REACHING SEMANTIC INTEROPERABILITY THROUGH SEMANTIC ASSOCIATION OF FACETED TAXONOMIES

By

HUNG-JU CHU

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2010

I dedicate this to my parents, who always encouraged me to pursue my dreams, and to all my teachers, who have helped me to realize those dreams.

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my advisor, Dr. Randy Y.C. Chow, for the outstanding guidance and support he has provided throughout the course of this research. I also would like to extend my gratitude to Dr. Su-Shing Chen for his unfailing encouragement. Heartfelt thanks are also due to Dr. Jonathan C.L. Liu, Dr. Richard Newman, and Dr. Raymond Issa for serving on my committee.

I would like to express my special thanks to all the CISE office staff, especially Mr. John Bowers and Ms. Joan Crisman, for taking care of the administrative details regarding my studies at the department. I am grateful to Shiwei Zhang and Varun Gupta for their help and many fruitful discussions during the design and implementation of this work. Also I would like to thank all my friends for their constant support and encouragement.

I would like to acknowledge National Science Foundation and the CISE department for providing me with research and conference traveling grants.

Last, but not least, I thank my family for their endless love. Without their support, this work would not have been possible.

TABLE OF CONTENTS

page

LIST OF TABLES

Table page

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

REACHING SEMANTIC INTEROPERABILITY THROUGH SEMANTIC ASSOCIATION OF FACETED TAXONOMIES

By

Hung-ju Chu

December 2010

Chair: Randy Y. C. Chow
Co-chair: Su-Shing Chen
Major: Computer Engineering

This dissertation addresses semantic interoperability, a common fundamental problem for research into database, semantic web, and information integration. A novel taxonomy-based information model is proposed for managing faceted domain knowledge (semantics) and thus extends its utility for a wide range of applications (interoperability). This approach was inspired by the building construction domain, where complementary taxonomies (or standards) are used to classify sets of related specifications and materials that then serve as reference models to enhance communication among stakeholders. The new approach is based on the hypothesis that faceted taxonomies (classification) and cross-referencing can be extended and formalized to create a functional generic model that addresses the semantic interoperability problem within a domain. Working examples of such an information model are defined and systematic approaches proposed for its application to the management of project requirements in the software engineering domain. Through an analysis of the working example and simulations of a prototype implementation, the proposed approaches are validated in terms of the resulting enhancement in

traceability, reusability, and testability in software development information artifacts, such as requirements, designs, and testing. The dissertation concludes by discussing the new model's potential broader impacts in workflow, proactive information retrieval, and ontology matching.

CHAPTER 1
INTRODUCTION

With the increasing diversity of interrelated data sources over the Internet, semantic interoperability for sharing digitalized knowledge has become essential for modern distributed applications. This has led to a great deal of research on ontology, the formal specification of shared conceptualizations [1], and the development of matching/alignment algorithms for use in the semantic web, database, and information reuse and integration research communities [2], [3], [4], [5], [6], [7], [8]. Although this line of research is fundamental and has brought valuable contributions to this endeavor, it has not as yet identified an adequate solution to the challenge of semantic heterogeneity that exists in most autonomously developed systems. Research in this area has primarily focused on the discovery of similarity semantics, such as equivalence or subsumption, between entities of ontologies. The performances of all the proposed approaches rely on the existence of such similarity relationships and the sufficiency of the data for inferring them. Unfortunately, many of today's independently developed information systems do not share a common knowledge modeling framework. Their semantics are complex and have often not been adequately specified. Moreover, the direct mapping of relationships between entities in such systems seldom exists. The difficulty is further compound by the polysemy problem, where the same word can represent several different meanings. As a result, a workable solution still requires significant intervention by domain experts.

As part of the effort to develop a more effective approach for tackling the semantic heterogeneity challenge, this dissertation proposes a novel domain information model that explores application context beyond the similarity semantic. The underlying

concept guiding the construction of this new approach is the observation that human beings naturally use classification techniques for comprehending and referencing concepts in order to convey their semantics. These phenomena are ubiquitous and examples of their use can be found throughout a wide range of human endeavors. For example, in the building construction domain, complementary hierarchically structured standards (or taxonomies) are often utilized to characterize complex building construction processes and objects used in these processes are often used as reference models to achieve semantic agreement among stakeholders during a building project. Examples of such standards are MasterFormat and UniformatII. MasterFormat [11] is a specification standard established by the Construction Specification Institute (CSI) for most nonresidential building construction projects in North America. UniformatII is a newer American Society of Testing and Materials (ASTM) standard aimed at providing a consistent reference for the description, economic analysis, and management of buildings during all phases of their life cycles [12]. The establishment and use of such standards could thus provide a way to construct a framework that can effectively facilitate the reconciliation of semantic heterogeneity in complex application domains.

In the building construction example, a highly structured taxonomy is used to anchor the semantics of complex objects during an extended communication process. At the other end of the spectrum, we can regard the World Wide Web as a loosely structured classification system with hyperlinks among the objects being classified. Here, most complex web pages and directories are hierarchically organized (classification) and URLs are often used to denote information (referencing). This study

hypothesizes that faceted taxonomies (classification) and cross-referencing principles can be used as the basis for a generic model that addresses the semantic interoperability problem within a domain by proposing a concept termed "semantic association" that extends the concept of similarity to affinity. The semantic of relatedness between two concepts can be characterized in two measurements: similarity (how closely objects resemble each other in their representation or attributes) and affinity (to what degree they are coupled in their application context). In one sense similarity tends to be more static, while affinity is more dynamic and often depends on application context. For example, a bicycle is similar to a car due to their physical structures and properties. However, gasoline has more affinity to a car, although they do not resemble each other. Developing information models which exploit affinity in addition to similarity is the focus of the research undertaken for this dissertation.

Recently, tools for constructing mind maps, tree-like diagrams used to represent ideas [13], have been gaining popularity. One example is Freemind, a mind-mapping software suite written in Java [14] that offers a very efficient tool for creating/editing a tree and linking/navigating its nodes. With the aid of such tools, a working model can be constructed to test the hypothesis, with software engineering as the target application domain. Considering the information artifacts in the domain, an information model and associated frameworks were designed for this study based on fundamental classification and referencing principles and a framework implemented using a prototype to validate the approaches. The following section gives an overview of the research carried out for this study and its limitations.

1.1 State of the Art

1.1.1 Semantic Web and Ontology

The World Wide Web (WWW) has had a profound impact on how people access digitalized information. Nevertheless, current web technology and search engines are still limited by the restrictions imposed by the automation of information extraction. For example, web pages returned from a search engine are often imprecise and the contents rely on human interpretation. The semantic web aims to support semantic interoperability, i.e., automated web content access/use based on the semantics of the information held on the WWW [9], [15]. The vision of semantic interoperability has motivated much of the research on ontology (formal specification of conceptualization) and its languages. One example of such a language is the Web Ontology Language (OWL), which is a semantic markup language for publishing and sharing ontologies on the WWW [10]. These languages provide primitives for specifying concepts, properties, explicit semantic relationships, and logical constraints on those concepts.

Although formal modeling languages such as Unified Modeling Language (UML) are very powerful in specifying semantics, they tend to be complex and hard to practice for regular system developers who do not have extensive training on knowledge modeling. Applying a formal modeling approach to a large scale application is still problematic, as shown by the challenges faced by those applying the Model Driven Development methodologies based on these kinds of languages [35][36].

1.1.2 Integration Principles

Once autonomously developed information artifacts (semantics) are formally specified in languages like OWL or UML, the next issue is how they can be effectively integrated and used in their applications. This subsection reviews the general principles

for integrating information systems. There are two general principles for system integration presented in the research literature: integration based on shared semantics across the whole application domain and integration through mapping, reconciling, and merging of related metadata such as database schemas or ontologies [2], [3], [4], [5], [6], [7], [8]. The former case (based on the shared semantic model) assumes a priori knowledge of heterogeneity during the initial system design, which has been proven to be impractical [5], [16], [7], [17], [18]. In the latter case, studies have also revealed that mapping between heterogeneous ontologies, which seems practical at first glance, is not in fact a simple task due to the existence of semantic heterogeneity and the need for bidirectional mappings [19]. Adopting a different viewpoint, this study addressed the challenge by seeking a relaxed mapping model, where concepts are matched under their application context through semantic association. To provide the context for this research, it is necessary to analyze some motivating examples in the following subsections.

1.2 Motivating Examples

1.2.1 Simple Example of Ontology Matching (OM)

This first example examines ontology from the perspective of computer science. An ontology specifies a conceptualization of a domain in terms of concepts, attributes, and relations [20]. The concepts are typically organized into a taxonomy tree[1] where each node represents a concept and each concept is a specialization of its parent.

[1] Hierarchical *taxonomy* has a tree topology and an important category of *ontology,* which are often denoted as a general graph. It is termed *taxonomy* in this dissertation for simplicity. This dissertation especially focuses on well-established taxonomies within a domain, termed *standard* to distinguish this type from the other two.

Figure 1-1 depicts a simplified ontology cited from GLUE [7], an ontology matching system developed at the University of Washington.

Each leaf node/concept in a taxonomy is associated with a set of instances. For example, the concept Associate-Professor has instances “Prof. Burn” and “Prof. Cook”, as shown in Figure 1-1. Each concept is also associated with a set of attributes. For example, the concept Associate-Professor in the figure has attributes name, degree, and granting-institution. An instance of a concept “materializes” its attributes with values. For example, instance “Professor Cook” has values name = “R. Cook”, degree= “Ph.D.”, and granting-institution = “U. of Sydney”. An ontology also specifies a set of relations among its concepts. For example, AdvisedBy(Student,Professor) is a relation (not shown in the figure).

For the ontology in Figure 1-1, representation elements consist of concepts (e.g., Professor and Assoc. Professor), attributes (e.g., name and degree), and relationships (e.g., AdvisedBy(Student,Professor), not shown). The ontology matching problem seeks semantic similarity mappings between representation elements. For the given example, the concept Professor in the left ontology is equivalent to the concept Professor in the right ontology. The concept Associate Professor in the left ontology is most similar to the concept Senior Lecturer in the right ontology. In actual data sets, the notion of similarity is often ambiguous and the mapping is likely to be complex (one to many); for example, Name could be mapped to First Name + Last Name. The formal definitions of the matching and the similarity semantics are beyond the scope of this section and will be discussed in more detail in a later section of this dissertation.

Most of the ontology matching approaches proposed in the current research literature rely on the lexical similarity between two entities of two ontologies for a semantic anchor and then explore structural, instance-level, and/or logical information for refining matching [2], [16], [7], [21], [22]. Hence, the performance of this line of research relies on the sufficiency in lexical overlap between two ontologies. Unfortunately it is not the case in many ontologies. The following section introduces a simple example illustrating the problem.

1.2.2 Faceted Taxonomies with Static Semantic Association (SA)

Consider the two simple ontologies shown in Figure 1-2 (modified from [23]). The lexical overlap between the entities of the ontologies is expected to be low due to the complementary nature of their aspects, diseases and causes. Moreover, "Intoxication" in the first ontology is semantically very different from "Toxic" in the second, although they are lexically similar. Contexts such as abnormalities and causes must be considered to capture their semantics. However, the contexts represent two different semantics and they do not have lexical similarity. This is a gap that would prevent most proposed similarity-based matching algorithms from inferring their relationship. Relationships such as "Heroin can cause intoxication" and "drugs can cause an overdose" represent the type of semantic associations that are of interest for this study. The following section introduces a non-trivial example of the type of problem that motivates this research.

1.2.3 Large Domain Standard with Dynamic Semantic Association

The semantic association shown in the above example is static in the sense that the items are facts and do not change over time or depend on a particular context. Another kind of semantic association that is also explored in this study is dynamic and

depends on a particular application context. Consider the following simple example, illustrated in Figure 1-3, where although the external wall of a house can be associated with steel studs in a particular design, it is not always true that steel studs are used for constructing an external wall.

To explore this kind of semantic association in an important application, consider a target application in the building construction domain where interoperability problems are prevalent and human interactions are commonplace. In this domain, a variety of competing and complementary taxonomy-based standards have been established but there is no uniform and systematic way to support efficient collaboration among project participants using different standards. The interoperability cost in such environment is tremendous. For example, a National Institute of Standards and Technology (NIST) report [24] estimated a conservative figure of $15.8 billion for the costs incurred due to a lack of interoperability in the capital facilities industry in the United States in 2002 alone.

To elaborate the problem, consider the two standards MasterFormat and UniformatII, which are actually complementary to each other. These standards were created by different stakeholders with different perspectives for different purposes. For instance, an architect is interested in the design and structure of a building, a contractor wants to know what materials are used and how much they cost, and a building inspector is concerned about building code compliance issues. MasterFormat classifies items primarily based on the specification of products and materials used in construction, so adopts the conceptual view of a contractor. In contrast, the taxonomical classification in UniformatII is primarily based on the attributes and location of structural building components, such as foundations and exterior walls, which reflects

the architect's view of a construction project. Although their viewpoints are different, they both address the same building object. In other words, the taxonomies of the standards classify the same set of objects but using different attributes. Clearly, cross-referencing or document conversion between the standards is inevitable for interaction among project participants in applications such as cost estimation and code compliance checking. For example, a wall (interior or exterior) in UniformatII needs to be associated with the material (metal, wood or fiberglass) given in MasterFormat and must also conform to its intended usage (hurricane or fire proof) according to building code regulations (standards yet to be formalized by the industry).

In general, UniformatII was designed to be more suitable as a participant communication framework than MasterFormat during the earlier phases of the life cycle. On the other hand, MasterFormat has been used for years and has gained widespread support in the construction industrial for specifying detailed project documents. To facilitate more efficient collaboration among project participants, it is a common practice to supplement UniformatII with Preliminary Project Descriptions (PPDs) or schematic design in earlier phases of a project, and then convert these to construction documents in MasterFormat during the later phases. In addition, conversion is also necessary for cost calculations since most databases operated by building materials suppliers are based on MasterFormat. It is therefore desirable to transform pre-bid elemental estimates to MasterFormat, and go on to estimate the trade costs of the project [12]. This process is often tedious and requires cross-area knowledge. Currently, this is done manually by domain experts and is a major roadblock hampering interoperability in the construction domain. Bridging the two standards is thus a key enabler for

enhancing interoperability. Directly matching approaches that are based on attributes of the entities of the standards are expected to be inefficient due to the heterogeneous nature of complementary standards. This dissertation proposes a practical compromise by redefining the notion of mapping using a semi-automatic semantic extraction framework to assist domain experts in achieving interoperability, which will be introduced in Chapter 3. It is also worth considering a further example in a different domain, discussed in the following subsection.

1.2.4 Project-Specific Taxonomies with Dynamic Semantic Association

In the software engineering domain, unlike the building construction domain, the wide variety of possible applications has led to a lack of consensus regarding large-scale standards for classifying domain objects. This subsection introduces an example that classifies information artifacts such as requirements, designs, and test cases into faceted taxonomies and relates them through indexing concepts. Figure 1.4 shows such an example. There are three facets shown in the taxonomy: story, indexing concept, and requirements. The indexing concept taxonomy (as shown in the middle subtree of Figure 1.4) is analogous to the standards in the building construction examples and is therefore used as a reference model for associating domain semantics. The differences are that the indexing taxonomies are usually smaller in scale and specific to a particular application. The dissertation also explores these kinds of information artifacts and proposes a framework for managing them in Section 4.2.

1.2.5 Project-Specific Taxonomies with Explicit Semantic Association

In the example given in the previous subsection, the relationship between an indexing concept and an indexed concept is approximated based on their semantic similarity. The information artifacts indexed by the same concept have implicit

association semantics. In contrast, this subsection also describes another example where the association semantics are explicitly specified by domain experts.

The example illustrated in Figure 1.5 refers to a requirement (R1) in a data integration application. It specifies one source schema, one target schema, and their mapping rule. The semantics of the rule are specified as {a + b}. As shown in the example, the association semantics are explicitly specified and thus can be efficiently interpreted in their application. The dissertation also explores this kind of information model and proposes a framework for managing it in Section 4.3.

After analyzing the motivating examples, several key research challenges will be identified that hinder the solutions to the semantic interoperability problem. The following chapter describes the challenges, defines the research goals and discusses the approaches adopted for tackling them. The contributions made by this study to this area of research are described in the last section of Chapter 2.

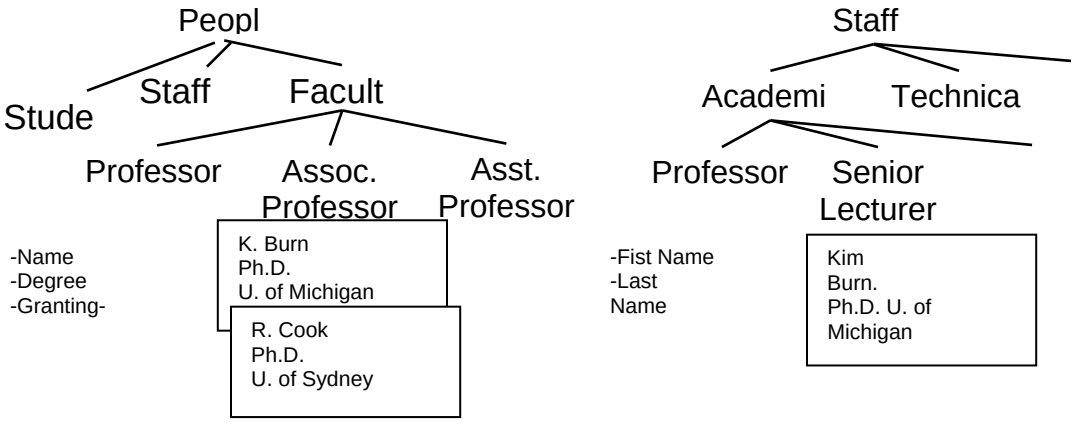


Figure 1-1. Two simple ontologies in the computer science domain

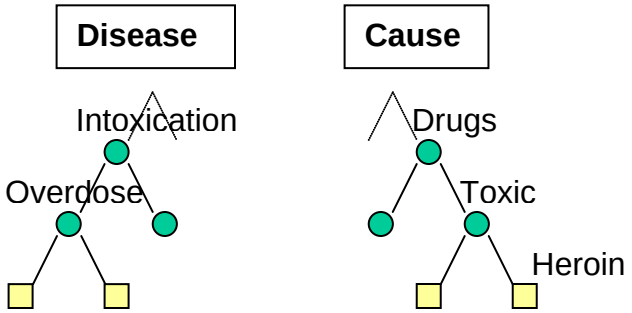


Figure 1-2. Two simple ontologies in the medical domain

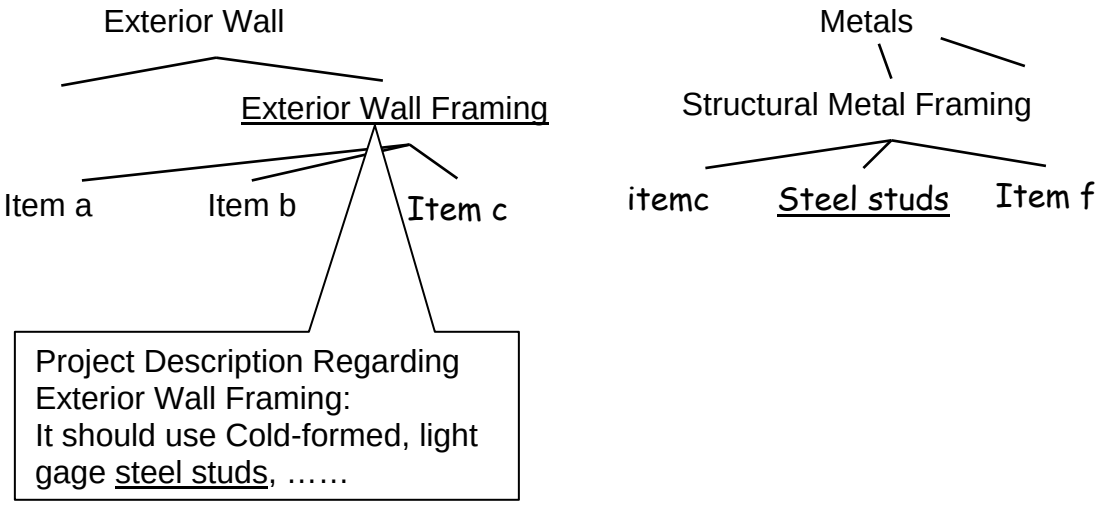


Figure 1-3. Two simple taxonomies in the building construction domain

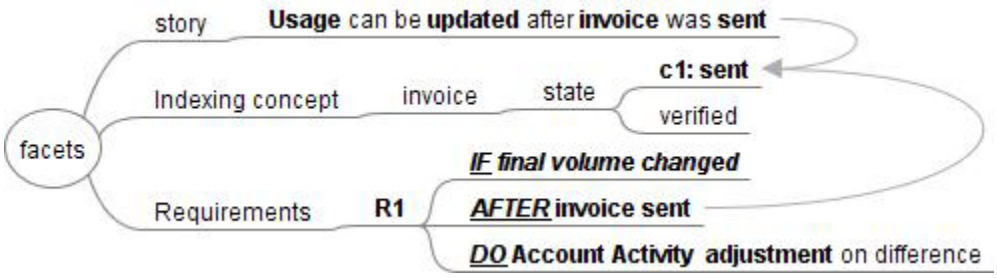


Figure 1-4. Example of faceted taxonomies in the software engineering domain

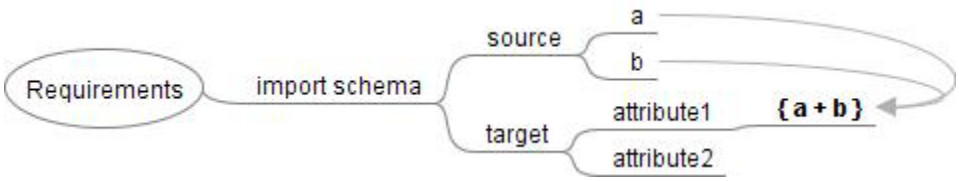


Figure 1-5. Example of taxonomy with explicit SA in the software engineering domain

CHAPTER 2
REACHING SEMANTIC INTEROPERABILITY

In computer science, most people interpret semantic interoperability as the ability to transmit digitalized information from one system to another and have it be correctly interpreted by the receiving system. In the context of this dissertation, semantic interoperability means the ability of domain information artifacts (semantics) in different perspectives and abstraction levels to be correctly transformed and efficiently used (interoperability) in their applications. With this goal in mind, this chapter examines some of the challenges that need to be overcome before automatic approaches for enhancing semantic interoperability can be realized.

2.1 Research Challenges

2.1.1 Gap between Standard and Computable

In a domain like building construction, the standards are initially designed for human use, therefore some domain knowledge that is obvious and assumed by the stakeholders is often omitted in the specifications. This presents a challenge for a systematic approach capable of annotating assumed semantics, clarifying complex concepts, and transforming them into formal representation before taxonomies can be effectively used for semantic association.

2.1.2 Implicit Affinity Semantics

Mining people's intentions through their documents is a well known difficult research challenge. This research inherits the difficulty. Although this study focuses on extracting concepts from domain project documents that are similar to those in given standards and assuming that the context around the concepts represents the affinity

semantics, this task remains very challenging because the concepts are often compound in nature and their semantics are thus hard to measure.

2.1.3 Context of Affinity Semantics

The difficulty in modeling the context of affinity semantics is implied in the lack of relevant research literature. However, it was possible to identify some context information. For example, domain experts' documents are based on a certain standard/taxonomy. The classification of this standard provides good context information that refines the semantics of the documents. In addition, the users who interpret the document have certain objectives, which in turn provide more context information. The best way to model the context to facilitate the discovery of the implicit affinity semantic remains a challenge, however.

2.1.4 Quantification of Affinity Semantics

Similar to those used in today's search engines, a ranking scheme must be devised to select the best semantic associations. The ranking is based on the degree of affinity, so affinity must be quantified. The affinity semantic is even more complex than the similarity semantic; therefore its quantification will be a challenge.

2.1.5 Reuse of Previously Discovered Semantic Association

Semantic association can be thought as a form of knowledge extraction. Previously extracted knowledge can help verify or suggest current matching. Incorporating this ingredient into the ranking formula presents another challenge.

2.1.6 Incorporation of Stakeholders' Feedback

Domain experts' feedback is the most direct and reliable semantic that the system can use for quantifying affinity. Devising a systematic way to interactively collect users'

feedback and then incorporate this feedback into the ranking formula is another challenge for this research.

