xml.tmpl</enum_value>
    </option>
    <option op_id="6" op_name="User_sql" description="sql table definition for User">
      <enum_value type="String">sql/User.sql.snippet.tmpl</enum_value>
    </option>
    <option op_id="7" op_name="Question_sql" description="sql table definition for Question">
      <enum_value type="String">sql/Question.sql.snippet.tmpl</enum_value>
    </option>
    <option op_id="8" op_name="Exam_sql" description="sql table definition for Exam">
      <enum_value type="String">sql/Exam.sql.snippet.tmpl</enum_value>
    </option>
    <option op_id="9" op_name="ExamAnswerPerStudent_sql" description="sql table definition for
ExamAnswerPerStudent">
      <enum_value type="String">sql/ExamAnswerPerStudent.sql.snippet.tmpl</enum_value>
    </option>
    <option op_id="10" op_name="QuestionAnswerPerStudent_sql" description="sql table definition for
QuestionAnswerPerStudent">
      <enum_value type="String">sql/QuestionAnswerPerStudent.sql.snippet.tmpl</enum_value>
    </option>
    <option op_id="11" op_name="StudentDAO_java" description="DAO class for Student">
      <enum_value type="String">edu/ucsc/cse/exam/dao/StudentDAO.java.tmpl</enum_value>
    </option>
    <option op_id="12" op_name="TeacherDAO_java" description="DAO class for Teacher">
      <enum_value type="String">edu/ucsc/cse/exam/dao/TeacherDAO.java.tmpl</enum_value>
    </option>
    <option op_id="13" op_name="MCDAO_java" description="DAO class for MC question">
      <enum_value type="String">edu/ucsc/cse/exam/dao/MCDAO.java.tmpl</enum_value>
    </option>
    <option op_id="14" op_name="TFDAO_java" description="DAO class for TF questions">
      <enum_value type="String">edu/ucsc/cse/exam/dao/TFDAO.java.tmpl</enum_value>
    </option>
  </vp_entity>




  <vp_entity vp_id="300" vp_name="appDef_Exam_props" description="Exam properties in various system
applications">
    <option op_id="1" op_name="DeleteExamConfirm" description="exam props used in the DeleteExamConfirm
application">
      <sub_option vp_entity_id="66" vp_entity_option_id="1" vp_entity_name="classcode"/>
      <sub_option vp_entity_id="73" vp_entity_option_id="1" vp_entity_name="creationdate"/>
      <sub_option vp_entity_id="74" vp_entity_option_id="1" vp_entity_name="difficulty"/>
      <sub_option vp_entity_id="77" vp_entity_option_id="1" vp_entity_name="tags"/>
    </option>
    <option op_id="2" op_name="EditExam" description="collect exam prop values in the EditExam applicaiton">
      <sub_option vp_entity_id="66" vp_entity_option_id="2" vp_entity_name="classcode"/>
      <sub_option vp_entity_id="74" vp_entity_option_id="2" vp_entity_name="difficulty"/>
      <sub_option vp_entity_id="77" vp_entity_option_id="2" vp_entity_name="tags"/>
      <sub_option vp_entity_id="75" vp_entity_option_id="2" vp_entity_name="ispublic"/>
    </option>
    <option op_id="3" op_name="ListExam" description="exam props used in the ListExam application table">
      <sub_option vp_entity_id="102" vp_entity_option_id="2" vp_entity_name="examId"/>
      <sub_option vp_entity_id="103" vp_entity_option_id="2" vp_entity_name="authorId"/>
      <sub_option vp_entity_id="66" vp_entity_option_id="3" vp_entity_name="classcode"/>
      <sub_option vp_entity_id="73" vp_entity_option_id="2" vp_entity_name="creationdate"/>
      <sub_option vp_entity_id="74" vp_entity_option_id="3" vp_entity_name="difficulty"/>
      <sub_option vp_entity_id="75" vp_entity_option_id="3" vp_entity_name="ispublic"/>
      <sub_option vp_entity_id="77" vp_entity_option_id="3" vp_entity_name="tags"/>
      <sub_option vp_entity_id="99" vp_entity_option_id="2" vp_entity_name="state"/>
    </option>
    <option op_id="4" op_name="StudentListGradedExam" description="exam props to show up in page for a student
to list graded exams">
      <sub_option vp_entity_id="102" vp_entity_option_id="3" vp_entity_name="examId"/>
      <sub_option vp_entity_id="74" vp_entity_option_id="1" vp_entity_name="difficulty"/>
      <sub_option vp_entity_id="77" vp_entity_option_id="1" vp_entity_name="tags"/>
    </option>
    <option op_id="5" op_name="StudentListOpenExam" description="exam props to show up in the page for a
student to list open exams so that he can select one to take that exam">
      <sub_option vp_entity_id="73" vp_entity_option_id="1" vp_entity_name="creationdate"/>
      <sub_option vp_entity_id="74" vp_entity_option_id="1" vp_entity_name="difficulty"/>
    </option>
    <option op_id="6" op_name="TeacherGradeExam" description="exam props to show up in the page for a teacher
to select one exam and start grading">
      <sub_option vp_entity_id="73" vp_entity_option_id="1" vp_entity_name="creationdate"/>
      <sub_option vp_entity_id="92" vp_entity_option_id="1" vp_entity_name="duration"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="301" vp_name="appDef_MC_props" description="MC properties in various system applications
">
    <option op_id="1" op_name="EditExamAddMC" description="MC question props used in the EditExamAddMC
application">
      <sub_option vp_entity_id="103" vp_entity_option_id="1" vp_entity_name="authorId"/>
      <sub_option vp_entity_id="73" vp_entity_option_id="1" vp_entity_name="creationdate"/>
      <sub_option vp_entity_id="74" vp_entity_option_id="1" vp_entity_name="difficulty"/>
      <sub_option vp_entity_id="75" vp_entity_option_id="1" vp_entity_name="ispublic"/>
      <sub_option vp_entity_id="76" vp_entity_option_id="1" vp_entity_name="isactive"/>
      <sub_option vp_entity_id="77" vp_entity_option_id="1" vp_entity_name="tags"/>
      <sub_option vp_entity_id="78" vp_entity_option_id="1" vp_entity_name="body"/>
      <sub_option vp_entity_id="79" vp_entity_option_id="1" vp_entity_name="correctanswers"/>
      <sub_option vp_entity_id="80" vp_entity_option_id="1" vp_entity_name="wronganswers"/>
    </option>
    <option op_id="2" op_name="EditExamMCDataView" description="MC question props used in the MCDataView in
the EditExam application">
      <sub_option vp_entity_id="73" vp_entity_option_id="1" vp_entity_name="creationdate"/>
      <sub_option vp_entity_id="78" vp_entity_option_id="1" vp_entity_name="body"/>
      <sub_option vp_entity_id="79" vp_entity_option_id="1" vp_entity_name="correctanswers"/>
      <sub_option vp_entity_id="80" vp_entity_option_id="1" vp_entity_name="wronganswers"/>
    </option>
    <option op_id="3" op_name="DeleteQuestionConfirm" description="MC question props to show up in the
DeleteQuestionConfirm application">
      <sub_option vp_entity_id="78" vp_entity_option_id="1" vp_entity_name="body"/>
      <sub_option vp_entity_id="73" vp_entity_option_id="1" vp_entity_name="creationdate"/>
      <sub_option vp_entity_id="74" vp_entity_option_id="1" vp_entity_name="difficulty"/>
    </option>
    <option op_id="4" op_name="EditQuestion" description="props to show up in the EditQuestion application">
      <sub_option vp_entity_id="74" vp_entity_option_id="2" vp_entity_name="difficulty"/>
      <sub_option vp_entity_id="75" vp_entity_option_id="2" vp_entity_name="ispublic"/>
      <sub_option vp_entity_id="77" vp_entity_option_id="2" vp_entity_name="tags"/>
      <sub_option vp_entity_id="78" vp_entity_option_id="2" vp_entity_name="body"/>
      <sub_option vp_entity_id="79" vp_entity_option_id="2" vp_entity_name="correctanswers"/>
      <sub_option vp_entity_id="80" vp_entity_option_id="2" vp_entity_name="wronganswers"/>
    </option>
    <option op_id="5" op_name="ListQuestion" description="question props used in the ListQuestion application
table">
```

```
      <sub_option vp_entity_id="103" vp_entity_option_id="2" vp_entity_name="authorId"/>
      <sub_option vp_entity_id="73" vp_entity_option_id="2" vp_entity_name="creationdate"/>
      <sub_option vp_entity_id="74" vp_entity_option_id="3" vp_entity_name="difficulty"/>
      <sub_option vp_entity_id="75" vp_entity_option_id="3" vp_entity_name="ispublic"/>
      <sub_option vp_entity_id="76" vp_entity_option_id="2" vp_entity_name="isactive"/>
      <sub_option vp_entity_id="77" vp_entity_option_id="3" vp_entity_name="tags"/>
      <sub_option vp_entity_id="78" vp_entity_option_id="3" vp_entity_name="body"/>
      <sub_option vp_entity_id="79" vp_entity_option_id="3" vp_entity_name="correctanswers"/>
      <sub_option vp_entity_id="80" vp_entity_option_id="3" vp_entity_name="wronganswers"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="302" vp_name="appDef_TF_props" description="TF properties in various system applications
">
    <option op_id="1" op_name="EditExamAddTF" description="TF question props used in the EditExamAddTF
application">
      <sub_option vp_entity_id="103" vp_entity_option_id="1" vp_entity_name="authorId"/>
      <sub_option vp_entity_id="73" vp_entity_option_id="1" vp_entity_name="creationdate"/>
      <sub_option vp_entity_id="74" vp_entity_option_id="1" vp_entity_name="difficulty"/>
      <sub_option vp_entity_id="75" vp_entity_option_id="1" vp_entity_name="ispublic"/>
      <sub_option vp_entity_id="76" vp_entity_option_id="1" vp_entity_name="isactive"/>
      <sub_option vp_entity_id="77" vp_entity_option_id="1" vp_entity_name="tags"/>
      <sub_option vp_entity_id="78" vp_entity_option_id="1" vp_entity_name="body"/>
      <sub_option vp_entity_id="81" vp_entity_option_id="1" vp_entity_name="correctanswer"/>
    </option>
    <option op_id="2" op_name="EditExamTFDataView" description="TF question props used in the TFDataView in
the EditExam application">
      <sub_option vp_entity_id="73" vp_entity_option_id="1" vp_entity_name="creationdate"/>
      <sub_option vp_entity_id="78" vp_entity_option_id="1" vp_entity_name="body"/>
      <sub_option vp_entity_id="81" vp_entity_option_id="1" vp_entity_name="correctanswer"/>
    </option>
    <option op_id="3" op_name="DeleteQuestionConfirm" description="TF question props to show up in the
DeleteQuestionConfirm application">
      <sub_option vp_entity_id="78" vp_entity_option_id="1" vp_entity_name="body"/>
      <sub_option vp_entity_id="73" vp_entity_option_id="1" vp_entity_name="creationdate"/>
      <sub_option vp_entity_id="74" vp_entity_option_id="1" vp_entity_name="difficulty"/>
    </option>
    <option op_id="4" op_name="EditQuestion" description="props to show up on the Edit Question page">
      <sub_option vp_entity_id="74" vp_entity_option_id="2" vp_entity_name="difficulty"/>
      <sub_option vp_entity_id="75" vp_entity_option_id="2" vp_entity_name="ispublic"/>
      <sub_option vp_entity_id="77" vp_entity_option_id="2" vp_entity_name="tags"/>
      <sub_option vp_entity_id="78" vp_entity_option_id="2" vp_entity_name="body"/>
      <sub_option vp_entity_id="81" vp_entity_option_id="2" vp_entity_name="correctanswer"/>
    </option>
    <option op_id="5" op_name="ListQuestion" description="question props used in the ListQuestion application
table">
      <sub_option vp_entity_id="103" vp_entity_option_id="2" vp_entity_name="authorId"/>
      <sub_option vp_entity_id="73" vp_entity_option_id="2" vp_entity_name="creationdate"/>
      <sub_option vp_entity_id="74" vp_entity_option_id="3" vp_entity_name="difficulty"/>
      <sub_option vp_entity_id="75" vp_entity_option_id="3" vp_entity_name="ispublic"/>
      <sub_option vp_entity_id="76" vp_entity_option_id="2" vp_entity_name="isactive"/>
      <sub_option vp_entity_id="77" vp_entity_option_id="3" vp_entity_name="tags"/>
      <sub_option vp_entity_id="78" vp_entity_option_id="3" vp_entity_name="body"/>
      <sub_option vp_entity_id="81" vp_entity_option_id="3" vp_entity_name="correctanswer"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="303" vp_name="appDef_ExamAnswerPerStudent_props" description="ExamAnswerPerStudent
properties in various system applications">
```

```
    <option op_id="1" op_name="StudentListGradedExam" description="props to be used in student list graded exam page">
      <sub_option vp_entity_id="91" vp_entity_option_id="1" vp_entity_name="score"/>
      <sub_option vp_entity_id="73" vp_entity_option_id="1" vp_entity_name="creationdate"/>
      <sub_option vp_entity_id="92" vp_entity_option_id="1" vp_entity_name="duration"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="304" vp_name="appDef_Teacher_props" description="Teacher properties in various system applications">
    <option op_id="1" op_name="DeleteUserConfirm" description="Teacher props to show up in the DeleteUserConfirm application">
      <sub_option vp_entity_id="61" vp_entity_option_id="1" vp_entity_name="username"/>
      <sub_option vp_entity_id="65" vp_entity_option_id="1" vp_entity_name="email"/>
      <sub_option vp_entity_id="66" vp_entity_option_id="1" vp_entity_name="classcode"/>
    </option>
    <option op_id="2" op_name="EditUserAccount" description="Teacher props to show up in the EditUserAccount application">
      <sub_option vp_entity_id="61" vp_entity_option_id="2" vp_entity_name="username"/>
      <sub_option vp_entity_id="62" vp_entity_option_id="2" vp_entity_name="password"/>
      <sub_option vp_entity_id="63" vp_entity_option_id="2" vp_entity_name="firstname"/>
      <sub_option vp_entity_id="64" vp_entity_option_id="2" vp_entity_name="lastname"/>
      <sub_option vp_entity_id="65" vp_entity_option_id="2" vp_entity_name="email"/>
      <sub_option vp_entity_id="66" vp_entity_option_id="2" vp_entity_name="classcode"/>
      <sub_option vp_entity_id="67" vp_entity_option_id="2" vp_entity_name="department"/>
      <sub_option vp_entity_id="71" vp_entity_option_id="2" vp_entity_name="office"/>
      <sub_option vp_entity_id="72" vp_entity_option_id="2" vp_entity_name="phonenum"/>
      <sub_option vp_entity_id="95" vp_entity_option_id="2" vp_entity_name="creditcardnum"/>
      <sub_option vp_entity_id="96" vp_entity_option_id="2" vp_entity_name="creditcardtype"/>
      <sub_option vp_entity_id="97" vp_entity_option_id="2" vp_entity_name="creditcardexpdate"/>
      <sub_option vp_entity_id="98" vp_entity_option_id="2" vp_entity_name="billingaddress"/>
    </option>
    <option op_id="3" op_name="ListUser" description="Teacher props to show up in the filtered table in ListUser application">
      <sub_option vp_entity_id="61" vp_entity_option_id="3" vp_entity_name="username"/>
      <sub_option vp_entity_id="63" vp_entity_option_id="3" vp_entity_name="firstname"/>
      <sub_option vp_entity_id="64" vp_entity_option_id="3" vp_entity_name="lastname"/>
      <sub_option vp_entity_id="65" vp_entity_option_id="3" vp_entity_name="email"/>
      <sub_option vp_entity_id="66" vp_entity_option_id="3" vp_entity_name="classcode"/>
      <sub_option vp_entity_id="67" vp_entity_option_id="3" vp_entity_name="department"/>
      <sub_option vp_entity_id="71" vp_entity_option_id="3" vp_entity_name="office"/>
      <sub_option vp_entity_id="72" vp_entity_option_id="3" vp_entity_name="phonenum"/>
    </option>
  </vp_entity>

  <vp_entity vp_id="305" vp_name="appDef_Student_props" description="Student properties in various system applications">
    <option op_id="1" op_name="DeleteUserConfirm" description="Student props to show up in the DeleteUserConfirm application">
      <sub_option vp_entity_id="61" vp_entity_option_id="1" vp_entity_name="username"/>
      <sub_option vp_entity_id="65" vp_entity_option_id="1" vp_entity_name="email"/>
      <sub_option vp_entity_id="66" vp_entity_option_id="1" vp_entity_name="classcode"/>
    </option>
    <option op_id="2" op_name="EditUserAccount" description="Student props to show up in the EditUserAccount application">
      <sub_option vp_entity_id="61" vp_entity_option_id="2" vp_entity_name="username"/>
      <sub_option vp_entity_id="62" vp_entity_option_id="2" vp_entity_name="password"/>
      <sub_option vp_entity_id="63" vp_entity_option_id="2" vp_entity_name="firstname"/>
      <sub_option vp_entity_id="64" vp_entity_option_id="2" vp_entity_name="lastname"/>
      <sub_option vp_entity_id="65" vp_entity_option_id="2" vp_entity_name="email"/>
      <sub_option vp_entity_id="66" vp_entity_option_id="2" vp_entity_name="classcode"/>
      <sub_option vp_entity_id="67" vp_entity_option_id="2" vp_entity_name="department"/>
      <sub_option vp_entity_id="68" vp_entity_option_id="2" vp_entity_name="major"/>
      <sub_option vp_entity_id="69" vp_entity_option_id="2" vp_entity_name="mailingaddress"/>
      <sub_option vp_entity_id="70" vp_entity_option_id="2" vp_entity_name="ista"/>
      <sub_option vp_entity_id="95" vp_entity_option_id="2" vp_entity_name="creditcardnum"/>
      <sub_option vp_entity_id="96" vp_entity_option_id="2" vp_entity_name="creditcardtype"/>
      <sub_option vp_entity_id="97" vp_entity_option_id="2" vp_entity_name="creditcardexpdate"/>
      <sub_option vp_entity_id="98" vp_entity_option_id="2" vp_entity_name="billingaddress"/>
    </option>
    <option op_id="2" op_name="ListUser" description="Student props to show up in the filtered table for list user
application">
      <sub_option vp_entity_id="61" vp_entity_option_id="3" vp_entity_name="username"/>
      <sub_option vp_entity_id="63" vp_entity_option_id="3" vp_entity_name="firstname"/>
      <sub_option vp_entity_id="64" vp_entity_option_id="3" vp_entity_name="lastname"/>
      <sub_option vp_entity_id="65" vp_entity_option_id="3" vp_entity_name="email"/>
      <sub_option vp_entity_id="66" vp_entity_option_id="3" vp_entity_name="classcode"/>
      <sub_option vp_entity_id="67" vp_entity_option_id="3" vp_entity_name="department"/>
      <sub_option vp_entity_id="68" vp_entity_option_id="3" vp_entity_name="major"/>
      <sub_option vp_entity_id="70" vp_entity_option_id="3" vp_entity_name="ista"/>
    </option>
  </vp_entity>


<!-- add more property entity here -->

<!-- END OF VP ENTITIES -->


<!-- START OF VP APPLICATIONS -->

  <vp_application vp_id="2000" vp_name="Entity_Definition_Class" description="Definitions of entity classes in the
style of JavaBeans, containing fields, getter and setter methods, etc. If an entity has a superclass, that super class
entity needs to be defined in a similar way.">
    <option description="includes basic entity definitions for user, question, question answer, exam, exam answer"
op_id="1" op_name="basic">
    <vp_entity_dependency vp_entity_id="30" vp_entity_option_id="1" description="User_types defines subclass user
type names and properties in the system">
      <ref_CGU ref_CGU_id="1" ref_CGU_name="Subclass_Entity_string_swap" description="replace template tags
with subclass entity names"/>
      <ref_CGU ref_CGU_id="2" ref_CGU_name="Subclass_Entity_tag_expansion" description="expand template
tags using subclass entity property names and types"/>
      <ref_CGU ref_CGU_id="3" ref_CGU_name="Nonsubclass_Entity_string_swap" description="replace template
tags with nonsubclass entity names"/>
      <ref_CGU ref_CGU_id="4" ref_CGU_name="Nonsubclass_Entity_tag_expansion" description="expand template
tags using nonsubclass entity property names and types"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="46" vp_entity_option_id="1" description="Question_types defines sublcass
question type names and properties in the system">
      <ref_CGU ref_CGU_id="1" ref_CGU_name="Subclass_Entity_string_swap" description="replace template tags
with subclass entity names"/>
      <ref_CGU ref_CGU_id="2" ref_CGU_name="Subclass_Entity_tag_expansion" description="expand template
tags using subclass entity property names and types"/>
      <ref_CGU ref_CGU_id="3" ref_CGU_name="Nonsubclass_Entity_string_swap" description="replace template
tags with nonsubclass entity names"/>
      <ref_CGU ref_CGU_id="4" ref_CGU_name="Nonsubclass_Entity_tag_expansion" description="expand template
tags using nonsubclass entity property names and types"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="54" vp_entity_option_id="1" description="Exam_types defines exam type
names and properties in the system">
```

