Domain Feature Model Recovery from Multiple Applications using Data Access Semantics and Formal Concept Analysis

Yiming Yang, Xin Peng, Wenyun Zhao
School of Computer Science, Fudan University, Shanghai 200433, China
{051021056, pengxin, wyzhao }@fudan.edu.cn

Abstract

Feature models are widely employed in domain-specific software development to specify the domain requirements with commonality and variability. A feature model is usually constructed by domain experts after comprehensive domain analysis. In this paper, we propose a method to recover an initial domain feature model from multiple existing domain applications using data access semantics and formal concept analysis (FCA). In the method, we first establish mappings among the database schemas of all the reference implementations. Then, we capture the data access semantics of each method in each reference implementation. Based on the pre-established data mapping, we can mix methods from different applications together and conduct formal concept analysis with the data access semantics as intention. After that, further concept merging/pruning and variability analysis are performed to produce the domain feature model. In order to evaluate the effectiveness of our method, we conduct a case study on three open-source forum applications and present comprehensive analysis and discussions on the results.

1. Introduction

When an organization plans to enter a new business domain or transfer to software product line (SPL) engineering, domain analysis is usually a necessary activity to provide a generic description of the requirements of the domain. The product of domain analysis is domain model, which is usually represented by feature models [1]. A feature model specifies the domain requirements with commonality and variability (optional, alternative and OR features).

A feature model is usually constructed by domain experts after comprehensive domain analysis, which is a tedious and time-consuming process. A more effective approach is to recover higher-level abstractions beyond those obtained by examining a program itself [2]. Therefore, if some representative reference implementations already exist in the domain, we can also recover the initial domain feature model from these implementations through reverse engineering. In the case of SPL transfer, the reference implementations are the variant products that have been developed by the team. In other cases, they can be applications with similar functionality and their source codes are available. In this way, the efforts of domain analysis can be reduced, and the representative and quality of the feature model can be ensured.

Requirement model recovery has been addressed in some recent works, e.g. goal model recovery in [3] and feature recovery in [4][5]. In this paper, we focus on recovering domain feature model from multiple reference applications. Different from requirement recovery for single system, domain model recovery should incorporate requirement models extracted from different applications. Therefore, a common analysis basis that aggregates consistent analysis information for different applications should be established to enable the commonality/variability analysis.

For domain-specific business applications, the way a program unit operates on data items largely characterize its business semantics. For example, in forum (discussion board) applications, “post thread” usually behaves as inserting a record to the thread table and updating the post number information in the user table, etc. The data accesses of the same business function in different applications are usually identical or similar. Moreover, the data models of different applications in the same domain, which can be represented by entity-relationship (ER) models, are usually similar, and direct mappings among data entities and properties can be established. Therefore, we can use the data access semantics of program units as the analysis basis, supported by the data schema mapping among different applications.

In this paper, we propose a method to recover an initial domain feature model from multiple existing domain applications. In the method, we use the data access semantics defined on database schemas as the business intent of all the program units (methods). Based on the direct mappings among the ER models of different applications, we can establish a common analysis basis for domain model recovery with consistent data access semantics. Then we can mix methods from different applications together and conduct formal concept analysis (FCA) [6] with the methods as objects and data access semantics as intentions. After that, further concept merging/pruning and variability analysis are performed on the original lattice to produce the feature structure. Finally, we can obtain the feature model by naming for each feature with the help of related intensions and extensions. In order to evaluate the effectiveness of our method, we conduct a case study on three open-source forum applications (JForum, MVNForum, JGossip) and present comprehensive analysis on the results.

The remainder of this paper is organized as follows. Section 2 introduces background knowledge. Our method is introduced in Section 3, including the overall process and main steps. The case study on three open-source forum systems is presented in Section 4 with detailed analysis and discussions. Finally, we present some related work in Section 5 and draw our conclusions in Section 6.

2. Background

2.1. Data Access Semantics

Data access semantics characterizes how each method in the program access business data items to accomplish its function. In our method, the business data items are data entities and properties defined in database schemas and corresponding ER models. Then we can define the data access semantics of a method in term of the access type and the data accessed.

Definition 1. (Data Access Semantics): The data access semantics of a method m is a data access set DAS(*m*) and each as∈DAS(*m*) can be represented by *t.f*+*a*, which can be interpreted as *m* conducts access type *a* on field list *f* of table *t*.