2.1.7 Modeling of Stakeholders' Explicit Semantics

Affinity semantics can be directly specified by domain experts. Modeling the semantics, applying them to an application domain, and then proving the benefits of this approach are additional challenges.

2.2 Research Goals and Approaches

2.2.1 Defining Semantic Association and Problems

The first goal of this dissertation is to bridge faceted taxonomies through application semantics. As the first step to reach this goal, semantic association must be formally defined in order to provide the theoretical foundation for the new model. After the foundation has been established, problems that capture the core issues of the semantic association model can be identified and defined. The next step is to implement the new definitions.

2.2.2 Implementing Semantic Association

The following core tasks have been identified for implementing semantic association: taxonomy formalization, concept extraction, and affinity semantic measurement.

Taxonomy Formalization

Taxonomy formalization is the first step towards implementing the idea of semantic association since the entities of the taxonomy will be used as the basis of an association semantic extraction algorithm. For this purpose, the metadata or schema of the taxonomy (standard) will be defined. Based on the metadata, systematic

approaches (manual, semi-automatic, or automatic) for annotating the standards will be developed.

Concept Identification and Extraction

This task extracts affinity semantics (explicit or implicit) from domain experts' documents. After standards are formalized, based on the concepts in the standards, interested entities must be extracted from the documents in order to identify the affinity semantics. This task involves two relevant research areas: semantic similarity measurement and ontology-based information extraction. Methods commonly used in the current research literature in these research areas, such as [7], [8], [28], [29], [46], [47], can be roughly grouped into two approaches: linguistic-based and probability-based. To devise efficient algorithms, both methods need to be explored.

Algorithms for Measuring Affinity Semantics

It is necessary to develop algorithms for quantifying the concepts/semantics and their relations in order to implement the previous tasks. The algorithms are expected to build on existing research on semantic similarity measures. Unlike previous research, such as [66], [67], however, the information sources addressed in this study are classified under taxonomies since entities classifying larger overlapped objects are more likely to be related to each other in their application contexts. This intuition is stronger if we assume all standards classify objects in the same domain. It also seems likely that the more semantic association that two entities have previously been discovered to have, the more likely they are to be related under the current application context. This task is therefore to develop algorithms for analyzing the objects and the semantic association database to quantify these intuitions, combining the linguistic and

probability approaches and incorporating user feedback to create a formula that measures affinity semantics.

2.2.3 Applying and Validating Semantic Association

Once the theoretical foundation (i.e., semantic association definition) and its implementation/algorithms have been created, the second goal of this study is to apply the new constructs to manage information artifacts in the two target application domains: building construction and software engineering. This dissertation proposes three information frameworks, the first of which is for the building construction domain and the other two for the software engineering domain. They are briefly described in the following subsections:

2.2.3.1 Standard-based information framework

The framework is designed for managing standard-based project specifications in domains such as building construction. It relies on the following assumptions.

- There exist hierarchically structured standards for characterizing common processes and objects within a specific domain. That is, standards characterize domain context.
- There exist text documents that, based on the standards, specify application processes and objects. That is, standard instances characterize specific application context.
- Application context involves multiple stakeholders who have knowledge of the standards.
- The affinity semantics explicitly and/or implicitly exist in the documents.

Based on these assumptions, the tasks identified in the previous subsections (standard formalization, affinity measurement algorithm, and concept extraction) can be implemented as parts of the framework.

2.2.3.2 Concept-indexed information classification framework

This information framework is designed for managing software requirements. It hierarchically classifies primitive domain concepts into faceted taxonomies and uses their nodes (composite concepts) as indexes for automatic searches of existing requirements to facilitate the analysis of new requirements.

2.2.3.3 Taxonomy-based information specification framework

Based on the principles of classification and semantic referencing, this framework is designed for specifying information artifacts generated during the course of software development such that the development can be efficiently conducted and the resulting system can be enhanced.

2.2.3.4 Validating frameworks

This study validates the first two frameworks through an analysis of working examples, and the third through prototype simulation.

2.3 Contributions

The contributions of this research can be summarized as follows:

2.3.1 A Model for Anchoring Affinity Semantics

Affinity semantics are both practical and important, as shown in the motivating examples. Within a domain, they are closely related to the application context and thus both relevant and useful for expressing the intentions of the stakeholder who is communicating with other participants. However, they are not well addressed in the current research literature. The reasons for this may be two-fold: first, affinity semantics are hard to define due to their strong dependency on context and second there is generally a lack of information for inferring the semantics. However, analyzing the motivating examples from the building construction industry suggests that it is indeed

possible to utilize the established domain communication framework (i.e. standards, project specifications, and so on) to model the application context. The analysis also indicates that pieces of knowledge related to the affinity semantics are in the domain experts' minds and are often revealed in documents such as project specifications. These observations support the potential utility of a semantic integration model based on affinity semantics. Also, assuming the instances being classified by all domain standards are in the same set, namely the domain object set, this becomes another quality instance-level information source for inferring affinity semantics. This assumption is practical, as shown in the building construction example. With the focus of domain information elements and these working assumptions, discovering the affinity semantics becomes possible, thus distinguishing this research from other research on semantic interoperability. Preliminary research results in this area have already been published [25], [26].

2.3.2 A Model for Integrating Domain Knowledge

The time-honored principle of "separation of concerns" is often applied to manage complex knowledge. Knowledge is broken down into smaller pieces in a particular direction. As a result, faceted taxonomies are created within a domain. Examples are standards in the building construction domain and the requirement analysis methodology advocated in Aspect-Oriented Requirement Engineering (AORE) [30]. However, there are as yet no methodologies that can be used to integrate these faceted taxonomies. This dissertation proposes a taxonomy-based information model capable of providing such a methodology, published in [27].

2.3.3 Application to Software Requirement Analysis

It is widely acknowledged in the software industry that requirement analysis plays a critical role in the success of software engineering projects. However the analysis process is often arduous and delicate psychological skills and a thorough understanding of existing systems are usually required in order to conduct a thorough analysis for each new requirement. As a result, current practice still mainly depends on the use of manual processes. As part of the effort to develop a more effective automatic approach, this dissertation proposes the Concept-Indexed Information Framework that enables searching semantically related requirements, thus facilitating the analysis of new requirements. As demonstrated in a working example, the framework can help users to identify related business rules, allowing them to either create similar requirements or consolidate new ones into existing requirements.

2.3.4 Application to Software Development Process

The separation of abstractions or models from computing algorithms is one of the key principles involved in improving the quantity and quality of software systems. Based on this principle, Model Driven Development (MDD) [31] and associated tools such as Microsoft Domain Specific Language (DSL) and Eclipse Modeling Frameworks have been the focus of intense interest in both the academic and industrial communities [32], [33], [34]. However, their successful implementation in many IT projects remains problematic due to several management issues and the intrinsic rigidity and inflexibility introduced by tools built based on these methodologies [35], [36]. There remains a need for a flexible approach that not only enhances traditional software development methods but is also practical in IT development environments. With this goal in mind, this dissertation proposes a taxonomy-based modeling approach that can be used to

specify, relate, and transform information artifacts created in a software development process. The simulation of a prototype implementation shows that the new model improves the traceability, usability, and testability of the artifacts. This research has been submitted for publication in [81].

2.3.5 Potential Broad Impacts

This dissertation contends the semantic association model can be applied in other research areas, such as ontology matching, proactive information retrieval, and workflow automation. Support for this argument will be given in the following subsections.

2.3.5.1 Complementary approach to ontology matching

Semantic association and ontology matching offer complementary approaches that can be beneficial for many real-world problems. Semantic association frameworks require ontology matching technologies to identify semantic similarity between standards and domain experts' documents for extracting affinity semantics. At the same time, affinity semantics (semantic associations) can provide valuable background knowledge for enhancing ontology matching, as shown in [37]. Doan and Halevy [38] also identified complex matching, such as one-to-many or many-to-many mapping, as one of the most important open research problems in ontology matching and suggested the potential utility of domain knowledge in providing a solution. Incorporating semantic association facilitates the establishment of domain knowledge bases.

2.3.5.2 Proactive information retrieval

Traditional category-based information retrieval only considers the relationship between objects and taxonomies [39]. For example, given an object such as a keyword, the system finds related objects under one or more taxonomies. With

semantic association, it is possible to implement a novel information retrieval approach that also takes into account the relationship between taxonomies. For example, given an object under a source taxonomy, the system finds related objects under other taxonomies that have semantic associations with the source taxonomy. This approach allows searchers to specify not only their intentions when using keywords, but also the context in which they are seeking to apply the source taxonomy. Utilizing this context information and the semantic association database, the system then has the potential to infer might-desire-to-know information for the searchers, which cannot be done by traditional search models.

2.3.5.3 Workflow automation

Most workflow systems are either process- or product-based, both of which imply prior knowledge about the overall processes or products. With increasing numbers of autonomously-developed systems available over the Internet and the need for them to interact effectively, it is becoming even more important to ensure that no such assumptions exist. Therefore, newer workflow models, such as the information-based framework Web Services Choreography [40], have become an active research area. The automation of workflow based on such models or frameworks will require semantic interoperability. Semantic association, along with ontology matching, offers a useful approach to satisfying this requirement.

CHAPTER 3
SEMANTIC ASSOCIATION

This chapter describes the approach adopted for the first two tasks described in the previous chapter: defining and implementing semantic association. It starts by defining the concept of semantic association and goes on to discuss its implementation, which consists of three components: the formalization of taxonomies, the ontology-based semantic extraction and the measurement of affinity. The first component is a simple and yet novel approach for annotating a standard using primitive descriptive statements constructed by a set of necessary and sufficient orthogonal relations. These are then normalized and generalized into ontology. The second component shows how the ontology is used to extract relevant information from the instances (i.e. project descriptions) of complementary standards. The third component quantifies the affinity such that it can be used to rank the associations. To demonstrate how the semantic association model can be applied to an information system, a generic framework is introduced in Section 3.5. Finally, related works are described in the last two sections.

3.1 Definitions

3.1.1 Semantic Association

To define semantic association, a fragment of building construction project description is considered as follows:

Exterior wall should use fiberglass and need to conform to building code regulations to meet hurricane and fire proofing requirements. Also its cost should not exceed the budget of $10,000.

From the perspective of semantic association, the specification Exterior wall can be interpreted as an entity of standard UniformatII. It is associated with fiberglass in

MasterFormat with semantic made-from under the application context cost estimation. The same entity Exterior wall is also associated with hurricane and fire proof entities in a code regulation standard with semantic meet-requirements-of under the application context code conformation checking.

There are several information elements involved or implied in the statement. These are:

- The two entities having the association, such as Exterior wall and metal joists.
- The context information from which the association is found. This includes the whole project description, representing the intention of its author and an application context specified by the user performing the semantic association, such as cost estimation.
- The extracted association semantics, such as made-from.

The semantic association can now be formally defined as follows:

Semantic Association (SA) [2] is defined as the 6-tuple (id, e, e', c, a, s) where

- id is a unique identifier of a particular association.
- e is the entity of the source taxonomy.
- e' is the entity of the target standard.
- c is the context under which the association is established.
- a is the quantification of the degree of affinity. This is the value returned by an association function (AF) that associates e and e' based on context c.
- s is the association semantic that interprets the association from e to e'.

[2] The literature [41] defines a relationship, also termed semantic association, between two entities of two metadata representations in RDF [42]. The relationship is defined as either semantically similar or semantically connected in their RDF graphs. The semantic association defined in this dissertation is fundamentally different from this treatment in both the definition itself and the assumptions upon which it is based.

Here, c is the context information previously explained. The association function AF takes c as parameter and associates e to e'. The context information, c, is used in the implementation of the function for discovering the connection between e and e'. The implementation/algorithm seeks similarities among the context information and annotated standards (introduced later) to provide evidence of the connection. The algorithms can be generalized to a series of functions over a similarity function. The formal definitions of similarity and association functions are given later in this section.

Virtually all ontology matching works define a similarity function that takes a source element and a target element and returns a value (usually between 0 and 1) to indicate the degree of similarity. Inherited from the uncertainty, association function returns a value, a, indicating the degree of affinity.

The association established by AF has richer semantics than similarity. The association semantic, namely s, is extracted from the input context information and can be modeled by a set of ontologies that are involved in the similarity functions. Figure 3-1 shows an example of a semantic association.

3.1.2 Association Function

Association function is a key component in the definition of semantic association. It can be constructed over similarity functions. For example, let us reconsider the example shown in Figure 3-1. Exterior Walls is associated to Metal Joists with the semantic relation Made-from. Association function establishes the connection through similarity function over

- The concept of Exterior wall specified in the project description and the Exterior walls in UniformatII.
- The concept of metal joist specified in the project description and the metal joists in Masterformat.

To formally define the association function, it is necessary to review some works in the formal definitions of ontology matching and similarity semantics in Section 3.6. In the section, notions of interpretation and representation are introduced. There are four representations involved in semantic association, namely source standard S, Context C, target standard T, and user representation U. The authors of the four representations have their own interpretations of the objects in real world, namely I_S, I_C, I_T, and I_U respectively. Following the definition of the similarity function, the association function can be defined as the production of the similarity functions are defined over association domain **A**. Association Domain **A** is the union of elements in S, C, and T and their compound elements, constructed using a set of operators O and rules R over the representation U. The association function is formally defined as follows:

$$AF(c, s, e, e') = \sum_j \lambda_j * \prod_i (SF_k(M(a_m), M(b_n))_i$$

where λj is the weight, indexed by j, of a particular information source type that contains elements a_m and b_n; both a_m and b_n are elements of **A**. The subscript i indicates the ith pair of elements for similarity measurement during the course of the association. SF_k refers to the similarity function (algorithm) indexed by k, which is most efficient for evaluating the information source type j. λj indicates the importance of the information source type. For example, elements coming from different levels of taxonomy (the degree of specialization) have different weights. Also, elements that come from a semantic association database (the I_U domain) carry different weights from elements that come directly from the project description, where they represent the direct intention of a stakeholder (the I_C domain).

The value returned by AF indicates the degree of affinity between two elements in S and T. As observed in the related works, the semantics of all pieces of information are based on the interpretation of the matcher, i.e. I_U, so the result is just an approximation of the "true" affinity. The approximation is accumulated and can best be characterized by their parameterized production, $\prod \lambda j *$.

3.1.3 Problem Definitions

Based on the definitions described in the previous subsections, several concrete problems can be defined as follows:

Problems:

(Words in italics are the information elements defined in the Standard-based Information Framework described in Section 3.5.)

Problem 1 (One Node Semantic Association) : Given an instance under a node of the source standard, its application context and a target standard, output a set of SA between the node and the nodes of the target taxonomy that best characterizes the intention carried in the instance.

Problem 2 (Standard Selection): Given an application context and a set of standards, output a taxonomy that best characterizes the application.

Problem 3 (One Node Complex Semantic Association): Given an instance under a node of the source standard, an application context and a set of target standards, output a selected target standard and SA between the node and the target taxonomy that best characterize the application intention.

Problem 4 (SA Application to Information Retrieval): Given a set of keywords under a node of the source standard, output top N instances of associated nodes in target standards that best characterize the semantics of the keywords.

Problem 5 (SA Application to Proactive Information Retrieval): Given a set of keywords under a node of the source standard, an application context, output the top N instances of associated nodes in the target standards that might be useful for the application.

3.2 Taxonomy Formalization

This section describes the approach adopted for formalizing the large-scale natural language-based standards that have been established in domains like building construction. Standards are initially designed for human use, so some domain knowledge that is obvious and assumed by stakeholders is often omitted in their specifications. Moreover, standards classifying large and complex items usually have the following characteristics:

- The entities being classified and the attributes upon which the classification is based, are themselves complex concepts.
- Multiple attributes (different concepts) might be used to classify entities at the same level.
- Attributes are not orthogonal and may result in overlapping concepts in low-level entities (an object can fit into multiple categories).

There is a need for a systematic approach for annotating assumed semantics, clarifying complex concepts, and transforming them into formal representation before taxonomies can be effectively used for semantic association. Hence, a standard for metadata is proposed in the following subsection.

3.2.1 Taxonomy Metadata

Semantics depend on context and context depends on applications. In other words, the semantics of a standard may be open to interpretation depending on how they are used. Although it is not possible to specify the application semantics of a standard at the outset of this study, its intrinsic semantics without context can be annotated and should include the following:

- The attributes being used for the classification under the general perception in the application domain (i.e. the perspective of the classification).
- The entities under the inheritance of the taxonomy and their own attributes.

To model/store the semantics, this study proposes the use of the following metadata as a standard:

Classification goal: every taxonomy should have a goal [43]. This metadata takes the form of a text paragraph describing the goal or purpose of the standard. For example, cost estimation or procurement in MasterFormat. This annotation will be helpful when answering the Standard Selection problems by associating the application context specified by the users to this text.

Classification Perspective: this refers to the attributes being used for the classification under the general perception in the application domain. Every taxonomy classifies objects with certain attributes to meet its goal. For example, MasterFormat classifies building elements primarily based on attributes such as the material used (perspective) to facilitate contractors' jobs such as cost estimation or procurement (goal). The attributes are important for semantic association since they represent a perspective of the domain object set. If stakeholders want to communicate with each other, their

project specifications should include different perspectives and serve as an important information source for the association inference.

Relation: the relation between parent and child nodes. Is-a (inheritance) and Consist-of (composition) are two fundamental relations on which the taxonomy is constructed. The classification-based Is-a relation is the instantiation of parent objects' perspective attributes. For example, man and woman Is-a human. They are instantiations of the attribute sex (perspective attribute) of human. Therefore, the relation is more about the intrinsic properties of objects. On the other hand, the relation Consist-of is more about the applications of objects. The classification based on the relation implies that child objects have functional relationships and that these have something to do with their parent. For example, human Consist-of eyes and mouth, a house Consist-of windows, wall, and doors, or success Consist-of efforts and fortune. Based on these observations, entity, the node of the taxonomy, should be annotated to explicitly specify the semantics.

Entity: this is the main information element that the system relies on when seeking semantic association. It should contain the following three components of information:

- Semantics of entity itself: this consists of a descriptive name or label that best characterizes its semantics and a text description for elaboration.
- Semantics of structure: the annotation of the semantics described in Relation.
- Applications of the entity: the real usage of the entities. This can be obtained through previously discovered semantic associations. For example, exterior wall can be made-from steel stud, steel studs can be-used for constructing exterior wall.

To implement the metadata, ontology is considered in this research. The following subsection describes a systematic approach for transforming taxonomy into ontology.

3.2.2 Ontologizing Natural Language Standard

The term ontology has been widely used in several disciplines, particularly philosophy, epistemology, and computer science. There is much confusion in its definition. For example, in philosophy ontology refers to the subject of existence, while in epistemology it is about knowledge and knowing. In computer science, many people use Gruber's definition [1] – an explicit specification of a shared conceptualization. In the context of this research, it is interpreted as a description of the concepts/terms and relationships that can exist in an application domain. Centered on terms and relations, the transformation of taxonomy into ontology is described in the following steps:

Step 1: relation set identification

The goal of this step is to identify a sufficient and necessary set of orthogonal relations for a given taxonomy/standard so that assumed domain knowledge and complex concepts can be formally specified. This step should be manually performed by standards committees who are experts in the original intended use of the standards. The set could be constructed from two types of relations: primitive and derived. Primitive relations are those that are unambiguously understood by the general public and the relationship between concepts connected by them does not change over time. Moreover, they reflect the intrinsic properties of objects or describe time and space and the intention of users when the objects are used. In addition, their definitions should include set relationships, such as instance-instance, instance-class, and class-class, to avoid ambiguity. For example, part_of is ambiguous since it could mean a subcomponent of an object or the membership of an object in a class. Its meaning can

be clarified if the instance-instance relationship is specified. Derived relations are those that can be composed/modeled from primitive relations.

To elaborate this step, a small portion of the top three levels in MasterFormat taxonomy, Division 5 (D5) Metals and Division 6 (D6) Wood and Plastic, both rooted from Material, is exemplified as follows:

Division 5- Metals
  05100 Structural Metal Framing
    05120 Structural steel
    05140 Structural aluminum
    05160 Metal framing systems
  05400 Cold formed metal framing
    05410 Load bearing metal studs
    05420 Cold formed metal joists
    05430 Slotted channel framing
Division 6 - Wood and Plastics
  06100 Rough carpentry
    06110 Wood framing
  06400 Architectural woodwork
    06460 Wood frames

The following relations are identified for formalizing the above example:

- used_for (class-class, human intention): purpose,
- kind_of (class-class, intrinsic): containment relation of attributes of instances,
- instance_of (instance-class, intrinsic): membership,
- made_of (class-class, intrinsic): material component

Table 3-1 shows the mathematical properties of these relations that are used in the subsequent steps for data normalization. They are also used for reasoning in knowledge extraction.

Step 2: relation statements construction

This step is to construct simple statements using the relations defined in step one and all keywords in the taxonomy. The statements are then processed in the subsequent steps for constructing ontology. There are two advantages to be gained by

using this bottom-up approach for formalizing taxonomies. One is that it can better address the dynamic nature of standards by enabling incremental updates and modifications of the statements and their resulting ontology. The other advantage is that domain experts who are not familiar with ontology can directly express their knowledge in the form of simple statements without the need for complex communication with knowledge modeling experts.

The following are examples of relation statements that partially describe the example shown in the previous step.

- Metals (D5), Wood (D6), Plastics (D6_1) are instance_of Material (root) → (D5_root, D6_root, D6_1_root)
- Metals (D5) are used_for framing → 05100_1
- Structural is a kind_of "metal framing" (05100_1) → 05100
- Cold formed is a kind_of "metal framing" (05100_1) → 05400
- Studs are made_of Metals (D5) → (05410_1)
- "Load bearing metal studs" are kind_of Metal studs (05410_1) → 05410
- 05410 is used_for 05400 → (05400_05410)

Note that each statement is given a unique identifier (following →) derived from the original identifier of a taxonomy entity.

Step 3: normalization

It is likely that redundant or conflict statements will be generated along the way as domain experts annotate their taxonomies in the above steps. Based on the mathematical properties of the relations, this step normalizes the statements by:

- redundancy elimination (removing same or equivalent statements),
- conflict detection (for example: A-r1-B, and B-r1-A statements are in conflict if r1 has asymmetric properties) (Note: A and B represent two concepts in a relation statement and r1 represents a certain relation.),
- implication detection (for example, A-r1-B, and B-r1 C statements imply A-r1-C through transitive property)

Step 4: semi-automatic generalization

This step generalizes the resulting statements from step 3 into higher-level concepts connected by the same set of relations. The intervention of a human operator is required for this step due to the complexity of the process. For example, if there exist A-r1-C, A-r1-D, B-r1-C, and B-r1-D, these can be generalized to concept1{A,B}-r1-concept2{C,D} by union. However, it becomes difficult when the above example is extended to include concept1{A,B}-r1-E and concept2{C,D}-r2-F. It is ot possible to conclude concept1{A,B}-r1-concept2{C,D,E} unless an exception indicating no E-r2-F is added. Alternatively, it can be generalized to concept1{A,B}-r1-concept3{E,concept2{C,D}}. The system interacts with users by providing prompts identifying dilemmas that require resolution for the entire taxonomy. Figure 3-2 depicts the generalized view or ontology of the relation statements shown in the previous steps.

3.3 Concept Identification and Extraction

As in ontology matching research that mines all available information seeking similarity, semantic association explores all kinds of information to discover association semantics. The tasks involved in this process include interpreting the intention described in documents like project specifications and the application intention specified by the user performing the association. Other supporting clues regarding the association can be found in semantic associations contained in historical data, inherited properties under the source taxonomy, and the direct similarity between the two elements to be associated. The following section exemplifies one way to capture semantic association implied in the intention of a project specification through a linguistic approach.

3.3.1 Association through Linguistics

Standards such as UniformatII and MasterFormat are functionally complementary to each other in an application domain but are costly to cross-reference as this must be performed by domain experts in workflows due to their complexity (vast many-to-many mappings). This module basically aims to automate the process by mimicking a domain expert carrying out the cross-referencing from the context of a standard-compliant project specification and a script representation index of the standard, which defines intentionality. For example, the following text is quoted from a PPD [44] under entity B2010 in UniformatII taxonomy:

B SHELL
B20 EXTERIOR CLOSURE
B2010 EXTERIOR WALLS
Exterior Wall **Framing**: **Cold-formed**, light gage **steel studs**, C-shape, galvanized finish, 6" metal thickness as designed by manufacturer according to American Iron and Steel Institute (AISI) Specification for the Design of Cold Formed Steel Structural Members, for L/240 deflection. Downside: specifications often contain note-style sentences.