```
     <ref_CGU ref_CGU_id="1" ref_CGU_name="Subclass_Entity_string_swap" description="replace template tags
with subclass entity names"/>
     <ref_CGU ref_CGU_id="2" ref_CGU_name="Subclass_Entity_tag_expansion" description="expand template
tags using subclass entity property names and types"/>
     <ref_CGU ref_CGU_id="3" ref_CGU_name="Nonsubclass_Entity_string_swap" description="replace template
tags with nonsubclass entity names"/>
     <ref_CGU ref_CGU_id="4" ref_CGU_name="Nonsubclass_Entity_tag_expansion" description="expand template
tags using nonsubclass entity property names and types"/>
   </vp_entity_dependency>
   <vp_entity_dependency vp_entity_id="140" vp_entity_option_id="1" description="ExamAnswerPerStudent_types
defines exam answer type names and properties in the system">
     <ref_CGU ref_CGU_id="1" ref_CGU_name="Subclass_Entity_string_swap" description="replace template tags
with subclass entity names"/>
     <ref_CGU ref_CGU_id="2" ref_CGU_name="Subclass_Entity_tag_expansion" description="expand template
tags using subclass entity property names and types"/>
     <ref_CGU ref_CGU_id="3" ref_CGU_name="Nonsubclass_Entity_string_swap" description="replace template
tags with nonsubclass entity names"/>
     <ref_CGU ref_CGU_id="4" ref_CGU_name="Nonsubclass_Entity_tag_expansion" description="expand template
tags using nonsubclass entity property names and types"/>
   </vp_entity_dependency>
   <vp_entity_dependency vp_entity_id="141" vp_entity_option_id="1"
description="QuestionAnswerPerStudent_types defines question answer type names and properties in the system
   ">
     <ref_CGU ref_CGU_id="1" ref_CGU_name="Subclass_Entity_string_swap" description="replace template tags
with subclass entity names"/>
     <ref_CGU ref_CGU_id="2" ref_CGU_name="Subclass_Entity_tag_expansion" description="expand template
tags using subclass entity property names and types"/>
     <ref_CGU ref_CGU_id="3" ref_CGU_name="Nonsubclass_Entity_string_swap" description="replace template
tags with nonsubclass entity names"/>
     <ref_CGU ref_CGU_id="4" ref_CGU_name="Nonsubclass_Entity_tag_expansion" description="expand template
tags using nonsubclass entity property names and types"/>
   </vp_entity_dependency>
   </option>
  </vp_application>

  <vp_application vp_id="2001" vp_name="Entity_ORM_Mapping" description="intermediate layer between entity
classes and its persistence representation in the physical storage">
   <option op_id="1" op_name="basic" description="ORM mapping definitions for basic entity types">
   <vp_entity_dependency vp_entity_id="30" vp_entity_option_id="1" description="User_types and its child vp
entities provide user entity names, entity properties and constraint information on the properties such as
lengthLimitUpper and lengthLimitLower. Because this solution has User as superclass and other entities such as
Teacher and Student as subclasses, we need to have a User Hibernate mapping file which models this inheritance
structure. We also have separate Hibernate mapping files for each subclass User entities, although this is not
necessary. Also note that ">
     <ref_CGU ref_CGU_id="5" ref_CGU_name="User_Entity_hbm_snippet_join" description="join code snippets to
create User Hibernate mapping file"/>
     <ref_CGU ref_CGU_id="6" ref_CGU_name="User_Child_Entity_hbm_snippet_string_swap" description="string
swap for subclass User entity snippets for User Hibernate mapping file"/>
     <ref_CGU ref_CGU_id="7" ref_CGU_name="User_Child_Entity_hbm_snippet_tag_expansion" description="tag
expansion for subclass User entity snippets for User Hibernate mapping file"/>
     <ref_CGU ref_CGU_id="8" ref_CGU_name="User_Entity_hbm_snippet_string_swap" description="string swap
for User entity snippet for User Hibernate mapping file"/>
     <ref_CGU ref_CGU_id="9" ref_CGU_name="User_Entity_hbm_snippet_tag_expansion" description="tag
expansion for User Hibernate mapping file"/>
     <ref_CGU ref_CGU_id="15" ref_CGU_name="User_Subclass_hbm_string_swap" description="string swap for
User subclass entity Hibernate mapping files"/>
     <ref_CGU ref_CGU_id="16" ref_CGU_name="User_Subclass_hbm_tag_expansion" description="tag expansion
for User subclass entity Hibernate mapping files"/>
   </vp_entity_dependency>
```

```
    <vp_entity_dependency vp_entity_id="46" vp_entity_option_id="1" description="Question_types and its child vp
entities provide question entity names, entity properties and constraint information on the properties such as
lengthLimitLower and lengthLimitUpper. Because this solution has Question as superclass and other entities such as
MC and TF as subclasses, we need to have a Question Hibernate mapping file which models this inheritance
structure. We also have separate Hibernate mapping files for each subclass Question entities, although this is not
necessary.">
      <ref_CGU ref_CGU_id="10" ref_CGU_name="Question_Entity_hbm_snippet_join" description="join code
snippets to create Question Hibernate mapping file"/>
      <ref_CGU ref_CGU_id="11" ref_CGU_name="Question_Child_Entity_hbm_snippet_string_swap"
description="string swap for subclass Question entity snippets for Question Hibernate mapping file"/>
      <ref_CGU ref_CGU_id="12" ref_CGU_name="Question_Child_Entity_hbm_snippet_tag_expansion"
description="tag expansion for subclass Question entity snippets for Question Hibernate mapping file"/>
      <ref_CGU ref_CGU_id="13" ref_CGU_name="Question_Entity_hbm_snippet_string_swap" description="string
swap for Question entity snippet for Question Hibernate mapping file"/>
      <ref_CGU ref_CGU_id="14" ref_CGU_name="Question_Entity_hbm_snippet_tag_expansion" description="tag
expansion for Question entity snippet for Question Hibernate mapping file"/>
      <ref_CGU ref_CGU_id="17" ref_CGU_name="Question_Subclass_hbm_string_swap" description="string swap
for Question subclass entity Hibernate mapping files"/>
      <ref_CGU ref_CGU_id="18" ref_CGU_name="Question_Subclass_hbm_tag_expansion" description="tag
expansion for Question subclass entity Hibernate mapping files"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="54" vp_entity_option_id="1" description="Exam_types and its child vp
entities provide Exam entity names and properties">
      <ref_CGU ref_CGU_id="19" ref_CGU_name="Exam_Entity_hbm_string_swap" description="string swap for
Exam entity Hibernate mapping file"/>
      <ref_CGU ref_CGU_id="20" ref_CGU_name="Exam_Entity_hbm_tag_expansion" description="tag expansion for
Exam entity Hibernate mapping file"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="140" vp_entity_option_id="1" description="ExamAnswer_types and its child
vp entities provide exam answer entity information such as entity names, property names, types, constraints,
collection property information and various Hibernate mapping settings">
      <ref_CGU ref_CGU_id="21" ref_CGU_name="ExamAnswer_Entity_hbm_string_swap" description="string swap
for ExamAnswerPerStudent entity Hibernate mapping file"/>
      <ref_CGU ref_CGU_id="22" ref_CGU_name="ExamAnswer_Entity_hbm_tag_expansion" description="tag
expansion for ExamAnswerPerStudent entity Hibernate mapping file"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="141" vp_entity_option_id="1" description="QuestionAnswer_types and its
child vp entities provide question answer entity information such as entity names, property names, constraints,
collection property information and various Hibernate mapping settings">
      <ref_CGU ref_CGU_id="23" ref_CGU_name="QuestionAnswer_Entity_hbm_string_swap" description="string
swap for QuestionAnswerPerStudent entity Hibernate mapping file"/>
      <ref_CGU ref_CGU_id="24" ref_CGU_name="QuestionAnswer_Entity_hbm_tag_expansion" description="tag
expansion for QuestionAnswerPerStudent entity Hibernate mapping file"/>
    </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2002" vp_name="Entity_Repository_Storage" description="Physical storage structure for
system objects persistence, we use MySQL relational database">
    <option op_id="1" op_name="basic_relational" description="basic relational database table definitions">
    <vp_entity_dependency vp_entity_id="30" vp_entity_option_id="1" description="User_types and its child vp
entities define user entity names, propert details like property name, property value type, property constraints like
length limit, etc.">
      <ref_CGU ref_CGU_id="25" ref_CGU_name="createdb_sql_snippet_join" description="join table snippets to
create an SQL file"/>
      <ref_CGU ref_CGU_id="26" ref_CGU_name="parent_tables_string_swap" description="swap strings to create
table snippets that represent parent entity classes"/>
      <ref_CGU ref_CGU_id="27" ref_CGU_name="parent_tables_tag_expansion" description="expand tags to create
table snippets that represent parent entity classes"/>
```

```
      <ref_CGU ref_CGU_id="28" ref_CGU_name="child_tables_string_swap" description="swap strings to create
table snippets that represent child entity classes"/>
      <ref_CGU ref_CGU_id="29" ref_CGU_name="child_tables_tag_expansion" description="expand tags to create
table snippets that represent child entity classes"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="46" vp_entity_option_id="1" description="Question_types and its child vp
entities define quetion entity names, property deatils like property name, property value type, property constraints like
length limit, etc.">
      <ref_CGU ref_CGU_id="25" ref_CGU_name="createdb_sql_snippet_join" description="join table snippets to
create an SQL file"/>
      <ref_CGU ref_CGU_id="26" ref_CGU_name="parent_tables_string_swap" description="swap strings to create
table snippets that represent parent entity classes"/>
      <ref_CGU ref_CGU_id="27" ref_CGU_name="parent_tables_tag_expansion" description="expand tags to create
table snippets that represent parent entity classes"/>
      <ref_CGU ref_CGU_id="28" ref_CGU_name="child_tables_string_swap" description="swap strings to create
table snippets that represent child entity classes"/>
      <ref_CGU ref_CGU_id="29" ref_CGU_name="child_tables_tag_expansion" description="expand tags to create
table snippets that represent child entity classes"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="54" vp_entity_option_id="1" description="Exam_types defines exam entity
and details about its properties">
      <ref_CGU ref_CGU_id="30" ref_CGU_name="exam_table_string_swap" description="swap strings to create
exam table snippet that represents Exam entity class"/>
      <ref_CGU ref_CGU_id="31" ref_CGU_name="exam_table_tag_expansion" description="expand tags to create
exam table snippet that represents Exam entity class"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="140" vp_entity_option_id="1" description="ExamAnswerPerStudent_types
defines exam answer entities and various details about its properties">
      <ref_CGU ref_CGU_id="32" ref_CGU_name="examAnswerPerStudent_table_tag_expansion"
description="expand tags to create exam answer table snippet that represents ExamAnswerPerStudent entity class
"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="141" vp_entity_option_id="1"
description="QuestionAnswerPerStudent_types defines question answer entities and various deatils about its
properties">
      <ref_CGU ref_CGU_id="33" ref_CGU_name="questionAnswerPerStudent_table_tag_expansion"
description="expand tags to create question answer table snippet that represents QuestionAnswerPerStudent entity
class"/>
    </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2003" vp_name="Spring_Configuration" description="Spring configuration that includes
various parameters, proxy for Hibernate session, transaction management, and Spring Bean definitions">
    <option op_id="1" op_name="basic" description="Spring configuration files for basic entity types">
    <vp_entity_dependency vp_entity_id="30" vp_entity_option_id="1" description="User_types include user entity
names to be used in Spring config file">
      <ref_CGU ref_CGU_id="34" ref_CGU_name="spring_config_tag_expansion" description="expand tags to create
Spring Bean configuration file"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="46" vp_entity_option_id="1" description="Question_types include user
entity names to be used in Spring config file">
      <ref_CGU ref_CGU_id="34" ref_CGU_name="spring_config_tag_expansion" description="expand tags to create
Spring Bean configuration file"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="54" vp_entity_option_id="1" description="Exam_types include user entity
names to be used in Spring config file">
      <ref_CGU ref_CGU_id="34" ref_CGU_name="spring_config_tag_expansion" description="expand tags to create
Spring Bean configuration file"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="140" vp_entity_option_id="1" description="ExamAnswerPerStudent_types
include user entity names to be used in Spring config file">
      <ref_CGU ref_CGU_id="34" ref_CGU_name="spring_config_tag_expansion" description="expand tags to create
Spring Bean configuration file"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="141" vp_entity_option_id="1"
description="QuestionAnswerPerStudent_types include user entity names to be used in Spring config file">
      <ref_CGU ref_CGU_id="34" ref_CGU_name="spring_config_tag_expansion" description="expand tags to create
Spring Bean configuration file"/>
    </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2004" vp_name="ExamMainApplication" description="ExamMainApplication is the entry
point for the whole Web application">
    <option op_id="1" op_name="basic" description="main application activities">
    <vp_entity_dependency vp_entity_id="30" vp_entity_option_id="1" description="User_types include user entity
names to be used in ExamMainApplication">
      <ref_CGU ref_CGU_id="35" ref_CGU_name="ExamMainApplication_java_tag_expansion" description="expand
tags to create entity dao definitions, getDao methods and initialization method"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="46" vp_entity_option_id="1" description="Question_types include user
entity names to be used in ExamMainApplication">
      <ref_CGU ref_CGU_id="35" ref_CGU_name="ExamMainApplication_java_tag_expansion" description="expand
tags to create entity dao definitions, getDao methods and initialization method"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="54" vp_entity_option_id="1" description="Exam_types include user entity
names to be used in ExamMainApplication">
      <ref_CGU ref_CGU_id="35" ref_CGU_name="ExamMainApplication_java_tag_expansion" description="expand
tags to create entity dao definitions, getDao methods and initialization method"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="140" vp_entity_option_id="1" description="ExamAnswerPerStudent_types
include user entity names to be used in ExamMainApplication">
      <ref_CGU ref_CGU_id="35" ref_CGU_name="ExamMainApplication_java_tag_expansion" description="expand
tags to create entity dao definitions, getDao methods and initialization method"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="141" vp_entity_option_id="1"
description="QuestionAnswerPerStudent_types include user entity names to be used in ExamMainApplication">
      <ref_CGU ref_CGU_id="35" ref_CGU_name="ExamMainApplication_java_tag_expansion" description="expand
tags to create entity dao definitions, getDao methods and initialization method"/>
    </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2005" vp_name="ExamSession" description="ExamSession models the Web application
session including the data contained in it">
    <option op_id="1" op_name="basic" description="session related activities">
    <vp_entity_dependency vp_entity_id="30" vp_entity_option_id="1" description="User_types include user entity
names to be used in Spring config file">
      <ref_CGU ref_CGU_id="36" ref_CGU_name="ExamSession_java_tag_expansion" description="expand tags to
generate code for user entity objects in the session, get and set methods for user entity objects, check user login
status"/>
    </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2006" vp_name="UserLogin" description="UserLogin shows a page for existing users to
login, it also has links to register for new users">
    <option op_id="1" op_name="existing_new" description="allows existing user to log in and new users to register
    r">
```

```xml
    <vp_entity_dependency vp_entity_id="30" vp_entity_option_id="1" description="user entity names affect login
process and user types that can be registered for a new user">
      <ref_CGU ref_CGU_id="37" ref_CGU_name="UserLoginPage_java_tag_expansion" description="expand tags
using user entity names for UserLogin page"/>
      <ref_CGU ref_CGU_id="38" ref_CGU_name="UserLoginPage_html_tag_expansion" description="expand tags
using editable user entity names for UserLogin page html code"/>
    </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2007" vp_name="EntityDAO" description="we use the DAO design pattern to manage entity
classes">
    <option op_id="1" op_name="basic" description="DAO settings for basic entities">
    <vp_entity_dependency vp_entity_id="30" vp_entity_option_id="1" description="Subclass user entities extend
User entity class, but their DAO classes don't extend UserDAO class, there is not UserDAO class actually. Need
subclass user entity names for string swap and tag expansion.">
      <ref_CGU ref_CGU_id="39" ref_CGU_name="UserDAO_java_string_swap" description="swap strings for
subclass user entity DAO classes"/>
      <ref_CGU ref_CGU_id="40" ref_CGU_name="UserDAO_java_tag_expansion" description="expand tags for user
entity properties in subclass user entity DAO classes"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="46" vp_entity_option_id="1" description="Subclass question entities extend
Question entity class, but there is not QuestionDAO superclass to which these subclass entities extend. Need
subclass question entity names for string swap and tag expansion.">
      <ref_CGU ref_CGU_id="41" ref_CGU_name="QuestionDAO_java_string_swap" description="swap strings for
subclass question entity DAO classes"/>
      <ref_CGU ref_CGU_id="42" ref_CGU_name="QuestionDAO_java_tag_expansion" description="expand tags for
question entity properties in subclass question entity DAO classes"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="54" vp_entity_option_id="1" description="there is not Exam superclass and
Exam DAO entities are just independent classes. Need exam entity names for string swap and tag expansion.">
      <ref_CGU ref_CGU_id="43" ref_CGU_name="ExamDAO_java_string_swap" description="swap strings for exam
entity names in Exam DAO entity classes"/>
      <ref_CGU ref_CGU_id="44" ref_CGU_name="ExamDAO_java_tag_expansion" description="expand tags for
exam entity properties in Exam DAO entity classes"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="140" vp_entity_option_id="1" description="Exam answer DAO class does
not extend any superclass. Need exam answer entity names for string sawp and tag expansion.">
      <ref_CGU ref_CGU_id="45" ref_CGU_name="ExamAnswerPerStudentDAO_java_string_swap"
description="swap strings for exam answer entity names in ExamAnswerPerStudentDAO class"/>
      <ref_CGU ref_CGU_id="46" ref_CGU_name="ExamAnswerPerStudentDAO_java_tag_expansion"
description="expand tags for exam answer entity properties in ExamAnswerPerStudentDAO class"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="141" vp_entity_option_id="1" description="Question answer DAO class
does not extend any superclass. Need question answer entity names for string swap and tag expansion.">
      <ref_CGU ref_CGU_id="47" ref_CGU_name="QuestionAnswerPerStudentDAO_java_string_swap"
description="swap strings for question answer entity names in QuestionAnswerPerStudentDAO class"/>
      <ref_CGU ref_CGU_id="48" ref_CGU_name="QuestionAnswerPerStudentDAO_java_tag_expansion"
description="expand tags for question answer entity properties in QuestionAnswerPerStudentDAO class"/>
    </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2008" vp_name="BasePage" description="most of the Page class in the system extends
BasePage class, which provides shared features such as logout and homepage links, also provides quick access to
DAO objects in the session">
    <option op_id="1" op_name="basic" description="base page activities">
    <vp_entity_dependency vp_entity_id="30" vp_entity_option_id="1" description="Need subclass user entity names
for string swap and tag expansion for homepage and logout applications, also need those for get user entity dao
methods.">
```

```
      <ref_CGU ref_CGU_id="49" ref_CGU_name="BasePage_java_tag_expansion" description="expand tags for
BasePage java class"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="46" vp_entity_option_id="1" description="Need subclass question entity
names for get question entity dao methods.">
      <ref_CGU ref_CGU_id="49" ref_CGU_name="BasePage_java_tag_expansion" description="expand tags for
BasePage java class"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="54" vp_entity_option_id="1" description="Need exam entity names for get
exam entity dao methods.">
      <ref_CGU ref_CGU_id="49" ref_CGU_name="BasePage_java_tag_expansion" description="expand tags for
BasePage java class"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="140" vp_entity_option_id="1" description="Need exam answer entity names
for get exam answer entity dao methods.">
      <ref_CGU ref_CGU_id="49" ref_CGU_name="BasePage_java_tag_expansion" description="expand tags for
BasePage java class"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="141" vp_entity_option_id="1" description="Need question answer entity
names for get question answer entity dao methods.">
      <ref_CGU ref_CGU_id="49" ref_CGU_name="BasePage_java_tag_expansion" description="expand tags for
BasePage java class"/>
    </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2009" vp_name="Util" description="Util class is a place where shared constants and
methods are defined">
    <option op_id="1" op_name="basic" description="utilities settings">
    <vp_entity_dependency vp_entity_id="30" vp_entity_option_id="1" description="Need subclass user entity names
for user entity type constants.">
      <ref_CGU ref_CGU_id="50" ref_CGU_name="Util_java_tag_expansion" description="expand tags for Util java
class"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="46" vp_entity_option_id="1" description="Need subclass question entity
names for question entity type constants.">
      <ref_CGU ref_CGU_id="50" ref_CGU_name="Util_java_tag_expansion" description="expand tags for Util java
class"/>
    </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2100" vp_name="EditStudentAccount" description="input property values to create a new
Student or edit an existing Student">
    <option op_id="1" op_name="basic" description="elements related to edit student account">
    <vp_entity_dependency vp_entity_id="43" 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="43" 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="43" vp_entity_option_id="2" description="for each student property, create
its widget on the edit page">
      <ref_CGU ref_CGU_id="131" ref_CGU_name="EditUserAccountPage_html_string_swap" description="string
swap for edit user account html page"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="43" vp_entity_option_id="2" description="for each student property, create
its widget on the edit page">
      <ref_CGU ref_CGU_id="132" ref_CGU_name="EditUserAccountPage_html_tag_expansion" description="tag
expansion for edit user account html page"/>
    </vp_entity_dependency>
    <vp_entity_dependency vp_entity_id="43" vp_entity_option_id="2" description="for each student property, create
its widget on the edit page">
      <ref_CGU ref_CGU_id="133" ref_CGU_name="EditUserAccountPage_property_tag_expansion"
description="tag expansion for edit user account page property file"/>
    </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2101" vp_name="EditTeacherAccount" description="input property values to create a new
Teacher or edit an existing Teacher">
    <option op_id="1" op_name="basic" description="elements related to edit teacher account">
      <vp_entity_dependency vp_entity_id="42" vp_entity_option_id="2" description="for each teacher 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 teacher 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 teacher property, create
its widget on the edit page">
        <ref_CGU ref_CGU_id="131" ref_CGU_name="EditUserAccountPage_html_string_swap" 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 teacher property, create
its widget on the edit page">
        <ref_CGU ref_CGU_id="132" ref_CGU_name="EditUserAccountPage_html_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 teacher property, create
its widget on the edit page">
        <ref_CGU ref_CGU_id="133" ref_CGU_name="EditUserAccountPage_property_tag_expansion"
description="expand tags for edit user account page"/>
      </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2102" vp_name="ListStudents" description="display existing students">
    <option op_id="1" op_name="allStudentsAllProps" description="display all the students in a table format with
column filter and sortable columns, use multiple page to improve performance">
      <vp_entity_dependency description="use a multipage, sortable table with column filter" vp_entity_id="3"
vp_entity_option_id="6">
        <ref_CGU ref_CGU_id="134" ref_CGU_name="ListUserPage_java_string_swap" description="swap strings to
create list user java page"/>
        <ref_CGU ref_CGU_id="135" ref_CGU_name="ListUserPage_java_tag_expansion" description="expand tags
to create list user java page"/>
        <ref_CGU ref_CGU_id="136" ref_CGU_name="ListUserPage_html_string_swap" description="swap strings to
create list question html page"/>
      </vp_entity_dependency>
      <vp_entity_dependency description="each student property will be displayed as a column in the table"
vp_entity_id="43" vp_entity_option_id="2">
        <ref_CGU ref_CGU_id="134" ref_CGU_name="ListUserPage_java_string_swap" description="swap strings to
create list user java page"/>
```