In our method, the access types on fields can be *select*, *insert*, *delete*, *update* and *condition filtering*. The last one, *condition filtering*, is a special type that means involving some fields in the condition statement for record filtering. The field list *f* can be empty, denoting that the access is conducted on the whole table, e.g. *insert* and *delete*. For example, *log.id*+*select* means a *select* access to the field *id* of table *log*, and *log*+*delete* denotes a *delete* access to the table *log*.

The data access semantics of the methods with direct access to the database can be elicited by static or dynamic analysis. However, in an application, usually there are also some methods that do not involve any direct data access. These methods can be categorized into two cases: the method is irrelative to business logic, e.g. part of certain technique framework; the method involves indirect data access through method call, data sharing and data passing, etc. Typical examples for the former case in forum applications are pure technical implementations, e.g. the exception handler in JGossip (org.jresearch.gossip.actions. ExceptionHandlerAction). These methods can be eliminated from the analysis basis. For the latter case, we can derive indirect data access semantics through dependency analysis between different methods.

2.2. Formal Concept Analysis

FCA [6] is a mathematical method that provides a way to identify meaningful clusters of objects that have common attributes. It can visualize data and its inherent structures, implications and dependencies. By FCA-based clustering on all the methods from different applications and their data access semantics, we can obtain the initial functional clusters for further feature analysis.

FCA is based on the definitions of formal context, formal concept, concept order and sparse lattice. Here we introduce the definitions given in [7] as follows.

Definition 2. (Formal context) Concept analysis deals with a relation $I \subseteq O \times A$ between a set of objects *O* and a set of attributes *A*. The tuple $C = (O, A, I)$ is called a formal context.

Definition3. (Formal concept) A tuple $c = (O, A)$ is called a concept iff $A = \sigma(O)$ and $O = \tau(A)$,where $\sigma(A) = \{a \in A \mid (o, a) \in I \text{ for } all\ o \in O\}$ and $\tau(O) = \{o \in O \mid (o, a) \in I \text{ for all a} \in \text{A}\}$ .That is, all objects in *c* share all attributes in *c*. For a concept $c = (O, A)$, *O* is called the extent of *c*, denoted by *extent*(*c*), and *A* is called the intent of *c*, denoted by *intent*(*c*).

Definition 4. (Concept order) The set of all concepts of a given formal context forms a partial order via the superconcept-subconcept ordering ≤ : $(O_1, A_1) \le (O_2, A_2) \Leftrightarrow O_1 \subseteq O2$ or, dually, with, $(O_1, A_1) \le (O_2, A_2) \Leftrightarrow A_1 \supseteq A2$.

Definition 5. (Concept Lattice) The set L of all concepts of a given formal context and the partial order ≤ form a complete lattice, called concept lattice.

The concept lattice can be represented in a more readable equivalent way as a s**parse representation of the lattice**. The content of a node N in this representation can be derived as follows:

- The objects of *N* are all objects at (**specific extent of concept**) and below *N*.
- The attributes of *N* are all attributes at (**specific intent of concept**) and above *N*.

We decided to use FCA in domain feature model recovery because of the following benefits: 1) FCA provides an intentional description for each cluster, which makes groupings more interpretable; 2) the hierarchical concept lattice provides structural information for feature structure reconstruction; 3) FCA is widely supported by general-purpose tools with concept lattice construction and visualization, e.g. ConExp [8].

3. Our Method

The process of our method is depicted in Figure 1. It consists of the following major activities:

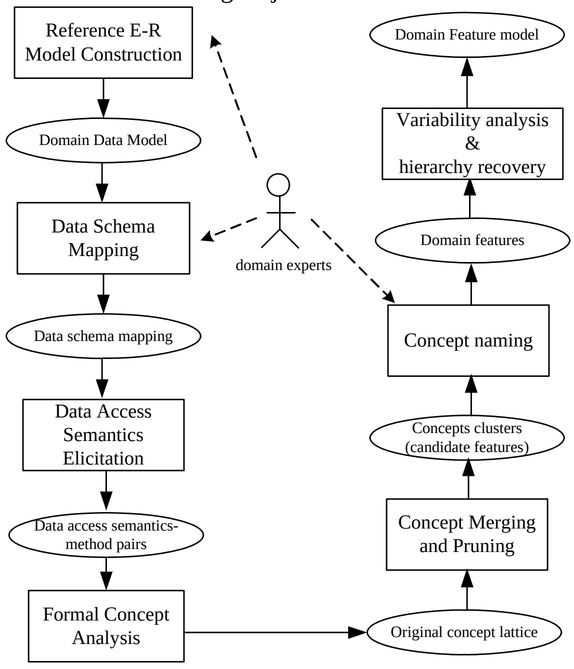
Reference E-R Model Construction
Domain Data Model
Data Schema Mapping
Data schema mapping
Data Access Semantics Elicitation
Data access semantics-method pairs
Formal Concept Analysis
Original concept lattice
Concept Merging and Pruning
Concepts clusters (candidate features)
Concept naming
Domain features
Variability analysis & hierarchy recovery
Domain Feature model
domain experts