Supposedly, the PPD is written by an architect to be used by a contractor who wants to estimate cost for exterior walls. He might comprehend that the wall framing will be made of cold-formed steel studs (semantic). Based on his expertise, he identifies that its corresponding entity in MasterFormat is 05410 Load bearing metal studs (association). Figure 3-3 shows how the ontology/relation statements being used for discovering the semantic under the context of entity B2010 that links the entity to MasterFormat entity 05410 (semantic association):

In the figure, “steel” and “framing” match the statement 05100_1 (one of the identifiers of the relation statements exemplified in previous subsection) which is Metals (D5) used_for framing. The “steel” matches “Metals” through the transitive property of

the relation, kind_of. The match is extended to statement 05400, which includes “cold-formed”. Finally “studs” is added to the match of statement 05410, through statement (05400_05410). Indeed the entity B2010 Exterior Wall in UniformatII has a semantic relationship with 05410 Load bearing metal studs in MasterFormat and the semantic can be described by the relation made_of.

One characteristic worth mentioning is that the entity B2010 Exterior Wall in the taxonomy provides a good context to help refine the association. For instance, the above matching, even without the “framing” keyword, is still possible since the inherited semantic of the hierarchy, shell, closure, and exterior walls, has a very close meaning to framing.

As shown in the above example, the documents or specifications that this research addresses have the following characteristics:

- Content has limited scope. It often details what, where, how, and when objects and activities are involved in a domain application. It usually contains rich semantics (author’s intention for communicating with other stakeholders) related to standards (due to the agreement among stakeholders) that coordinate objects and activities in the domain.

- Content are categorized according to taxonomy. In other words, text in a document has some assumption or context, which is inherited along the taxonomy hierarchy.

- Terminologies are relatively unified and unambiguous.

- Sentences are relatively free styled, such as note-styled or template-styled due to writing convention or standards.

These characteristics distinguish this research from others, such as [8] and [45] which extract shallow information from general or web documents.

In addition to the intrinsic semantics of standards, this module also explores their application or context semantics in order to achieve more effective semantic extraction. The application semantics depend on the stakeholders' view or interests, such as the particular information they need to perform their jobs. For example, a cost estimator might look for MasterFormat items and some numerical information so that they can link them to their own MasterFormat-based cost databases, while an inspector might be interested in the same information but from a different view point, thus leading to different semantics. For example, to a cost estimator, "6" metal thickness" in the PPD is significant in terms of how much the studs with such thickness cost. However, for an inspector, the importance of the 6" thickness specified indicates compliance with the associated building code for the structure.

In summary, this module extracts semantics from the instances (specifications) of multiple standards based on three kinds of ontologies: the ontology of the source standard, the ontology of target standard, and the application ontology based on the stakeholders' views. The extracted semantics provide evidence for the semantic association of entities between the source and target standards.

3.3.2 Association through Probability

Another intuition that requires further study is that entities classifying larger overlapping objects are more likely to be related to each other in their application contexts. This falls under the same assumption that many ontology matching researches share. For example, [7] defines similarity based on the joint probability distribution of the concepts involved. Another example is the well known Jaccard measure [47] that computes the similarity between two concepts A and B to be P(A ∩

S) / P(A ∪ S). The approaches developed based on this intuition can be integrated into the formula of semantic association.

3.4 Measurement of Affinity

The ontology-based semantic extraction module proposed in the previous section can be implemented via a matching process between relation statements and text. The goal is to identify a set of matched relation statements of related entities with respect to their standards. For a given entity, its associated relation statements carry different weights depending on their positions in the taxonomy and the information content [47] of their keywords. The measurement of affinity quantifies the weights so that the degree of the closeness between matched relation statements and their associated entity can be determined. Based on this measurement, a ranking scheme can be devised to identify optimal semantic associations among all matches. The ranking scheme can be modeled as a function of the following factors:

- number of relation statements matched,
- number of keywords matched,
- quality of the matches

The measurement of quality is an open question. Basically, the more specific a match is, the higher its quality. One effective way often used to model the quality is in terms of the position in the taxonomy (a higher level means it is less specific and thus carries less weight) and by the information content of the keywords. The information content can be quantified by their inverse document frequency (IDF) [48] combined with their counts in the taxonomy (more appearances means a potential match is less specific and it thus carries less weight).

Let us reconsider the example in Section 3.1.1. Here, there are several entities in MasterFormat that contain “framing” and “Metals”, all of which are candidates for semantic association. The entity 05410 is considered optimal because it matches more keywords along its taxonomy hierarchy and some of the keyword matches, such as studs, are very specific with respect to both position and IDF.

3.5 Standard-based Information Framework

This section defines a generic information framework that is based on the semantic association model discussed in the previous sections. The Information elements involved in the framework can be summarized as follows:

- Standards: annotated taxonomies,
- Instances: of inter-node (for example, project descriptions), objects being classified (for example, building elements),
- Application Context: application intention/purpose, (this is used for target taxonomy selection.)
- keywords: a set of words that best characterize the target objects being looked for,
- Semantic Association Database: previously extracted semantic associations,
- External Resource: such as thesauri, and WordNet [49],
- Domain Expert Input: feedback on current suggested association

Figure 3-4 shows the framework.

The major thrust of the research described in this chapter is to develop an integration framework that facilitates the exploitation of semantics from taxonomy-based standards and instantiations of the standards to achieve higher interoperability between domain participants and their information systems. To demonstrate the applicability of the proposed approach toward the goal, Appendix A presents an overall architecture of a semantic integration framework based on the SA model, as illustrated in Figure A-1. This depicts one possible implementation and its relationship with other related technologies.

3.6 Related Work

A search of the existing literature revealed that the concept of semantic association among domain standards through affinity semantics has not previously been proposed. The closest research areas are schema and ontology matching in the database, semantic web and AI research communities. There is a great deal of overlap between approaches to matching database schema and ontology. Nevertheless, compared to database schema, ontology usually specifies richer semantics (such as class relationships, definitions of class properties, and logic axioms) and is closer to the research reported in this dissertation. This section reviews related work in ontology matching and discusses how previous research can contribute to developing solutions for the defined problems.

3.6.1 Ontology Matching

With the advance of technologies in structured data representation (such as XML), semantic-enabled applications have become realistic and have consequently stimulated much research on ontology, a framework for specifying semantics. As mentioned earlier, the most commonly used definition of ontology is a formal explicit specification of a shared conceptualization [1]. What exactly counts as an ontology has been an open topic of discussion, and the conclusions reached often reflect the different perspectives of different communities. Despite these divergent views, their common core is that ontology consists of a set of terms or concepts in a domain and the specifications of their meaning grounded in some logic theories [50]. The success of the ontological approach to semantic-enabled applications fundamentally depends on the ease of its management such as ontology alignment, reconciliation, transformation and merging. Ontology matching is the enabler for those operations. Ontology

matching seeks semantic similarity (as elaborated in earlier sections), such as equivalence and subsumption, between concepts in different ontologies.

Research in this area roughly fall into two groups based on a shared upper ontology and direct mapping using heuristic and machine learning [51]. The first group develops very general ontologies formalizing common notions such as time, space, event, processes, and physical objects, and aims to provide a semantic grounding for domain/application-specific ontology development. Some of these ontologies are becoming accepted as standards, for example Suggested Upper Merged Ontology (SUMO) [52], DOLCE [53] and the Process Specification Language (PSL) [54]. There have been reports on both the success and the difficulties of this approach [55], [56]. Several semantic integration tools have also been implemented based on this approach, such as [57]. Although extending a standard upper ontology seems a promising solution to the semantic heterogeneity problem, its success relies on comprehensive *a priori* consensus. As mentioned earlier, this assumption rarely holds true in real life. Moreover, many databases hold tremendous amounts of historical data originally created for different purposes that now need to be integrated with other data with different standards. Hence the need for direct matching heterogeneous ontologies will always remain.

Without a common inherited ontology, the second group seeks to discover the similarity between elements in ontologies based on all available data, such as class definition, entity annotation, structure information, logic constrains, and their instances. Heuristics rules, linguistic technologies, information theories [58], and machine learning

are used to explore the evidence for similarity [2], [3], [4], [7], [59], and [60]. There are two common assumptions/intuitions that this group of research relies on:

- The more syntactically similar two entities are, the more semantically similar they are. (linguistic-based approaches). From this assumption,
- The more instances that two concepts represent/classify, the more similar they are. (probability-based approaches).

The two assumptions still hold in the context of this research, therefore much of the technology developed in these areas can be used for finding semantic association.

3.6.2 Formal Definition of Matching and Similarity Semantics

Association function AF was defined in Section 3.1.2. However, in order to formally define AF, related works in the definitions of matching and similarity semantics are reviewed as follows:

Semantics are subjective. Every representation system has its own interpretation of real world objects. Let us define an interpretation of a representation to be a function that maps elements of the representation into objects in the real world. With this definition, it is possible to define what a match means. Several researchers have formally defined matching semantics based on observation. In [61], Biskup and Convent introduced the notion of integration assertions that relate (match) two elements in two schemas. Consider an expression f(s) defined over the element set s in schema S (Source) and an expression g(t) defined over element set t in schema T (Target). An integration assertion takes the form f(s) = g(t), meaning there exists an interpretation from S, namely I_S, and an interpretation from T, namely I_T, that map f(s) and g(t) into the same objects in the real world. In [62], Madhavan and coworkers extend the matching

semantics to the form f(s) op g(t), where op could be =, $\subseteq$, or $\in$, depending on the output types of f(s) and g(t).

It is possible that no interpretations exist that relate two elements in two representation systems into the same objects in the real world even though the two elements are in fact semantically related. An example of this would be the case where f(s) refers to an assistant professor at the University of Florida in the USA and g(t) refers to a senior lecturer at Cheng-Kung University in Taiwan. There is thus a need for a common reference model so that the equivalence of the two example elements can be inferred. In [63], Doan introduced the user representation U for this purpose. U is known by the user, who interprets the matching from S to T. It is assumed that the user understands S and T and this understanding is modeled as the mapping from any element of representations S and T into a semantically equivalent element in U. Doan characterized this mapping process with a function M. The notations M (s) and M (t) then denote the elements in U that are equivalent to s and t, respectively. The function M was formally defined in [63] as follows:

Definition (Mapping Function M) Let S and T be two given representations, and let U be a user representation. Function M pairs elements from S and T with those in U such that there exist interpretations I_S, I_T , I_U (for S, T, and U, respectively) that satisfy:

$$\forall s \in S, \{ I_S, I_U \} \models \{s = M(s)\} \text{ and } \forall t \in T, \{ I_T, I_U \} \models \{t = M(t)\}$$

Consider the meaning of the first formula. The statement “element s of S is semantically equivalent to element M(s) of U” means that there exists interpretations I_S for S and I_U for U such that they map s and M(s) into the same object/concept in the real world.

Similarity semantic is assumed by most ontology matching works. To formally define the notion, Madhavan and coworkers [62] assume the existence of a user-defined similarity function SF over the concepts of U. SF takes a pair of concepts, namely u1 and u2 in U, and returns a value that indicates the degree of similarity between u1 and u2. The larger this value is, the higher their similarity. Without loss of generality, the returning value falls inside the interval [0-1], which is consistent with the definition of probability. Using these definitions of U, M, and SF, for this study the **1-1 matching problem** is defined as follows:

Given two representations S and T, for each element s of S, find the element t of T that maximizes SF(M(s),M(t)).

As mentioned earlier, these interpretations are inevitably subjective. In practice, the interpretations for target and source representations, i.e. I_S and I_T, are not accessible from the user/tool that does the matching but exist only in the minds of the authors of the representations. All the data concerning the representations are in the form of what are referred to in this research as syntactic clues. Therefore, most of the proposed ontology matching algorithms can be considered as approximating true similarity values using these syntactic clues. There is an important assumption behind this approximation: The more similar the syntactic clues of the two elements s and t, the more semantically similar the two elements are. Therefore the definition of the **1-1 matching problem** approximately maximizes SF(M(s),M(t)), which introduces an uncertainty factor known as the returning value. Hence, for this study the term **complex mapping** is defined to relate an element s of a representation S to a formula F that is constructed over the elements of another representation T using a set of operators O

and rules R. The operators of O and rules of R are assumed to be well-defined over the user representation U so that M(F) can be recursively defined in a trivial manner. If F = $o_i(t_1, t_2)$, where o_i is an operator and t_1, t_2 are elements of T, then $M(F) = o_i(M(t_1), M(t_2))$, and so on. With the definition of M(F), the **complex mapping** can be defined in such a way as to maximize SF(M(s),M(F)).

As shown in the above reviews, similarity is the common presumption of the work as given in the formal definitions. In contrast, semantic association regards the similarity as a way to seek a connection between source and target elements through associated context information. Its formal definition was given earlier in section 3.1.1.

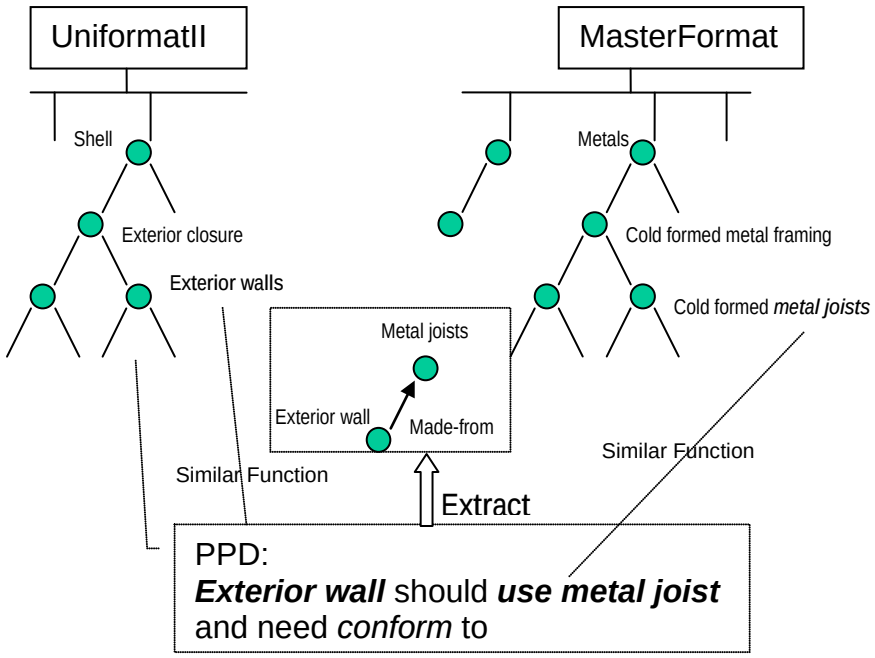
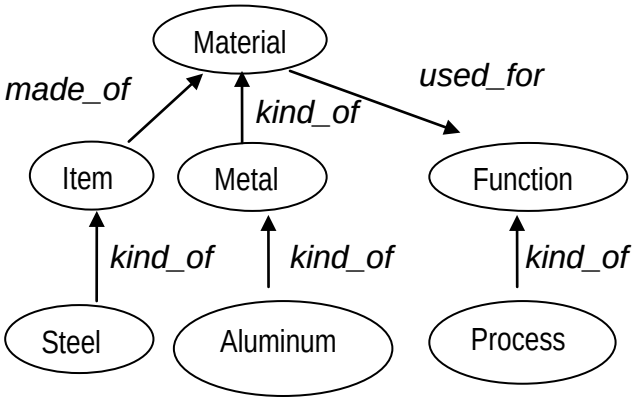


Figure 3-1. Example of semantic association

Table 3-1. Mathematical properties of the relations

Relations	Transitive	reflexive	antisymmetric
used_for	no	no	no
kind_of	yes	yes	yes
instance_of	yes	yes	yes
made_of	yes	no	no



{metals, wood, plastics ..} are instance_of Material
{stud, joist ..} are instance_of Item
{framing, ..} are instance_of Function
{cold formed, structural ..} are instance_of Process

Figure 3-2. Ontology **e**xample

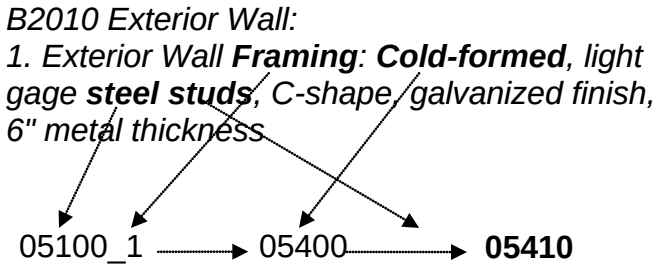


Figure 3-3. Example of semantic discovery

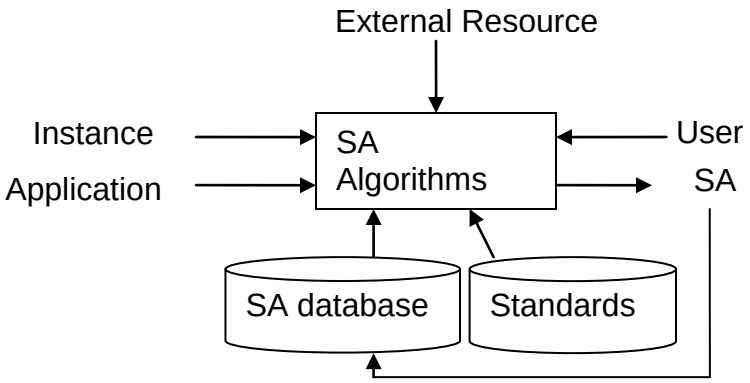


Figure 3-4. Semantic association framework

CHAPTER 4
TAXONOMY-BASED INFORMATION MODEL

In the previous chapter, the concept of semantic association was defined, approaches for implementing the concept proposed, and the new approaches applied to manage building construction project documents. In this chapter, the same principles (i.e. the classification and the cross-referencing) are applied to model generic domain knowledge such that it can be effectively accessed and used in real-world applications. The model is then used to enhance semantic interoperability in the software engineering domain. More specifically, the model is applied to a software requirement management process to facilitate requirement analysis and enhance the system's traceability, reusability, and testability.

4.1 Overview of the Model

This dissertation proposes four aspects that a generic information model must include: 1. object classification, 2. object specification, 3. query language, and 4. applications. To give an overview of the proposed model, some well known information models/systems are compared based on these four aspects in the following subsection:

4.1.1 Comparison of Existing Information Models

Consider the first two systems shown in Table 4-1. The Google search system can be regarded as a very loosely specified and classified information system. The objects being managed in the system are web pages that are autonomously specified. The web pages reference each other via hyperlinks and they are loosely classified and indexed by the Google system. At the other end of the spectrum (the degree of formalism), a relational model strictly normalizes/specifies data into tables within an application domain and relates/classifies them through predefined primary and foreign

key constraints. From the viewpoint of the spectrum, the information model proposed in this dissertation sits somewhere between the Google and Relational models. The following table shows a comparison between the characteristics of the proposed model and other systems.

4.1.2 Specification

Any parse-able forms can be used in the specifications of objects. Figure 4-1 shows an example that specifies a use case. Form or Template should have the following characteristics:

Transformability

The forms should be transformable (via transformers) to other forms for some applications. Examples of such transformations will be provided in later sections of this chapter.

Reusability

Any object being documented incurs cost (authoring and maintenance). Therefore it is important that the specifications should be frequently reusable in the lifecycle of the project development to justify that cost. In the target application, the usage oriented requirement specification is identified as shown in Figure 4-1. Requirements are documented in the viewpoints of how actors use the features and the features are elaborated in appendixes and references using existing structures/diagrams such as UML. This is relatively efficient since they can later be transformed into a regression automatic test suite, which reduces the cost periodically incurred in QA (Quality Assurance).

4.1.3 Classification

In the new model proposed here, objects are classified into five major facets: function, system concern (aspect), is-a, part-of, and atomic semantics. The function taxonomy classifies the core features of the application system. The purpose of this classification is to facilitate information identification via navigation or queries. The system concerning taxonomy is used for untangling unrelated information and for addressing the repeated or patterned information that is otherwise scattered across the functional hierarchy. The first two taxonomies are motivated by the objectives advocated by those in the AORE community. The remaining three facets are included in the domain ontology and are used for indexing concepts in the instances of the forms and for the semantic measurements in answering a Concept Indexed Structured Query Language (CISQL) query, which will be introduced in the following subsection. The following is an example of such a classification:

4.1.4 Query Language

Before a knowledge base can be applied, it is first necessary to select a language for Information Retrieval. To be able to answer searchers' queries, information from different sources must be merged and focused. In SQL, this information merging is performed via a key and foreign key relation and the SELECT and WHERE clauses focus the returning data to the questions of interests. In the Google system, only a specific query is considered. Elements in a query consist of keywords, logic (AND and OR), the attributes of the web pages, and the attributes for rendering the results. The information merging and focusing are not obvious in the Google system but rely on users to interpret and merge returning pages across multiple queries in order to answer their questions. In CISQL, the information merging proceeds via indexing concepts and

the focus is on the classifications of the taxonomy and the structures of the information artifacts themselves. The following shows an example of CISQL:

select useCaseTemplate.actors

under Contract&module, Invoice&Module

where is-a(contract&state) and is-a(GAS&Roles)

Since the requirements are written in natural language, concept indexing is based on semantics rather than syntax and is implemented using semantic similarity measurement. The information merging through indexing concept is analogous to the key- foreign-key relation in the Relational model. The differences are that the relations in CISQL are not predefined and the merging is approximated.

4.1.5 Applications

An information model should have a primary application. For example, in the Google system, identifying a web page is its main application. In a Relational system, data management is the main application. In this information model, domain knowledge management is the main application. More specifically, data management includes knowledge retrieval, analysis, consolidation, and transformation (under an application context). Examples of these are given in Section 4.2 and Section 4.3.

4.2 Concept-indexed Information Classification Framework

Section 4.1.3 introduced a way of classifying information into five orthogonal facet taxonomies: core features, system concerns (aspects), is-a, part-of, and atomic semantics. The first facet is naturally and commonly used in most application domains. The second facet organizes system concerns that otherwise tangle multiple information objects classified under the first facet taxonomy. This section introduces a classification framework that focuses the idea of indexing concepts that, in some sense, can be

considered as a special kind of system concern since indexing concepts usually exist in multiple information objects classified under the core features taxonomy. As described in Section 4.1.4, these indexing concepts are used for information merging so they are usually important for some applications. The following subsection introduces an example of how the indexing concepts are used for software requirement retrieval, analysis and consolidation.

4.2.1 A Working Example

Figure 4-3 shows an example of faceted taxonomy that is typical of those used for managing requirements in the target application of this study- an accounting system for the utility industry. There are three facets shown in the taxonomy: story, indexing concept, and requirements. This example describes an issue raised by an accountant, Mary, who points out that it is possible in business practice that the final volumes of customers' transactions may need to be changed after they have already been invoiced (shown as s1 in the top sub-tree of Figure 4-3). After analysis, the Business Analyst (BA) specifies the requirement as follows:

IF ([final volume changed] **AND** [invoice status is sent]) **THEN** ([calculate transaction differences] as adjustment **AND** [automatically post adjustment to account activity] **AND** [send notification to users]).

The text within [] denotes an indexed Statement concept, which will be defined in the following section. The operators, IF, THEN, and ELSE, are rule constructors which are

kinds of application semantic constructors [3]. The requirement is labeled R1, as shown in the bottom sub-tree of Figure 4-3.

Consider another example: accounts clerk John tells another BA that there are multiple adjustments for the same transaction in a customer's account activity (labeled s2 in the story sub-tree). Due to his lack of knowledge regarding R1, which was specified by the previous BA, he might experience some difficulty in understanding the relationship between transaction editing and the account activity adjustments. With the help of the semantic association, however, he finds R1 through the Indexing Concept, c2:system (shown in the first branch of the middle subtree in Figure 4-3), and the indexed concept adjustment to account activity, in R1. After studying R1, he realizes that the multiple adjustments are caused by multiple transaction edits so he is able to consolidate the new requirement introduced by the story to R1 by adding additional checking to the existing transaction adjustment.