```
        <ref_CGU ref_CGU_id="135" ref_CGU_name="ListUserPage_java_tag_expansion" description="expand tags
to create list user java page"/>
        <ref_CGU ref_CGU_id="136" ref_CGU_name="ListUserPage_html_string_swap" description="swap strings to
create list question html page"/>
      </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2103" vp_name="ListTeachers" description="display existing teachers">
    <option op_id="1" op_name="allTeachersAllProps" description="display all the teachers in a table format with
column filter and sortable columns, use multiple page to improve performance">
      <vp_entity_dependency description="use a multipage, sortable table with column filter" vp_entity_id="3"
vp_entity_option_id="6">
        <ref_CGU ref_CGU_id="134" ref_CGU_name="ListUserPage_java_string_swap" description="swap strings to
create list user java page"/>
        <ref_CGU ref_CGU_id="135" ref_CGU_name="ListUserPage_java_tag_expansion" description="expand tags
to create list user java page"/>
        <ref_CGU ref_CGU_id="136" ref_CGU_name="ListUserPage_html_string_swap" description="swap strings to
create list question html page"/>
      </vp_entity_dependency>
      <vp_entity_dependency description="each teacher property will be displayed as a column in the table"
vp_entity_id="42" vp_entity_option_id="2">
        <ref_CGU ref_CGU_id="134" ref_CGU_name="ListUserPage_java_string_swap" description="swap strings to
create list user java page"/>
        <ref_CGU ref_CGU_id="135" ref_CGU_name="ListUserPage_java_tag_expansion" description="expand tags
to create list user java page"/>
        <ref_CGU ref_CGU_id="136" ref_CGU_name="ListUserPage_html_string_swap" description="swap strings to
create list question html page"/>
      </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2016" vp_name="DeleteUserConfirm" description="delete an existing user in the system">
    <option op_id="1" description="This action combines delete confirmation action for all the user types that allow
deletion. It asks for user confirmation before executing the delete operation">
      <vp_entity_dependency description="for each deletable user type, add widgets to remove that user from the
system" vp_entity_id="110" vp_entity_option_id="1">
        <!-- DeleteUserConfirmPage -->
        <ref_CGU ref_CGU_id="125" ref_CGU_name="DeleteUserConfirmPage_java_snippet_join_tag_expansion"
description="expand tags to create the template file for snippet join CGU"/>
        <ref_CGU ref_CGU_id="126" ref_CGU_name="DeleteUserConfirmPage_java_snippet_join_string_swap"
description="swap strings to create code snippets for user type in DeleteUserConfirm java page"/>
        <ref_CGU ref_CGU_id="127" ref_CGU_name="DeleteUserConfirmPage_user_snippet_join_tag_expansion"
description="expand tags to create code snippets for user type in DeleteUserConfirmPage java page" />
      </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2017" vp_name="EditMCQuestion" description="input property values to create a new MC
question or edit an existing MC question">
    <option op_id="1" op_name="newQuestion" description="create a new MC question from scratch">
      <vp_entity_dependency description="for each MC property, create its widegt on the edit page" vp_entity_id="48"
vp_entity_option_id="1">
        <!-- Edit$QuestionEntity$$QuestionPage -->
        <ref_CGU ref_CGU_id="107" ref_CGU_name="EditQuestionPage_properties_tag_expansion"
description="expand tags to create property files for the edit question page"/>
        <ref_CGU ref_CGU_id="109" ref_CGU_name="EditQuestionPage_java_string_swap" description="swap
strings to create edit question java page"/>
        <ref_CGU ref_CGU_id="110" ref_CGU_name="EditQuestionPage_java_tag_expansion" description="expand
tags to create edit question java page"/>
```

```
    <ref_CGU ref_CGU_id="111" ref_CGU_name="EditQuestionPage_html_tag_expansion" description="expand
tags to create edit question html page"/>
   </vp_entity_dependency>
  </option>
  <option op_id="2" op_name="existingQuestion" description="edit an existing MC question">
   <vp_entity_dependency description="for each MC property, create its widegt on the edit page" vp_entity_id="48"
vp_entity_option_id="1">
    <!-- Edit$QuestionEntity$$QuestionPage -->
    <ref_CGU ref_CGU_id="107" ref_CGU_name="EditQuestionPage_properties_tag_expansion"
description="expand tags to create property files for the edit question page"/>
    <ref_CGU ref_CGU_id="109" ref_CGU_name="EditQuestionPage_java_string_swap" description="swap
strings to create edit question java page"/>
    <ref_CGU ref_CGU_id="110" ref_CGU_name="EditQuestionPage_java_tag_expansion" description="expand
tags to create edit question java page"/>
    <ref_CGU ref_CGU_id="111" ref_CGU_name="EditQuestionPage_html_tag_expansion" description="expand
tags to create edit question html page"/>
   </vp_entity_dependency>
  </option>
 </vp_application>

 <vp_application vp_id="2018" vp_name="EditTFQuestion" description="input property values to create a new TF
question or edit an existing TF question">
  <option op_id="1" op_name="newQuestion" description="create a new TF question from scratch">
   <vp_entity_dependency description="for each TF property, create its widget on the edit page" vp_entity_id="49"
vp_entity_option_id="1">
    <!-- Edit$QuestionEntity$$QuestionPage -->
    <ref_CGU ref_CGU_id="107" ref_CGU_name="EditQuestionPage_properties_tag_expansion"
description="expand tags to create property files for the edit question page"/>
    <ref_CGU ref_CGU_id="109" ref_CGU_name="EditQuestionPage_java_string_swap" description="swap
strings to create edit question java page"/>
    <ref_CGU ref_CGU_id="110" ref_CGU_name="EditQuestionPage_java_tag_expansion" description="expand
tags to create edit question java page"/>
    <ref_CGU ref_CGU_id="111" ref_CGU_name="EditQuestionPage_html_tag_expansion" description="expand
tags to create edit question html page"/>
   </vp_entity_dependency>
  </option>
  <option op_id="2" op_name="existingQuestion" description="edit an existing TF question">
   <vp_entity_dependency description="for each TF property, create its widget on the edit page" vp_entity_id="49"
vp_entity_option_id="1">
    <!-- Edit$QuestionEntity$$QuestionPage -->
    <ref_CGU ref_CGU_id="107" ref_CGU_name="EditQuestionPage_properties_tag_expansion"
description="expand tags to create property files for the edit question page"/>
    <ref_CGU ref_CGU_id="109" ref_CGU_name="EditQuestionPage_java_string_swap" description="swap
strings to create edit question java page"/>
    <ref_CGU ref_CGU_id="110" ref_CGU_name="EditQuestionPage_java_tag_expansion" description="expand
tags to create edit question java page"/>
    <ref_CGU ref_CGU_id="111" ref_CGU_name="EditQuestionPage_html_tag_expansion" description="expand
tags to create edit question html page"/>
   </vp_entity_dependency>
  </option>
 </vp_application>

 <vp_application vp_id="2019" vp_name="ListMCQuestion" description="display existing MC questions">
  <option op_id="1" op_name="system1" description="display MC questions in a table format with column filter and
sortable columns, use multiple pages to improve performance">
   <vp_entity_dependency description="use a multipage, sortable table with column filter" vp_entity_id="3"
vp_entity_option_id="6">
    <!-- List$QuestionEntity$$QuestionPage -->
    <ref_CGU ref_CGU_id="112" ref_CGU_name="ListQuestionPage_java_string_swap" description="swap strings
to create list question java page"/>
```

```
    <ref_CGU ref_CGU_id="113" ref_CGU_name="ListQuestionPage_java_tag_expansion" description="expand
tags to create list question java page"/>
    <ref_CGU ref_CGU_id="114" ref_CGU_name="ListQuestionPage_html_string_swap" description="swap strings
to create list question html page"/>
   </vp_entity_dependency>
   <vp_entity_dependency description="each MC property will be displayed as a column in the table"
vp_entity_id="48" vp_entity_option_id="1">
    <!-- List$QuestionEntity$$QuestionPage -->
    <ref_CGU ref_CGU_id="112" ref_CGU_name="ListQuestionPage_java_string_swap" description="swap strings
to create list question java page"/>
    <ref_CGU ref_CGU_id="113" ref_CGU_name="ListQuestionPage_java_tag_expansion" description="expand
tags to create list question java page"/>
    <ref_CGU ref_CGU_id="114" ref_CGU_name="ListQuestionPage_html_string_swap" description="swap strings
to create list question html page"/>
    <!-- $QuestionEntity$$QuestionDataProvider -->
    <ref_CGU ref_CGU_id="93" ref_CGU_name="QuestionDataProvider_java_string_swap" description="swap
strings to create question selection wrappers"/>
    <!-- Detachable$QuestionEntity$$QuestionModel -->
    <ref_CGU ref_CGU_id="96" ref_CGU_name="DetachableQuestionModel_java_string_swap" description="swap
strings to create detachable question model"/>
    <!-- QuestionActionPanel -->
    <ref_CGU ref_CGU_id="117" ref_CGU_name="QuestionActionPanel_java_tag_expansion"
description="expand tags to create question action panel java file"/>
    <ref_CGU ref_CGU_id="118" ref_CGU_name="QuestionActionPanel_html_file_copy" description="copy files to
create question action panel html files"/>
   </vp_entity_dependency>
  </option>
 </vp_application>

 <vp_application vp_id="2020" vp_name="ListTFQuestion" description="display existing TF questions">
  <option op_id="1" op_name="system1" description="display TF questions in a table format with column filter and
sortable columns, use multiple pages to improve performance">
   <vp_entity_dependency description="use a multipage, sortable table with column filter" vp_entity_id="3"
vp_entity_option_id="6">
    <!-- List$QuestionEntity$$QuestionPage -->
    <ref_CGU ref_CGU_id="112" ref_CGU_name="ListQuestionPage_java_string_swap" description="swap strings
to create list question java page"/>
    <ref_CGU ref_CGU_id="113" ref_CGU_name="ListQuestionPage_java_tag_expansion" description="expand
tags to create list question java page"/>
    <ref_CGU ref_CGU_id="114" ref_CGU_name="ListQuestionPage_html_string_swap" description="swap strings
to create list question html page"/>
   </vp_entity_dependency>
   <vp_entity_dependency description="each TF property will be displayed as a column in the table"
vp_entity_id="49" vp_entity_option_id="1">
    <!-- List$QuestionEntity$$QuestionPage -->
    <ref_CGU ref_CGU_id="112" ref_CGU_name="ListQuestionPage_java_string_swap" description="swap strings
to create list question java page"/>
    <ref_CGU ref_CGU_id="113" ref_CGU_name="ListQuestionPage_java_tag_expansion" description="expand
tags to create list question java page"/>
    <ref_CGU ref_CGU_id="114" ref_CGU_name="ListQuestionPage_html_string_swap" description="swap strings
to create list question html page"/>
    <!-- $QuestionEntity$$QuestionDataProvider -->
    <ref_CGU ref_CGU_id="93" ref_CGU_name="QuestionDataProvider_java_string_swap" description="swap
strings to create question selection wrappers"/>
    <!-- Detachable$QuestionEntity$$QuestionModel -->
    <ref_CGU ref_CGU_id="96" ref_CGU_name="DetachableQuestionModel_java_string_swap" description="swap
strings to create detachable question model"/>
    <!-- QuestionActionPanel -->
    <ref_CGU ref_CGU_id="117" ref_CGU_name="QuestionActionPanel_java_tag_expansion"
description="expand tags to create question action panel java file"/>
```

```
        <ref_CGU ref_CGU_id="118" ref_CGU_name="QuestionActionPanel_html_file_copy" description="copy files to
create question action panel html files"/>
      </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2021" vp_name="PreviewQuestion" description="preview a selected question as it appears
in a real exam session">
    <option op_id="1" description="display diffennent question content based on the question type">
      <vp_entity_dependency description="each question type needs its own rendering mechanism to be added in the
preview question page" vp_entity_id="46" vp_entity_option_id="1" >
        <!-- Preview$QuestionEntity$$QuestionPanel -->
        <ref_CGU ref_CGU_id="119" ref_CGU_name="PreviewQuestionPanel_java_string_swap" description="swap
strings to create preview question panel java file"/>
        <ref_CGU ref_CGU_id="120" ref_CGU_name="PreviewQuestionPanel_java_tag_expansion"
description="expand tags to create preview question panel java file"/>
        <ref_CGU ref_CGU_id="121" ref_CGU_name="PreviewQuestionPanel_html_tag_expansion"
description="expand tags to create preview question panel html file"/>
        <!-- PreviewQuestionPage -->
        <ref_CGU ref_CGU_id="115" ref_CGU_name="PreviewQuestionPage_java_tag_expansion"
description="expand tags to create preview question java page"/>
        <ref_CGU ref_CGU_id="116" ref_CGU_name="PreviewQuestionPage_html_file_copy" description="copy files
to create preview question html page"/>
      </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2022" vp_name="ListExam" description="list exams">
    <option op_id="1" op_name="all_exams" description="list all existing exams in a table with column filter and
sortable columns, use multiple pages to improve performance">
      <vp_entity_dependency description="use a multipage, sortable table with column filter" vp_entity_id="3"
vp_entity_option_id="6">
        <!-- ListExamPage -->
        <ref_CGU ref_CGU_id="66" ref_CGU_name="ListExamPage_java_tag_expansion" description="expand tags in
ListExamPage java file"/>
        <ref_CGU ref_CGU_id="67" ref_CGU_name="ListExamPage_html_file_copy" description="copy template
content into the newly generated file for html page"/>
        <!-- ListExamPage$ExamActionPanel -->
        <ref_CGU ref_CGU_id="68" ref_CGU_name="ListExamPage_action_html_file_copy" description="copy
template content into the newly generated file for action panel"/>
      </vp_entity_dependency>
      <vp_entity_dependency description="each exam property will be rendered as a column in the table"
vp_entity_id="55" vp_entity_option_id="1">
        <!-- ListExamPage -->
        <ref_CGU ref_CGU_id="66" ref_CGU_name="ListExamPage_java_tag_expansion" description="expand tags in
ListExamPage java file"/>
        <ref_CGU ref_CGU_id="67" ref_CGU_name="ListExamPage_html_file_copy" description="copy template
content into the newly generated file for html page"/>
        <!-- ListExamPage$ExamActionPanel -->
        <ref_CGU ref_CGU_id="68" ref_CGU_name="ListExamPage_action_html_file_copy" description="copy
template content into the newly generated file for action panel"/>
        <!-- ExamDataProvider -->
        <ref_CGU ref_CGU_id="101" ref_CGU_name="ExamDataProvider_java_file_copy" description="copy template
file to create exam data provider"/>
        <!-- DetachableExamModel -->
        <ref_CGU ref_CGU_id="99" ref_CGU_name="DetachableExamModel_java_file_copy" description="copy
template file to create detachable exam model"/>
        </vp_entity_dependency>
    </option>
  </vp_application>
  <vp_application vp_id="2023" vp_name="EditExam" description="edit exam properties, add or remove quetions">
    <option op_id="1" op_name="newExamFromScratch" description="create a brand new Exam from scratch by
adding questions">
      <vp_entity_dependency description="each question type will have its own table for adding questions into the new
exam" vp_entity_id="46" vp_entity_option_id="1">
        <!-- EditExamPages -->
        <ref_CGU ref_CGU_id="58" ref_CGU_name="EditExamPage_java_snippet_join" description="join code
snippets to create some content for EditExamPage"/>
        <ref_CGU ref_CGU_id="59" ref_CGU_name="EditExamPageQuestionDataView_snippet_string_swap"
description="swap strings to create snippets to be used in EditExamPage"/>
        <ref_CGU ref_CGU_id="60" ref_CGU_name="EditExamPageQuestionDataView_snippet_tag_expansion"
description="expand tags to create snippets to be used in EditExamPage"/>
        <ref_CGU ref_CGU_id="61" ref_CGU_name="EditExamPage_java_tag_expansion" description="expand tags
in EditExamPage"/>
        <ref_CGU ref_CGU_id="62" ref_CGU_name="EditExamPage_html_snippet_join" description="join code
snippets to create EditExamPage html file"/>
        <ref_CGU ref_CGU_id="63" ref_CGU_name="EditExamPageQuestionDataView_html_snippet_string_swap"
description="swap strings to create snippets for EditExamPage html file"/>
        <ref_CGU ref_CGU_id="64" ref_CGU_name="EditExamPageQuestionDataView_html_snippet_tag_expansion"
description="expand tags to create snippets for EditExamPage html file"/>
      </vp_entity_dependency>
    </option>
    <option op_id="2" op_name="newExamWithSelections" description="create a brand new Exam with a list of
question selections">
      <vp_entity_dependency description="each question type will have its own table for adding questions into the new
exam" vp_entity_id="46" vp_entity_option_id="1">
      <!-- EditExamPage -->
        <ref_CGU ref_CGU_id="58" ref_CGU_name="EditExamPage_java_snippet_join" description="join code
snippets to create some content for EditExamPage"/>
        <ref_CGU ref_CGU_id="59" ref_CGU_name="EditExamPageQuestionDataView_snippet_string_swap"
description="swap strings to create snippets to be used in EditExamPage"/>
        <ref_CGU ref_CGU_id="60" ref_CGU_name="EditExamPageQuestionDataView_snippet_tag_expansion"
description="expand tags to create snippets to be used in EditExamPage"/>
        <ref_CGU ref_CGU_id="61" ref_CGU_name="EditExamPage_java_tag_expansion" description="expand tags
in EditExamPage"/>
        <ref_CGU ref_CGU_id="62" ref_CGU_name="EditExamPage_html_snippet_join" description="join code
snippets to create EditExamPage html file"/>
        <ref_CGU ref_CGU_id="63" ref_CGU_name="EditExamPageQuestionDataView_html_snippet_string_swap"
description="swap strings to create snippets for EditExamPage html file"/>
        <ref_CGU ref_CGU_id="64" ref_CGU_name="EditExamPageQuestionDataView_html_snippet_tag_expansion"
description="expand tags to create snippets for EditExamPage html file"/>
      </vp_entity_dependency>
    </option>
    <option op_id="3" op_name="existingExam" description="edit an existing Exam">
      <vp_entity_dependency description="each question type will have its own table for adding questions into the new
exam" vp_entity_id="46" vp_entity_option_id="1">
        <!-- EditExamPage -->
        <ref_CGU ref_CGU_id="58" ref_CGU_name="EditExamPage_java_snippet_join" description="join code
snippets to create some content for EditExamPage"/>
        <ref_CGU ref_CGU_id="59" ref_CGU_name="EditExamPageQuestionDataView_snippet_string_swap"
description="swap strings to create snippets to be used in EditExamPage"/>
        <ref_CGU ref_CGU_id="60" ref_CGU_name="EditExamPageQuestionDataView_snippet_tag_expansion"
description="expand tags to create snippets to be used in EditExamPage"/>
        <ref_CGU ref_CGU_id="61" ref_CGU_name="EditExamPage_java_tag_expansion" description="expand tags
in EditExamPage"/>
        <ref_CGU ref_CGU_id="62" ref_CGU_name="EditExamPage_html_snippet_join" description="join code
snippets to create EditExamPage html file"/>
        <ref_CGU ref_CGU_id="63" ref_CGU_name="EditExamPageQuestionDataView_html_snippet_string_swap"
description="swap strings to create snippets for EditExamPage html file"/>
```