Figure 1. The analysis process

1. Domain experts construct a domain data model and establish mappings from application data schemas to the domain data model. The result of this step is data schema mappings.

2. Database access statements in the programs are transformed to direct data access semantics. Besides direct data access semantics, we also use method invocations to derive indirect data access semantics. Finally, the data access semantics of each method should be further standardized based on the data schema mappings. The result of this step is data access semantics-method pairs.

3. Based on the data access semantics of all the methods, we can conduct formal concept analysis with all the methods as objects and their data access semantics as attributes. The result of this step is the original concept lattice.

4. To reduce concept size and cluster related concepts together, we apply concept merging and pruning on the original concept lattice. The result of this step is concepts clusters（candidate features） set.

5. The analysts pick out meaningful concept clusters and name them with the help of rich information from related intensions and extensions. Each named concepts cluster is regarded as a domain feature.

6. The method identifies refinement relations among the features to reconstruct the feature structure. Based on the reconstructed feature structure, feature variability evaluation further identifies feature variations and determines their variation types.

3.1 Data Schema Mapping

The reference applications chose for reverse analysis are often developed by different teams, so their database schemas are usually heterogeneous. In order to provide a common definition basis for the data access semantics, we propose to construct a domain data model and establish mappings from application data schemas to the domain data model. Table 1 lists the data schema mappings for the business entity *Category* among the three forum applications. The mappings are defined on fields or attributes of business entities. According to our experience, most of the data schema mappings are obvious, since the names for the same entity and field are usually identical or similar in different applications. For example, ***Category*** or ***Categories*** are used in two applications, and the other one uses the similar name *Group* (see Table 1).

3.2 Data Access Semantics Elicitation

In our method, direct data access semantics are elicited from the database access statements in the programs (usually SQL statements). Usually, the data access semantics can be elicited through static analysis on source code[9] or dynamic analysis on execution traces[10]. The former is inapplicable and quite

Table 1. Data schema mappings for *Category* among the three forum applications

Domain	JForum	MVNForum	JGossip
Category.categoryId	jforum_categories.categories_id	Mvnforumcategory.CategoryID	jrf_group.groupid
Category.name	jforum_categories.title	Mvnforumcategory.CategoryName	jrf_group.group_name
Category.description	N/A	Mvnforumcategory.CategoryDesc	N/A
Category.order	jforum_categories.display_order	Mvnforumcategory.CategoryOrder	jrf_group.group_sort

expensive due to the dynamically generated SQL statements and the quite different ways of database access. For example, among the three forum applications, JForum conducts database access by loading SQL statements from a java properties file(%APP%/WEB-INF/config/database/) at runtime. Therefore, we use aspect-oriented programming (AOP), especially AspectJ, to implement aspect-based SQL tracing for data access semantics elicitation. AOP is an appropriate technique for dynamic SQL extraction since it crosscuts the SQL execution points without any alteration of the original source code [10].

Based on the SQL records, further SQL syntax analysis can be conducted to elicit the data access semantics for each method. The data access semantics of a SQL statement is characterized by three parts: the table(s) accessed, the fields involved and the constraints (the where clause). The rules for data access semantics elicitation from different kinds of SQL statements are listed in Table 2. We can see that different contents are elicited for different SQL statements, e.g. involved fields in insert statements are not extracted, but they are essential semantics in select and update statements. Each field used in WHERE clause is also an essential semantics.

Besides direct data access semantics, we also use method invocations to derive indirect data access semantics. The rule for indirect data access semantics derivation can be declared as: if method m1 invokes method m2, then there is DAS(*m2*) $\subseteq$ DAS(*m1*). In order to provide a common analysis basis, the data access semantics of each method should be further standardized based on the data schema mappings of corresponding application to the domain data model. Thus, the data access semantics of all the methods from different applications are represented by the data entities and fields defined in the domain data model.