On another occasion, Lucy tells a new BA the third story (labeled s3 in the story sub-tree), pointing out that retail customers' gas usage can be updated after their invoices have already been sent. The new BA might have no idea how to start to analyze the impacts of this new story. Since the concept invoice status is sent in R1 is indexed and the story contains two keywords, invoice and sent, by conducting a semantic association search the new BA finds R1 and uses it as a model to facilitate his analysis. He realizes that the new requirement can actually be merged into R1 by modifying the IF condition as follows: IF ([final volume changes] OR [usage updated]). As shown in the example, the indexed concept in the story and the requirements are

[3] The IF, AND, THEN are keywords defined in our prototype for constructing business rules. Application semantic constructors can be any formal conceptual modeling language, such as UML.

linked via indexing concepts. For example, s3 and r1 are linked through c1, as illustrated by the left two arrows in Figure 4-3. From a different viewpoint, indexing concepts are semantically associated through requirements. For example, c1 and c2 are semantically associated (SA) through R1 (as illustrated by the lower two arrows in Figure 4-3). Since there are logical relationships, i.e. semantics, R1 can serve as the application semantics for the SA, so R1 is termed an SA statement. The semantic association can also be established through stories. For example, c1 and c2 are semantically associated (SA) through s2 and s3 (as illustrated by the upper two arrows in Figure 4-3). However, the application semantic of this SA is not explicit, as the relation between s2 and s3 is not clear. Due to R1's SA (which also associates c1 and c2), it may be possible to discover the implicit relations between s2 and s3.

Based on the concept of semantic association and its application to requirement analysis, it is now useful to formally define this concept starting with some terminology and concept definitions.

4.2.2 Revisit Semantic Association

4.2.2.1 Term definition

Keyword: a word or a phrase for the basic element that represents a concept. Examples are the nodes of is-a taxonomies in WordNet.

Indexing Concept: the semantic of a taxonomy node. For example, the c1 node in Figure 4-3 signifies an invoice with state "sent".

Indexed Story Concept: the semantic extracted from the text to be analyzed/searched (such as user stories or use cases). This can be characterized by several keywords, for example "final volume changed" in s1.

SA Statement: an instance of the data structure defined by the application semantic constructors. For example, <if> A <then> B where A and B are indexed statement concepts.

Indexed Statement Concept: the semantics of an operand in an SA statement that has strong similarity to an Indexing Concept. An example is "automatically posting adjustment to account activity" in the previous example.

Semantic Connectivity: two concepts are semantically connected if there exists an SA statement that contains both concepts.

Semantic Similarity-α: two concepts, c1 and c2, are semantically similar if there exists a similarity measurement that grades them above α where 0 <= α >= 1.

SA Statement Associability: two SA statements, s1, s2, are semantically associated if there exist concepts, c1, c2 and c3, where c1 ∈ s1, c2 ∈ s2, c3 ∈ indexing concept, and both c1 and c2 are similar to c3. For example, c1 and c2 are concepts indexed by c3.

Indexing Concept Semantic Associability: two Indexing concepts, c1, c2, are semantically associated if there exist two Indexed Statement Concepts, c1' and c2' where c1' and c2' are either ∈ a SA statement or ∈ associated SA statements, and (c1 and c1' are similar) and (c2 and c2' are similar). In other words, if an SA statement or associated statements contain concepts that are indexed by c1 and c2, c1 and c2 are semantically associated.

Association Graph: an association graph consists of concepts that are connected by either Semantic Connectivity or Semantic Similarity–α (as defined above). In the

graph, Semantic Connectivity and Semantic Similarity–α are visualized as a directed edge and a weighted undirected edge, respectively.

Direct Semantic Associability: two Indexing Concepts, c1, c2, are directly associated if there exists a path in the association graph that starts with c1 and ends with c2 and the path contains at least one directed edge.

Indirect Semantic Associability: two Indexing Concepts, c1, c2, are indirectly associated if there exist two paths in the association graph that start with c1 and c2, respectively, and intersect in either an Indexed Concept or two Indexed Concepts in the same SA statement, but there is no directed path between them. In other words, c1 and c2 indexed SA statement concepts have no explicit logic or dependency relation but may have an implicit relation.

Note that due to multiple associations, two indexing concepts can have both direct and indirect associations.

4.2.2.2 Semantic association definition

As with the semantic association definition introduced in Section 3.1, semantic association can now be defined as follows:

Semantic Association (SA) is defined as a 7-tuple (id, e, e', s, c, as, a) where

- id is a unique identifier of a particular association.
- e is a source indexing Concept
- e' is a target indexing Concept
- s is a list of Indexed Concepts connected by relations
- c is the context that denotes the semantics of the indexed concepts. This can be represented as a list of keyword vectors, each of which is associated with an indexed concept for determining the senses of the keywords in the concept.

- as is the application semantics used to interpret the association from e to e'.
- a is affinity. This is the value returned by an association function AF and will be defined in the following section. It is a real number between 0 and 1.

As for the association function defined in Section 3.1.2, the association function AF takes c and s as parameters and applies semantic similarity algorithms to discover the association between e and e'. It returns a to quantify their affinity, indicating the reliability of the association. The association has semantics, namely as. In this framework, it is implemented by the SA statements.

Two core problems must be solved before the semantic association and the association function can be implemented - the semantic similarity measurement between indexing and indexed concepts and the affinity quantification between two indexed concepts. These problems are addressed and solutions proposed in the following section.

4.2.3 Semantic Similarity Measures

4.2.3.1 Vector-space-model based algorithm

The feasibility of the proposed model depends on the performance of semantic similarity measures between indexing and indexed concepts. Semantic Similarity measures between two atomic terms have been well studied in areas such as information retrieval, natural language processing, and ontology matching [66], [67]. Methods commonly used in the proposed approaches for deriving semantic similarity can be roughly grouped into two types:

- co-occurrence statistics from corpora,
  (This is based on theories such as Distributional Hypothesis and Information Content. The Distributional Hypothesis theory states that synonyms tend to

appear in similar linguistic contexts. The Information Content theory states that words with global low occurrence frequency often carry more information)

- is-a ontology specified in lexical database like WordNet
  (This is based on intuitions like the idea that path length and common subsume can characterize semantic closeness.)

Built upon these intuitions/theories, a new approach can be constructed using the Vector Space Model (VSM) to approximate the semantics of the complex concepts referred to in our research context. Each of the unique keyword senses in the taxonomies and the domain knowledge forms a coordinate basis vector of the vector space. Therefore, the number of all unique keyword senses represents the dimensions of the vector space. A unique keyword sense is represented as a set of synonym with a vector of context words and an optional pointer to an is-a ontology to capture its semantic neighborhood. A match of two words in our approach means either:

- matching on any word in synonym with high similarity in context words,
  (Matching on a word without similar context could indicate polysemy, i.e. words that have more than one meaning)

- matching on different words that are within the semantic neighborhood specified in the is-a ontology

For simplicity, keywords are referred to unique keyword senses later in this section. In this new approach, each concept consists of a set of keywords and represents a vector in the vector space. For example, an Indexing Concept Vector consists of keywords from the node of a taxonomy, its parent nodes and the associated nodes. In Figure 4-2, node **1.1** and **1.1.1** are associated (**1.1.1** represents the components of **1.1**). The keywords are weighted based on their usage (reference frequency) and information content. Intuitively a keyword that has more references should have a higher likelihood of being used for future references. Information content is a static statistic metric. It

indicates how effectively the keyword contributes in identifying an indexed concept. The information content is decided by a local parameter, Facet Information Content (FIC), in a concept and a global parameter, Inverse Concept frequency (ICF) across indexed concepts in taxonomies. FIC of a keyword represents its importance in indentifying the semantic of the concept. For example, the semantics of node **1.1** can be described as "the edit function of physical transaction, which consists of price and final volume". The word "edit" has a high FIC since the facet is about function and edit is a kind of function. ICF of a keyword quantifies its specialization and represents its capability in discriminating concepts. Based on these metrics, an Indexed Concept is represented as:

$$V_c = \left[W_{1,c}, W_{1,c}, W_{3,c}, \dots, W_{N,c}\right] where\ W_{k,c} = SF_k * FIC_k * ICF_k$$

$SF_k = \frac{sf_k}{SF}$ where sf_k is the frequency of keyword k being associated and SF is the sum of the frequency of all keywords being associated.

$FIC_k = \rho * kind_of(k, f)$where k is the keyword k, f is the facet keyword, $kind\_of(k, f)$ =1 if k is "is-a" derivable from f ; 0 otherwise. ρ is a parameter between 0 and 1.

$ICF_k = log\frac{|C|}{1+|\{c:t_k \in C\}|}$ where |C| is the total number of concepts in the taxonomy and $|\{c: t_k \in C\}|$ is the number of concepts where the term t_k appears. Note that, due to the nature of taxonomy (terms in a higher level node are shared by all its child nodes), one occurrence of a term can only appear in a concept once. With the above definition, similarity between two concepts can be quantified as follows:

$$sim(c1, c2) = \frac{V_{c1} \cdot V_{c2}}{|V_{c1}|| V_{c2}|} = \frac{\sum_{i=1}^{N} W_{i,c1} * W_{i,c2}}{\sqrt{\sum_{i=1}^{N} W_{i,c1}^2} * \sqrt{\sum_{i=1}^{N} W_{i,c2}^2}}$$

where N is the dimension of the vector space and * denotes two word matches for a unique keyword sense. The definition of match was described earlier.

4.2.3.2 Graph based algorithm

In this subsection, another algorithm is proposed for measuring semantic similarity. Semantic similarity measures differences between two atomic terms/concepts that represent fundamental research in areas such as information retrieval, natural language processing, and ontology matching. When these concepts are invoked in the present context they are often composited, hence the task of measuring their semantic similarity is even more challenging. The approach described in the previous subsection is based on the Vector Space Model (VSM) [68] for approximating the semantics of these complex concepts. This is termed FIC-ICF weighting in the current study as it combines the application of Facet Information Content (FIC) in a concept with a global parameter, Inverse Concept frequency (ICF), across indexing concepts in taxonomies. This is a modified version of the tf–idf weighting formula (term frequency–inverse document frequency) [69] often used in information retrieval and text mining. In the alternative approach proposed in this subsection, a graph-based algorithm is used to identify the most similar concept to a given domain concept that is represented in the node of an indexing concept taxonomy. Before introducing the algorithm, it is important to first define what is meant by a Match for a keyword in this new approach. As defined in the previous subsection, every keyword has a set of unique keyword senses. A unique keyword sense is represented as a set of synonyms with a vector of context words. A Match in the new approach means matching any word or its synonym with a high similarity in context words, with the high similarity defined as follows:

High_Similarity(k_a, k_b) = $\frac{A \cap B}{A \cup B}$ > α, where A and B are the context word vectors for keywords $k_a and\ k_b$, respectively, α is a threshold between 0 and 1. The value of High_Similarity(k_a, k_b) is either 0 or 1.

The definition of Match is the following:

$\mathrm{Match}(\mathrm{k_a}) \equiv (\mathrm{k_a} = \mathrm{k_b})$ and High_Similarity($\mathrm{k_a}, \mathrm{k_b}$) $\mathrm{where}\ \mathrm{k_b} \in$ SynonymSet(k).

Beyond the keyword matching, it is preferable to match several keywords that denote a concept. For example, Figure 4-4 illustrates a simple example of indexing concept taxonomy.

Suppose a user inputs a domain concept that consists of two words: “transaction” and “edit”, meaning a transaction is edited. The concept node in the figure that has the most similar meaning is the node “change” in the bottom sub-tree of the figure. This node denotes a concept regarding the change of gas final volume. It is semantically similar to the input domain concept because the final volume is an attribute of transaction based on the top sub-tree and change is a synonym of edit.

Given a domain concept, the most similar concept node in an indexing concept taxonomy is defined as the most specific node of the most condensed spanning sub-tree. The spanning sub-tree (K) is defined as a sub-tree of the taxonomy that minimally covers one occurrence of K Matched input keywords where 1 <= K <= M; M is the number of input keywords. The most condensed means that it contains fewer edges and leaf nodes. Intuitively, it is possible to list the following characteristics: a sub-tree with one path is considered to be a desirable sub-tree, having only one leaf node. The shorter the path, the more condensed it is. A sub-tree with only one node is considered the most condensed. The fewer leave nodes that the sub-tree has, the more

condensed it is. Based on these intuitions, the following weight function can be formulated for a spanning sub-tree:

$$\text{Weight(Subtree)} = (1 - \mu - v)\sum_{i=n}\left(\text{Match}(k_i)\right) + \frac{\mu}{l} + \frac{v}{e+1}$$

where n, l, e are the numbers of matched keyword, leave nodes, and edge respectively, $(1 - \mu - v)$, μ, v are the weights for keyword, leaf, and edge respectively, $0 \leq \mu + v < 1$ and $0 \leq \mu, v < 1$.

The new algorithm for finding the most similar concept is outlined as follows:

Input: 1. a tree with N nodes; 2. M keywords representing the concept to search.

- Traverse the input tree to find Matched keywords and group them based on input keywords, where each group contains all occurrences of Match for an input keyword. Trim all sub-trees with no word matches. Complexity: O(N*M)

- List all combinations of K matched keywords in different groups. Each combination contains one occurrence of the K matched input keywords, which represents a spanning tree (K). Iterate K from 1 to L where L is the number of all Matched input keywords. Complexity: $O(N^M M^M)$

- Count the numbers of edges and leaf nodes and apply the weight function for each combination/sub-tree. Complexity: $O(N^{M+1} * M^M)$

- Pick the sub-tree with the maximum weight, which will be the most condensed spanning tree.

Output: return null if no sub-tree is found; if the leaf node number of the sub-tree > 1 return to the root of the sub-tree; otherwise return the lowest node in the path

Since the input keywords represent a concept, the number of the keywords is limited to a constant, say 3. Therefore the complexity of the algorithm is $O(N^{3+1})$.

Revisiting the example shown in Figure 4-4 and applying the algorithm, "transaction" has one match, as shown in the node labeled "2" in the top sub-tree of Figure 4-4, and "edit" has two matches, represented by the two nodes labeled "1" in the bottom sub-tree. The arrow from "final" node to "final volume" node signifies that these two nodes are semantically the same. Therefore, equivalently the sub-tree of the "final" node can be copied and inserted under the "final volume" node. In this case, the sub-tree is just a node "change". After the duplication, the resulting taxonomy effectively contains three spanning sub-trees that cover the matched words <transaction, edit>, <transaction, change>, and <transaction, change (the duplicate)>[4]. The next step is to determine which of these is the most condensed. The edges of the three spanning sub-trees contain 4, 4 and 2 edges respectively and all have only one leaf node. Therefore, the node selected is the node containing the duplicate. This is equivalent to the "change" node selected manually earlier. As shown in the weight function, the semantic similarity between an indexing concept and an indexed concept are approximated. The degree of similarity can be quantified by normalizing the weight of the returning sub-tree to between 0 – 1.

4.2.4 Affinity

4.2.4.1 Local affinity

After similar indexing and indexed concepts have been identified ($\text{concept\_similarity}(c1, c2)$ >= a threshold), they are associated and association paths are formed. These association paths can be direct or indirect, as noted in the definitions in Section 4.2.2.1. Local Affinity is thus defined as follows: Given an

[4] Ignore spanning trees with one matched keyword for simplicity.

association path (direct association) or two intersecting paths (indirect association), a real number between 0 and 1 indicates the degree of association between the first concept and the last concept in the path(s). This value is determined by a particular implementation of the abstract association function defined in Section 3.1.2. For example, assume all the representation systems are homogeneous and the reliabilities of all data sources are equal. If the input is a direct association, the local affinity is simply the product of the weights of all the similarity edges in the path. If the input is an indirect association, the "gap" where there is no explicit relation that connects the two concepts means that an uncertainty factor needs to be considered. These intuitions can be formulated as follows:

$$LA_D(E_{s,}\, E_c) = \prod_i W_{Si}$$

$$LA_I(E_{s,}\, E_c) = \sigma \prod_i W_{Si}$$

where LA_D and LA_I are the local affinities for the direct and indirect paths, respectively, and $E_{s,}\, E_c$ are the sets of similarity and connectivity edges for the input path(s), respectively. W_{Si} is the weight of the ith similarity edge; and σ is the quantification of the uncertainty whose value is between 0 and 1. Note that it is also possible to interpret σ as "un-annotated semantics". The identification of indirect associations is likely to be important since they are the places to explore implicit semantics.

4.2.4.2 Global affinity

Two concepts can be associated for multiple related or unrelated semantics. Their global affinity accounts for all known semantics. Given two indexing concepts and an association database, the Global Affinity (GA) is a real number between 0 and 1, representing the total affinity between the two concepts normalized by the total affinity

of all pairs of concepts in the database. This can be evaluated using the following two steps: search for all paths linking the two given concepts, namely Path(s)_C1_C2, and search for all paths that exist in the database, namely Path(s)_ALL. The GA can then be defined as follows:

$$GA(C1, C2) = \sum_{i}^{M} LA(E_{si,} E_{ci}) / \sum_{i}^{N} LA(E_{si,} E_{ci})$$

where $E_{si,} E_{ci}$ are, respectively, sets of similarity and connectivity edges of the ith path(s) in Path(s)__C1_C2 or in Path(s)_ALL. If the path is direct, LA_D applies, otherwise, LA_I applies. M is the number of path(s) that the two given concepts share and N is the total number of association paths that exist in the given database.

The indexing concepts and their GA can be visualized as a map where concepts are nodes and their GA are edges similar to van Ham et al.'s Phrase Nets, which offer visual overviews of unstructured text [70]. The size of a node is proportional to the sum of its GA with other nodes. The size of an edge is proportional to its GA. The map provides an overview of the concepts and their SA. It is possible to apply a filter to SA to change the map for a particular viewpoint. Other potential applications of the map are open to exploration; one possible application is to facilitate requirement searches and mining.

4.2.5 Search Semantic Association

Searching associations is an important operation when discovering implicit semantics and consolidating association semantics. For example, in addition to searching all associations for a particular indexing concept in order to consolidate their semantics, it is also possible to search for associations between two indexing concepts to analyze their association semantics and find indirect paths and thus implicit

semantics. In this implementation, semantic associations are stored in a relational database, so searches can be implemented using SQL as:

Select top n * from SemanticAssociation sa where hashkey(Indexing-concept1) = sa.startingConcept and hashkey(Indexing-concept2) = sa.endingConcept order by sa.affinity desc

4.2.6 Implementation

Figure 4-11 shows the implementation overview of the Taxonomy-based Semantic Association Framework (TSAF) which is based on the information model introduced in this chapter. The top two branches of the figure show the data model of the framework. Its UML diagram is illustrated in Figure B-1 (Appendix B). There are three types of concepts (TaxonomyConceptVO, ApplicationConceptVO, and SAConceptVO), all of which are inherited from an abstract class called ConceptVO (as shown in the top branch of Figure 4-12). VO is a naming convention signifying Value Object (this type of class is mapped to a table in database). The concept represented by a (leaf) node in taxonomy is implemented as TaxonomyConceptVO. This can be used as an indexing concept for information retrieval or can serve as context information for anchoring the semantic of the object classified under the node. The second top branch of Figure 4-12 shows the components of a Semantic Association (SA). The association relates two TaxonomyConceptVO objects based on the context information, consisting of association semantic and application semantic (implemented as SASemanticVO and ApplicationConceptVO respectively). The definitions and the relations of the classes in this data model can be found in the UML diagram illustrated in Figure B-1.

Concept indexing and semantic measurement are the primary operations for seeking semantic association. The implementation outline of this module is illustrated in

the top third branch of Figure 4-12. The driver class is SimilaritySearcher. This contains an indexer, namely iafIndexer, which uses TsaVectorGenerator to parse taxonomies and build an aspect-word matrix. One column in the matrix represents a vector that encodes a taxonomy concept. The values of the vector are numbers generated by the algorithm described in Section 4.2.3.1. Once the matrix/index is built, it is possible to search a concept (a vector representing an indexed concept) against the matrix based on similarity measurement methods like cosine, described in Section 4.2.3.1. The two branches in the bottom of Figure 4-12 show two main classes in the service layer of the implementation architecture, namely Semantic Association (SA) Manager and Taxonomy Manager. The primary operation of Semantic Association Manager is to find a list of SAs based on inputs SASemanticVO and ApplicationConceptVO objects. The SASemanticVO object is a set of SAConceptVO objects (indexed concepts) connected to each other through semantic constructors illustrated in the left-hand side portion of Figure B-1. The operation calls SimilaritySearcher to search for indexing concepts which establish the semantic association list. The input ApplicationConceptVO is used for validating the resulting concepts. The definitions and the relations of the classes in this similarity measurement module can be found in the UML diagram illustrated in Figure B-2.

4.3 Taxonomy-based Information Specification Framework

Section 4.2 introduced an information classification framework that carries out three of the four main applications of the proposed information model (i.e. requirement retrieval, analysis, and consolidation). This section describes an information specification framework that carries out the fourth application – transformation under an

application, specifically transforming software designs and test cases into Java code under the application – software development.

The background that motivated the design of this framework is as follows:

It is highly desirable to model software artifacts and their tests in the early stages of a development process since it allows designers to focus on higher level abstraction and thus produce more reliable final products that have better alignment with their requirements. Although there is no lack of such modeling tools, their actual implementation in many IT projects remains challenging. As part of the effort to provide a flexible and easier solution, this subsection introduces a novel taxonomy-based approach for specifying, relating, transforming, and testing the artifacts. A working example is used to demonstrate the feasibility of this systematic approach to model requirements, class designs, and test cases using a set of facilities based on the principle of classification and semantic referencing which is the hypothesis of this dissertation. Testing is then performed based on the resulting model. This new modeling approach is both intuitive and practical, offering several unique advantages such as the direct integration of different aspect artifacts and the enhancement of their traceability, reusability, and testability. This taxonomy-based modeling approach is expected to have the potential for semantic inference and discovery, both of which facilitate semantic applications in software testing.

The following subsections introduce the proposed approach.

4.3.1 Building the Meta Model Construct

To realize the ideal of classification and referencing, it is first necessary to construct a meta model to describe the artifacts of interest. As shown in the bottom sub-tree of Figure 4-5, the meta model is made up of three components:

- a set of meta taxonomies that are used as either templates or classifications of the artifacts (i.e. specification and classification),

- a set of key concepts for annotating the artifacts (These represent the semantics to be conveyed or referenced, i.e. semantic referencing.),

- a set of semantic interpreters for realizing the semantics (This may be, for example, transforming one artifact to another or applying the artifacts to a particular application such as software testing.)

The following section describes the essential elements involved in building such a meta-model. These are termed meta model constructs as they are at the meta-meta-model abstraction level.

4.3.1.1 Meta model construct

The meta-model construct, shown in the top sub-tree of Figure 4-5, consists of three components: taxonomy, notations, and semantic link.

- Taxonomy is the basic element for constructing objects in all abstraction levels of the modeling. It is similar to class in the object-oriented modeling paradigm.

- Notations are the symbols used for annotating the nodes of the taxonomy.

- A Semantic Link is a symbol that is used for annotating the semantic referencing.

The following subsection discusses these three components in more detail.

4.3.1.2 Meta taxonomy

There are two types of taxonomies: classification and template.

- Classification type: this is used for classifying referenced domain concepts in order to refine their semantics. For example, in the domain of software development, class is a concept and it can be classified under the categories of Java type -> architectural layer -> service in a hierarchical taxonomy. When the concept class is referenced by an object, the object is then annotated as a class under the service layer.

- Template type: this is used for specifying domain objects. An object annotated with this kind of taxonomy must conform to the template.

4.3.1.3 Taxonomy nodes and notations

Taxonomy can be constructed with five types of nodes: Key Concept (labeled as either <> or <>? depending on whether or not it has parameters), Semantics (labeled as {}), Abstraction (labeled as ()), Composition (labeled as []), and Entity (with no labeling). A Key Concept node represents the semantics of a system-defined domain concept and is similar to keywords in a programming language. A Semantics node represents the semantics of a domain concept and is arbitrarily defined by application designers. An Abstraction node signifies that its child nodes are its instances (is-a relation), while a Composition node indicates that its child nodes are its components (part-of relation). An Entity node represents an instance of domain objects, for example a segment of text that describes a business requirement.

4.3.1.4 Semantic link

A semantic link consists of three parts: source node, target node, and relation. The target node is either a Semantic Node or a Key Concept Node and the associated relation is either function application or semantic annotation, both of which are determined by the target node. Function application takes the source node and the specified inputs and returns the specified output node. For example, the first arrow shown in Figure 4-5 is a semantic link of type function application. The <modeling approach> is the target node of type Key Concept. When the semantic link is evaluated, <modeling approach> takes domain as input and then outputs a [meta model]. The second arrow in Figure 4-5 is a semantic link of type semantic annotation. It is used for semantic annotation; the semantic of the target node annotates the semantic of the source node. In this example, the semantics of [meta model] node is annotated as the output of the <modeling approach> function.