```
        <ref_CGU ref_CGU_id="64" ref_CGU_name="EditExamPageQuestionDataView_html_snippet_tag_expansion"
description="expand tags to create snippets for EditExamPage html file"/>
      </vp_entity_dependency>
    </option>
    <option op_id="4" op_name="existingExamWithSelections" description="edit an existing Exam with a list of
question selections">
      <vp_entity_dependency description="each question type will have its own table for adding questions into the new
exam" vp_entity_id="46"  vp_entity_option_id="1">
      <!-- EditExamPage -->
        <ref_CGU ref_CGU_id="58" ref_CGU_name="EditExamPage_java_snippet_join" description="join code
snippets to create some content for EditExamPage"/>
        <ref_CGU ref_CGU_id="59" ref_CGU_name="EditExamPageQuestionDataView_snippet_string_swap"
description="swap strings to create snippets to be used in EditExamPage"/>
        <ref_CGU ref_CGU_id="60" ref_CGU_name="EditExamPageQuestionDataView_snippet_tag_expansion"
description="expand tags to create snippets to be used in EditExamPage"/>
        <ref_CGU ref_CGU_id="61" ref_CGU_name="EditExamPage_java_tag_expansion" description="expand tags
in EditExamPage"/>
        <ref_CGU ref_CGU_id="62" ref_CGU_name="EditExamPage_html_snippet_join" description="join code
snippets to create EditExamPage html file"/>
        <ref_CGU ref_CGU_id="63" ref_CGU_name="EditExamPageQuestionDataView_html_snippet_string_swap"
description="swap strings to create snippets for EditExamPage html file"/>
        <ref_CGU ref_CGU_id="64" ref_CGU_name="EditExamPageQuestionDataView_html_snippet_tag_expansion"
description="expand tags to create snippets for EditExamPage html file"/>
      </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2024" vp_name="EditExamAddMC" description="add MC questions to the exam">
    <option op_id="1" op_name="newExam" description="use a list of selected questions to create a new Exam">
      <vp_entity_dependency description="use a normal table to show existing questions in the exam, each exam
property is displayed as a column" vp_entity_id="3" vp_entity_option_id="7">
        <!-- EditExamAdd$QuestionEntityCamelCase$$Page -->
        <ref_CGU ref_CGU_id="53" ref_CGU_name="EditExamAddQuestionPage_java_string_swap"
description="swap strings for questoin entity name in editExamAddQuestionPage"/>
        <ref_CGU ref_CGU_id="54" ref_CGU_name="EditExamAddQuestionPage_java_tag_expansion"
description="expand tags for add question page"/>
        <!-- Exam$QuestionEntity$$DataProvider -->
        <ref_CGU ref_CGU_id="101" ref_CGU_name="ExamDataProvider_java_file_copy" description="copy template
file to create exam data provider"/>
        <!-- DetachableExam$QuestionEntity$$Model -->
        <ref_CGU ref_CGU_id="98" ref_CGU_name="DetachableExamQuestionModel_java_string_swap"
description="swap string to create detachable exam question model"/>
        <ref_CGU ref_CGU_id="99" ref_CGU_name="DetachableExamModel_java_file_copy" description="copy
template file content to create detachable exam model"/>
        <!-- $QuestionEntity$$QuestionSelectionWrapper -->
        <ref_CGU ref_CGU_id="94" ref_CGU_name="QuestionSelectionWrapper_java_string_swap"
description="swap strings to create question selection wrappers"/>
      </vp_entity_dependency>
    </option>
    <option op_id="2" op_name="existingExam" description="edit an existing Exam using a list of selected questions">
      <vp_entity_dependency description="use a normal table to show existing questions in the exam, each exam
property is displayed as a column" vp_entity_id="3" vp_entity_option_id="7">
        <!-- EditExamAdd$QuestionEntityCamelCase$$Page -->
        <ref_CGU ref_CGU_id="53" ref_CGU_name="EditExamAddQuestionPage_java_string_swap"
description="swap strings for questoin entity name in editExamAddQuestionPage"/>
        <ref_CGU ref_CGU_id="54" ref_CGU_name="EditExamAddQuestionPage_java_tag_expansion"
description="expand tags for add question page"/>
        <!-- Exam$QuestionEntity$$DataProvider -->
        <ref_CGU ref_CGU_id="101" ref_CGU_name="ExamDataProvider_java_file_copy" description="copy template
file to create exam data provider"/>
```

```
        <!-- DetachableExam$QuestionEntity$$Model -->
        <ref_CGU ref_CGU_id="98" ref_CGU_name="DetachableExamQuestionModel_java_string_swap"
description="swap string to create detachable exam question model"/>
        <ref_CGU ref_CGU_id="99" ref_CGU_name="DetachableExamModel_java_file_copy" description="copy
template file content to create detachable exam model"/>
        <!-- $QuestionEntity$$QuestionSelectionWrapper -->
        <ref_CGU ref_CGU_id="94" ref_CGU_name="QuestionSelectionWrapper_java_string_swap"
description="swap strings to create question selection wrappers"/>
      </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2025" vp_name="EditExamAddTF" description="add TF questions to the exam">
    <option op_id="1" op_name="newExam" description="use a list of selected questions to create a new Exam">
      <vp_entity_dependency description="use a normal table to show existing questions in the exam, each exam
property is displayed as a column" vp_entity_id="3" vp_entity_option_id="7">
        <!-- EditExamAdd$QuestionEntityCamelCase$$Page -->
        <ref_CGU ref_CGU_id="53" ref_CGU_name="EditExamAddQuestionPage_java_string_swap"
description="swap strings for questoin entity name in editExamAddQuestionPage"/>
        <ref_CGU ref_CGU_id="54" ref_CGU_name="EditExamAddQuestionPage_java_tag_expansion"
description="expand tags for add question page"/>
        <!-- Exam$QuestionEntity$$DataProvider -->
        <ref_CGU ref_CGU_id="101" ref_CGU_name="ExamDataProvider_java_file_copy" description="copy template
file to create exam data provider"/>
        <!-- DetachableExam$QuestionEntity$$Model -->
        <ref_CGU ref_CGU_id="98" ref_CGU_name="DetachableExamQuestionModel_java_string_swap"
description="swap string to create detachable exam question model"/>
        <ref_CGU ref_CGU_id="99" ref_CGU_name="DetachableExamModel_java_file_copy" description="copy
template file content to create detachable exam model"/>
        <!-- $QuestionEntity$$QuestionSelectionWrapper -->
        <ref_CGU ref_CGU_id="94" ref_CGU_name="QuestionSelectionWrapper_java_string_swap"
description="swap strings to create question selection wrappers"/>
      </vp_entity_dependency>
    </option>
    <option op_id="2" op_name="existingExam" description="edit an existing Exam using a list of selected questions">
      <vp_entity_dependency description="use a normal table to show existing questions in the exam, each exam
property is displayed as a column" vp_entity_id="3" vp_entity_option_id="7">
      <!-- EditExamAdd$QuestionEntityCamelCase$$Page -->
        <ref_CGU ref_CGU_id="53" ref_CGU_name="EditExamAddQuestionPage_java_string_swap"
description="swap strings for questoin entity name in editExamAddQuestionPage"/>
        <ref_CGU ref_CGU_id="54" ref_CGU_name="EditExamAddQuestionPage_java_tag_expansion"
description="expand tags for add question page"/>
        <!-- Exam$QuestionEntity$$DataProvider -->
        <ref_CGU ref_CGU_id="101" ref_CGU_name="ExamDataProvider_java_file_copy" description="copy template
file to create exam data provider"/>
        <!-- DetachableExam$QuestionEntity$$Model -->
        <ref_CGU ref_CGU_id="98" ref_CGU_name="DetachableExamQuestionModel_java_string_swap"
description="swap string to create detachable exam question model"/>
        <ref_CGU ref_CGU_id="99" ref_CGU_name="DetachableExamModel_java_file_copy" description="copy
template file content to create detachable exam model"/>
        <!-- $QuestionEntity$$QuestionSelectionWrapper -->
        <ref_CGU ref_CGU_id="94" ref_CGU_name="QuestionSelectionWrapper_java_string_swap"
description="swap strings to create question selection wrappers"/>
      </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2026" vp_name="PreviewExam" description="preview exam questions as they will be
rendered in a real exam session">
    <option op_id="1" op_name="single_page" description="preview all questions on a single page">
```

```
      <vp_entity_dependency description="each question type has its own rendering mechanism to display on the
preview page" vp_entity_id="46" vp_entity_option_id="1">
        <!-- PreviewExamPage -->
        <ref_CGU ref_CGU_id="69" ref_CGU_name="PreviewExamPage_java_tag_expansion" description="expand
tags to create preview exam page java file"/>
        <ref_CGU ref_CGU_id="70" ref_CGU_name="PreviewExamPage_html_file_copy" description="copy template
content to create preview exam page html file"/>
        <!-- Preview$QuestionEntity$$QuestionPanel -->
        <ref_CGU ref_CGU_id="73" ref_CGU_name="PreviewQuestionPanel_java_tag_expansion"
description="expand tags to create preview question panels java file"/>
        <ref_CGU ref_CGU_id="74" ref_CGU_name="PreviewQuestionPanel_html_tag_expansion"
description="expand tags to create preview question panels html file"/>
      </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2027" vp_name="StudentListGradedExam" description="a student lists all his exams that
have been graded">
    <option op_id="1" description="show graded exams that match the student's classcode, and provide pointers to
the view exam answer report action">
      <vp_entity_dependency description="there needs an action to show the graded exam answer for a student"
vp_entity_id="2031" vp_entity_option_id="1">
        <!-- StudentListGradedExamPage -->
        <ref_CGU ref_CGU_id="75" ref_CGU_name="StudentListGradedExamPage_java_tag_expansion"
description="expand tags to create page for a student to list graded exams"/>
        <ref_CGU ref_CGU_id="76" ref_CGU_name="StudentListGradedExamPage_html_tag_expansion"
description="expand tags to create html page for a student to list graded exams"/>
      </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2028" vp_name="StudentListOpenExam" description="a student lists open exams that he
can start answering">
    <option op_id="1" description="show open exams that match the student's classcode, also provide pointers to the
take exam action">
      <vp_entity_dependency description="there needs an action for a student to take a selected exam"
vp_entity_id="2028" vp_entity_option_id="1">
        <!-- StudentListOpenExamPage -->
        <ref_CGU ref_CGU_id="77" ref_CGU_name="StudentListOpenExamPage_java_tag_expansion"
description="expand tags to create page for a student to list open exams"/>
        <ref_CGU ref_CGU_id="78" ref_CGU_name="StudentListOpenExamPage_html_tag_expansion"
description="expand tags to create html page for a student to list open exams"/>
      </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2029" vp_name="StudentTakeExam" description="a student starts answering questions in
the exam and submits the question answers">
    <option op_id="1" op_name="single_page" description="all the questions will be rendered on the same page">
      <vp_entity_dependency description="each question type has its own rendering mechanism to show up on the
page, each question answer needs to submit individually, then submit the whole exam answer" vp_entity_id="46"
vp_entity_option_id="1">
        <!-- StudentTakeExamPage -->
        <ref_CGU ref_CGU_id="79" ref_CGU_name="StudentTakeExamPage_java_tag_expansion"
description="expand tags to create page for a student to take an exam"/>
        <ref_CGU ref_CGU_id="80" ref_CGU_name="StudentTakeExamPage_html_file_copy" description="copy
template content to create take exam html page"/>
        <!-- StudentTakeExamPage$TakeMcQuestionPanel -->
        <ref_CGU ref_CGU_id="81"
ref_CGU_name="StudentTakeExamPage_TakeMCQuestionPanel_html_tag_expansion" description="MC question
panel used in take exam page"/>
        <!-- StudentTakeExamPage$TakeTfQuestionPanel -->
        <ref_CGU ref_CGU_id="82"
ref_CGU_name="StudentTakeExamPage_TakeTFQuestionPanel_html_tag_expansion" description="TF question
panel used in take exam page"/>
      </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2030" vp_name="TeacherGradeExam" description="a teacher lists all the student exam
answers of a selected exam that are completed and ready for grading">
    <option op_id="1" description="use the simple table to show all the student exam answers of a selected exam, for
those un-graded exam answers, show pointers to the grade action, for those graded exam answers, show pointers to
view the exam grade report">
      <vp_entity_dependency description="depends on the exam properties" vp_entity_id="54"
vp_entity_option_id="1">
        <!-- TeacherGradeExamPage -->
        <ref_CGU ref_CGU_id="87" ref_CGU_name="TeacherGradeExamPage_java_tag_expansion"
description="expand tags to create the page for a teacher to select an exam and start grading"/>
        <ref_CGU ref_CGU_id="88" ref_CGU_name="TeacherGradeExamPage_html_tag_expansion"
description="expand tags to create html page for a teacher to select and exam and start grading"/>
      </vp_entity_dependency>
      <vp_application_dependency description="there should be an action to view the grade for a selected exam
answer" vp_application_id="2031" variant_element_id="1"/>
    </option>
  </vp_application>

  <vp_application vp_id="2031" vp_name="TeacherGradeExamAnswer" description="a teacher opens a student
answer to a selected exam, enters score based on the student answers to questions, and submit the grade
information">
    <option op_id="1" description="for each question type, show correct answer, student answer and allow user input
for the score, then all the scores are submitted">
      <vp_entity_dependency description="each question type needs its own rendering mechanism for correct answer,
student answer, and input field for the score" vp_entity_id="46" vp_entity_option_id="1">
        <!-- TeacherGradeExamAnswerPage -->
        <ref_CGU ref_CGU_id="83" ref_CGU_name="TeacherGradeExamAnswerPage_java_tag_expansion"
description="expand tags to create page for a teacher to grade an exam answer"/>
        <ref_CGU ref_CGU_id="84" ref_CGU_name="TeacherGradeExamAnswerPage_html_file_copy"
description="expand tags to create html page for a teacher to grade an exam answer"/>
        <!-- TeacherGradeExamAnswerPage$GradeMcQuestionPanel -->
        <ref_CGU ref_CGU_id="85"
ref_CGU_name="TeacherGradeExamAnswerPage_GradeMCQuestionPanel_html_file_copy" description="copy
template content to create panel for grading MC questions"/>
        <!-- TeacherGradeExamAnswerPage$GradeTfQuestionPanel -->
        <ref_CGU ref_CGU_id="86"
ref_CGU_name="TeacherGradeExamAnswerPage_GradeTFQuestionPanel_html_file_copy" description="copy
template content to create panel for grading TF questions"/>
      </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2032" vp_name="ViewExamAnswerGrade" description="view the graded exam answer
submitted by a student and compare with the correct answer">
    <option op_id="1" description="for each question type, show correct answer, student answer and score, it might
include grader comments, etc. as well">
      <vp_entity_dependency description="each question type needs it own rendering mechanism for correct answer,
student answer, score and other related information" vp_entity_id="46" vp_entity_option_id="1">
        <!-- ViewExamAnswerGradePage -->
        <ref_CGU ref_CGU_id="89" ref_CGU_name="ViewExamAnswerGradePage_java_tag_expansion"
description="expand tags to create page to view exam answer grades"/>
        <ref_CGU ref_CGU_id="90" ref_CGU_name="ViewExamAnswerGradePage_html_file_copy"
description="expand tags to create html page to view exam answer grades"/>
        <!-- ViewExamAnswerGradePage$ViewMcQuestionGradePanel -->
        <ref_CGU ref_CGU_id="91"
ref_CGU_name="ViewExamAnswerGradePage_ViewMCQuestionGradePanel_html_file_copy" description="copy
template to create view MC question grade panel"/>
        <!-- ViewExamAnswerGradePage$ViewTfQuestionGradePanel -->
        <ref_CGU ref_CGU_id="92"
ref_CGU_name="ViewExamAnswerGradePage_ViewTFQuestionGradePanel_html_file_copy" description="copy
template to create view TF question grade panel"/>
      </vp_entity_dependency>
    </option>
  </vp_application>

  <vp_application vp_id="2033" vp_name="DeleteExamConfirm" description="delete an existing exam in the system
">
    <option op_id="1" description="asks for user confirmation before deleting an existing exam">
      <vp_entity_dependency vp_entity_id="300" vp_entity_option_id="1" description="exam properties to show on the
delete exam confirmation page">
        <!-- DeleteExamConfirmPage -->
        <ref_CGU ref_CGU_id="51" ref_CGU_name="DeleteExamConfirmPage_java_tag_expansion"
description="expand tags for DeleteExamConfirmPage"/>
        <ref_CGU ref_CGU_id="52" ref_CGU_name="DeleteExamConfirmPage_html_tag_expansion"
description="expand tags for DeleteExamConfirm html page"/>
      </vp_entity_dependency>

    </option>
  </vp_application>


<!-- END OF VP APPLICATIONS -->

<!-- START OF CODE GENERATION UNITS -->


  <code_generation_unit unit_id="1" unit_name="Subclass_Entity_string_swap" description="swap strings for global
parameters in an entity that extends a parent entity definition"
template_location="edu/ucsc/cse/exam/datamodel/$Entity$$.java.tmpl.Subclass"
template_param_lookup_type="Function" template_param_lookup_value="findAllSubclassEntityNames">
    <string_swap original_string="Entity">
      <replacement_content lookup_type="Dictionary">entityDefs.$Entity$$.name</replacement_content>
    </string_swap>
    <string_swap original_string="EntitySuper">
      <replacement_content lookup_type="Dictionary">entityDefs.$Entity$$.parent</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="2" unit_name="Subclass_Entity_tag_expansion" description="generate fields
based on property definitions for an entity that extends a parent entity definition"
template_location="edu/ucsc/cse/exam/datamodel/$Entity$$.java.tmpl.Subclass"
template_param_lookup_type="Function" template_param_lookup_value="findAllSubclassEntityNames">
    <tag_expansion tag_id="field_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="propType">entityDefs.$Entity$$.props.propType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">entityDefs.$Entity$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="constructor">
```

```
      <parameter_content lookup_type="Dictionary"
parameter_name="propType">entityDefs.$Entity$$.props.propType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">entityDefs.$Entity$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="id_getter_setter">
    </tag_expansion>
    <tag_expansion tag_id="property_getter_setter">
      <parameter_content lookup_type="Dictionary"
parameter_name="propType">entityDefs.$Entity$$.props.propType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">entityDefs.$Entity$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="equals_method">
      <parameter_content lookup_type="Dictionary"
parameter_name="propType">entityDefs.$Entity$$.props.propType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">entityDefs.$Entity$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="hashCode_method">
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">entityDefs.$Entity$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="toString_method">
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">entityDefs.$Entity$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="compareTo_method">
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">entityDefs.$Entity$$.props.propName</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="3" unit_name="Nonsubclass_Entity_string_swap" description="swap strings for
global parameters in an entity that extends a parent entity definition"
template_location="edu/ucsc/cse/exam/datamodel/$Entity$$.java.tmpl.Nonsubclass"
template_param_lookup_type="Function" template_param_lookup_value="findAllNonSubclassEntityNames">
    <string_swap original_string="Entity">
      <replacement_content lookup_type="Dictionary">entityDefs.$Entity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="4" unit_name="Nonsubclass_Entity_tag_expansion" description="generate fields
based on property definitions for an entity that does not extend a parent entity definition"
template_location="edu/ucsc/cse/exam/datamodel/$Entity$$.java.tmpl.Nonsubclass"
template_param_lookup_type="Function" template_param_lookup_value="findAllNonSubclassEntityNames">
    <tag_expansion tag_id="field_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="propType">entityDefs.$Entity$$.props.propType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">entityDefs.$Entity$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="collection_field_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="colType">entityDefs.$Entity$$.collections.colType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="colMemberType">entityDefs.$Entity$$.collections.colMemberType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="colName">entityDefs.$Entity$$.collections.colName</parameter_content>
```

```
      <parameter_content lookup_type="Dictionary"
parameter_name="colTypeImpl">entityDefs.$Entity$$.collections.colTypeImpl</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="constructor">
      <parameter_content lookup_type="Dictionary"
parameter_name="propType">entityDefs.$Entity$$.props.propType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">entityDefs.$Entity$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="property_getter_setter">
      <parameter_content lookup_type="Dictionary"
parameter_name="propType">entityDefs.$Entity$$.props.propType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">entityDefs.$Entity$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="collection_property_getter_setter">
      <parameter_content lookup_type="Dictionary"
parameter_name="colType">entityDefs.$Entity$$.collections.colType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="colMemberType">entityDefs.$Entity$$.collections.colMemberType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="colName">entityDefs.$Entity$$.collections.colName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="equals_method">
      <parameter_content lookup_type="Dictionary"
parameter_name="propType">entityDefs.$Entity$$.props.propType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">entityDefs.$Entity$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="hashCode_method">
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">entityDefs.$Entity$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="toString_method">
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">entityDefs.$Entity$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="compareTo_method">
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">entityDefs.$Entity$$.props.propName</parameter_content>
    </tag_expansion>
  </code_generation_unit>