Table 2. Rules for data access semantics elicitation

SQL Statement	Data Semantics
insert into table values (? ? ..)	table+insert
delete from table where condition = ?	table+delete table.condition+cond
update fields = ? where condition = ?	table.fields+update table.condition+cond
select fields from table where condition = ?	table.fields+select table.condition+cond

3.3 Concept Analysis and Concept Pruning /Merging

Based on the data access semantics of all the methods, we can conduct formal concept analysis with all the methods as *objects* and their data access semantics as *attributes*. The concept analysis can be implemented by applying existing FCA tools, e.g. ConExp used in our case study. In implementation, we can convert the analysis information into formal context with the format that can be recognized by ConExp. Then the concept lattice can be constructed by ConExp, and the concept structure and the extents/intents of all the concepts can be read for further feature analysis.

The original concept lattice produced after FCA is usually too large and scattered for feature analysis. For example, the original concept lattice for the three forum applications has more than 3,000 concepts, and most of them are meaningless structural nodes. Besides, other concepts may be too small and incomplete as candidate features. Therefore, we should conduct concept pruning on the original concept lattice to remove meaningless concepts and concept merging to generate candidate features with appropriate granularity.

Our concept pruning/merging algorithm is based on our observations to the four types of concepts in the original concept lattice as in Table 3. These four types of concepts are characterized by their specific extents and intents in the sparse concept lattice. Their interpretations provide the illumination of how to determine the pruning/merging policies for different types of concepts as follows.

1) A concept of the first type implements more concrete functions than its parent concept, embodied by its specific intents. Therefore, it can be a starting point for a new functional cluster if its specific intents are significant (with meaningful enrichment to its parent concepts), otherwise it should be merged with its parent concept if its specific intents are slight.

2) Concepts of the second type are functional abstracts, but they (actually most of them) may only involve some incomplete business semantics. Therefore, they can be identified as alternative features if considered to be meaningful. It can be evaluated by whether its combination of related intents can be interpreted as a meaningful business function.

3) A Concept of the third type combines intents (data access semantics) from different parent concepts. Among its parent concepts, there may be a dominant one that provides most of the intents, and other combined intents are secondary. In this case, the concept can be merged with the dominant parent concept. Otherwise, it can be treated as a candidate composite feature and a starting point for a new concept cluster.

4) The fourth type of concepts is meaningless and can be ignored in feature analysis.

Based on our observations to the four types of concepts, we propose a recursive algorithm for concept pruning/merging as shown in Table 4. Inputs of the algorithm include original sparse lattice graph and threshold of intent similarity. Output is the concepts clusters set. In the algorithm, function *intentSimilarity* is used to measure the intent similarity between two concepts as follows.

Table 3. The four types of concepts

Concept Type	Interpretation
with specific intents and specific extents	Concrete functions with specific intents (data access semantics)
with specific intents but without specific extents	Functional abstracts, all the objects (methods) are gathered from the child concepts
with specific extents but without specific intents	Concrete functions, all the intents (data access semantics) are inherited from the parent concepts
without specific extents or specific intents	Meaningless structural nodes only to form the concept lattice

Definition 6. (Intent Similarity) The intent similarity of two concepts is dependent on the intersection of two intents. It is the evidence to judge whether two concepts should be merged.

$$IntentSimilarity(C_1, C_2) = \frac{|\,Intent(C_1) \cap Intent(C_2)\,| * 2}{|\,Intent(C_1)\,| + |\,Intent(C_2)\,|}$$

Table 4. The algorithm of concept pruning/merging

```
input L : sparse lattice graph
   threshold_i : Threshold of intent similarity
output CCS : Concepts Clusters Set
variables le: parent element in traverse path
traverseLattice(LatticeElement e , Cluster C) //start from top element of L
begin
  if specific intent of e is not empty and specific extent of e
     is not empty
     if intentSimilarity(e , le) > threshold_i
        C := C ∪ {e };  le:= e ;
        continue traverse all children of node e;
     else
        CCS := CCS ∪ { C }; create C := {e} ;
        le:= e ; identify e as primitive feature;
        traverse all children of node e;
  else if specific intent of e is empty and specific extent of
e is not empty
     if intentSimilarity(e,le)> threshold_i
        C := C ∪ {e }; le := e ;
        continue traverse all children of node e;
     else
        CCS := CCS ∪ { C }; create C := {e} ;
        le := e ; identify e as composite feature;
        traverse all children of node e;
  else
     continue traverse all children of node e;
end
```

By traversing the original concept lattice, the algorithm merges neighboring concepts whose intents are similar and have specific extents. To judge whether two concepts could be merged, we use a simple intent similarity metric equation. Concepts which can't be merged will be a staring point for a new functional cluster. Concepts without specific extents are used as structure node for traversing.