With these three building blocks, one can build a meta-model using the following steps:

- Analyze a domain process, identify artifacts, and design meta taxonomies
- Identify key concepts
- Implement the key concepts

The following section shows how the new approach is applied to the software development domain.

4.3.2 Building the Meta Model

4.3.2.1 Identifying artifacts and designing meta taxonomies

This step analyzes a software development process, identifies the artifacts to be annotated, and designs their meta templates. The left hand side of Figure 4-6 shows a typical software development process which consists of four basic stages: eliciting the raw requirements, requirement refining, designing, and use case composition. (The right hand side of Figure 4-6 shows the regression testing framework which will be discussed in Section 5.3.3). Three important artifacts are identified in the process: requirement, design, and use case. The next step is to use the meta-modeling construct introduced in Section .4.3.1 to design meta taxonomies for each artifact, which may be templates or forms for requirement specifications or use cases in the form of taxonomies. These will be used for specifying requirements or use cases.

4.3.2.2 Identifying key concepts

The next step is to identify the key concepts that are essential for characterizing the software development process and artifacts. For example, the two important concepts emphasized in the software production line methodology [71], variability and commonality, should be defined as key concepts as <f:Variable and Values> and <Common>. Note that the f: in <f:Variable and Values> indicates that when the node is

referenced, the semantic link is a type of function application and it expects inputs. In this case, the input will be the values that can be assigned to the variable.

Table 4-2 lists some of the key concepts used in the demonstration example that will be introduced in the following section. As mentioned earlier, complex key concepts can be classified into specific categories to enrich their semantics. The top sub-tree in Figure 4-7 is an example that represents categories of keywords used for annotating Java type and architectural layers. When <class> is referenced by an object, the object's architectural layer and type are determined. The second sub-tree of the figure is an example of the other type of meta taxonomy – template. This represents the syntax for Java class and interface inheritance (extends and implements) through either the taxonomy structure itself or explicit semantic links. When a particular Java class design references the Key Concept <Java Structure>, it means that the design taxonomy conforms to the syntax and can be transformed into code by the semantic interpreter of <Java Structure>.

4.3.2.3 Implementing key concepts

For any given domain, a taxonomy-based language is needed to describe the domain knowledge, along with an interpreter to perform the semantics denoted in an application specified in the language. So far, the first step has identified several meta taxonomies to specify or to organize artifacts that represent the domain knowledge while the second step identified the key concepts that characterize a domain application. The meta taxonomy and key concepts are similar to the grammar and the keywords in a programming language, respectively. The next step is therefore to implement the semantics of the key concepts, which is analogous to the interpreter in a programming language. For example, the implementation of the key concept <Java

Structure>'s semantic is a Java code generator that takes a design taxonomy as input and outputs Java code.

4.3.3 Building the Application Model for Software Engineering

This section uses a simple example to explain how the language is used. Consider a simple requirement in data integration. This is described in R1: "imports various data sources to common schema", as illustrated in Figure 4-8. The following subsections show how this requirement is processed using the new approach proposed here. Before starting the process, however, first examine the figures to be used and clarify their types. Figures 4-8 and 4-10 are instances of a requirement meta taxonomy, while Figure 4-9 is an instance of a use case meta taxonomy and Figure 4-11 is an instance of a design meta taxonomy [5].

4.3.3.1 Requirements and use cases

Requirement Annotation

Two semantic links are shown in Figure 4-8. The link to <issue> indicates the creation of a tracking issue with versions 1.0 and assignee Mary. This link binds the requirement to a particular version and the information can be traced by the system for regression testing, which will be introduced later. The other link is devoted to the testing system. Requirements-based testing (RBT) [72] is an important technique that identifies a necessary and sufficient set of test cases from the requirements to ensure that the design and code fully meet those requirements. Identifying a variable that contains possible values for testing is therefore a vital step in the process [73]. In the example shown in Figure 4-9, the segment "various data sources" in the text of R1 is

[5] Note that the graphical links and the referenced key concept nodes can be hidden such that only the design portion is visible.

recognized as a variable. Therefore, R1 is further refined to include the values "schema 1" and "schema 2". A new associated node RA1 is created to accommodate this variable and its values and is then annotated with the key concept <Variable and Values>.

Associating Requirement to Use Case

Driven by the annotated requirement (RA1), a use case can be created as depicted in Figure 4-9. Individual test cases and scenarios can then be derived using the values of the variable specified in RA1.1.

Requirement Refinement

Figure 4-10 shows how the requirement R1 is refined and annotated. As the figure illustrates, schema 1 and schema 2 are annotated with a semantic node {input data} such that their example data files are associated. This information is useful in the design, implementation, and testing phases. Due to the annotation, it is now possible to implement an automatic approach for tracking the information across different development stages. The second annotation specifies the rule that transform the attribute2.1 and attribute2.2 of schema2 into the c_attribute1 of the common schema. The expression {a+b-c} is an application specific rule and its content, i.e. the formula, is not specified. Due to its reference to **<**Rule>, a Key Concept with semantic “Generate Java Comment for the rule“ , the text of the rule will appear as a Java comment in the associated code when the referenced design taxonomy, introduced in the following section, is evaluated by the Java code generator. The comments can then be interpreted and implemented by coders using a general programming language. Note that this example combines model driven and traditional development methods. Finally

the whole refined requirement taxonomy is annotated with <design> such that the associated design template (meta taxonomy) can be provided when the taxonomy is evaluated.

4.3.3.2 Associating requirements to design

Figure 4-11 shows a class design for requirement R1. Three Java classes (SchemaClass1, SchemaClass2, and CommonSchema) are designed according to schema1, schema2, and c_schmea, specified in R1. The design taxonomy is an instance of <Java Structure> meta taxonomy. Based on the structure and the {extends} annotation in the taxonomy, both SchemaClass1 and SchemaClass2 extend AbstractSourceSchema, which in turn extends AbstractTestSupport. The class SchemaService converts source schemas into the common schema through its method convertToCommon. This method is so important that it is annotated with a semantic node {regression test}. The following subsection describes how this design can be interpreted for regression testing.

4.3.3.3 Semantic interpretation and inference

Modeling the gap between artifacts in different layers of abstractions is an active research area. For example, Ubayashi and coworkers have proposed a solution for bridging architectural design and code [74]. Adopting a different perspective, this research focuses on how the semantics behind the gap can be interpreted in the context of the overall taxonomy system. In order to provide some idea of the potential utility of this new approach, this subsection presents an example of semantic interpretation that employs context information for regression testing. When evaluating the design taxonomy shown in Figure 4-11, applying the semantic annotation {regression test} on the method convertToCommon can be implemented as an aspect

and a join point on the method. (An aspect is a standalone code that addresses a system crosscutting concern and can be woven into the join points of a base program [75]). The logic of the aspect is to call the sendValuesToRegressionServer method of AbstractTestSupport through the inheritance of the input parameter AbstractSourceSchema via Java reflection. The implementation of sendValuesToRegressionServer obtains the relevant version information from the context provided by the <issue> annotation on the requirement R1 shown in Figure 4-8. It then serializes the CommonSchema object's data, which is the output of the method that is interwoven with the aspect, tags the data with version information via Java Reflection and sends them to a regression test server. At run time, when the Use case specified in Figure 4-9 is performed as shown in the upper portion of the left hand side of Figure 4-6 through tools like Selenium (a portable software testing framework for web applications [76]), the aspect is triggered and the serialized object data are sent to the server for regression analysis. This example shows a possible implementation that utilizes information from artifacts across different stages via semantic links. The best way to test this modeling approach will be discussed in detail in Chapter 5

4.3.4 Benefits

Unlike current modeling languages, such as UML or DSL, this new approach is informal and less constrained, which should make it easier to use and more flexible to implement. It should therefore offer a practical way for developers who are less familiar with formal modeling to enhance their systems. The proposed approach offers the following benefits:

Semantic Inference and Discovery: Although taxonomy or the tree layout has limited ability in specifying knowledge, it has some unique characteristics. For example, it

naturally incorporates inheritance and composition relations and provides good context information for anchoring the semantics of the objects being classified. A previous study proposed a new approach for measuring the semantic similarity between two objects based on taxonomies and presented a potential application for requirement consolidation [17]. The use of context information and semantic links offers further potential for semantic inference and discovery.

Efficient Editing: The features of tree topology generally favor graphical editing. For example, sub-tree operations facilitate bulk updates, cloning, and swapping. Also, a sub-tree can be collapsed so that the scope of the graphical presentation is flexible and scalable. As mentioned in the Introduction, many efficient tools already exist for creating, editing, linking, and navigation trees, all of which play a role in the success of the proposed approach.

Traceability: This benefit is directly captured by the semantic links, as shown in the example in Section 4.3.3.

Reusability: The Key Concepts and the Semantic Nodes themselves are reused when they are referenced. Also, the Key Concept <common> is specifically designed for this purpose.

Testability: As demonstrated in the example in Section 4.3.3.3, objects can be tested through annotations driven by both the requirements and use cases.

Table 4-1. Comparison among information systems

	Classification	Specification	Query Language	Applications
Google	none/less structured	web pages	specific query	identifying web pages
Relational	fully structured, constrained by entity relations	tables of normalized data	SQL (Structured Query Language)	data management
This Research	semi-structured; faceted taxonomy; indexed by concepts organized in is-a and composition relations	templates (parse-able forms) and, relations (ad hoc and system defined), and transformers	CIQL(Concept Indexed Structured Query Language)	knowledge retrieval, merging, expansion, and transformation
Aspect-oriented Requirement Engineering (AORE)	faceted taxonomy; pointcuts and weaving	None	none/regular expression	methodology for managing system concerns in requirement stage
Unified Modeling Language (UML)	None	various well defined structures/diagrams	None	specifying business rules, architectural design, system …, etc.

Description:	
Actor(s):	
Pre-Conditions:	
Post-Conditions:	
Normal Course of Events:	

Use Case Appendix

Appendix Name	Note	Reference

Figure 4-1. Example of specification: use case form

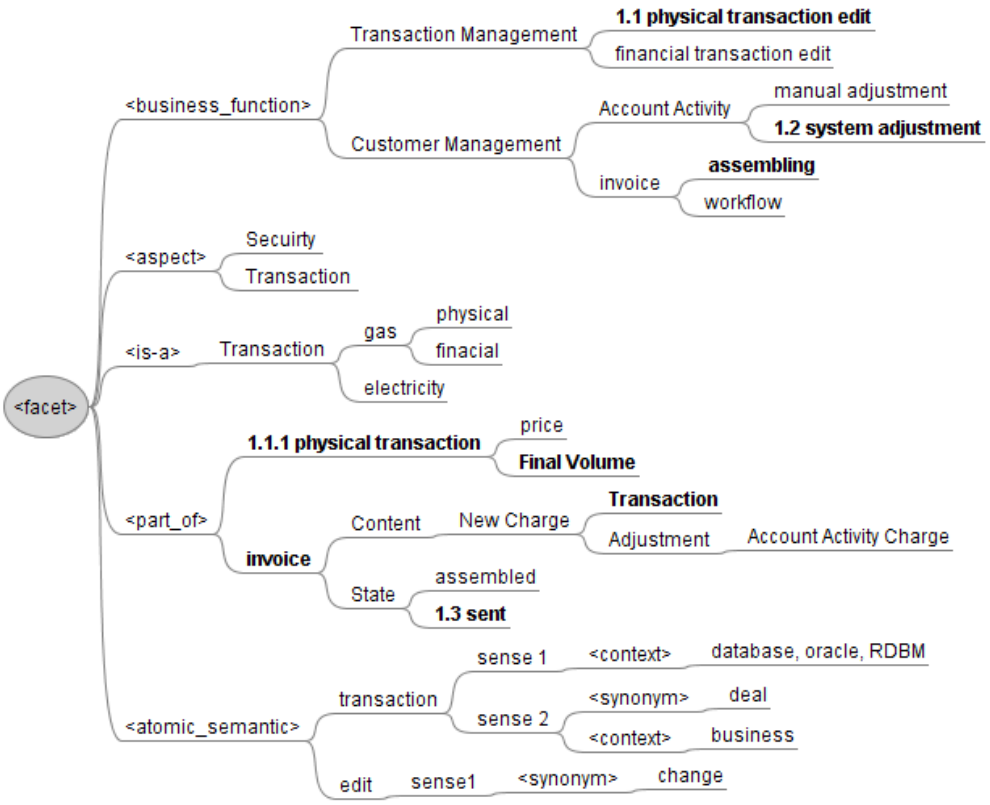


Figure 4-2. Example of proposed taxonomies for classifying software requirements

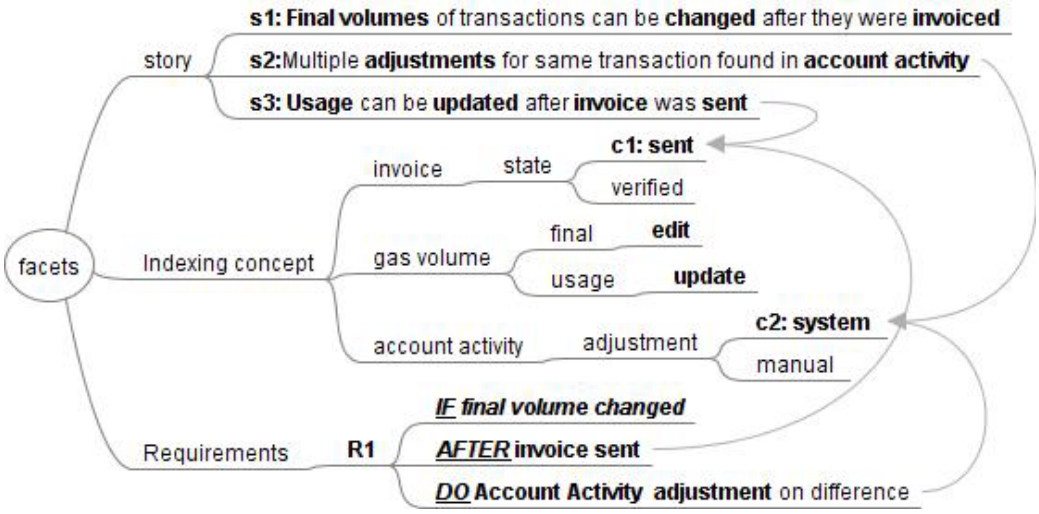


Figure 4-3. Example of faceted taxonomies with indexing concepts

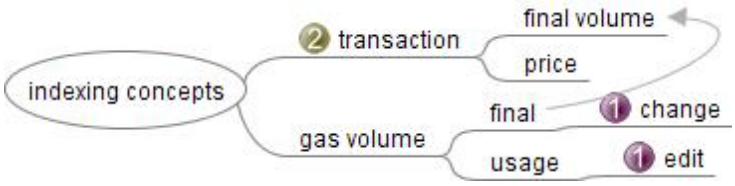


Figure 4-4. Example of indexing concept taxonomy

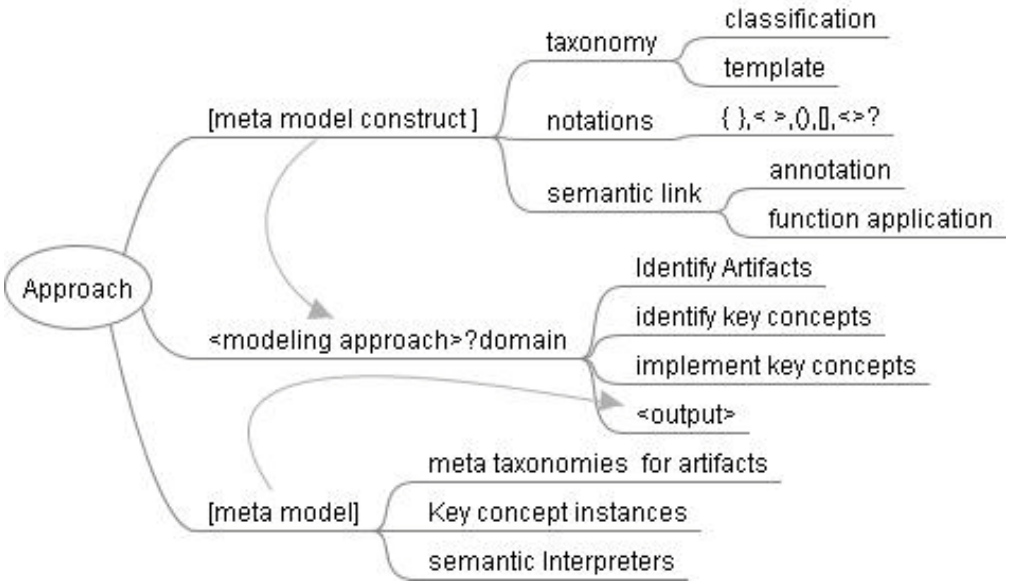


Figure 4-5. The taxonomy-based modeling approach

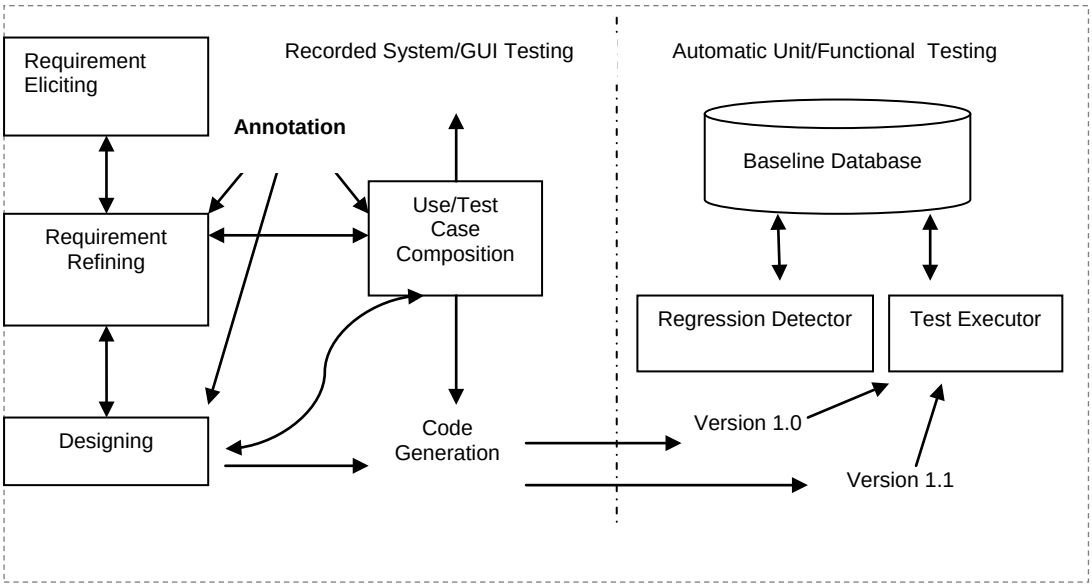


Figure 4-6. Annotation, software development process, and testing

Table 4-2. Examples of key concepts

Keywords	Reference Semantics
<Common>	Reuse
<f:Variable and Values>?List	Alternative test Scenarios
<Rule>	Generate Java Comment for the rule
<Design>	Java Class Design
<Test>	Regression Test
<f:Issue>? Version, Assignee	Create Tracking Issue