  <code_generation_unit unit_id="5" unit_name="User_Entity_hbm_snippet_join" description="join code snippets to
create the Hibernate mapping file to a parent entity"
template_location="edu/ucsc/cse/exam/datamodel/$Parent$$.hbm.xml.tmpl" template_param_lookup_type="String"
template_param_lookup_value="User">
    <snippet_join snippet_id="child_mapping_snippet">
      <snippet_file_location ref_CGU_ids="6 7" param_lookup_type="Function"
param_lookup_value="findSubclassUserEntityNames">
        edu/ucsc/cse/exam/datamodel/$Child$$.hbm.snippet
      </snippet_file_location>
    </snippet_join>
    <snippet_join snippet_id="parent_mapping_snippet">
      <snippet_file_location ref_CGU_ids="8 9" param_lookup_type="String" param_lookup_value="User">
        edu/ucsc/cse/exam/datamodel/$Parent$$.hbm.snippet
      </snippet_file_location>
    </snippet_join>
  </code_generation_unit>
  <code_generation_unit unit_id="6" unit_name="User_Child_Entity_hbm_snippet_string_swap" description="swap
strings for the child user entity Hibernate mapping definition"
template_location="edu/ucsc/cse/exam/datamodel/$Child$$.hbm.snippet.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
    <string_swap original_string="Child">
      <replacement_content lookup_type="Dictionary">userDefs.$Child$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="7" unit_name="User_Child_Entity_hbm_snippet_tag_expansion"
description="expand tags for the child user entity Hibernate mapping definition"
template_location="edu/ucsc/cse/exam/datamodel/$Child$$.hbm.snippet.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
  <tag_expansion tag_id="property_def">
    <parameter_content lookup_type="Dictionary"
parameter_name="propName">userDefs.$Child$$.props.propName</parameter_content>
    <parameter_content lookup_type="Dictionary"
parameter_name="propType">userDefs.$Child$$.props.propType</parameter_content>
    <parameter_content lookup_type="Dictionary"
parameter_name="lengthLimitUpper">userDefs.$Child$$.props.lengthLimitUpper</parameter_content>
  </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="8" unit_name="User_Entity_hbm_snippet_string_swap" description="swap strings
for the User entity Hibernate mapping definition"
template_location="edu/ucsc/cse/exam/datamodel/$Parent$$.hbm.snippet.tmpl"
template_param_lookup_type="String" template_param_lookup_value="User">
    <string_swap original_string="Parent">
      <replacement_content lookup_type="Dictionary">userDefs.$Parent$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="9" unit_name="User_Entity_hbm_snippet_tag_expansion" description="expand
tags for the User entity Hibernate mapping definition"
template_location="edu/ucsc/cse/exam/datamodel/$Parent$$.hbm.snippet.tmpl"
template_param_lookup_type="String" template_param_lookup_value="User">
    <tag_expansion tag_id="property_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">userDefs.$Parent$$.props.propName</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propType">userDefs.$Parent$$.props.propType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="lengthLimitUpper">userDefs.$Parent$$.props.lengthLimitUpper</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="collection_property_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="type">userDefs.$Parent$$.mappings.type</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="name">userDefs.$Parent$$.mappings.name</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="targetClass">userDefs.$Parent$$.mappings.targetClass</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="table">userDefs.$Parent$$.mappings.table</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="column">userDefs.$Parent$$.mappings.column</parameter_content>
    </tag_expansion>
  </code_generation_unit>
  <code_generation_unit unit_id="10" unit_name="Question_Entity_hbm_snippet_join" description="join code
snippets to create the Hiberante mapping file to a parent Question entity"
template_location="edu/ucsc/cse/exam/datamodel/$Parent$$.hbm.xml.tmpl" template_param_lookup_type="String"
template_param_lookup_value="Question">
    <snippet_join snippet_id="child_mapping_snippet">
      <snippet_file_location ref_CGU_ids="11 12" param_lookup_type="Function"
param_lookup_value="findSubclassQuestionEntityNames">
        edu/ucsc/cse/exam/datamodel/$Child$$.hbm.snippet
      </snippet_file_location>
    </snippet_join>
    <snippet_join snippet_id="parent_mapping_snippet">
      <snippet_file_location ref_CGU_ids="13 14" param_lookup_type="String" param_lookup_value="Question">
        edu/ucsc/cse/exam/datamodel/$Parent$$.hbm.snippet
      </snippet_file_location>
    </snippet_join>
  </code_generation_unit>

  <code_generation_unit unit_id="11" unit_name="Question_Child_Entity_hbm_snippet_string_swap"
description="swap strings for the child question entity Hiberante mapping definition"
template_location="edu/ucsc/cse/exam/datamodel/$Child$$.hbm.snippet.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <string_swap original_string="Child">
      <replacement_content lookup_type="Dictionary">questionDefs.$Child$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="12" unit_name="Question_Child_Entity_hbm_snippet_tag_expansion"
description="expand tags for the child user entity Hibernate mapping definition"
template_location="edu/ucsc/cse/exam/datamodel/$Child$$.hbm.snippet.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <tag_expansion tag_id="property_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">questionDefs.$Child$$.props.propName</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propType">questionDefs.$Child$$.props.propType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="lengthLimitUpper">questionDefs.$Child$$.props.lengthLimitUpper</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="13" unit_name="Question_Entity_hbm_snippet_string_swap" description="swap
strings for the Question entity Hibernate mapping definition"
template_location="edu/ucsc/cse/exam/datamodel/$Parent$$.hbm.snippet.tmpl"
template_param_lookup_type="String" template_param_lookup_value="Question">
    <string_swap original_string="Parent">
      <replacement_content lookup_type="Dictionary">questionDefs.$Parent$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="14" unit_name="Question_Entity_hbm_snippet_tag_expansion" description="tag
expansions for the Question entity Hibernate mapping definition"
template_location="edu/ucsc/cse/exam/datamodel/$Parent$$.hbm.snippet.tmpl"
template_param_lookup_type="String" template_param_lookup_value="Question">
    <tag_expansion tag_id="property_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">questionDefs.$Parent$$.props.propName</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propType">questionDefs.$Parent$$.props.propType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="lengthLimitUpper">questionDefs.$Parent$$.props.lengthLimitUpper</parameter_content>
```

```
    </tag_expansion>
    <tag_expansion tag_id="collection_property_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="type">questionDefs.$Parent$$.mappings.type</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="name">questionDefs.$Parent$$.mappings.name</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="targetClass">questionDefs.$Parent$$.mappings.targetClass</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="table">questionDefs.$Parent$$.mappings.table</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="column">questionDefs.$Parent$$.mappings.column</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="15" unit_name="User_Subclass_hbm_string_swap" description="swap strings for
the User subclass entities Hibernate mapping definitions"
template_location="edu/ucsc/cse/exam/datamodel/$Entity$$.hbm.xml.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
    <string_swap original_string="Entity">
      <replacement_content lookup_type="Dictionary">userDefs.$Entity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="16" unit_name="User_Subclass_hbm_tag_expansion" description="tag expansions
for the User sublcass entities Hibernate mapping definitions"
template_location="edu/ucsc/cse/exam/datamodel/$Entity$$.hbm.xml.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
    <tag_expansion tag_id="property_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">userDefs.$Entity$$.props.propName</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propType">userDefs.$Entity$$.props.propType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="lengthLimitUpper">userDefs.$Entity$$.props.lengthLimitUpper</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="collection_property_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="type">userDefs.$Entity$$.mappings.type</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="name">userDefs.$Entity$$.mappings.name</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="targetClass">userDefs.$Entity$$.mappings.targetClass</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="table">userDefs.$Entity$$.mappings.table</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="column">userDefs.$Entity$$.mappings.column</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="17" unit_name="Question_Subclass_hbm_string_swap" description="swap strings
for the Question subclass entities Hibernate mapping definitions"
template_location="edu/ucsc/cse/exam/datamodel/$Entity$$.hbm.xml.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <string_swap original_string="Entity">
      <replacement_content lookup_type="Dictionary">questionDefs.$Entity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>
  <code_generation_unit unit_id="18" unit_name="Question_Subclass_hbm_tag_expansion" description="tag
expansions for the Question sublcass entities Hibernate mapping definitions"
template_location="edu/ucsc/cse/exam/datamodel/$Entity$$.hbm.xml.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <tag_expansion tag_id="property_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">questionDefs.$Entity$$.props.propName</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propType">questionDefs.$Entity$$.props.propType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="lengthLimitUpper">questionDefs.$Entity$$.props.lengthLimitUpper</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="collection_property_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="type">questionDefs.$Entity$$.mappings.type</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="name">questionDefs.$Entity$$.mappings.name</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="targetClass">questionDefs.$Entity$$.mappings.targetClass</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="table">questionDefs.$Entity$$.mappings.table</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="column">questionDefs.$Entity$$.mappings.column</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="19" unit_name="Exam_Entity_hbm_string_swap" description="swap strings for the
Exam entity Hibernate mapping definitions"
template_location="edu/ucsc/cse/exam/datamodel/$Entity$$.hbm.xml.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findExamEntityNames">
    <string_swap original_string="Entity">
      <replacement_content lookup_type="Dictionary">examDefs.$Entity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="20" unit_name="Exam_Entity_hbm_tag_expansion" description="tag expansions
for the Exam entity Hibernate mapping definitions"
template_location="edu/ucsc/cse/exam/datamodel/$Entity$$.hbm.xml.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findExamEntityNames">
    <tag_expansion tag_id="property_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">examDefs.$Entity$$.props.propName</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propType">examDefs.$Entity$$.props.propType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="lengthLimitUpper">examDefs.$Entity$$.props.lengthLimitUpper</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="collection_property_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="type">examDefs.$Entity$$.mappings.type</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="name">examDefs.$Entity$$.mappings.name</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="targetClass">examDefs.$Entity$$.mappings.targetClass</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="table">examDefs.$Entity$$.mappings.table</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="column">examDefs.$Entity$$.mappings.column</parameter_content>
    </tag_expansion>
  </code_generation_unit>
  <code_generation_unit unit_id="21" unit_name="ExamAnswer_Entity_hbm_string_swap" description="swap strings
for the ExamAnswerPerStudent entity Hibernate mapping definitions"
template_location="edu/ucsc/cse/exam/datamodel/$Entity$$.hbm.xml.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findExamAnswerEntityNames">
    <string_swap original_string="Entity">
      <replacement_content lookup_type="Dictionary">examAnswerDefs.$Entity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="22" unit_name="ExamAnswer_Entity_Subclass_hbm_tag_expansion"
description="tag expansions for the ExamAnswerPerStudent entity Hibernate mapping definitions"
template_location="edu/ucsc/cse/exam/datamodel/$Entity$$.hbm.xml.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findExamAnswerEntityNames">
    <tag_expansion tag_id="property_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">examAnswerDefs.$Entity$$.props.propName</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propType">examAnswerDefs.$Entity$$.props.propType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="lengthLimitUpper">examAnswerDefs.$Entity$$.props.lengthLimitUpper</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="collection_property_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="type">examAnswerDefs.$Entity$$.mappings.type</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="name">examAnswerDefs.$Entity$$.mappings.name</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="targetClass">examAnswerDefs.$Entity$$.mappings.targetClass</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="table">examAnswerDefs.$Entity$$.mappings.table</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="column">examAnswerDefs.$Entity$$.mappings.column</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="23" unit_name="QuestionAnswer_Entity_hbm_string_swap" description="swap
strings for the QuestionAnswerPerStudent entity Hibernate mapping definitions"
template_location="edu/ucsc/cse/exam/datamodel/$Entity$$.hbm.xml.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findQuestionAnswerEntityNames">
    <string_swap original_string="Entity">
      <replacement_content lookup_type="Dictionary">questionAnswerDefs.$Entity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="24" unit_name="QuestionAnswer_Entity_hbm_tag_expansion" description="tag
expansions for the QuestionAnswerPerStudent entity Hibernate mapping definitions"
template_location="edu/ucsc/cse/exam/datamodel/$Entity$$.hbm.xml.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findQuestionAnswerEntityNames">
    <tag_expansion tag_id="property_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">questionAnswerDefs.$Entity$$.props.propName</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propType">questionAnswerDefs.$Entity$$.props.propType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="lengthLimitUpper">questionAnswerDefs.$Entity$$.props.lengthLimitUpper</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="collection_property_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="type">questionAnswerDefs.$Entity$$.mappings.type</parameter_content>
```

```
      <parameter_content lookup_type="Dictionary"
parameter_name="name">questionAnswerDefs.$Entity$$.mappings.name</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="targetClass">questionAnswerDefs.$Entity$$.mappings.targetClass</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="table">questionAnswerDefs.$Entity$$.mappings.table</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="column">questionAnswerDefs.$Entity$$.mappings.column</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit  unit_id="25" unit_name="createdb_sql_snippet_join" description="join code snippets to
generate an SQL file that will create all the tables" template_location="sql/createdb.sql.tmpl">
    <snippet_join snippet_id="parent_tables_snippet">
      <snippet_file_location ref_CGU_ids="26 27" param_lookup_type="Function"
param_lookup_value="findAllParentclassEntityNames">
        sql/$Entity$$.sql.snippet
      </snippet_file_location>
    </snippet_join>
    <snippet_join snippet_id="child_tables_snippet">
      <snippet_file_location ref_CGU_ids="28 29" param_lookup_type="Function"
param_lookup_value="findAllSubclassEntityNames">
        sql/$SubEntity$$.sql.snippet
      </snippet_file_location>
    </snippet_join>
    <snippet_join snippet_id="exam_table_snippet">
      <snippet_file_location ref_CGU_ids="30 31" param_lookup_type="String" param_lookup_value="Exam">
        sql/$Entity$$.sql.snippet
      </snippet_file_location>
    </snippet_join>
    <snippet_join snippet_id="examAnswerPerStudent_table_snippet">
      <snippet_file_location ref_CGU_ids="32">
        sql/ExamAnswerPerStudent.sql.snippet
      </snippet_file_location>
    </snippet_join>
    <snippet_join snippet_id="questionAnswerPerStudent_table_snippet">
      <snippet_file_location ref_CGU_ids="33">
        sql/QuestionAnswerPerStudent.sql.snippet
      </snippet_file_location>
    </snippet_join>
  </code_generation_unit>

  <code_generation_unit unit_id="26" unit_name="parent_tables_string_swap" description="swap strings for the
parent table SQL definition" template_location="sql/$Entity$$.sql.snippet.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findAllParentclassEntityNames">
    <string_swap original_string="Entity">
      <replacement_content lookup_type="Dictionary">entityDefs.$Entity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="27" unit_name="parent_tables_tag_expansion" description="expand tags for the
parent table SQL definition" template_location="sql/$Entity$$.sql.snippet.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findAllParentclassEntityNames">
    <tag_expansion tag_id="column_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">entityDefs.$Entity$$.props.propName</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propSQLType">entityDefs.$Entity$$.props.propSQLType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="lengthLimitUpper">entityDefs.$Entity$$.props.lengthLimitUpper</parameter_content>
```

```
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="28" unit_name="child_tables_string_swap" description="swap strings for the child
table SQL definition" template_location="sql/$SubEntity$$.sql.snippet.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findAllSubclassEntityNames">
    <string_swap original_string="SubEntity">
      <replacement_content lookup_type="Dictionary">entityDefs.$SubEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="29" unit_name="child_tables_tag_expansion" description="expand tags for the
child table SQL definition" template_location="sql/$SubEntity$$.sql.snippet.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findAllSubclassEntityNames">
    <tag_expansion tag_id="column_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">entityDefs.$SubEntity$$.props.propName</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propSQLType">entityDefs.$SubEntity$$.props.propSQLType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="lengthLimitUpper">entityDefs.$SubEntity$$.props.lengthLimitUpper</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="30" unit_name="exam_table_string_swap" description="swap strings for the exam
table SQL definition" template_location="sql/$Entity$$.sql.snippet.tmpl" template_param_lookup_type="Function"
template_param_lookup_value="findExamEntityNames">
    <string_swap original_string="Entity">
      <replacement_content lookup_type="Dictionary">examDefs.$Entity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="31" unit_name="exam_table_tag_expansion" description="expand tags for the
exam table SQL definition" template_location="sql/$Entity$$.sql.snippet.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findExamEntityNames">
    <tag_expansion tag_id="column_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">examDefs.$Entity$$.props.propName</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propSQLType">examDefs.$Entity$$.props.propSQLType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="lengthLimitUpper">examDefs.$Entity$$.props.lengthLimitUpper</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="32" unit_name="examAnswerPerStudent_table_tag_expansion"
description="expand tags for the exam answer table SQL definition"
template_location="sql/ExamAnswerPerStudent.sql.snippet.tmpl">
    <tag_expansion tag_id="column_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">examAnswerDefs.ExamAnswerPerStudent.props.propName</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propSQLType">examAnswerDefs.ExamAnswerPerStudent.props.propSQLType</parameter_con
tent>
      <parameter_content lookup_type="Dictionary"
parameter_name="lengthLimitUpper">examAnswerDefs.ExamAnswerPerStudent.props.lengthLimitUpper</paramet
er_content>
    </tag_expansion>
  </code_generation_unit>
  <code_generation_unit unit_id="33" unit_name="questionAnswerPerStudent_table_tag_expansion"
description="expand tags for the question answer table SQL definition"
template_location="sql/QuestionAnswerPerStudent.sql.snippet.tmpl">
    <tag_expansion tag_id="column_def">
      <parameter_content lookup_type="Dictionary"
parameter_name="propName">questionAnswerDefs.QuestionAnswerPerStudent.props.propName</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="propSQLType">questionAnswerDefs.QuestionAnswerPerStudent.props.propSQLType</parameter_content>
      <parameter_content lookup_type="Dictionary"
parameter_name="lengthLimitUpper">questionAnswerDefs.QuestionAnswerPerStudent.props.lengthLimitUpper</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="34" unit_name="spring_config_tag_expansion" description="expand tags to create
Spring applicationContext.xml configuration file" template_location="applicationContext.xml.tmpl">
    <tag_expansion tag_id="resource_mapping">
      <parameter_content lookup_type="Function"
parameter_name="nonSubclassEntityName">findAllNonSubclassEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="dao_def">
      <parameter_content lookup_type="Function"
parameter_name="entityName">findAllNonParentclassEntityNames</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="35" unit_name="ExamMainApplication_java_tag_expansion" description="expand
tags to create ExamMainApplication, the entry point for the whole online exam application"
template_location="edu/ucsc/cse/exam/ExamMainApplication.java.tmpl">
    <tag_expansion tag_id="dao_import">
      <parameter_content lookup_type="Function"
parameter_name="entityName">findAllNonParentclassEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="dao_def">
      <parameter_content lookup_type="Function"
parameter_name="entityName">findAllNonParentclassEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="init_method">
      <parameter_content lookup_type="Function"
parameter_name="userEntityName">findNonParentclassUserEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="getDao">
      <parameter_content lookup_type="Function"
parameter_name="entityName">findAllNonParentclassEntityNames</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="36" unit_name="ExamSession_java_tag_expansion" description="expand tags to
create the ExamSession class that represents the session class"
template_location="edu/ucsc/cse/exam/auth/ExamSession.java.tmpl">
    <tag_expansion tag_id="entity_import">
      <parameter_content lookup_type="Function"
parameter_name="userEntityName">findNonParentclassUserEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="entity_def">
      <parameter_content lookup_type="Function"
parameter_name="userEntityName">findNonParentclassUserEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="entity_operation">
      <parameter_content lookup_type="Function"
parameter_name="userEntityName">findNonParentclassUserEntityNames</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="37" unit_name="UserLoginPage_java_tag_expansion" description="expand tags to
create UserLoginPage class that represents the login page for the system"
template_location="edu/ucsc/cse/exam/auth/UserLoginPage.java.tmpl">
    <tag_expansion tag_id="lib_import">
      <parameter_content lookup_type="Function"
parameter_name="userEntityName">findNonParentclassUserEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="editUser_lib_import">
      <parameter_content lookup_type="Function"
parameter_name="editableUserEntityName">findEditableUserEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="page_redirection">
      <parameter_content lookup_type="Function"
parameter_name="userEntityName">findNonParentclassUserEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="new_user">
      <parameter_content lookup_type="Function"
parameter_name="userEntityName">findNonParentclassUserEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="get_dao">
      <parameter_content lookup_type="Function"
parameter_name="userEntityName">findNonParentclassUserEntityNames</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="38" unit_name="UserLoginPage_html_tag_expansion" description="expand tags to
create UserLoginPage html that represents the login page for the system"
template_location="edu/ucsc/cse/exam/auth/UserLoginPage.html.tmpl">
    <tag_expansion tag_id="user_reg">
      <parameter_content lookup_type="Function"
parameter_name="editableUserEntityName">findEditableUserEntityNames</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="39" unit_name="UserDAO_java_string_swap" description="swap strings for user
entity names to create user dao classes" template_location="edu/ucsc/cse/exam/dao/$EntityName$$DAO.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
    <string_swap original_string="EntityName">
      <replacement_content lookup_type="Dictionary">userDefs.$EntityName$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="40" unit_name="UserDAO_java_tag_expansion" description="expand tags for user
entity names to create user dao classes" template_location="edu/ucsc/cse/exam/dao/$EntityName$$DAO.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
    <tag_expansion tag_id="super_prop_const">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">userDefs.User.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="prop_const">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">userDefs.$EntityName$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="find_by_super_property">
```