During the merging process, a concept may be merged with multiple parent concepts. If a concept node exists in multiple (more than $threshold_{pruning}$) concept clusters, they are regarded as irrelevant functions for each feature and should be pruned from concept clusters. These pruned concepts may be composite feature candidates. They may also be some utility functions (e.g. net.jforum.drivers.generic. AutoKeys.getStatementForAutoKeys (String)).

3.4 Feature Naming and Structure Reconstruction

After concept pruning/merging, the analysts examine each of the generated candidate features (concept clusters) to evaluate its business meanings. Meaningless candidate features are removed, whilst meaningful candidate features are chosen as domain features. Then the analysts can name each domain feature with the help of its intents and extents. After this manual examination and naming, all the domain features are determined with significant names denoting their business functions.

Based on their structural relations in the original concept lattice, our method then identifies refinement relations among domain features to reconstruct the feature structure. Basically the feature structure can be constructed from the hierarchical structures of the concept clusters. The original concept lattice is based on a mixed formal context with objects and attributes from multiple different applications, which usually have different functional structures. Therefore, the concept clusters may have inconsistent hierarchical structure, and the structural relations among domain features should be re-organized to produce a representative feature structure for the whole domain. In the algorithm presented in Table 4, primitive features and composite features have been distinguished. Therefore, the feature structure reconstruction in our method is mainly to re-organize the composite features in the feature structure. It is based on our qualification to sub-feature relations.

Definition 7. (Sub-Feature Qualification) A feature *C2* is qualified to be a sub-feature of another feature *C1* if and only if: PFS(*C2*) $\subset$ PFS(*C1*) and there does not exist any other feature *C3* satisfying PFS (*C2*) $\subset$ PFS (*C3*) $\wedge$ PFS (*C3*) $\subset$ PFS (*C1*). PFS is the Primitive Feature Set of feature *c*. Each primitive feature *f* in PFS is direct or indirectly sub feature of *c*.

An illustrating example of feature structure reconstruction is presented in Figure 2. The primitive feature sets of the composite features in Figure 2(a) are listed in Table 5. From the table, we can see that there are PFS (*C2*) $\subset$ PFS (*C1*), PFS (*C3*) $\subset$ PFS (*C1*) and PFS (*C5*) $\subset$ PFS (*C4*). According to Definition 7, we can obtain the reconstructed feature structure as shown in Figure 2(b).

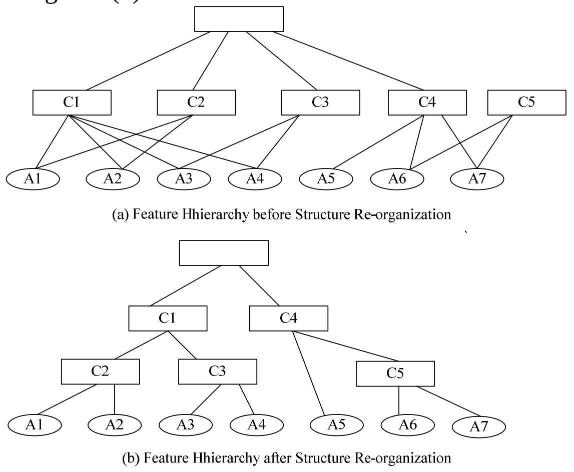
C1
C2
C3
C4
C5
A1
A2
A3
A4
A5
A6
A7
(a) Feature Hhierarchy before Structure Re-organization
C1
C4
C2
C3
C5
A1
A2
A3
A4
A5
A6
A7
(b) Feature Hhierarchy after Structure Re-organization

Figure 2. An example of feature structure reconstruction

Table 5. The atomic feature sets of the composite features in Figure 2(a)

Composite Feature	Primitive Feature Set
C1	*A1, A2, A3, A4*
C2	*A1,A2*
C3	*A3, A4*
C4	*A5, A6, A7*
C5	*A6, A7*

3.5 Feature Variability Evaluation

Based on the reconstructed feature structure, feature variability evaluation further identifies feature variations and determines their variation types (optional, alternative, OR). We will first distinguish functional decomposition and specialization from all the feature refinement relations. Then we can further identify optional, alternative and OR features.