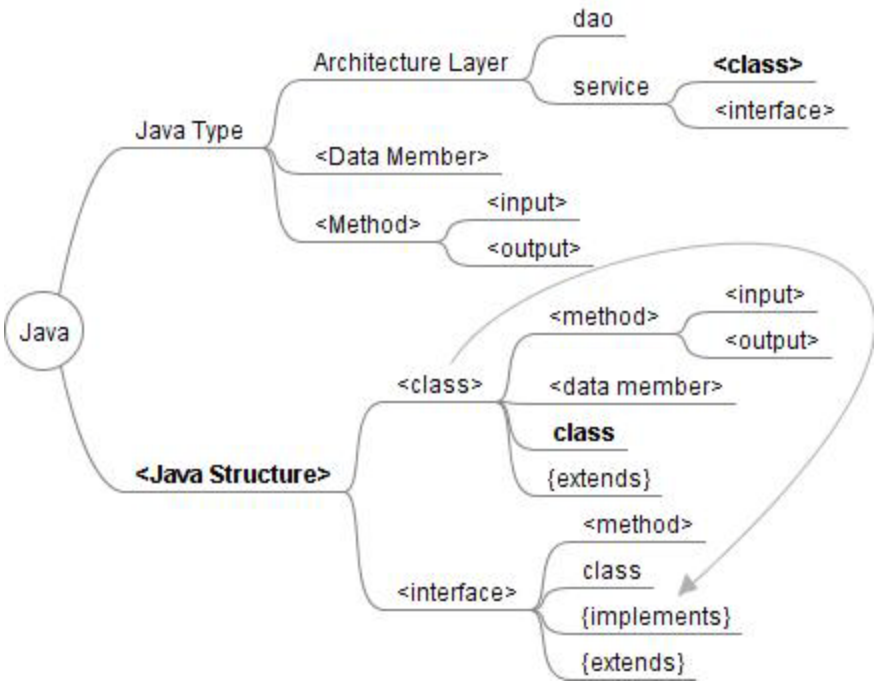


Figure 4-7. Key concepts in Java code generation

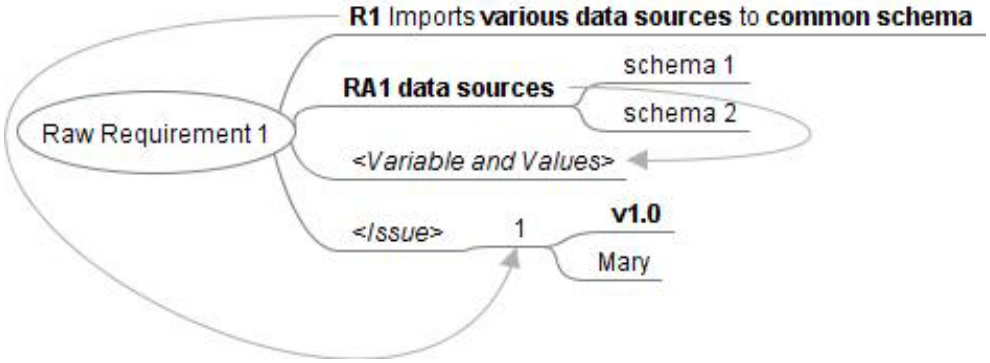


Figure 4.8. An example of annotated requirements

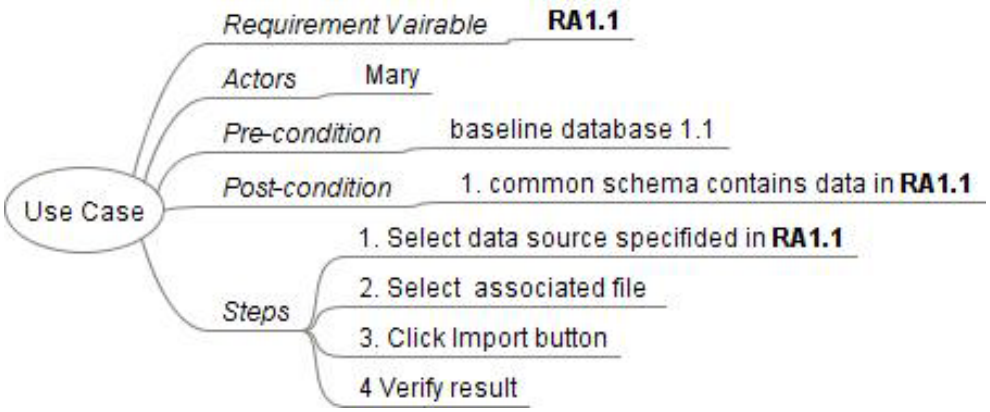


Figure 4-9. An example of use case

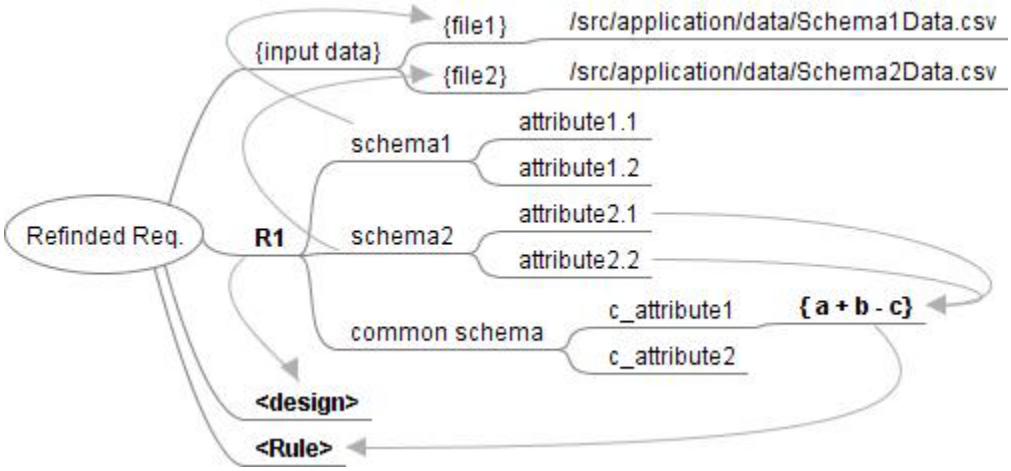


Figure 4-10. An example of refined requirements

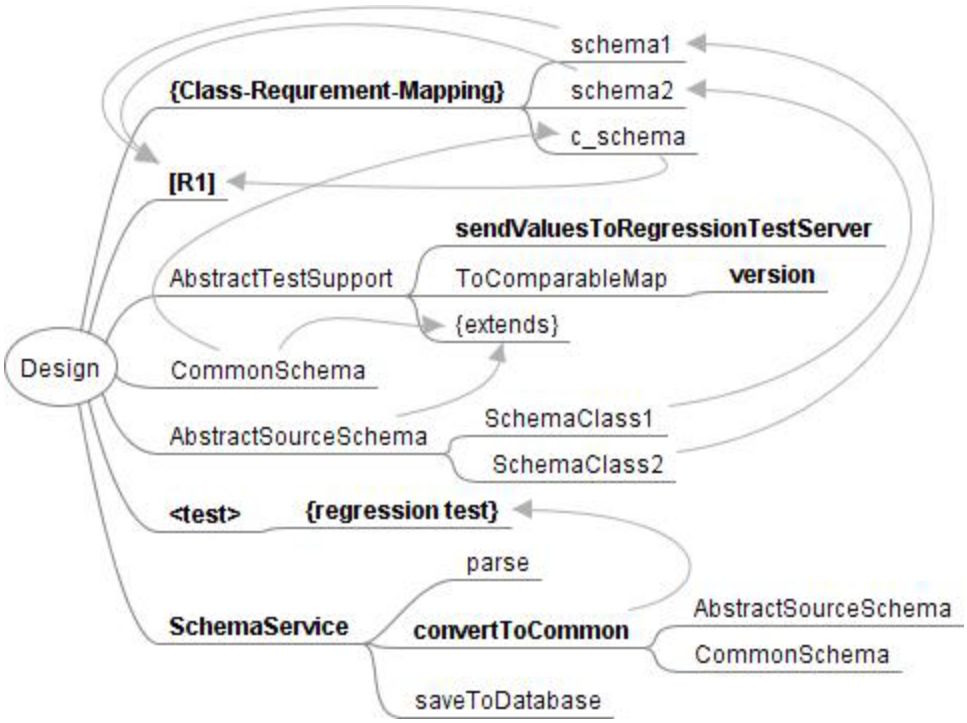


Figure 4-11. An example of class design

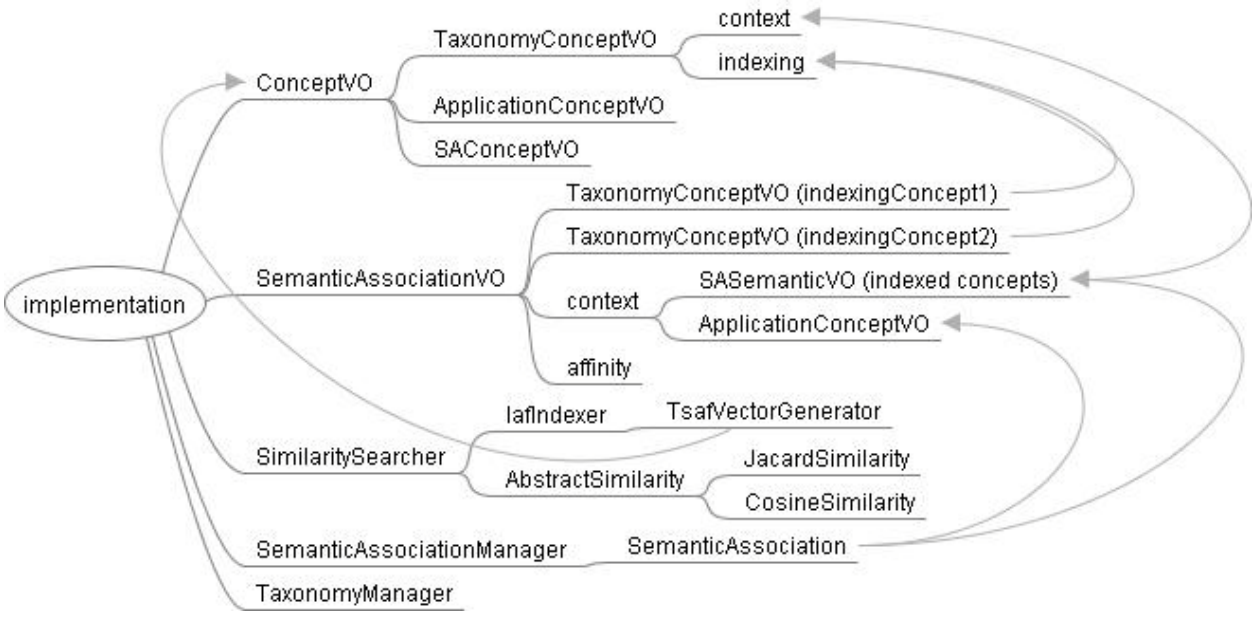


Figure 4-12. An overview of TSAF implementation

CHAPTER 5
ARCHITECTURE VALIDATION: PROTOTYPE AND SIMULATION

5.1 Introduction

This chapter presents a prototype implementation for the Taxonomy-based Information Specification Framework described in Section 4.3. The prototype implements the meta model construct and the meta model for Java software development. The data integration application illustrated in Section 4.3 is reused and extended as an example and shows how the system can model the application using the meta model. The model is then transformed into Java code and tested. In order to demonstrate how the model evolves during the lifecycle of a typical software development process, some additional changes are made to the existing model to represent the next version of the application. How the system adapts to the new version is examined and testing performed to detect any software regression defects.

5.2 Architecture and Implementation

As described in Section 4.3, the components of the proposed framework are in three levels of abstraction: meta model construct, meta model, and application model. The following sections describe the implementation of each in turn.

5.2.1 Meta Model Construct

The meta model construct is primarily implemented in two classes, TaxonomyNode and Transformer, as shown in the two sub-trees of Figure.5-1. TaxonomyNode is made up of two components: the node semantics and structure information. As described in section 4.3.1 3, the node semantics contain the key concept (system predefined semantics), semantics (application defined semantics), abstraction (is-a relation between parent and children nodes), composition (part-of

relation), and entity (undefined semantic). The structure information is specified by the link branch in the upper sub-tree of Figure 5-1. The parentId of the link class is a pointer to its parent node. The targetId is a pointer to the node of other taxonomy representing a semantic referencing. There could be multiple semantic references for a node. It is specified by spawning a child node <reference>, which is a key concept representing a semantic reference of its parent node (not shown in the figure).

The second component of the meta construct, Transformer, is shown in the lower subtree of Figure 5-1. The Transformer mainly contains two methods: parse and evaluate. The parse method takes an application model as input and outputs a taxonomy constructed by TaxonomyNod. The idToNodeIndexMap is the index of the taxonomy. The map takes a node id as key and returns the pointer of the TaxonomyNode object represented by the id. The parsing is analogous to the first step of a typical program interpreter (i.e. translating source code into intermediate code). The second method, evaluate, is an abstract method which is to be implemented by classes which extent Transformer. The implemented transformers are the semantic interpreters of the parsed data structure, which are specific to particular applications. The following subsection will introduce two transformers, JavaCodeTransformer and TestCaseTransformer, which are specific to Java program development applications.

The two components (classes) are used in building a meta model for a particular application domain. For example, the following section describes a meta model that is built by using the two classes and it is specific to Java programming.

5.2.2 Meta Model

The upper sub-tree of Figure 5-2 shows a kind of Transformer which transforms a Java class design into Java code. The middle sub-tree shows another kind of

Transformer which transforms a test case design into Java Test code. The lower sub-tree shows some examples of key concepts which are recognized by the evaluators of the transformers for semantic interpretations (i.e. code translations). For example, consider javaCodeTransformer. When <class> is encountered while it is evaluating a particular application class design, processClass is called to generate a Java class for the node which is annotated with the <class>. The following shows the code snippets of this example:

```
public void evaluate() throws IOException {
  …
  while (!stack.isEmpty()) {
    String text = taxonomyNode[idToNodeIndexMap
    .get(taxonomyNode[i].getTargetId())].getText();
    if (text.equals(KeyConcept.CLASS))
      processClass(treeRoot, suffix);}}

public void processClass(Element treeRoot, String suffix) {
  …
  out.write("public class " + classname + suffix + "{\n");
  while (it.hasNext()) {
    if (taxonomyNode[idToNodeIndexMap.get(taxonomyNode[i]
        .getTargetId())].getText().equals(KeyConcept.METHOD))
      processMethod(current, out);
    else if (taxonomyNode[idToNodeIndexMap.get(taxonomyNode[i]
             .getTargetId())].getText().equals(KeyConcept.CLASS)) {
           String parentID = taxonomyNode[i].getParentId();
           if (idToNodeIndexMap.containsKey(parentID))
             processClass(current, " extends "+ taxonomyNode[idToNodeIndexMap
                                                .get(parentID)].getText());
      … }}}
```

The definitions and the relations of the major classes described in Sections 5.2.1 and 5.2.2 can be found in their UML diagram, illustrated in Figure B-3 (Appendix B).

5.2.3 Application Model

The resulting meta model generated in Section 5.2.2 can then be used for a particular application model like the data integration example discussed in Section 4.3.3. Figure 5-3 shows the relation between the model and the codebase of the application. The model basically represents the abstraction of the application.

However, it is not realistic to be able to immediately model all the logic contained in an application; some generated code can be directly used in its codebase but other code must be adapted before it can be used. Examples of generated code that can be directly used include interface and pure data objects that are used as parameters of functions. The use of pure data objects in regression testing will be discussed in Section 5.3.3.1. Examples of generated code that cannot be directly used include the portions of the application model that specify the implementation logic of business services. These usually need modification before the whole codebase of the application can compile and run [6]. The following section elaborates how the model is transformed to Java Code using the data integration sample application.

5.3 Simulation

5.3.1 Data Integration Application Version 1.0

5.3.1.1 Requirements and class design

Here, the requirement illustrated in Figure 4-9 and the class design in Figure 4-11 are reused to serve as Version 1.0 of the application. Figure 5-4 illustrates a class design for the sample application, which is extended from Figure 4-11. The extension portion consists of the additions of several arrows[7], as shown in the taxonomy nodes in Figure 5-4, and some implementation logic (of the parsing method), as shown in the first branch of the upper sub-tree of Figure 5-4. {csvParser} shown in the branch is an application defined concept whose semantic is to parse a csv file and return a list of

[6]. The generated code/interfaces usually cause the application codebase compilation errors revealed in most Integrated Development Environment (IDE) tools like Eclipse. This prototype takes advantage of this feature to make code adaption easier and more practical.

[7] The arrows shown in the nodes represent hyperlinks (semantic links) to other taxonomy nodes or Internet/intranet resources. If a designer clicks on an arrow, the focus will change to the specified target node or the target resource will be opened, depending on the type of target.

object of type AbstractSourceSchema. For example, the Schema1Data shown in the branch is the path (url) to a file named file1 (not shown in the figure). The parser opens the file, parses the rows in the file into array and assigns the parts to the corresponding attributes of the target source schema based on the specified semantic (in this example, the semantic is “equal”). The following shows the code snippet of this example that is generated from the model.

```
public List<AbstractSourceSchema> parse(String filePath) throws Exception {
  BufferedReader br = new BufferedReader(new FileReader(filePath));
  CSVReader csvReader = new CSVReader(br);
  String[] parts = csvReader.readNext();
  List<AbstractSourceSchema> srcSchemaDataList = new
  ArrayList<AbstractSourceSchema>();
  parts = csvReader.readNext();
  while (parts != null) {
    if (filePath.indexOf("Schema1Data") != -1) {
      SchemaClass1 srcSchemaData = new SchemaClass1();
                        srcSchemaData.setS1Att1(parts[0]);
                        srcSchemaData.setS1Att2(parts[1]);
                        srcSchemaDataList.add(srcSchemaData);
                  } else if …
      parts = csvReader.readNext(); }
            return srcSchemaDataList;}
```

5.3.1.2 Test case

Figure 5-5 shows a test suite named importSchemaTest as shown in the upper sub-tree of Figure 5-5. The test suite contains two portions. The first portion is operations as shown in the middle sub-tree of Figure 5-5. The operations have nodes, {parse}, {convertCommon}, {saveToDatabase}, and {getCommonSchema} with semantics referencing the methods of SchemaService, which is shown in the top sub-tree of Figure 5-4. In the operations sub-tree (Figure 5-5), the hyperlinks among <input> and <output> specify parameter passing among the methods. The sub-tree is transformed into the following code snippet:

```
SchemaService schemaSerive = ManagerHandlerFactory.getInstance()
                            .getSchemaServiceHandler();
```

```
void importSchema(String filePath) throws Exception {
  List<AbstractSourceSchema> srcSchema = schemaSerive.parse(filePath);
  for (AbstractSourceSchema src : srcSchema) {
    CommonSchemaVO targetSchema = schemaSerive.convertToCommon(src);
    schemaSerive.saveToDatabase(targetSchema);}}

List<CommonSchemaVO> getCommonSchema() throws Exception {
      return schemaSerive.getCommonSchema();}
```

The second portion is test instances (as shown in the bottom sub-tree of Figure 5-5). The test instances are based on the requirement segment that is annotated with <VariableAndValues> which is RA1.1, in this case, as shown in Figure 4-8. It is to test the importing of Schema1 and Schema2. As indicated in the bottom sub-tree of Figure 5-5, the tests are designed to create an object of ImportSchemaTest named testObject1 and then perform importSchema ({file1}, importSchema{file2}, and getCommonSchema respectively. The sub-tree is transformed into the following code:

```
ImportSchemaTest testObject1 = new ImportSchemaTest();
String currentVersion = "1.0"; //obtained from requirement
dbUtil.saveVersions(baseVersion, currentVersion);
String filePath1 = "/src/application/data/Schema1Data.csv";
testObject1.importSchema(filePath1);
String filePath2 = "/src/application/data/Schema2Data.csv";
testObject1.importSchema(filePath2);
List<CommonSchemaVO> commonSchema = testObject1.getCommonSchema();
```

The value of the variable currentVersion in the line 2 is obtained from the <version> annotation on the requirement shown in Figure 4-8. The version information is used for regression testing, which will be introduced in the following section.

5.3.2 Data Integration Application Version 1.1

5.3.2.1 Model change

Figure 5-6 shows some changes (bold portions) to the model (Version 1.0) exemplified in Section 5.3.1 to represent a new version (Version 1.1). The changes are the additions of a new source schema and an annotation {regressionTest} on convertToCommon method.

5.3.2.2 Code generation and adaption

As described in Section 5.2.3, some generated code needs to be modified before it can be used in the application. For example, in the middle sub-tree of Figure 5-6, a test instance (the bold node) is added. The following code is generated as a result of the addition.

```
/* GENERATED FROM MODEL
   @importSchema, {file3}
   //the following code is new. please move it to real implementation
   String filePath3 = "/src/application/data/Schema3Data.csv";
   testObjct1.importSchema(filePath3); */
```

Some generated code can be directly used in the application. For example, the newly added source schema SchemaClass3 is a pure value object (DTO) which can be directly used. The generated code snippet is shown as follows:

```
package di.dto;
public class SchemaClass3 extends AbstractSourceSchema {
  String s3Att1;
  String s3Att2; … }
```

The following section demonstrates how annotation can be used to perform regression testing across the two versions.

5.3.3 Regression Testing Framework

Regression testing is used to detect software defects (bugs) that result from changes to an existing system. One of the common scenarios that introduce regression bugs is software refactoring, a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior [77]. In this scenario, since the external observable behavior is not changed, it is useful to design a regression testing framework capable of detecting behavior changes before and after the refactoring.

An appropriate framework was designed based on the hypothesis that, for a function, its returning objects should be the same across versions if

- The semantic of the function does not change over versions.
- The input and the context (i.e. such as system state, including database state) are the same.

With the aid of the framework, it is possible to show how people can use annotation (semantic referencing) defined in an application model to perform regression testing. The following subsection introduces the architecture of such a framework.

5.3.3.1 Overview of regression test framework

Figure 5-7 shows an overview of the regression framework. The upper sub-tree of the figure shows two layers of the demonstration application system introduced in the previous sections. The Service Layer is where the main application logic resides and contains classes whose methods carry out the business services. The Data Transfer Object (DTO) is a class of objects whose values provide the parameters for the methods. The Data Access Object (DAO) is another architectural layer that contains logic supporting data persistence, and Value Object (VO) represents the value objects used in the layer. VO objects are commonly mapped to database tables via Object-Relation mapping frameworks such as Hibernate [78].

The regression testing framework possesses two kinds of semantic annotations that will be transformed into Java code to perform the testing. One is the object annotation, as shown in the middle sub-tree of Figure 5-7. DTO and VO objects are usually the targets for this kind of annotation, which is used to annotate those objects with information such as the version number that can then be used when performing object comparison between two different versions. It is implemented by Java annotation

[79] in the model prototype. The other kind of annotation is model annotation, as shown in the bottom sub-tree of Figure 5-7. One example is the {regression test} introduced in Section 3.2. The semantic interpreter of this annotation can be implemented using AOP, as described in that section. For simplicity, in this demonstration implementation, Java dynamic proxy is used. This is a feature of the Java language that allows the interception of method calls such that additional behavior between a class caller and its callee can be interposed [79]. The following subsections provide more details of the framework.

5.3.3.2 Implementation of regression test framework

The UML diagram of the major classes in the Service Layer is illustrated in Figure B-4 (Appendix B). The class SchemaService is the interface generated from the bottom branch of Figure 4-11 and SchemaServiceImpl is its implementation. When a caller (such as test case driver) needs the services, it first obtains the handler of the service object through ManagerHandlerFactory. The factory returns the Java dynamic proxy of the SchemaServiceImpl which contains the handler. Figure B-5 illustrates the UML diagram of the major classes in the DAO and VO layers. The diagram shows the definitions of the classes and their relations. When a caller needs to retrieve or persists a VO object, such as CommonSchemaVO, it first gets a corresponding DAO object from DAOFactory. The DAO object in turn obtains a generic Hibernate [78] DAO object which is held in a HibernateContext object. The context object is held in a ThreadLocal object such that it is shared during the lifecycle of the computing thread. The use of the ThreadLocal object increases the performance of database accesses and enables transactional management across different data sources.

5.3.3.3 Regression test annotation on object

Based on the hypothesis described in Section 4.2, the objective is to compare the value objects that are used as output parameters of a method across two versions. They must be both serializable and version annotated. To implement these two features, all classes in Data Transfer Object layer (DTO) and in Value Object (VO) layer must extend a class called AbstractTestSupport, illustrated in Figure 5-5 in Section 3.2. As mentioned in the previous subsection, the objects in DTO are used as input and output parameters of the methods of business service objects. The objects in VO are used as input and output parameters of the methods of Data Access Objects' (DAO). Therefore, these objects must be compared to identify for behavior changes (regression bugs) between the two versions. AbstractTestSupport (that DTO and VO extend) has a method called toComparableMap which takes version and isShadow (type of boolean) as inputs. The method recursively serializes all data members of DTO and VO objects (depending on the boolean value isShadow) that have regression annotation and whose version equals or is less than the input version. The returning serialized objects are compared by a regression detector for change detection. The following shows an example of a VO object which has an annotated data member for regression testing. The attribute commonAtt1 will be serialized by toComparableMap if the base version of the regression testing is 1.0 or larger.

```
public class CommonSchemaVO extends AbstractTestSupport{
      @Regression (version=”1.0)
      String commonAtt1;}
```

5.3.3.4 Regression test annotation on model

It is now possible to compare objects across versions. The next question is to specify where the behavior changes are to be detected. As mentioned earlier, in the

new framework, this is simply done by the {regressionTest} annotation. In this demonstration implementation, the semantic of the annotation is interpreted by injecting a piece of code after the annotated method's invocation through Java dynamic proxy. The code obtains version information, called the toComparableMap of the returning object, and saves the resulting map (serialized data). The following shows the code snippet:

```
returnObject = method.invoke(service, args);
String methodName = method.getName();
// The following code is injected for regression test
if (methodName.indexOf("convertToCommon") != -1) {
  String[] versions = dbUtil.getBaseVersion();
  dbUtil.saveObject(key, convertToComparableMap(returnObject, versions[0],
  versions[1]);}
```

5.3.3.5 Detecting regression defect

As illustrated in the right hand side of Figure 4-6, the regression detecting demonstration involves the following steps:

Baseline Database Setup: this step controls the initial context state.

Run Tests Version 1.0: this step runs the tests generated from the <testCase> model discussed in Section 4.1.

Run Tests Version 1.1: this step runs the tests generated from a changed <testCase> model (not shown here due to space limitations). At this moment, the database (or context) contains the version information and the serialized returning objects of the annotated methods for both versions 1.0 and 1.1.

Run Regression Detector: this step compares the returning objects of the two versions and reports any differences. Any difference indicates a regression bug.

5.4 Conclusion

This prototype implementation showed the following:

- How the proposed framework is implemented.
- How an application (data integration) is modeled using the resulting meta model (class design and test case) which is based on the classification and referencing principles.
- How the class design and the test case (semantics) are transformed (interoperability) into Java testing code (application).
- How information artifacts are related (traceability), how existing code is used in the code generation (reusability), and how regression testing is performed (testability).