```
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">userDefs.User.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="find_by_property">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">userDefs.$EntityName$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="filter_super_property">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">userDefs.User.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="filter_property">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">userDefs.$EntityName$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="match_filter_super_parameter">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">userDefs.User.props.propName</parameter_content>
      <parameter_content parameter_name="propType"
lookup_type="Dictionary">userDefs.User.props.propType</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="match_filter_parameter">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">userDefs.$EntityName$$.props.propName</parameter_content>
      <parameter_content parameter_name="propType"
lookup_type="Dictionary">userDefs.$EntityName$$.props.propType</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="41" unit_name="QuestionDAO_java_string_swap" description="swap strings for
question entity names to create question dao classes"
template_location="edu/ucsc/cse/exam/dao/$EntityName$$DAO.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <string_swap original_string="EntityName">
      <replacement_content lookup_type="Dictionary">questionDefs.$EntityName$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="42" unit_name="QuestionDAO_java_tag_expansion" description="expand tags for
question entity names to create question dao classes"
template_location="edu/ucsc/cse/exam/dao/$EntityName$$DAO.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <tag_expansion tag_id="super_prop_const">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">questionDefs.Question.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="prop_const">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">questionDefs.$EntityName$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="find_by_super_property">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">questionDefs.Question.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="find_by_property">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">questionDefs.$EntityName$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="filter_super_property">
```

```
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">questionDefs.Question.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="filter_property">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">questionDefs.$EntityName$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="match_filter_super_parameter">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">questionDefs.Question.props.propName</parameter_content>
      <parameter_content parameter_name="propType"
lookup_type="Dictionary">questionDefs.Question.props.propType</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="match_filter_parameter">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">questionDefs.$EntityName$$.props.propName</parameter_content>
      <parameter_content parameter_name="propType"
lookup_type="Dictionary">questionDefs.$EntityName$$.props.propType</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="43" unit_name="ExamDAO_java_string_swap" description="swap strings for exam
entity names to create exam dao classes"
template_location="edu/ucsc/cse/exam/dao/$NonInheritanceEntityName$$DAO.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findExamEntityNames">
    <string_swap original_string="NonInheritanceEntityName">
      <replacement_content
lookup_type="Dictionary">examDefs.$NonInheritanceEntityName$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="44" unit_name="ExamDAO_java_tag_expansion" description="expand tags for
exam entity names to create exam dao classes"
template_location="edu/ucsc/cse/exam/dao/$NonInheritanceEntityName$$DAO.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findExamEntityNames">
    <tag_expansion tag_id="prop_const">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">examDefs.$NonInheritanceEntityName$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="find_by_property">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">examDefs.$NonInheritanceEntityName$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="filter_property">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">examDefs.$NonInheritanceEntityName$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="match_filter_parameter">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">examDefs.$NonInheritanceEntityName$$.props.propName</parameter_content>
      <parameter_content parameter_name="propType"
lookup_type="Dictionary">examDefs.$NonInheritanceEntityName$$.props.propType</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="45" unit_name="ExamAnswerPerStudentDAO_java_string_swap"
description="swap strings for ExamAnswerPerStudent entity names to create ExamAnswerPerStudent dao classes"
template_location="edu/ucsc/cse/exam/dao/$NonInheritanceEntityName$$DAO.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findExamAnswerEntityNames">
    <string_swap original_string="NonInheritanceEntityName">
```

```
      <replacement_content
lookup_type="Dictionary">examAnswerDefs.$NonInheritanceEntityName$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="46" unit_name="ExamAnswerPerStudentDAO_java_tag_expansion"
description="expand tags for ExamAnswerPerStudent entity names to create ExamAnswerPerStudent dao classes"
template_location="edu/ucsc/cse/exam/dao/$NonInheritanceEntityName$$DAO.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findExamAnswerEntityNames">
    <tag_expansion tag_id="prop_const">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">examAnswerDefs.$NonInheritanceEntityName$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="find_by_property">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">examAnswerDefs.$NonInheritanceEntityName$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="filter_property">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">examAnswerDefs.$NonInheritanceEntityName$$.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="match_filter_parameter">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">examAnswerDefs.$NonInheritanceEntityName$$.props.propName</parameter_content>
      <parameter_content parameter_name="propType"
lookup_type="Dictionary">examAnswerDefs.$NonInheritanceEntityName$$.props.propType</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="47" unit_name="QuestionAnswerPerStudentDAO_java_string_swap"
description="swap strings for QuestionAnswerPerStudent entity names to create QuestionAnswerPerStudent dao
classes" template_location="edu/ucsc/cse/exam/dao/$NonInheritanceEntityName$$DAO.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findQuestionAnswerEntityNames">
    <string_swap original_string="NonInheritanceEntityName">
      <replacement_content
lookup_type="Dictionary">questionAnswerDefs.$NonInheritanceEntityName$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="48" unit_name="QuestionAnswerPerStudentDAO_java_tag_expansion"
description="expand tags for QuestionAnswerPerStudent entity names to create QuestionAnswerPerStudent dao
classes" template_location="edu/ucsc/cse/exam/dao/$NonInheritanceEntityName$$DAO.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findQuestionAnswerEntityNames">
    <tag_expansion tag_id="prop_const">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">questionAnswerDefs.$NonInheritanceEntityName$$.props.propName</parameter_conten
t>
    </tag_expansion>
    <tag_expansion tag_id="find_by_property">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">questionAnswerDefs.$NonInheritanceEntityName$$.props.propName</parameter_conten
t>
    </tag_expansion>
    <tag_expansion tag_id="filter_property">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">questionAnswerDefs.$NonInheritanceEntityName$$.props.propName</parameter_conten
t>
    </tag_expansion>
    <tag_expansion tag_id="match_filter_parameter">
```

```
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">questionAnswerDefs.$NonInheritanceEntityName$$.props.propName</parameter_conten
t>
      <parameter_content parameter_name="propType"
lookup_type="Dictionary">questionAnswerDefs.$NonInheritanceEntityName$$.props.propType</parameter_content>
    </tag_expansion>
  </code_generation_unit>


  <code_generation_unit unit_id="49" unit_name="BasePage_java_tag_expansion" description="expand tags for
BasePage java class" template_location="edu/ucsc/cse/exam/web/BasePage.java.tmpl">
    <tag_expansion tag_id="hompage_redirection">
      <parameter_content parameter_name="userEntityName" lookup_type="Function">
        findSubclassUserEntityNames
      </parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="logout_redirection">
      <parameter_content parameter_name="userEntityName" lookup_type="Function">
        findSubclassUserEntityNames
      </parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="get_user_dao">
      <parameter_content parameter_name="userEntityName" lookup_type="Function">
        findSubclassUserEntityNames
      </parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="get_question_dao">
      <parameter_content parameter_name="questionEntityName" lookup_type="Function">
        findSubclassQuestionEntityNames
      </parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="get_exam_dao">
      <parameter_content parameter_name="examEntityName" lookup_type="Function">
        findExamEntityNames
      </parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="get_examAnswer_dao">
      <parameter_content parameter_name="examAnswerEntityName" lookup_type="Function">
        findExamAnswerEntityNames
      </parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="get_questionAnswer_dao">
      <parameter_content parameter_name="questionAnswerEntityName" lookup_type="Function">
        findQuestionAnswerEntityNames
      </parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="50" unit_name="Util_java_tag_expansion" description="expand tags for Util java
class" template_location="edu/ucsc/cse/exam/web/Util.java.tmpl">
    <tag_expansion tag_id="question_constant">
      <parameter_content parameter_name="questionEntityName"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
      <parameter_content parameter_name="index"
lookup_type="Function">findSubclassQuestionEntityIndices</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="user_constant">
      <parameter_content parameter_name="userEntityName"
lookup_type="Function">findSubclassUserEntityNames</parameter_content>
```

```
      <parameter_content parameter_name="index"
lookup_type="Function">findSubclassUserEntityIndices</parameter_content>
    </tag_expansion>
  </code_generation_unit>


  <code_generation_unit unit_id="51" unit_name="DeleteExamConfirmPage_java_tag_expansion"
description="expand tags for DeleteExamConfirmPage class that asks for confirmation before deleting an exam"
template_location="edu/ucsc/cse/exam/web/exam/DeleteExamConfirmPage.java.tmpl">
    <tag_expansion tag_id="form_add_exam_props">
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.Exam.DeleteExamConfirm.props.GUIWidget</parameter_content>
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.Exam.DeleteExamConfirm.props.propName</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="52" unit_name="DeleteExamConfirmPage_html_tag_expansion"
description="expand tags for DeleteExamConfirmPage html that renders the page"
template_location="edu/ucsc/cse/exam/web/exam/DeleteExamConfirmPage.html.tmpl">
    <tag_expansion tag_id="exam_prop_table_row">
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.Exam.DeleteExamConfirm.props.GUIWidget</parameter_content>
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.Exam.DeleteExamConfirm.props.propName</parameter_content>
    </tag_expansion>
  </code_generation_unit>



  <code_generation_unit unit_id="53" unit_name="EditExamAddQuestionPage_java_string_swap" description="swap
string for question entity name in editExamAddQuestionPage classes"
template_location="edu/ucsc/cse/exam/web/exam/EditExamAdd$QuestionEntity.CamelCase$$Page.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <string_swap original_string="QuestionEntity">
      <replacement_content lookup_type="Dictionary">questionDefs.$QuestionEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="54" unit_name="EditExamAddQuestionPage_java_tag_expansion"
description="expand tags in editExamAddQuestionPage classes"
template_location="edu/ucsc/cse/exam/web/exam/EditExamAdd$QuestionEntity.CamelCase$$Page.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <tag_expansion tag_id="question_item_props">
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.$QuestionEntity$$.EditExamAdd$QuestionEntity$$.props.GUIWidget</paramete
r_content>
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.$QuestionEntity$$.EditExamAdd$QuestionEntity$$.props.propName</parameter
_content>
    </tag_expansion>
  </code_generation_unit>



  <code_generation_unit unit_id="55" unit_name="EditExamAddQuestionPage_html_string_swap"
description="swap string for question entity name in editExamAddQuestionPage html pages"
template_location="edu/ucsc/cse/exam/web/exam/EditExamAdd$QuestionEntity.CamelCase$$Page.html.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <string_swap original_string="QuestionEntity">
```

```
     <replacement_content lookup_type="Dictionary">questionDefs.$QuestionEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="56" unit_name="EditExamAddQuestionPage_html_tag_expansion"
description="expand tags in editExamAddQuestionPage html pages"
template_location="edu/ucsc/cse/exam/web/exam/EditExamAdd$QuestionEntity.CamelCase$$Page.html.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <tag_expansion tag_id="table_header">
     <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.$QuestionEntity$$.EditExamAdd$QuestionEntity$$.props.propName</parameter
_content>
    </tag_expansion>
    <tag_expansion tag_id="table_item">
     <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.$QuestionEntity$$.EditExamAdd$QuestionEntity$$.props.propName</parameter
_content>
    </tag_expansion>
  </code_generation_unit>



  <code_generation_unit unit_id="57" unit_name="EditExamAddQuestionPageActionPanel_html_file_copy"
description="replace parameters in the template file location and copy template content into the newly generated files
for action panel html pages"
template_location="edu/ucsc/cse/exam/web/exam/EditExamAdd$QuestionEntity.CamelCase$$Page$CreateExam$
QuestionEntity$$ActionPanel.html.tmpl" template_param_lookup_type="Function"
template_param_lookup_value="findSubclassQuestionEntityNames">
    <file_copy>nothing to do</file_copy>
  </code_generation_unit>



  <code_generation_unit unit_id="58" unit_name="EditExamPage_java_snippet_join" description="join code snippets
to create some content for EditExamPage class"
template_location="edu/ucsc/cse/exam/web/exam/EditExamPage.java.tmpl">
    <snippet_join snippet_id="question_data_view">
     <snippet_file_location ref_CGU_ids="59 60" param_lookup_type="Function"
param_lookup_value="findSubclassQuestionEntityNames">
       edu/ucsc/cse/exam/web/exam/EditExamPage$QuestionEntity$$DataView.java.snippet
     </snippet_file_location>
    </snippet_join>
    <snippet_join snippet_id="question_data_view2">
     <snippet_file_location ref_CGU_ids="59 60" param_lookup_type="Function"
param_lookup_value="findSubclassQuestionEntityNames">
       edu/ucsc/cse/exam/web/exam/EditExamPage$QuestionEntity$$DataView.java.snippet
     </snippet_file_location>
    </snippet_join>
  </code_generation_unit>

  <code_generation_unit unit_id="59" unit_name="EditExamPageQuestionDataView_snippet_string_swap"
description="swap QuestionEntity strings to create snippets to be used in EditExamPage"
template_location="edu/ucsc/cse/exam/web/exam/EditExamPage$QuestionEntity$$DataView.java.snippet.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <string_swap original_string="QuestionEntity">
     <replacement_content lookup_type="Dictionary">questionDefs.$QuestionEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>
  <code_generation_unit unit_id="60" unit_name="EditExamPageQuestionDataView_snippet_tag_expansion"
description="expand tags to create snippets to be used in EditExamPage"
template_location="edu/ucsc/cse/exam/web/exam/EditExamPage$QuestionEntity$$DataView.java.snippet.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <tag_expansion tag_id="question_props">
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.$QuestionEntity$$.EditExamAdd$QuestionEntity$$.props.GUIWidget</parameter_content>
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.$QuestionEntity$$.EditExamAdd$QuestionEntity$$.props.propName</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="61" unit_name="EditExamPage_java_tag_expansion" description="expand tags in
EditExamPage" template_location="edu/ucsc/cse/exam/web/exam/EditExamPage.java.tmpl">
    <tag_expansion tag_id="question_selections">
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="question_ids">
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="add_question">
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="collect_question_selections">
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="exam_prop_widgets">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.Exam.EditExam.props.propName</parameter_content>
      <parameter_content parameter_name="propType"
lookup_type="Dictionary">appDefs.Exam.EditExam.props.propType</parameter_content>
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.Exam.EditExam.props.GUIWidget</parameter_content>
      <parameter_content parameter_name="lengthLimitUpper"
lookup_type="Dictionary">appDefs.Exam.EditExam.props.lengthLimitUpper</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="exam_confirm_items">
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="add_selections_from_map">
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="exam_prop_widgets2">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.Exam.EditExam.props.propName</parameter_content>
      <parameter_content parameter_name="propType"
lookup_type="Dictionary">appDefs.Exam.EditExam.props.propType</parameter_content>
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.Exam.EditExam.props.GUIWidget</parameter_content>
      <parameter_content parameter_name="lengthLimitUpper"
lookup_type="Dictionary">appDefs.Exam.EditExam.props.lengthLimitUpper</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="exam_confirm_items2">
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="question_item_class">
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
  </code_generation_unit>


  <code_generation_unit unit_id="62" unit_name="EditExamPage_html_snippet_join" description="join code snippets
to create EditExamPage html file" template_location="edu/ucsc/cse/exam/web/exam/EditExamPage.html.tmpl">
    <snippet_join snippet_id="question_data_table">
      <snippet_file_location ref_CGU_ids="63 64" param_lookup_type="Function"
param_lookup_value="findSubclassQuestionEntityNames">
        edu/ucsc/cse/exam/web/exam/EditExamPage$QuestionEntity$$DataView.html.snippet
      </snippet_file_location>
    </snippet_join>
  </code_generation_unit>

  <code_generation_unit unit_id="63" unit_name="EditExamPageQuestionDataView_html_snippet_string_swap"
description="swap QuestionEntity strings to create snippets to be used in EditExamPage html file"
template_location="edu/ucsc/cse/exam/web/exam/EditExamPage$QuestionEntity$$DataView.html.snippet.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <string_swap original_string="QuestionEntity">
      <replacement_content lookup_type="Dictionary">questionDefs.$QuestionEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="64" unit_name="EditExamPageQuestionDataView_html_snippet_tag_expansion"
description="expand tags to create snippets to be used in EditExamPage html file"
template_location="edu/ucsc/cse/exam/web/exam/EditExamPage$QuestionEntity$$DataView.html.snippet.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <tag_expansion tag_id="table_header">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.$QuestionEntity$$.EditExamAdd$QuestionEntity$$.props.propName</parameter
_content>
    </tag_expansion>
    <tag_expansion tag_id="table_item">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.$QuestionEntity$$.EditExamAdd$QuestionEntity$$.props.propName</parameter
_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="65" unit_name="EditExamPage_html_tag_expansion" description="expand tags in
EditExamPage html file" template_location="edu/ucsc/cse/exam/web/exam/EditExamPage.html.tmpl">
    <tag_expansion tag_id="exam_prop_widgets">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.Exam.EditExam.props.propName</parameter_content>
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.Exam.EditExam.props.GUIWidget</parameter_content>
    </tag_expansion>
  </code_generation_unit>



  <code_generation_unit unit_id="66" unit_name="ListExamPage_java_tag_expansion" description="expand tags in
ListExamPage java file" template_location="edu/ucsc/cse/exam/web/exam/ListExamPage.java.tmpl">
```

```
    <tag_expansion tag_id="exam_props">
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.Exam.ListExam.props.GUIWidget</parameter_content>
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.Exam.ListExam.props.propName</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="67" unit_name="ListExamPage_html_file_copy" description="copy template
content into the newly generated file for list exam html pages"
template_location="edu/ucsc/cse/exam/web/exam/ListExamPage.html.tmpl">
    <file_copy>nothing to do</file_copy>
  </code_generation_unit>

  <code_generation_unit unit_id="68" unit_name="ListExamPage_action_html_file_copy" description="copy template
content into the newly generated file for list exam action panel"
template_location="edu/ucsc/cse/exam/web/exam/ListExamPage$ExamActionPanel.html.tmpl">
    <file_copy>copy template file</file_copy>
  </code_generation_unit>

  <code_generation_unit unit_id="69" unit_name="PreviewExamPage_java_tag_expansion" description="expand
tags to create preview exam page java file"
template_location="edu/ucsc/cse/exam/web/exam/PreviewExamPage.java.tmpl">
    <tag_expansion tag_id="preview_question_panel">
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="70" unit_name="PreviewExamPage_html_file_copy" description="copy template
file to create preview exam page html file"
template_location="edu/ucsc/cse/exam/web/exam/PreviewExamPage.html.tmpl">
    <file_copy>copy template file</file_copy>
  </code_generation_unit>

  <code_generation_unit unit_id="71" unit_name="PreviewQuestionPage_java_tag_expansion" description="expand
tags to create preview question page java file"
template_location="edu/ucsc/cse/exam/web/question/PreviewQuestionPage.java.tmpl">
    <tag_expansion tag_id="preview_question_panel">
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="72" unit_name="PreviewQuestionPage_html_file_copy" description="copy
template file to create preview question page html file"
template_location="edu/ucsc/cse/exam/web/question/PreviewQuestionPage.html.tmpl">
    <file_copy>copy template file</file_copy>
  </code_generation_unit>

  <code_generation_unit unit_id="73" unit_name="PreviewQuestionPanel_java_tag_expansion" description="expand
tags to create preview question panels java file"
template_location="edu/ucsc/cse/exam/web/question/Preview$QuestionEntity$$QuestionPanel.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <tag_expansion tag_id="library_import">
      <parameter_content parameter_name="questionType"
lookup_type="Dictionary">questionDefs.$QuestionEntity$$.name</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="add_widgets">
```

```
   <parameter_content parameter_name="questionType"
lookup_type="Dictionary">questionDefs.$QuestionEntity$$.name</parameter_content>
  </tag_expansion>
 </code_generation_unit>

 <code_generation_unit unit_id="74" unit_name="PreviewQuestionPanel_html_tag_expansion" description="expand
tags to create preview question panels html file"
template_location="edu/ucsc/cse/exam/web/question/Preview$QuestionEntity$$QuestionPanel.html.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
  <tag_expansion tag_id="add_widgets">
   <parameter_content parameter_name="questionType"
lookup_type="Dictionary">questionDefs.$QuestionEntity$$.name</parameter_content>
  </tag_expansion>
 </code_generation_unit>


 <code_generation_unit unit_id="75" unit_name="StudentListGradedExamPage_java_tag_expansion"
description="expand tags to create page for a student to list graded exams"
template_location="edu/ucsc/cse/exam/web/exam/StudentListGradedExamPage.java.tmpl">
  <tag_expansion tag_id="answer_props">
   <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.ExamAnswerPerStudent.StudentListGradedExam.props.propName</parameter_
content>
   <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.ExamAnswerPerStudent.StudentListGradedExam.props.GUIWidget</parameter_
content>
  </tag_expansion>
  <tag_expansion tag_id="exam_props">
   <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.Exam.StudentListGradedExam.props.propName</parameter_content>
   <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.Exam.StudentListGradedExam.props.GUIWidget</parameter_content>
  </tag_expansion>
 </code_generation_unit>

 <code_generation_unit unit_id="76" unit_name="StudentListGradedExamPage_html_tag_expansion"
description="expand tags to create page for a student to list graded exams"
template_location="edu/ucsc/cse/exam/web/exam/StudentListGradedExamPage.html.tmpl">
  <tag_expansion tag_id="answer_props_header">
   <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.ExamAnswerPerStudent.StudentListGradedExam.props.propName</parameter_
content>
  </tag_expansion>
  <tag_expansion tag_id="exam_props_header">
   <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.Exam.StudentListGradedExam.props.propName</parameter_content>
  </tag_expansion>
  <tag_expansion tag_id="answer_props_row">
   <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.ExamAnswerPerStudent.StudentListGradedExam.props.propName</parameter_
content>
  </tag_expansion>
  <tag_expansion tag_id="exam_props_row">
   <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.Exam.StudentListGradedExam.props.propName</parameter_content>
  </tag_expansion>
 </code_generation_unit>
```