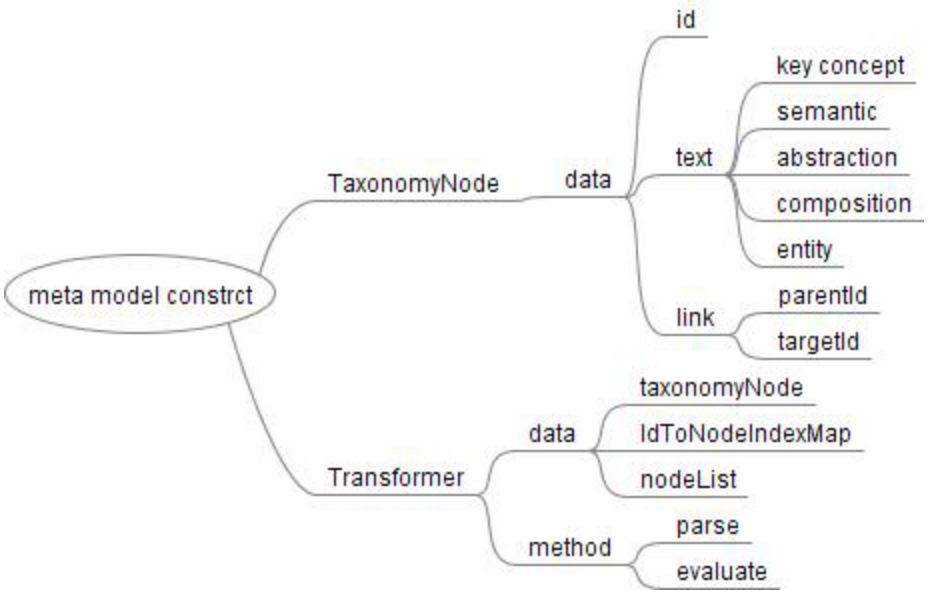


Figure 5-1. Class design of meta model construct

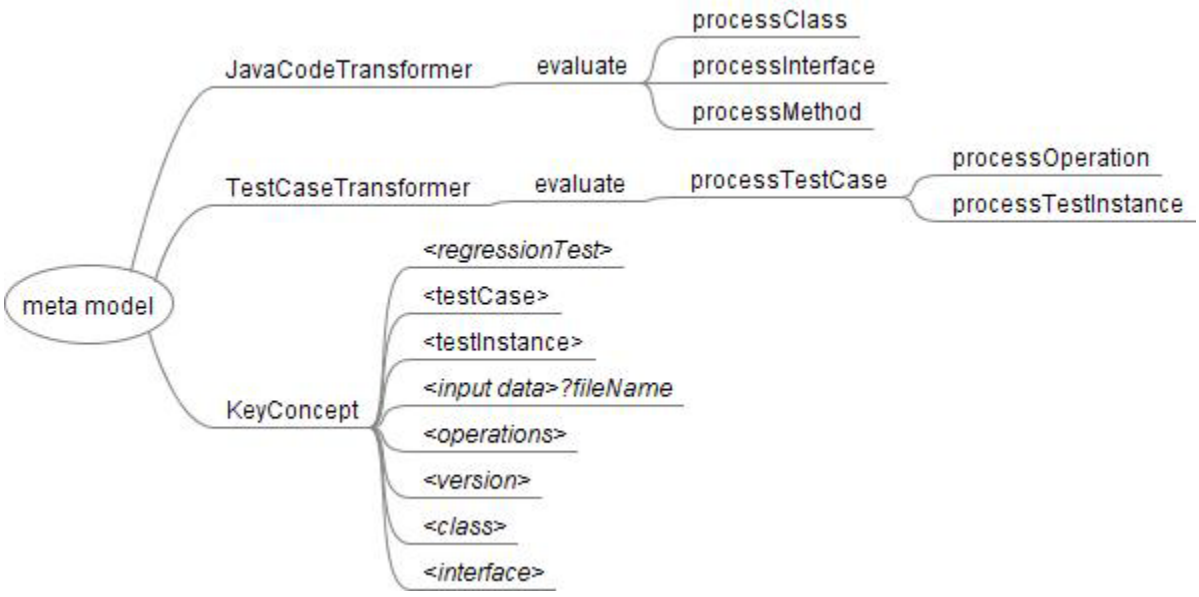


Figure 5-2. An example of a meta model for Java application development

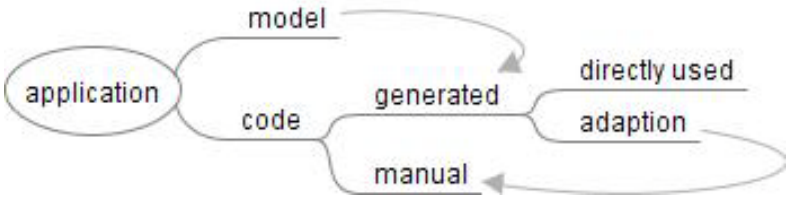


Figure 5-3. Application model and transformation

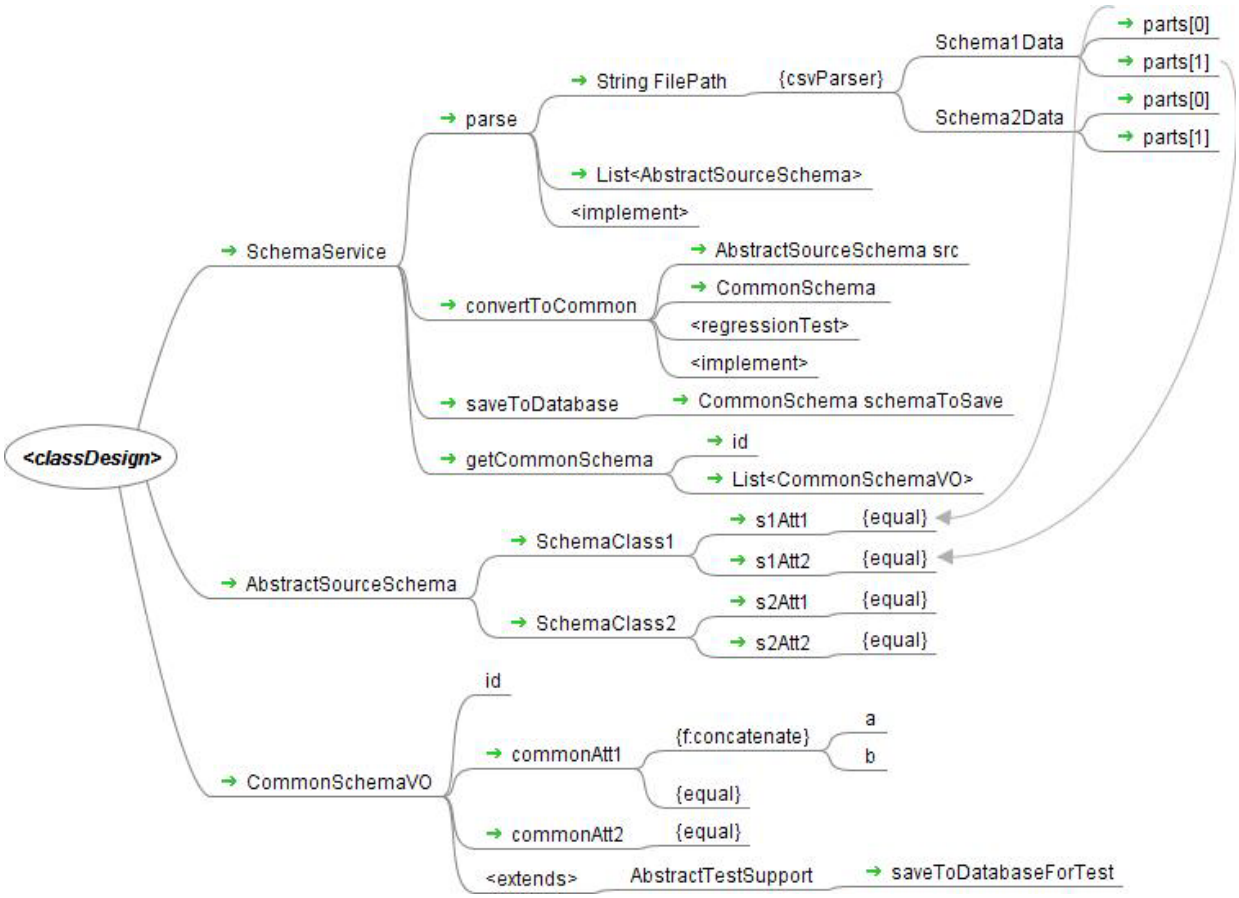


Figure 5-4. An example of class design: an extension of Figure 4-11

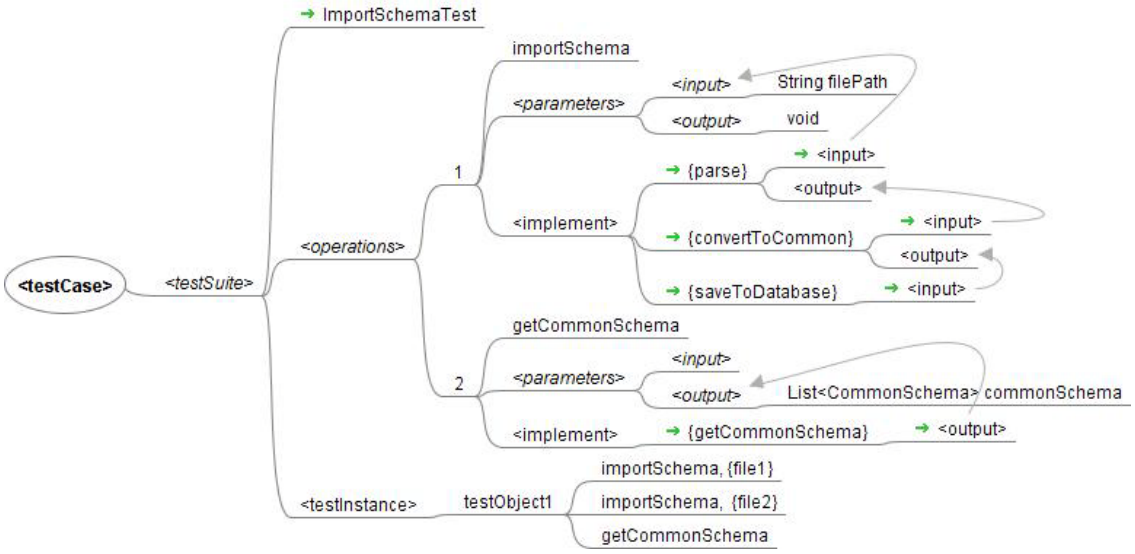


Figure 5-5. An example of a test case

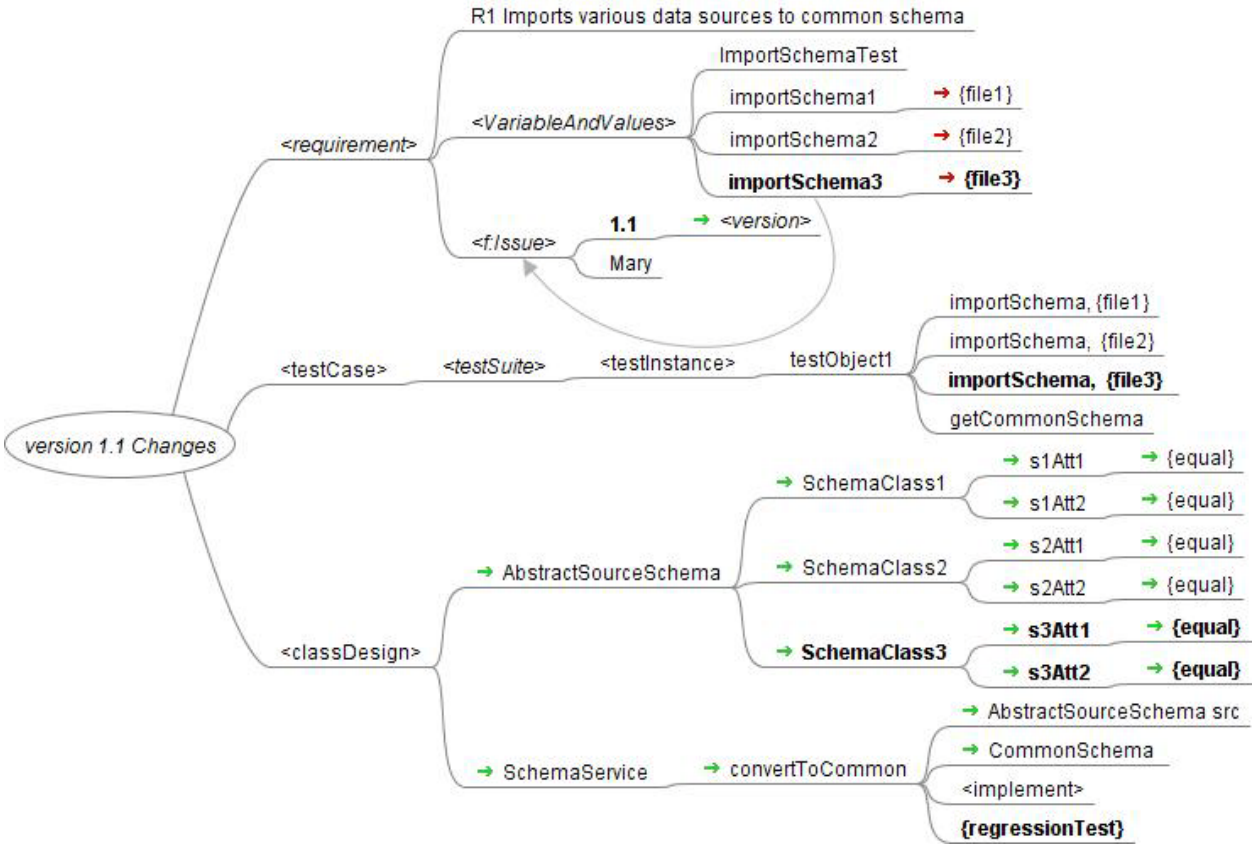


Figure 5-6. An example of an application model evolution

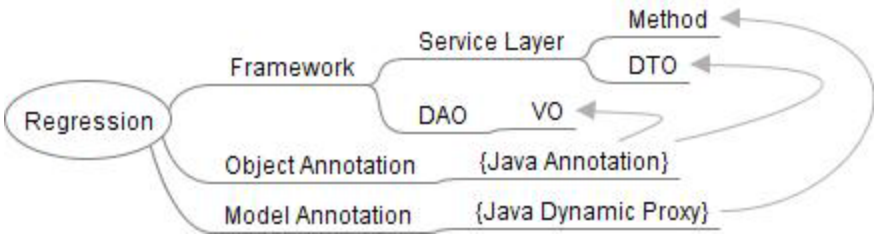


Figure 5-7. Overview of the regression test framework

CHAPTER 6
CONCLUSION AND FUTURE WORK

The research reported in this dissertation identified an open research area via a motivating example in the building construction domain. It formally defined the concept, namely semantic association, identified some research challenges and defined associated problems. New approaches were then proposed for implementing some of the identified concepts and problems. This foundation was then extended to cover generic information frameworks and applied to the software engineering domain. A prototype implementation was presented to validate one of the three proposed frameworks.

However, several problems remain unanswered and undefined in this new and only partially explored research territory. Some future research tasks have been identified and are itemized as follows:

- Building a firm theoretical foundation for semantic association: In [64], Kalfoglou and Schorlemmer try to provide a theoretical basis and formal methodologies to the ontology matching problems using information flow theory and channel theory described by Dretske[65] and Barwise and Seligman [58]. Semantic association derives its association function from the similarity function of ontology matching problem. It should be possible to extend the work reported in [64] to include the theoretical background of semantic association.
- Developing algorithms for answering the problems defined in Section 3.1.3.
- Validating the semantic similarity algorithms proposed in Section 4.2.3 with real data.
- Continuing the implementation of the complete taxonomy-based information model, including CISQL.
- Applying the prototype implementation described in Chapter 5 to larger real-world applications.

APPENDIX A
SEMANTIC ASSOCIATION-BASED INTEGRATION FRAMEWORK

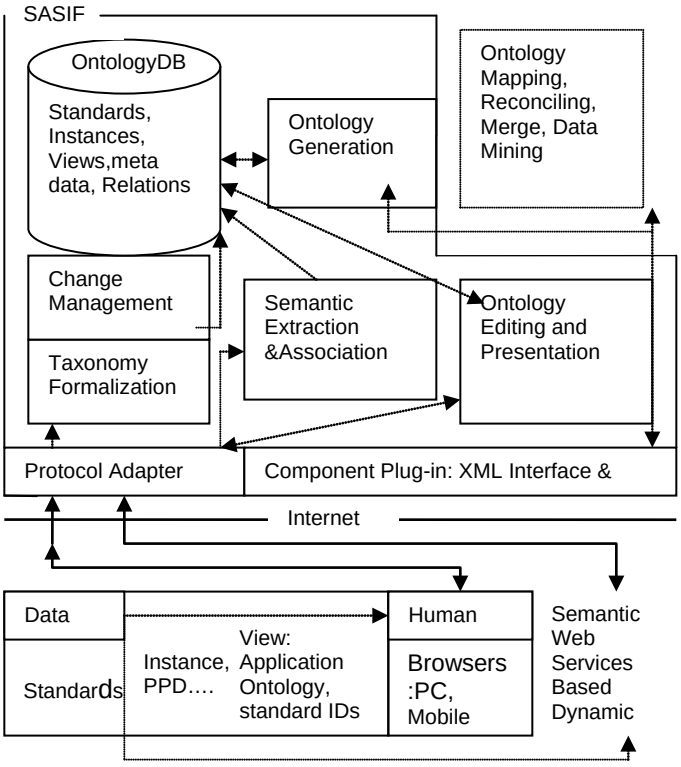


Figure A-1. Overview of Semantic Association-based Integration Framework

In the framework illustrated in Figure A-1, relations and relation statements of various versions of standards written in natural languages are developed and uploaded via web-based tools to the system by stakeholders in the application domain. The taxonomy formalization along with the change management modules process them through parsing, normalization, generalization, linguistic processing (such as inflection, derivation, compounds, and synonyms), and indexing for incremental update in the ontology database. For a particular application, the stakeholders upload instances of the source standard (e.g., PPDs), target standard, and its application ontology. After processing the free text of PPD instances through linguistic techniques such as tokenization, chunk parsing, and grammatical function recognition [45], the system

applies semantic extraction and ranking algorithms, and returns/deposits extracted metadata and semantic association to the ontology database and also to the users or clients, if applicable, for feedback.

The integration of competing and complementary standards is a critical step for enhancing interoperability among heterogeneous systems using the standards. The proposed semantic association is only one aspect of this effort. It should be supplemented with other technologies such as ontology mapping, reconciling, and merging to provide a practical and complete solution. The framework includes a plug-in mechanism via XML-based interfaces and API for external software component integration.

The formalized standards, their instances, users' application ontologies, and extracted metadata form a semantic rich ontology repository. Integrating the repository with other ontology techniques through the plug-in mechanism allows the effective construction of application domain ontology.

Web services enriched with the vision of the semantic web have emerged as a mainstream solution to system integration over the Internet. Following the same trend, the implementation of the proposed framework adopts the Web Ontology Language (OWL) with the intention of integrating building construction workflow systems via semantic web services.