```
  <code_generation_unit unit_id="77" unit_name="StudentListOpenExamPage_java_tag_expansion"
description="expand tags to create page for a student to list open exams"
template_location="edu/ucsc/cse/exam/web/exam/StudentListOpenExamPage.java.tmpl">
    <tag_expansion tag_id="exam_props">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.Exam.StudentListOpenExam.props.propName</parameter_content>
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.Exam.StudentListOpenExam.props.GUIWidget</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="78" unit_name="StudentListOpenExamPage_html_tag_expansion"
description="expand tags to create page for a student to list open exams"
template_location="edu/ucsc/cse/exam/web/exam/StudentListOpenExamPage.html.tmpl">
    <tag_expansion tag_id="exam_props_header">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.Exam.StudentListOpenExam.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="exam_props_row">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.Exam.StudentListOpenExam.props.propName</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="79" unit_name="StudentTakeExamPage_java_tag_expansion"
description="expand tags to create page for a student to take an exam"
template_location="edu/ucsc/cse/exam/web/exam/StudentTakeExamPage.java.tmpl">
    <tag_expansion tag_id="add_question_panel">
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="question_panel_def">
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="80" unit_name="StudentTakeExamPage_html_file_copy" description="copy
template file to create take exam html page for a student"
template_location="edu/ucsc/cse/exam/web/exam/StudentTakeExamPage.html.tmpl">
    <file_copy>copy template file to create student take exam page</file_copy>
  </code_generation_unit>

  <code_generation_unit unit_id="81"
unit_name="StudentTakeExamPage_TakeMCQuestionPanel_html_tag_expansion" description="MC question panel
used in the take exam page for a student"
template_location="edu/ucsc/cse/exam/web/exam/StudentTakeExamPage$TakeMCQuestionPanel.html.tmpl">
    <file_copy>copy template file to create student take MC question panel</file_copy>
  </code_generation_unit>

  <code_generation_unit unit_id="82"
unit_name="StudentTakeExamPage_TakeTFQuestionPanel_html_tag_expansion" description="TF question panel
used in the take exam page for a student"
template_location="edu/ucsc/cse/exam/web/exam/StudentTakeExamPage$TakeTFQuestionPanel.html.tmpl">
    <file_copy>copy template file to create student take TF question panel</file_copy>
  </code_generation_unit>

  <code_generation_unit unit_id="83" unit_name="TeacherGradeExamAnswerPage_java_tag_expansion"
description="expand tags to create page for a teacher to grade a submitted student exam"
template_location="edu/ucsc/cse/exam/web/exam/TeacherGradeExamAnswerPage.java.tmpl">
```

```
    <tag_expansion tag_id="add_question_panel">
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="question_panel_def">
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="84" unit_name="TeacherGradeExamAnswerPage_html_file_copy"
description="copy template file to create grade exam html page for a teacher"
template_location="edu/ucsc/cse/exam/web/exam/TeacherGradeExamAnswerPage.html.tmpl">
    <file_copy>copy template file</file_copy>
  </code_generation_unit>

  <code_generation_unit unit_id="85"
unit_name="TeacherGradeExamAnswerPage_GradeMCQuestionPanel_html_file_copy" description="copy template
file to create grade MC question panel html page for a teacher"
template_location="edu/ucsc/cse/exam/web/exam/TeacherGradeExamAnswerPage$GradeMCQuestionPanel.html.t
mpl">
    <file_copy>copy template file to create grade MC question panel</file_copy>
  </code_generation_unit>

  <code_generation_unit unit_id="86"
unit_name="TeacherGradeExamAnswerPage_GradeTFQuestionPanel_html_file_copy" description="copy template
file to create grade TF question panel html page for a teacher"
template_location="edu/ucsc/cse/exam/web/exam/TeacherGradeExamAnswerPage$GradeTFQuestionPanel.html.t
mpl">
    <file_copy>copy template file to create grade TF question panel</file_copy>
  </code_generation_unit>

  <code_generation_unit unit_id="87" unit_name="TeacherGradeExamPage_java_tag_expansion"
description="expand tags to create the page for a teacher to select an exam and start grading"
template_location="edu/ucsc/cse/exam/web/exam/TeacherGradeExamPage.java.tmpl">
    <tag_expansion tag_id="exam_props">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.Exam.TeacherGradeExam.props.propName</parameter_content>
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.Exam.TeacherGradeExam.props.GUIWidget</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="88" unit_name="TeacherGradeExamPage_html_tag_expansion"
description="expand tags to create page for a teacher to select an exam and start grading"
template_location="edu/ucsc/cse/exam/web/exam/TeacherGradeExamPage.html.tmpl">
    <tag_expansion tag_id="exam_props_header">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.Exam.TeacherGradeExam.props.propName</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="exam_props_row">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.Exam.TeacherGradeExam.props.propName</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="89" unit_name="ViewExamAnswerGradePage_java_tag_expansion"
description="expand tags to create page to view exam answer grade"
template_location="edu/ucsc/cse/exam/web/exam/ViewExamAnswerGradePage.java.tmpl">
    <tag_expansion tag_id="add_view_question_panel">
```

```
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="view_question_panel_def">
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="90" unit_name="ViewExamAnswerGradePage_html_file_copy" description="copy
template file to create html page to view exam answer grade"
template_location="edu/ucsc/cse/exam/web/exam/ViewExamAnswerGradePage.html.tmpl">
    <file_copy>copy template file</file_copy>
  </code_generation_unit>

  <code_generation_unit unit_id="91"
unit_name="ViewExamAnswerGradePage_ViewMCQuestionGradePanel_html_file_copy" description="copy
template file to create view MC question grade panel"
template_location="edu/ucsc/cse/exam/web/exam/ViewExamAnswerGradePage$ViewMCQuestionGradePanel.html.
tmpl">
    <file_copy>copy template file to create vew MC question grade panel</file_copy>
  </code_generation_unit>

  <code_generation_unit unit_id="92"
unit_name="ViewExamAnswerGradePage_ViewTFQuestionGradePanel_html_file_copy" description="copy template
file to create view TF question grade panel"
template_location="edu/ucsc/cse/exam/web/exam/ViewExamAnswerGradePage$ViewTFQuestionGradePanel.html.t
mpl">
    <file_copy>copy template file to create vew TF question grade panel</file_copy>
  </code_generation_unit>


  <code_generation_unit unit_id="93" unit_name="QuestionDataProvider_java_string_swap" description="swap
QuestionEntity strings to create question data providers"
template_location="edu/ucsc/cse/exam/web/model/$QuestionEntity$$QuestionDataProvider.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <string_swap original_string="QuestionEntity">
      <replacement_content lookup_type="Dictionary">questionDefs.$QuestionEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="94" unit_name="QuestionSelectionWrapper_java_string_swap" description="swap
QuestionEntity strings to create question selection wrappers"
template_location="edu/ucsc/cse/exam/web/model/$QuestionEntity$$QuestionSelectionWrapper.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <string_swap original_string="QuestionEntity">
      <replacement_content lookup_type="Dictionary">questionDefs.$QuestionEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="95" unit_name="UserDataProvider_java_string_swap" description="swap
UserEntity strings to create user data providers"
template_location="edu/ucsc/cse/exam/web/model/$UserEntity$$DataProvider.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
    <string_swap original_string="UserEntity">
      <replacement_content lookup_type="Dictionary">userDefs.$UserEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>
  <code_generation_unit unit_id="96" unit_name="DetachableQuestionModel_java_string_swap" description="swap
QuestionEntity strings to create detachable question model"
template_location="edu/ucsc/cse/exam/web/model/Detachable$QuestionEntity$$QuestionModel.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <string_swap original_string="QuestionEntity">
      <replacement_content lookup_type="Dictionary">questionDefs.$QuestionEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="97" unit_name="DetachableUserModel_java_string_swap" description="swap
UserEntity strings to create detachable user model"
template_location="edu/ucsc/cse/exam/web/model/Detachable$UserEntity$$Model.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
    <string_swap original_string="UserEntity">
      <replacement_content lookup_type="Dictionary">userDefs.$UserEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="98" unit_name="DetachableExamQuestionModel_java_string_swap"
description="swap QuestionEntity strings to create detachable exam question model"
template_location="edu/ucsc/cse/exam/web/model/DetachableExam$QuestionEntity$$Model.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <string_swap original_string="QuestionEntity">
      <replacement_content lookup_type="Dictionary">questionDefs.$QuestionEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="99" unit_name="DetachableExamModel_java_file_copy" description="copy
template file to create detachable exam model"
template_location="edu/ucsc/cse/exam/web/model/DetachableExamModel.java.tmpl">
    <file_copy>copy template file to create detachable exam model</file_copy>
  </code_generation_unit>

  <code_generation_unit unit_id="100" unit_name="ExamQuestionDataProvider_java_string_swap"
description="swap QuestionEntity strings to create exam question data providers"
template_location="edu/ucsc/cse/exam/web/model/Exam$QuestionEntity$$DataProvider.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <string_swap original_string="QuestionEntity">
      <replacement_content lookup_type="Dictionary">questionDefs.$QuestionEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="101" unit_name="ExamDataProvider_java_file_copy" description="copy template
file to create exam data provider" template_location="edu/ucsc/cse/exam/web/model/ExamDataProvider.java.tmpl">
    <file_copy>copy template file to create exam data provider</file_copy>
  </code_generation_unit>



  <code_generation_unit unit_id="102" unit_name="DeleteQuestionConfirmPage_java_snippet_join_tag_expansion"
description="expand tags to create the template file for the snippet join CGU"
template_location="edu/ucsc/cse/exam/web/question/DeleteQuestionConfirmPage.java.tmpl.tmpl">
    <tag_expansion tag_id="generate_snippet_tags">
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="add_question_props">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.MC.DeleteQuestionConfirm.props.propName</parameter_content>
```

```
   <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.MC.DeleteQuestionConfirm.props.GUIWidget</parameter_content>
  </tag_expansion>
 </code_generation_unit>

 <code_generation_unit unit_id="103" unit_name="DeleteQuestionConfirmPage_java_snippet_join"
description="join code snippets to create DeleteQuestionConfirmPage java file"
template_location="edu/ucsc/cse/exam/web/question/DeleteQuestionConfirmPage.java.tmpl">
  <snippet_join snippet_id="MC_props">
   <snippet_file_location ref_CGU_ids="104 105">
    edu/ucsc/cse/exam/web/question/DeleteQuestionConfirmPageMC.snippet
   </snippet_file_location>
  </snippet_join>
  <snippet_join snippet_id="TF_props">
   <snippet_file_location ref_CGU_ids="104 105">
    edu/ucsc/cse/exam/web/question/DeleteQuestionConfirmPageTF.snippet
   </snippet_file_location>
  </snippet_join>
 </code_generation_unit>

 <code_generation_unit unit_id="104" unit_name="DeleteQuestionConfirmPage_question_snippet_string_swap"
description="swap strings to create code snippet for question type in DeleteQuestionConfirm java page"
template_location="edu/ucsc/cse/exam/web/question/DeleteQuestionConfirmPage$questionType$$.snippet.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
  <string_swap original_string="questionType">
   <replacement_content lookup_type="Dictionary">questionDefs.$questionType$$.name</replacement_content>
  </string_swap>
 </code_generation_unit>

 <code_generation_unit unit_id="105" unit_name="DeleteQuestionConfirmPage_question_snippet_tag_expansion"
description="expand tags to create code snippet for question type in DeleteQuestionConfirm java page"
template_location="edu/ucsc/cse/exam/web/question/DeleteQuestionConfirmPage$questionType$$.snippet.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
  <tag_expansion tag_id="add_question_props">
   <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.$questionType$$.DeleteQuestionConfirm.props.propName</parameter_content>
   <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.$questionType$$.DeleteQuestionConfirm.props.GUIWidget</parameter_content>
  </tag_expansion>
 </code_generation_unit>

 <code_generation_unit unit_id="106" unit_name="DeleteQuestionConfirmPage_html_tag_expansion"
description="expand tags to create DeleteQuestionConfirm html page"
template_location="edu/ucsc/cse/exam/web/question/DeleteQuestionConfirmPage.html.tmpl">
  <tag_expansion tag_id="add_question_props">
   <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.MC.DeleteUserConfirm.props.propName</parameter_content>
   <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.MC.DeleteConfirm.props.GUIWidget</parameter_content>
  </tag_expansion>
 </code_generation_unit>



 <code_generation_unit unit_id="107" unit_name="EditQuestionPage_properties_tag_expansion"
description="expand tags to create property file for the edit question page"
template_location="edu/ucsc/cse/exam/web/question/Edit$QuestionEntity$$QuestionPage.properties.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
  <tag_expansion tag_id="prop_hint">
```

```
      <parameter_content parameter_name="questionType"
lookup_type="Dictionary">questionDefs.$QuestionEntity$$.name</parameter_content>
    </tag_expansion>
  </code_generation_unit>


  <code_generation_unit unit_id="109" unit_name="EditQuestionPage_java_string_swap" description="swap strings
to create edit question page"
template_location="edu/ucsc/cse/exam/web/question/Edit$QuestionEntity$$QuestionPage.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <string_swap original_string="QuestionEntity">
      <replacement_content lookup_type="Dictionary">questionDefs.$QuestionEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="110" unit_name="EditQuestionPage_java_tag_expansion" description="expand
tags to create edit question page"
template_location="edu/ucsc/cse/exam/web/question/Edit$QuestionEntity$$QuestionPage.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <tag_expansion tag_id="question_props">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.$QuestionEntity$$.EditQuestion.props.propName</parameter_content>
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.$QuestionEntity$$.EditQuestion.props.GUIWidget</parameter_content>
      <parameter_content parameter_name="isRequired"
lookup_type="Dictionary">appDefs.$QuestionEntity$$.EditQuestion.props.isRequired</parameter_content>
      <parameter_content parameter_name="lengthLimitUpper"
lookup_type="Dictionary">appDefs.$QuestionEntity$$.EditQuestion.props.lengthLimitUpper</parameter_content>
    </tag_expansion>
  </code_generation_unit>


  <code_generation_unit unit_id="111" unit_name="EditQuestionPage_html_tag_expansion" description="expand
tags to create edit question page"
template_location="edu/ucsc/cse/exam/web/question/Edit$QuestionEntity$$QuestionPage.html.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <tag_expansion tag_id="question_props">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.$QuestionEntity$$.EditQuestion.props.propName</parameter_content>
    </tag_expansion>
  </code_generation_unit>


  <code_generation_unit unit_id="112" unit_name="ListQuestionPage_java_string_swap" description="swap strings
to create list question java page"
template_location="edu/ucsc/cse/exam/web/question/List$QuestionEntity$$QuestionPage.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <string_swap original_string="QuestionEntity">
      <replacement_content lookup_type="Dictionary">questionDefs.$QuestionEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="113" unit_name="ListQuestionPage_java_tag_expansion" description="expand
tags to create list question java page"
template_location="edu/ucsc/cse/exam/web/question/List$QuestionEntity$$QuestionPage.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <tag_expansion tag_id="props_count">
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.$QuestionEntity$$.ListQuestion.props.GUIWidget</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="question_props">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.$QuestionEntity$$.ListQuestion.props.propName</parameter_content>
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.$QuestionEntity$$.ListQuestion.props.GUIWidget</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="114" unit_name="ListQuestionPage_html_string_swap" description="swap strings
to create list question html page"
template_location="edu/ucsc/cse/exam/web/question/List$QuestionEntity$$QuestionPage.html.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <string_swap original_string="QuestionEntity">
      <replacement_content lookup_type="Dictionary">questionDefs.$QuestionEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>


  <code_generation_unit unit_id="115" unit_name="PreviewQuestionPage_java_tag_expansion"
description="expand tags to create preview question java page"
template_location="edu/ucsc/cse/exam/web/question/PreviewQuestionPage.java.tmpl">
    <tag_expansion tag_id="preview_question_panel">
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
  </code_generation_unit>


  <code_generation_unit unit_id="116" unit_name="PreviewQuestionPage_html_file_copy" description="copy files to
create preview question html page"
template_location="edu/ucsc/cse/exam/web/question/PreviewQuestionPage.html.tmpl">
    <file_copy>copy template file to create preview question page html file</file_copy>
  </code_generation_unit>



  <code_generation_unit unit_id="117" unit_name="QuestionActionPanel_java_tag_expansion" description="expand
tags to create question action panel java file"
template_location="edu/ucsc/cse/exam/web/question/QuestionActionPanel.java.tmpl">
    <tag_expansion tag_id="preview_question_panel">
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="edit_question_panel">
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="delete_question_panel">
      <parameter_content parameter_name="questionType"
lookup_type="Function">findSubclassQuestionEntityNames</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="118" unit_name="QuestionActionPanel_html_file_copy" description="copy files to
create question action panel html file"
template_location="edu/ucsc/cse/exam/web/question/QuestionActionPanel.html.tmpl">
    <file_copy>copy template file to create question action panel html file</file_copy>
  </code_generation_unit>
  <code_generation_unit unit_id="119" unit_name="PreviewQuestionPanel_java_string_swap" description="swap
strings to create preview question panel java file"
template_location="edu/ucsc/cse/exam/web/question/Preview$QuestionEntity$$QuestionPanel.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <string_swap original_string="QuestionEntity">
      <replacement_content lookup_type="Dictionary">questionDefs.$QuestionEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="120" unit_name="PreviewQuestionPanel_java_tag_expansion"
description="expand tags to create preview question panel java file"
template_location="edu/ucsc/cse/exam/web/question/Preview$QuestionEntity$$QuestionPanel.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <tag_expansion tag_id="library_import">
      <parameter_content parameter_name="questionType"
lookup_type="Dictionary">questionDefs.$QuestionEntity$$.name</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="add_widgets">
      <parameter_content parameter_name="questionType"
lookup_type="Dictionary">questionDefs.$QuestionEntity$$.name</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="121" unit_name="PreviewQuestionPanel_html_tag_expansion"
description="expand tags to create preview question panel html file"
template_location="edu/ucsc/cse/exam/web/question/Preview$QuestionEntity$$QuestionPanel.html.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassQuestionEntityNames">
    <tag_expansion tag_id="add_widgets">
      <parameter_content parameter_name="questionType"
lookup_type="Dictionary">questionDefs.$QuestionEntity$$.name</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="122" unit_name="UserHomePage_java_string_swap" description="swap strings to
create user home page java file"
template_location="edu/ucsc/cse/exam/web/user/$UserEntity$$HomePage.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
    <string_swap original_string="UserEntity">
      <replacement_content lookup_type="Dictionary">userDefs.$UserEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="123" unit_name="UserHomePage_html_file_copy" description="copy files to create
user home page html file" template_location="edu/ucsc/cse/exam/web/user/$UserEntity$$HomePage.html.tmpl">
    <file_copy>copy template file to create user home page html file</file_copy>
  </code_generation_unit>

  <code_generation_unit unit_id="124" unit_name="DeleteUserConfirmPage_java_snippet_join_tag_expansion"
description="expand tags to create the template file for snippet join CGU"
template_location="edu/ucsc/cse/exam/web/user/DeleteUserConfirmPage.java.tmpl.tmpl">
    <tag_expansion tag_id="generate_snippet_tags">
      <parameter_content parameter_name="userType"
lookup_type="Function">findSubclassUserEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="add_user_props">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.Teacher.DeleteUserConfirm.props.propName</parameter_content>
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.Teacher.DeleteUserConfirm.props.GUIWidget</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="125" unit_name="DeleteUserConfirmPage_java_snippet_join" description="join
code snippets to create DeleteUserConfirmPage java file"
template_location="edu/ucsc/cse/exam/web/user/DeleteUserConfirmPage.java.tmpl">
    <snippet_join snippet_id="Teacher_props">
      <snippet_file_location ref_CGU_ids="126 127">
        edu/ucsc/cse/exam/web/user/DeleteUserConfirmPageTeacher.snippet
      </snippet_file_location>
    </snippet_join>
    <snippet_join snippet_id="Student_props">
      <snippet_file_location ref_CGU_ids="126 127">
        edu/ucsc/cse/exam/web/user/DeleteUserConfirmPageStudent.snippet
      </snippet_file_location>
    </snippet_join>
  </code_generation_unit>

  <code_generation_unit unit_id="126" unit_name="DeleteUserConfirmPage_user_snippet_string_swap"
description="swap strings to create code snippet for user type in DeleteUserConfirm java page"
template_location="edu/ucsc/cse/exam/web/user/DeleteUserConfirmPage$userType$$.snippet.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
    <string_swap original_string="userType">
      <replacement_content lookup_type="Dictionary">userDefs.$userType$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="127" unit_name="DeleteUserConfirmPage_user_snippet_tag_expansion"
description="expand tags to create code snippet for user type in DeleteUserConfirm java page"
template_location="edu/ucsc/cse/exam/web/user/DeleteUserConfirmPage$userType$$.snippet.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
    <tag_expansion tag_id="add_user_props">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.$userType$$.DeleteUserConfirm.props.propName</parameter_content>
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.$userType$$.DeleteUserConfirm.props.GUIWidget</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="128" unit_name="DeleteUserConfirmPage_html_tag_expansion"
description="expand tags to create DeleteUserConfirmPage html file"
template_location="edu/ucsc/cse/exam/web/user/DeleteUserConfirmPage.html.tmpl">
    <tag_expansion tag_id="add_user_props">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.Teacher.DeleteUserConfirm.props.propName</parameter_content>
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.Teacher.DeleteUserConfirm.props.GUIWidget</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="129" unit_name="EditUserAccountPage_java_string_swap" description="swap
strings to create EditUserAccountPage java file"
template_location="edu/ucsc/cse/exam/web/user/Edit$UserEntity$$AccountPage.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
    <string_swap original_string="UserEntity">
      <replacement_content lookup_type="Dictionary">userDefs.$UserEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="130" unit_name="EditUserAccountPage_java_tag_expansion"
description="expand tags to create EditUserAccountPage java file"
```

```
template_location="edu/ucsc/cse/exam/web/user/Edit$UserEntity$$AccountPage.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
    <tag_expansion tag_id="user_home">
      <parameter_content parameter_name="userType"
lookup_type="Function">findSubclassUserEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="home_link_visibility">
      <parameter_content parameter_name="userType"
lookup_type="Function">findSubclassUserEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="user_logout">
      <parameter_content parameter_name="userType"
lookup_type="Function">findSubclassUserEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="logout_link_visibility">
      <parameter_content parameter_name="userType"
lookup_type="Function">findSubclassUserEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="user_props">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.$UserEntity$$.EditUserAccount.props.propName</parameter_content>
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.$UserEntity$$.EditUserAccount.props.GUIWidget</parameter_content>
      <parameter_content parameter_name="isRequired"
lookup_type="Dictionary">appDefs.$UserEntity$$.EditUserAccount.props.isRequired</parameter_content>
      <parameter_content parameter_name="lengthLimitUpper"
lookup_type="Dictionary">appDefs.$UserEntity$$.EditUserAccount.props.lengthLimitUpper</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="phonenum_class_def">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.$UserEntity$$.EditUserAccount.props.propName
      </parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="131" unit_name="EditUserAccountPage_html_string_swap" description="swap
strings to create EditUserAccountPage html file"
template_location="edu/ucsc/cse/exam/web/user/Edit$UserEntity$$AccountPage.html.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
    <string_swap original_string="UserEntity">
      <replacement_content lookup_type="Dictionary">userDefs.$UserEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="132" unit_name="EditUserAccountPage_html_tag_expansion"
description="expand tags to create EditUserAccountPage html file"
template_location="edu/ucsc/cse/exam/web/user/Edit$UserEntity$$AccountPage.html.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
    <tag_expansion tag_id="question_props">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.$UserEntity$$.EditUserAccount.props.propName</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="133" unit_name="EditUserAccountPage_property_tag_expansion"
description="expand tags to create EditUserAccountPage property file"
template_location="edu/ucsc/cse/exam/web/user/Edit$UserEntity$$AccountPage.properties.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
    <tag_expansion tag_id="prop_hint">
```

```
      <parameter_content parameter_name="userType"
lookup_type="Dictionary">userDefs.$UserEntity$$.name</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="134" unit_name="ListUserPage_java_string_swap" description="swap strings to
create list user java page" template_location="edu/ucsc/cse/exam/web/user/List$UserEntity$$Page.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
    <string_swap original_string="UserEntity">
      <replacement_content lookup_type="Dictionary">userDefs.$UserEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>



  <code_generation_unit unit_id="135" unit_name="ListUserPage_java_tag_expansion" description="expand tags to
create list user java page" template_location="edu/ucsc/cse/exam/web/user/List$UserEntity$$Page.java.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
    <tag_expansion tag_id="props_count">
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.$UserEntity$$.ListUser.props.GUIWidget</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="user_props">
      <parameter_content parameter_name="propName"
lookup_type="Dictionary">appDefs.$UserEntity$$.ListUser.props.propName</parameter_content>
      <parameter_content parameter_name="GUIWidget"
lookup_type="Dictionary">appDefs.$UserEntity$$.ListUser.props.GUIWidget</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="add_table">
      <parameter_content parameter_name="userType"
lookup_type="Dictionary">userDefs.$UserEntity$$.name</parameter_content>
    </tag_expansion>
  </code_generation_unit>


  <code_generation_unit unit_id="136" unit_name="ListUserPage_html_string_swap" description="swap strings to
create list question html page" template_location="edu/ucsc/cse/exam/web/user/List$UserEntity$$Page.html.tmpl"
template_param_lookup_type="Function" template_param_lookup_value="findSubclassUserEntityNames">
    <string_swap original_string="QuestionEntity">
      <replacement_content lookup_type="Dictionary">userDefs.$UserEntity$$.name</replacement_content>
    </string_swap>
  </code_generation_unit>

  <code_generation_unit unit_id="137" unit_name="UserActionPanel_java_tag_expansion" description="expand tags
to create user action panel java file" template_location="edu/ucsc/cse/exam/web/user/UserActionPanel.java.tmpl">
    <tag_expansion tag_id="edit_user_panel">
      <parameter_content parameter_name="userType"
lookup_type="Function">findSubclassUserEntityNames</parameter_content>
    </tag_expansion>
    <tag_expansion tag_id="delete_user_panel">
      <parameter_content parameter_name="userType"
lookup_type="Function">findSubclassUserEntityNames</parameter_content>
    </tag_expansion>
  </code_generation_unit>

  <code_generation_unit unit_id="138" unit_name="UserActionPanel_html_file_copy" description="copy files to
create user action panel html file" template_location="edu/ucsc/cse/exam/web/user/UserActionPanel.html.tmpl">
    <file_copy>copy template file to create user action panel html file</file_copy>
  </code_generation_unit>
```

```xml
  <code_generation_unit unit_id="139" unit_name="BasePage_html_file_copy" description="copy files to create base page html file" template_location="edu/ucsc/cse/exam/web/BasePage.html.tmpl">
    <file_copy>copy template file to create base page html file</file_copy>
  </code_generation_unit>

  <code_generation_unit unit_id="140" unit_name="WebAppProperty_file_copy" description="copy files to create the Web application property file" template_location="application.properties.tmpl">
    <file_copy>copy template file to create property file for Web Application</file_copy>
  </code_generation_unit>

  <code_generation_unit unit_id="141" unit_name="Log4JProperty_file_copy" description="copy files to create Log4J property file" template_location="log4j.properties.tmpl">
    <file_copy>copy template file to create property file for Log4J</file_copy>
  </code_generation_unit>

<!-- END OF CODE GENERATION UNITS -->

</var_model>
```

BIBLIOGRAPHY

[1] "Microsoft Office Website," http://office.microsoft.com/en-us/FX102855291033.aspx, last accessed: Oct 30, 2008.

[2] G. Chastek, P. Donohoe, K. C. Kang, and S. Thiel, "Product Line Analysis: A Practical Introduction (Technical Report)," Carnegie Mellon Software Engineering Institute CMU/SEI-2001-TR-001, June 2001.

[3] P. Clements, L. Northrop, and L. M. Northrop, *Software Product Lines : Practices and Patterns*, 3rd ed.: Addison-Wesley, Aug 20, 2001.

[4] D. L. Parnas, "On the Design and Development of Program Families," *IEEE Transactions on Software Engineering*, vol. 2, pp. 1-9, 1976.

[5] K. Pohl, G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*, 1st ed.: Springer, Sep 2005.

[6] J. Herrington, *Code Generation in Action*, Revised ed.: Manning Publications, 2003.

[7] K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*: Addison-Wesley Professional, 2000.

[8] J. Greenfield, K. Short, S. Cook, and S. Kent, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, 1st ed.: Wiley, 2004.

[9] L. Brownsword and P. C. Clements, "A Case Study in Successful Product Line Development," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 1996.

[10] P. C. Clements and L. M. Northrop, "Salion, Inc.: A Software Product Line Case Study," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 2002.

[11] H. Gomaa, *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*, 1st ed.: Addison-Wesley, July 7, 2004.

[12] P. P. S. L. Company, "XFeature Project Website," http://www.pnp-software.com/XFeature/, last accessed: Feb 20, 2008.

[13] H. Zhang and S. Jarzabek, "XVCL: A Mechanism for Handling Variants in Software Product Lines," *Science of Computer Programming*, vol. 53, pp. 381-407, Dec 2004.

[14] H. Zhang and S. Jarzabek, "An XVCL-based Approach to Software Product Line Development," in *15th International Conference on Software Engineering and Knowledge Engineering (SEKE'03)*, San Francisco, USA, July 2003, pp. 267-275.

[15] Microsoft, "Visual Studio MSDN," http://msdn.microsoft.com/en-us/vstudio/default.aspx, last accessed: Oct 30, 2008.

[16] Instantiations, "WindowBuilder Online Documentation," http://download.instantiations.com/DesignerDoc/integration/latest/docs/html/toc.html, last accessed: Oct 30, 2008.

[17] Sun Microsystems Inc., "Javadoc Online Documentation," http://java.sun.com/j2se/1.4.2/docs/tooldocs/javadoc/index.html, last accessed: Oct 30, 2008.

[18] Doxygen, "Doxygen Manual," http://www.stack.nl/~dimitri/doxygen/manual.html, last accessed: Oct 30, 2008.

[19] "ArcStyler, the leading platform for Model Driven Architecture (MDA)," Interactive Objects Software GmbH 2005.

[20] "JAG Getting Started Tutorial," http://jag.sourceforge.net/tutorial.html, last accessed: Oct 30, 2008.

[21] M. Raible, "AppFuse QuickStart," http://appfuse.org/display/APF/AppFuse+QuickStart, last accessed: Oct 30, 2008.

[22] S. J. Mellor, K. Scott, A. Uhl, and D. Weise, *MDA Distilled*: Addison-Wesley Professional, 2004.

[23] S. G. Cohen, J. L. Stanley, Jr., A. S. Peterson, and R. W. Krut, Jr., "Application of Feature-Oriented Domain Analysis to the Army Movement Control Domain (CMU/SEI-91-TR-028)," the Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA June 1992.

[24] K. C. Kang, S. G. Cohen, J. A. Hess, N. W. E., and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study (CMU/SEI-90-TR-21, ADA 235785)," the Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 1990.

[25] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures," *Annals of Software Engineering,* vol. 5, pp. 143-168, January 1998.

[26] K. Czarnecki, S. Helsen, and U. W. Eisenecker, "Formalizing Cardinality-based Feature Models and Their Specialization," *Software Process Improvement and Practice,* vol. 10, pp. 7-29, 2005.

[27] K. Czarnecki and C. H. Kim, "Cardinality-Based Feature Modeling and Constraints: A Progress Report," in *OOPSLA'05 Workshop on Software Factories* San Diego, California, USA, Oct 2005.

[28] K. C. Kang, K. Lee, and J. Lee, "Feature Oriented Product Line Software Engineering: Principles and Guidelines," in *Domain Oriented Systems Development: Perspectives and Practices* UK: Taylor & Francis, 2003, pp. 29-46.

[29] "The Object Management Group (OMG)," http://omg.org/, last accessed: May 26, 2008.

[30] "Eclipse Modeling Framework Documentation," http://www.eclipse.org/modeling/emf/docs/, last accessed: March 26, 2008.

[31] "Graphical Modeling Framework Documentation," http://wiki.eclipse.org/index.php/GMF_Documentation, last accessed: March 26, 2008.

[32] D. Djuric, D. Gaševic, and V. Devedžic, "The Tao of Modeling Spaces," *Journal of Object Technology,* vol. 5, pp. 125-147, Nov-Dec 2006.

[33] J. M. Neighbors, "Software Construction Using Components," Irvine, CA: University of California, Irvine, 1980.

[34] R. McCain, "Reusable Software Component Construction: A Product-Oriented Paradigm," in *5th AIAA/ACM/NASA/IEEE Computers in Aerospace Conference* Long Beach, CA, 1985, pp. 125-135.

[35] R. Prieto-Díaz, "Domain Analysis: An Introduction," *ACM SIGSOFT Software Engineering Notes,* vol. 15, pp. 47-54, 1990.

[36] R. Prieto-Díaz, "Domain Analysis for Reusability," in *COMPSAC'87* Tokyo, Japan, Oct 1987, pp. 23-29.

[37] K. C. Kang, S. Kim, J. Lee, and K. Lee, "Feature-Oriented Engineering of PBX Software for Adaptability and Reuseability," *Software: Practice and Experience,* vol. 29, pp. 875-896, Aug 5, 1999.

[38] Generative Software Development Lab, "Feature Modeling Plug-in Website," http://gsd.uwaterloo.ca/projects/fmp-plugin/overview/, last accessed: Nov 20, 2007.

[39] "Eclipse Modeling Project Official Website," http://www.eclipse.org/modeling/, last accessed: Dec 17, 2007.

[40] Object Management Group, "OCL 2.0 OMG Final Adopted Specification," www.omg.org/docs/ptc/03-10-14.pdf last accessed: May 26, 2008.

[41] H. R. Andersen, "An Introduction to Binary Decision Diagrams," Department of Information Technology, Technical University of Denmark Apr 1998.

[42] K. Czarnecki and M. Antkiewicz, "Mapping Features to Models: A Template Approach Based on Superimposed Variants," in *4th International Conference on Generative Programming and Component Engineering (GPCE'05)*, Tallinn, Estonia, 2005, pp. 422-437.

[43] K. Czarnecki, S. Helsen, and U. Eisenecker, "Staged Configuration Through Specialization and Multi-Level Configuration of Feature Models," *Software Process Improvement and Practice,* vol. 10, pp. 143-169, 2005.

[44] M. Antkiewicz and K. Czarnecki, "FeaturePlugin: Feature Modeling Plug-In for Eclipse," OOPSLA'04 Eclipse Technology eXchange (ETX) Workshop, Oct 2004.

[45] M. Clauß, "Untersuchung der Modellierung von Variabilität in UML (diploma thesis)," Fakultät Informatik, Technische Universität Dresden, 2001.

[46] S. Bühne, G. Halmans, and K. Pohl, "Modeling Dependencies between Variation Points in Use Case Diagrams," in *9th International Workshop on Requirements Engineering - Foundation for Software Quality (REFSQ'03)*, Klagenfurt/Velden, Austria, June 2003, pp. 59-70.

[47] A. Pasetti and O. Rohlik, "Technical Note on A Concept for the XFeature Tool," June 2005.

[48] v. Cechticky, A. Pasetti, O. Rohlik, and W. Schaufelberger, "XML-based Feature Modeling," in *Software Reuse: Methods, Techniques and Tools: 8th International Conference, ICSR 2004*, Madrid, Spain, July 5-9, 2004, pp. 101-114.

[49] "Document Schema Definition Languages (DSDL) - Part 3: Rule-based Validation - Schematron (ISO/IEC FDIS 19757-3)."

[50] K. Andersson, "A Vocabulary for Conceptual Design," in *IFIP TC5/WG5.2 Workshop on Formal Design Methods for CAD*, Tallinn, Estonia, June 16-19, 1994, pp. 157-171.

[51] S. Chase, "Using Logic to Specify Shapes and Spatial Relations in Design Grammar," in *Artificial Intelligence in Design '96 Workshop: Grammatical Design*, Stanford University, CA, June 1996.

[52] R. L. Carpenter, *The Logic of Typed Feature Structures: With Applications to Unification Grammars, Logic Programs and Constraint Resolution*: Cambridge University Press, 1992.

[53] R. F. Woodbury and A. L. Burrow, "Whither Design Space," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* vol. 20, pp. 63-82, April 2006.

[54] R. F. Woodbury and A. L. Burrow, "A Typology of Design Space Explorers," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* vol. 20, pp. 143-153, April 2006.

[55] R. F. Woodbury, A. Burrow, S. Datta, and M. T.-W. Chang, "Typed Feature Structures and Design Space Exploration," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* vol. 13, pp. 287-302, September 1999.

[56] R. D. Levine and W. D. Meurers, "Head-Driven Phrase Structure Grammar: Linguistic Approach, Formal Foundations, and Computational Realization," in *Encyclopedia of Language and Linguistics (2nd ed.)*, K. Brown, Ed. Oxford: Elsevier.

[57] M. Hamilton, "Introduction to the 001 Tool Suite," April 25, 2001.

[58] M. Hamilton, "001 Presentation: Development Before the Fact Environment for Building System Oriented Objects," Goddard Space Flight Center, April 10, 2003.

[59] S. Jarzabek, P. Bassett, and H. Zhang, "XVCL: XML-based Variant Configuration Language," in *International Conference on Software Engineering (ICSE'03)*, Porland, USA, May 2003, pp. 810-811.

[60] M. S. Soe, H. Zhang, and S. Jarzabek, "XVCL: A Tutorial," in *14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02),* Ischia, Italy, July 2002, pp. 341-349.

[61] B. Kuipers, "Introduction to Frame Based Knowledge Representation," http://www.cs.utexas.edu/users/qr/algy/algy-expsys/node2.html, last accessed: Nov 20, 2007.

[62] P. G. Bassett, *Framing Software Reuse--Lessons from the Real World*, 1st ed.: Prentice-Hall, NJ, 1997.

[63] D. Batory, "A Tutorial on Feature Oriented Programming and the AHEAD Tool Suite," in *Summer School on Generative and Transformation Techniques in Software Engineering*. vol. 4143, R. Laemmel, J. Saraiva, and J. Visser, Eds. Braga, Portugal: Springer-Verlag, 2006.

[64] D. Batory, "A Tutorial on Feature Oriented Programming and the AHEAD Tool Suite," in *Generative and Transformational Techniques in Software Engineering.* vol. 4143/2006, 2006, pp. 3-35.

[65] D. Batory, "Feature Models, Grammars, and Propositional Formulas," in *9th International Software Product Line Conference (SPLC'05),* Rennes, France, 2005, pp. 7-20.

[66] S. Apel and D. Batory, "When to Use Features and Aspects?: A Case Study," in *5th International Conference on Generative Programming and Component Engineering (GPCE'06),* Portland, Oregon, USA, 2006, pp. 59-68.

[67] M. M. I. Chisty, "An Introduction to Java Annotations," http://www.developer.com/java/other/article.php/10936_3556176_1, last accessed: Oct 30, 2008.

[68] "Tapestry User Guide," http://tapestry.apache.org/tapestry4.1/usersguide/index.html, last accessed: Oct 30, 2008.

[69] P. P. Chen, "The Entity-Relationship Model - Toward a Unified View of Data," *ACM Transactions on Database Systems (TODS),* vol. 1, pp. 9-36, 1976.

[70] F. G. Halasz, "Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems," *Communications of the ACM,* vol. 31, pp. 836-852, July 1988.

[71] "Blackboard Official Website," http://www.blackboard.com/us/index.aspx, last accessed: December 30, 2007.

[72] "Moodle Official Website," http://moodle.org, last accessed: December 30, 2007.

[73] "GIFT Format from MoodleDocs," http://docs.moodle.org/en/GIFT_format, last accessed: Jan 2, 2008.

[74] "IMS Question and Test Interoperability Overview (Version 2.1 Public Draft rev.2 Specification)," http://www.imsglobal.org/question/qtiv2p1pd2/imsqti_oviewv2p1pd2.html, last accessed: Jan 2, 2008.

[75] "Hot Potatoes Official Website," http://hotpot.uvic.ca/, last accessed: December 30, 2007.

[76] "Sakai Project Homepage," http://www.sakaiproject.org/portal, last accessed: Nov 20, 2008.

[77] "Claroline Wiki Documentation Page," http://doc.claroline.net/en/index.php/Main_Page, last accessed: Nov 20, 2008.

[78] "PayPal website," https://www.paypal.com/, last accessed: June 8, 2008.

[79] "Google Checkout website," https://checkout.google.com/, last accessed: June 8, 2008.

[80] D. Sadoski, "Client/Server Software Architectures--An Overview," http://www.sei.cmu.edu/str/descriptions/clientserver_body.html, last accessed: June 13, 2008.

[81] "GRE Frequently Asked Questions about the General Test: Computer-Based General Test," http://www.ets.org/portal/site/ets/menuitem.1488512ecfd5b8849a77b13bc3921509/?vgnextoid=302433c7f00c5010VgnVCM10000022f95190RCRD&vgnextchannel=7196e3b5f64f4010VgnVCM10000022f95190RCRD#Computer_Based_General_Test, last accessed: June 12, 2008.

[82] "Gradiance Online Testing System Overview," http://www.gradiance.com/idea.html, last accessed: Nov 20, 2008.

[83] "Template Engine Wikipedia Page," http://en.wikipedia.org/wiki/Template_engine_(web), last accessed: Nov 28, 2008.

[84] M. Dashorst and E. Hillenius, *Wicket in Action*: Manning Publications, 2008.

[85] YesSoftware, "CodeCharge Studio 4.0 Online Documentation," http://docs.codecharge.com/studio40/html/, last accessed: Dec 2, 2008.

[86] R. E. Johnson and B. Foote, "Designing Reusable Classes," *Journal of Object-Oriented Programming,* vol. 1, pp. 22-35, 1988.

[87] M. Fowler, "Inversion of Control," http://martinfowler.com/bliki/InversionOfControl.html, last accessed: Dec 2, 2008.