APPENDIX B
SYSTEM IMPLMENTATION UML CLASS DIAGRAMS

B.1 Taxonomy-based Semantic Association Framework (TSAF) Implementation

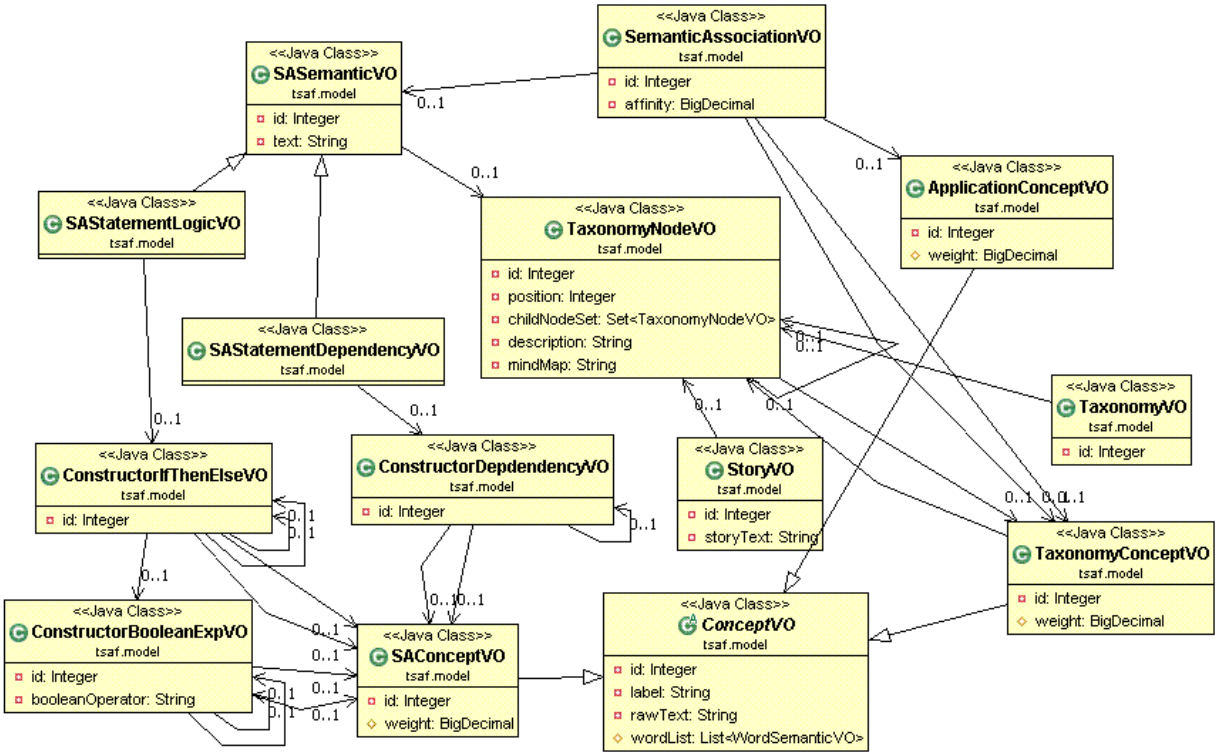


Figure B-1. TSAF data model

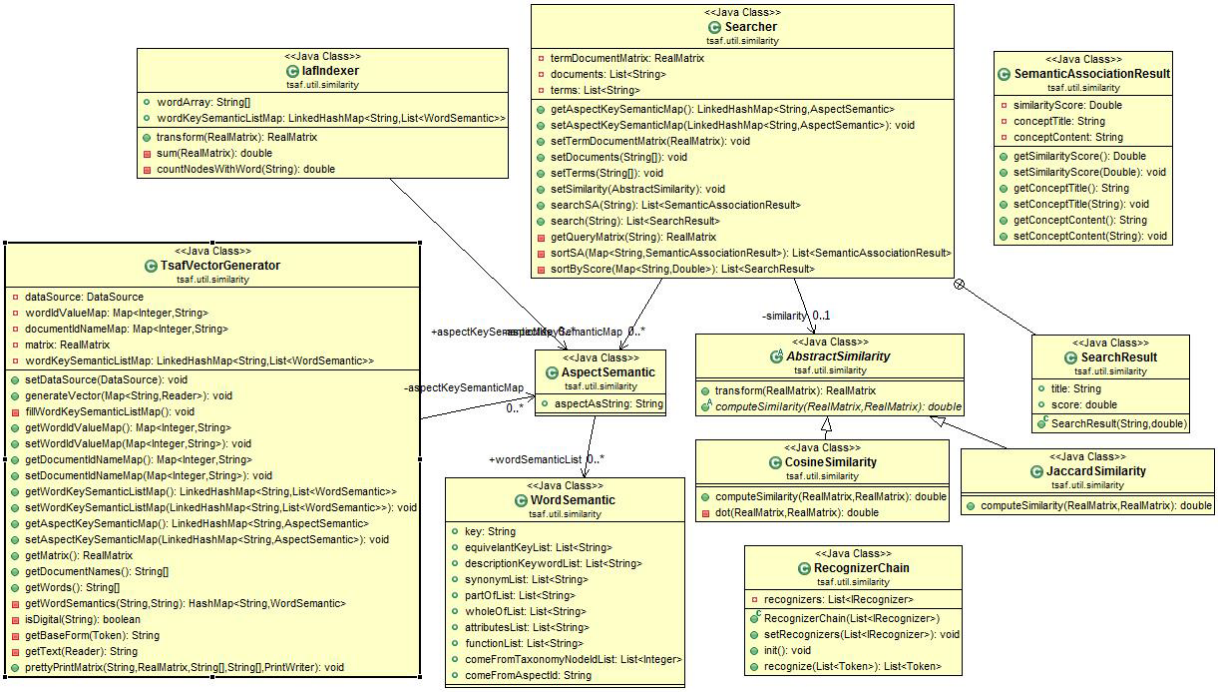


Figure B-2. TSAF similarity service model

B.2 Taxonomy-based Information Specification Framework (TISF) Implementation

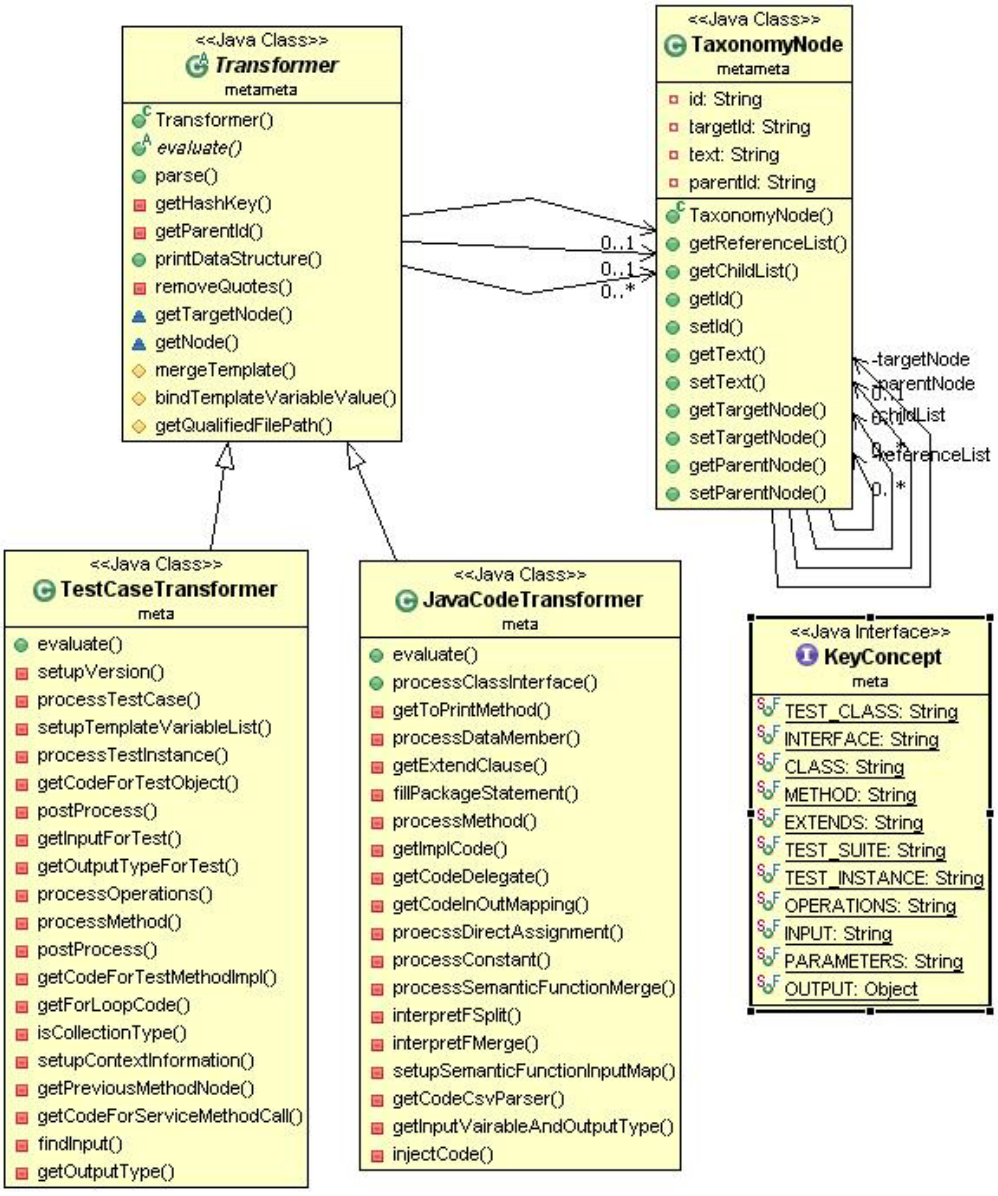


Figure B-3. TISF meta-meta and meta models

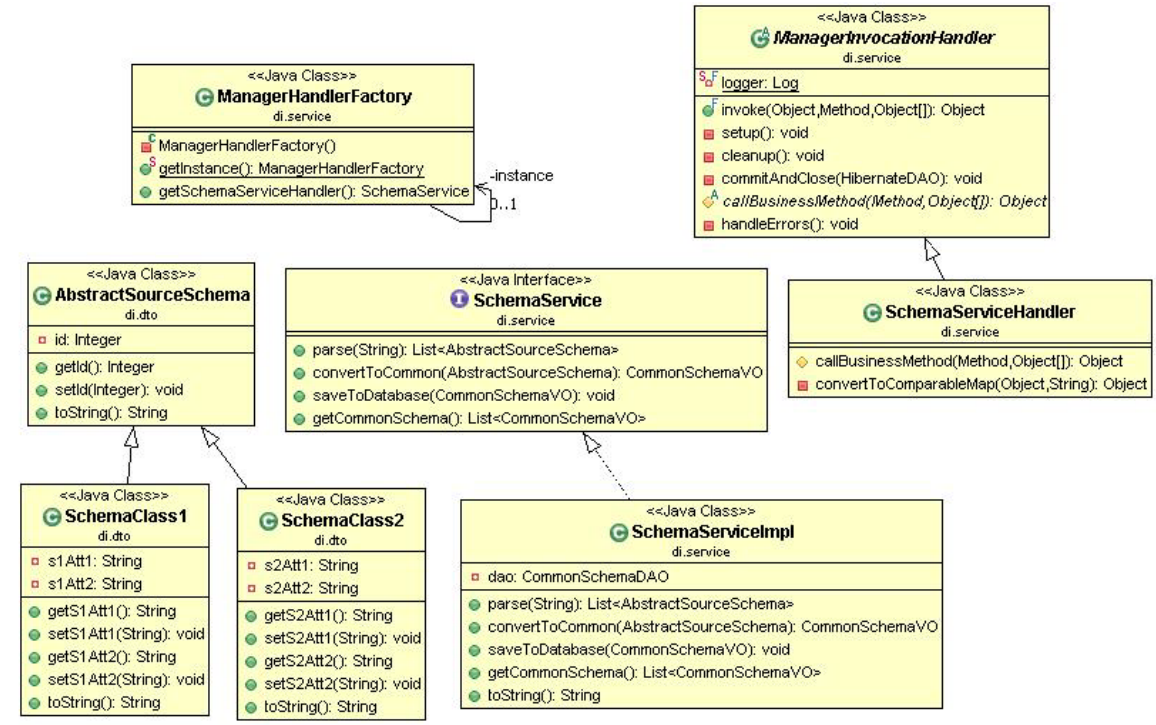


Figure B-4. TISF application model – service layer

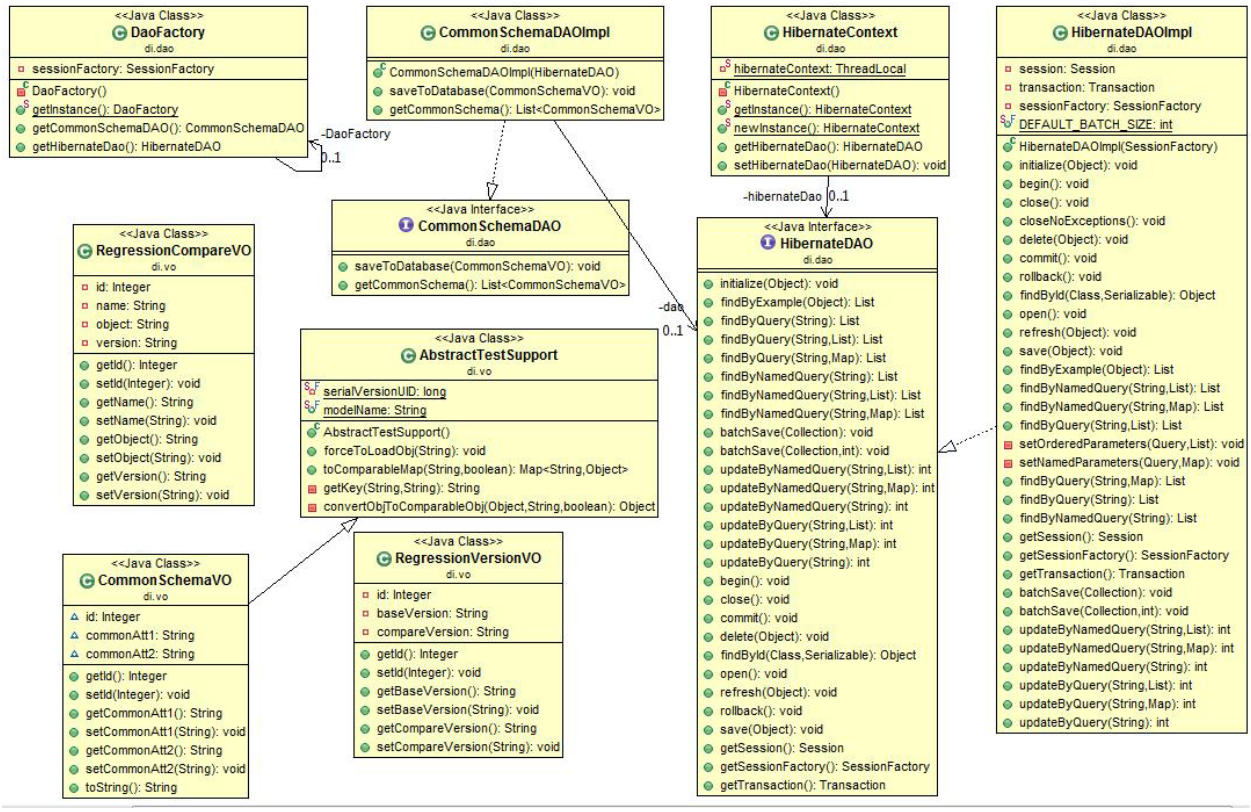


Figure B-5. TISF application model – data access layer

REFERENCES

[1] Gruber, T.R., “A translation approach to portable ontology specification,” Knowledge Acquisition 5(2): pp. 199-220, 1993.

[2] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm, “Generic schema matching with Cupid,” Paper presented at the Twenty Seventh International Conference on Very Large Databases (VLDB'2001), Roma, Italy.

[3] Natalya F. Noy, Mark A. Musen, “The PROMPT suite: Interactive tools for ontology merging and mapping,” International Journal of Human-Computer Studies, v.59 n.6, p.983-1024, December 2003.

[4] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara, “Semantic matching of web services capabilities,” In the First International Semantic Web Conference (ISWC), 2002.

[5] Hong Hai Do, Sergey Melnik, Erhard Rahm, “Comparison of schema matching evaluations,” Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems, p.221-237, October 07-10, 2002.

[6] Aberber, K., Cudré-Mauroux, P. and Hauswirth, M., “The Chatty Web: Element semantics through gossiping,” Proceedings of the 20th International World Wide Web Conference, 2003, pp. 197–206.

[7] Doan, A., Madhavan, J., Domingos, P. and Halevy, A., “Learning to map between ontologies on the Semantic Web,” The VLDB Journal, Vol. 12, pp. 303-319, 2003.

[8] David W. Embley, Douglas M. Campbell, Randy D. Smith, Stephen W. Liddle, “Ontology-based extraction and structuring of information from data-rich unstructured documents,” in Proc. of the Seventh International Conf. on Information and Knowledge Management, November, 1998, pp. 52-59.

[9] Nigel Shadbolt, Tim Berners-Lee, Wendy Hall, “The Semantic Web revisited,” IEEE Intelligent Systems, vol.21 no.3, pp. 96-101, May 2006.

[10] Antoniou G., van Harmelen F., “Web Ontology Language: OWL,” In: Staab S, Studer R, editors. Handbook on Ontologies in Information Systems. Berlin; New York: Springer; 2004. pp. 67-92.

[11] Construction Specifications Institute, MasterFormat 95™: Alexandria, VA: The Construction Specifications Institute, 1995.

[12] Charette, R. P. and Marshall, H. E., UNIFORMAT II Elemental Classification for Building Specifications, Cost Estimating, and Cost Analysis, NISTIR 6389, Gaithersburg, MD: National Institute of Standards and Technology, October 1999

[13] Barry Buzan, The Mind Map Book: How to Use Radiant Thinking to Maximize Your Brain's Untapped Potential, Plume March, 1996

[14] Jörg Müller, Daniel Polansky, Petr Novak, Christian Foltin, Dimitry Polivaev., et al. (2010, August 31), FreeMind - free mind mapping software, [Online]. Available: http://freemind.sourceforge.net/wiki/index.php/Main_Page

[15] Tim Berners-Lee, James Hendler and Ora Lassila (2001, May 17), The Semantic Web, Scientific American Magazine. [Online]. Available: http://www.sciam.com/article.cfm?id=the-semantic-web&print=true

[16] Rahm, E and Bernstein, P. A., "A survey of approaches to automatic schema matching," The VLDB Journal, Vol. 10, pp. 334-350, 2001.

[17] Mitchell, B.S., Mancoridis, S. and Traverso, M., "Reverse engineering: Search-based reserve engineering," in Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, 2002, pp. 431-438

[18] Firat, A., Madnick, S. and Grosof, B.N., "Knowledge integration to overcome ontological heterogeneity: Challenges from financial information systems," In Proc. of ICIS-2002, 2002.

[19] Amor, R. and Faraj, I., "Misconceptions about Integrated Project Database," ITCON, Vol. 6, pp. 57-66, 2001.

[20] D. Fensel, Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce. Springer-Verlag, 2001.

[21] W. Hu, N. Jian, Y. Qu, and Y. Wang, "GMO: A Graph Matching for Ontologies," In Proceedings of the K-CAP'05 Workshop on Integrating Ontologies, 2005.

[22] Qu, Y., Hu, W., and Cheng, G., "Constructing virtual documents for ontology matching," In Proc. of the 15th International World Wide Web Conference (WWW'06), 2006, pp. 23–31.

[23] Fausto Giunchiglia, Pavel Shvaiko, Mikalai Yatskevich, "Discovering missing background knowledge in ontology matching," In Proceeding of the 2006 conference on ECAI 2006: 17th European Conference on Artificial Intelligence, pp.382-386.

[24] Gallaher, M. P., O'Connor, A. C., Dettbarn, J. L., Jr.' Gilday, L. T., "Cost analysis of inadequate interoperability in the U.S. Capital Facilities Industry," NIST GCR 04-867, Gaithersburg, MD: National Institute of Standards and Technology, August, 2004.

[25] Hung-ju Chu, Randy Y.C. Chow, et al., "Semantic association of taxonomy-based standards using ontology," presented at the K-Cap Workshop on Integrating Ontologies, October 2-5, 2005.

[26] Hung-ju Chu, Randy Y.C. Chow, “Reaching semantic interoperability through semantic association of domain standards,” presented at the 11th International Workshop on Future Trends of Distributed Computing Systems, Sedona, Arizona, March 21-23 2007.

[27] Hung-ju Chu, Randy Y. C. Chow, “An information model for managing domain knowledge via faceted taxonomies,” presented at the 11th IEEE International Conference on Information Reuse and Integration IRI-2010, Las Vegas, August 4-6, 2010.

[28] Pablo Castells, Mriam Fernandez and David Vallet,” An Adaption of the Vector Space Model for Ontology based Information Retrieval”, IEEE Transaction on Knowledge and Data Engineering, vol. 2, pp.261-272, 2007.

[29] Yufei Li, Yuan Wang, and Xiaotao Huang, “A Relation-Based Search Engine in Semantic Web”, IEEE Transaction on Knowledge and Data Engineering, vol.19, pp.273-282,2007.

[30] P.L. Tarr et al., “N degrees of separation: Multi-dimensional separation of concerns,” In Proc. of the 21st International Conference on Software Eng. (ICSE 99), ACM Press, 1999, pp. 107–119.

[31] M. Volter and T. Stahl, “Model-driven software development: Technology, engineering, management,” John Wiley and Sons Ltd, 2006.

[32] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. J. Grose, Eclipse modeling framework: A developer's guide, Addison Wesley Professional, 2003.

[33] J. Bézivin, G. Hillairet, F. Jouault, I. Kurtev, and W. Piers, “Bridging the MS/DSL tools and the Eclipse modeling framework,” In Proceedings of the International Workshop on Software Factories (OOPSLA’05), San Diego, California, USA, 2005.

[34] Turhan Özgür, “Comparison of Microsoft DSL Tools and Eclipse Modeling Frameworks for Domain-Specific Modeling In the Context of the Model-Driven Development,” Masters Thesis, School of Engineering, Blekinge Institute of Technology, Ronneby, Sweden, January 2007

[35] Johan den Haan (2009, June 25), 8 Reasons Why Model-Driven Development is Dangerous, [Online]. Available: http://www.theenterprisearchitect.eu/archive/2009/06/25/8-reasons-why-model-drivendevelopment-is-dangerous

[36] B. Selic, "The pragmatics of model-driven development," IEEE Software, vol. 20, pp. 19-25, 2003.

[37] Z. Aleksovski, M. Klein, W. ten Katen, and F. van Harmelen, “Matching unstructured vocabularies using a background ontology,” In Proceedings of Knowledge Engineering and Knowledge Management, Springer-Verlag, 2006.

[38] A. Doan and A. Halevy, “Semantic integration research in the database community: A brief survey," AI Magazine, Special Issue on Semantic Integration, 2005.

[39] M. Iwayama, T. Tokunaga, “Cluster-based text categorization: A comparison of category search strategies,” in the Proc. of the 18th annual international ACM SIGIR Conference on Research and Development in Information Retrieval, 1995, pp. 273-280.

[40] D. Burdett, N. Kavantzas (2004, March), WS Choreography Model Overview, [Online]. Available: http://www.w3.org/TR/2004/WD-ws-chor-model-20040324/.

[41] Boanerges Aleman-Meza, Chris Halaschek, I. Budak Arpinar, and Amit Sheth, “Context-aware semantic association ranking,” in Proc. of the Semantic Web and Databases Workshop, Berlin, September 2003.

[42] D. Brickley and R.V. Guha.(2000, March), Resource Description Framework (RDF) Schema Specification 1.0. [Online]. Available: http://www.w3.org/TR/2000/CR-rdf-schema-20000327/.

[43] Andrew Grove, “Taxonomy,” Encyclopedia of Library and Information Science, 2003, pp. 2770–p.2777.

[44] Rosen, Harold J., Construction Specifications Writing: Principles and Procedures, 5th edition, Hoboken, N.J., J. Wiley, 2005.

[45] Maedche, A., Neumann, G., Staab, S., “Bootstrapping an ontology-based information extraction system,” Studies in Fuzziness and Soft Computing, Intelligent Exploration of the Web, Springer 2002.

[46] L. Ding, T. Finin, A. Joshi, R. Pan, R. Scott Cost, J. Sachs, V. Doshi,P. Reddivari, and Y. Peng, “Swoogle, a Search and Metadata Engine for the Semantic Web,” Proc. 13th ACM Conf. Information and Knowledge Management (CIKM ’04), Nov.2004

[47] Van Rijsbergen, Information Retrieval, London: Butterworths, 1979. Second Edition.

[48] Church, K. W. and Gale, W. A., “Inverse document frequency (IDF): A measure of deviations from Poisson,” in A. et al. (Ed.), NLP using Very Large Corpora, Kluwer Academic Publishers, 1999.

[49] Miller, G. A., Ed. “WordNet: An on-line lexical database,” International Journal of Lexicography 3, 4, Winter 1990, pp. 235-312.

[50] M. Uschold and M. Grüninger, “Ontologies and semantics for seamless connectivity,” SIGMOD Record, vol. 33, no. 3, 2004.

[51] N. Noy, “Semantic integration: A survey of ontology-based approaches,” SIGMOD Record, Special Issue on Semantic Integration, 2004.

[52] I. Niles and A. Pease, “Towards a standard upper ontology,” Paper presented at the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001), Ogunquit, Maine, 2001.

[53] A. Gangemi, N. Guarino, C. Masolo, and A. Oltramari, “Sweetening Wordnet with DOLCE,” AI Magazine, vol. 24, no. 3, pp. 13–24, 2003.

[54] M. Grüninger, “A guide to the ontology of the process specification language,” in S. Staab and R. Studer, editors, Handbook on Ontologies, Springer, 2003.

[55] S. Polyak, J. Lee, M. Gruninger, and C. Menzel, “Applying the process interchange format(PIF) to a supply chain process interoperability scenario,” in Workshop on Applications of Ontologies and Problem Solving Methods, ECAI’98, Brighton, England, 1998.

[56] A. Valente, T. Russ, R. MacGrecor, and W. Swartout, “Building and (re)using an ontology for air campaign planning,” IEEE Intelligent Systems, vol. 14, no. 1, pp. 27–36,1999.

[57] M. Grüninger and J. Kopena, “Semantic integration through invariants,” in Workshop on Semantic Integration at ISWC-2003, Sanibel Island, FL, 2003.

[58] J. Barwise and J. Seligman, Information Flow: The Logic of Distributed Systems, Cambridge University Press, 1997.

[59] E. Hovy, “Combining and standardizing large scale, practical ontologies for machine translation and other uses,” in Proceedings of the First International Conference on Language Resources and Evaluation (LREC), Granada, Spain, 1998, pp. 535–542.

[60] G. Stumme and A. Mädche. FCA-Merge, “Bottom-up merging of ontologies,” in Proceedings of the 7th International Conference on Artificial Intelligence (IJCAI ’01), Seattle, WA,2001, pp. 225–230.

[61] J. Biskup and B. Convent, “A formal view integration method,” in Proceedings of the ACM Conf. on Management of Data (SIGMOD), 1986.

[62] J. Madhavan, A. Halevy, P. Domingos, and P. Bernstein, “Representing and reasoning about mappings between domain models,” in Proceedings of the National AI Conference (AAAI-02), 2002.

[63] AnHai Doan, “Learning to map between structured representations of data,” Ph.D. Dissertation, the Department of Computer Science and Engineering, University of Washington 2002.

[64] Y. Kalfoglou and M. Schorlemmer, “IF-Map: An ontology mapping method based on information flow theory”, Journal on Data Semantics, vol. 1, no. 1, pp. 98–127, Oct. 2003.

[65] F. Dretske, Knowledge and the Flow of Information, MIT Press, 1981.

[66] Patwardhan S, Banerjee S, Pedersen T., “Using measures of semantic relatedness for word sense disambiguation,” in Proceedings of the Fourth International Conference on Intelligent Text Processing and Computational Linguistics, Mexico City, Mexico, 2003, pp. 241-57.

[67] Ted Pedersen, Serguei V. S. P. et al., “Measures of semantic similarity and relatedness in the biomedical domain,” Journal of Biomedical Informatics, vol. 40, issue 3, pp. 288-299, June 2007.

[68] D.L. Lee, H. Chuang, and K. Seamons. “Document ranking and the vector space model,” IEEE Transactions on Software, vol. 14, no. 2, pp. 67-75, 1997.

[69] S. Robertson, “Understanding inverse document frequency: on theoretical arguments for IDF,” Journal of Documentation, vol. 60, pp. 503-520, 2004

[70] Frank van Ham, Martin Wattenberg, Fernanda B. Viégas, “Mapping text with phrase nets,” IEEE Transaction on Visualization and Computer Graphics, vol. 15, no. 6, pp. 1169-1176, December 2009.

[71] Klaus Pohl•Günter Böckle, Frank van der Linden , Software Product Line Engineering Foundations, Principles, and Techniques, Springer New York, 2005

[72] Gary E. Mogyorodi, “What is requirements-based testing?,” in Proceedings of the Fifteenth Annual Software Technology Conference, Salt Lake City, UT, Apr.28-May 1, 2003.

[73] Peter Zielczynski (2006, Feb), Traceability from Use Cases to Test Cases. [Online]. Available: http://www.ibm.com/developerworks/rational/library/04/r-3217/index.html?S_TACT=105AGX15&S_CMP=EDU

[74] Naoyasu Ubayashi, Jun Nomura, Tetsuo Tamai, “Archface: A contract place where architectural design and code meet together,” in Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, May 2010.

[75] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin, “Aspect-oriented programming,” in

Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag LNCS 1241. June 1997

[76] Grig Gheorghiu, “A Look at Selenium,” Better software magazine, October 2005, [Online]. Available: http://agile.unisonis.com/articles/Grig_Gheorghiu_SeleniumToolLook.pdf

[77] Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts, Refactoring Improving the Design of Existing Code, Addison-Wesley Professional, July, 1999.

[78] Object and relational mapping (ORM) with Hibernate, Red Hat, Inc. [Online]. Available: http://www.jboss.com/pdf/HibernateBrochure-03_07.pdf

[79] M. M. Islam Chisty (2005, October 14), An Introduction to Java Annotations. [Online]. Available: http://www.developer.com/java/other/article.php/3556176/An-Introduction-to-Java-Annotations.htm.

[80] Mattias Merz, “Using the dynamic proxy approach to introduce role-based security to Java data objects,” in Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE 06), San Francisco, July 2006.

[81] Hung-ju Chu, Randy Y.C. Chow and Su-Shing Chen, “Model-driven testing with taxonomy and semantic referencing,” International Journal of Software Engineering and Knowledge Engineering (IJSEKE) - A Focused Topic Issue on Software Test Automation, Practice, and Standardization, 2011, (submitted for publication).

BIOGRAPHICAL SKETCH

Hung-ju Chu was born in Taiwan. He graduated from the Department of Radiological Technology, Chung-Tai Junior College, in 1987 and went on to serve as a Lieutenant in the Army of Taiwan for two years. In 1990, he started his undergraduate studies in electrical engineering at the National Cheng-Kung University, Tainan, Taiwan, graduating in the top 5% of his class in 1994. In Fall 1995, he entered the graduate program in computer science at the University of Florida, Gainesville, FL. His research focused primarily on active database management systems. After graduating with Master of Engineering degree in 1997, he worked for Motorola, Inc. and then Oracle, Inc. as a software engineer for 5 years. In 2002, he returned to Gainesville and commenced his Ph.D. studies, advancing to Ph.D. candidate in 2005. While attending graduate school, he also worked for W.S. Badcock Corporation, Mulberry, FL as an independent consultant from 2005 to 2007 and since 2007 has worked for Infinite Energy, Inc, Gainesville, FL as a team supervisor responsible for developing software systems. He received his Ph.D. degree from the University of Florida in the fall of 2010. His research interests include information modeling and its applications to software engineering.