Functional decomposition (AND) and specialization (OR/alternative) are the two typical refinement mechanisms in feature model. They can be distinguished according to the three concept types identified in Table 3. Each feature consisting of the second type of concepts (with specific intents but without specific extents) can be identified as an OR-decomposed feature, and its sub-features can be seen as variants for it. Features consisting of the first or the third type of concepts are AND-decomposed features.

Then optional features can be identified from AND-decomposed sub-features with the characteristics that their binding states (bound or removed) can be different in different applications. Therefore, we can identify optional features by evaluating the concept extents of related features, which originally are obtained by concept merging. The rule for optional feature qualification can be identified as follows.

Definition 8. (Optional Feature Qualification) An AND sub-feature f_2 can be qualified to be an optional feature of its parent feature f_1 if there is an application *app* satisfying:($\exists$ *m***.***m* $\in$ *app* $\wedge$ *m* $\in$ extent(f_1)) $\wedge$ ($\neg\exists$ *m***.** *m* $\in$ *app* $\wedge$ *m* $\in$ extent(f_2)).

As OR decompositions have been identified, identification of alternative/OR features is to further distinguish alternative features from OR-decomposed sub-features. Alternative features are exclusive specializations to their parent features, denoting that only one out of a set of alternative features can be bound for each application. Therefore, alternative features can be identified by the following rule.

Definition 9. (Alternative Feature Qualification) A set of OR sub-feature *FS* can be qualified to be alternative features of their parent feature *f* if there does not exist an application *app* satisfying: $\exists$ f_1, $f_2 \in FS$**.** ($\exists$ *m***.** *m* $\in$ *app* $\wedge$ *m* $\in$ extent(f_1)) $\wedge$ ($\exists$ *m***.** *m* $\in$ *app* $\wedge$ *m* $\in$ extent(f_2)).

According to definition 9, exclusive OR sub-features can be identified as alternative features, and other OR sub-features can be finally determined to be OR features.

4. Case study

This section presents our case study on three open-source forum applications with detailed discussions.

4.1. Case Settings

In our case study, three open-source forum applications are adopted for domain feature model recovery. The reasons for choosing the forum domain are as follows. First, forum systems are widely-used online applications with obvious and recognized business functions, facilitating the result validation. Second, there exist many open-source forum implementations developed by different teams. Third, forum systems are typical database-based applications.

The scales of the three forum applications are presented in Table 6 according to the numbers of lines of code (LOC), classes, methods and features. The features are counted on the manually constructed feature mode for each application. It takes two graduates who are familiar with forum systems two weeks to separately finish analysis for all the three applications. From Table 6, we can see MVNForum provides the most complete services from the size of source code and the feature number. Actually MVNForum does involve many unique functions, e.g. folder management for private messages, threatening content report and reply content split to new thread, etc. However, the other two smaller forum applications also involve some unique functions.

In order to provide a common definition basis for data access semantics, we identify 9 main domain entities and establish corresponding data entity mappings as listed in Table 7. It can be seen that a domain entity may be mapped to 0-2 application-specific data entities. Based on the data entity mappings, field mapping can be further identified for data access semantics elicitation.

Table 6. System characteristics

Application	#LOC	#Class	#Method	#Feature
JForum	33123	188	1618	68
MVNForum	105119	425	4578	141
JGossip	33507	260	1299	54

Table 7. Data entity mappings of the three systems

Entity	JForum	MVNForum	JGossip
Attachment	jforum_attach jforum_attach_desc	Mvnforumattachment	jrf_attach
Category	jforum_categories	Mvnforumcategory	jrf_group
Forum	jforum_forums	Mvnforumforum	jrf_forum
Thread	jforum_topics jforum_posts	Mvnforumthread	jrf_thread jrf_message
Post	jforum_posts jforum_posts_text	Mvnforumpost	jrf_message
Member	jforum_users	Mvnforummember	jrf_user
Group	jforum_groups	Mvnforumgroups	N/A
Message	jforum_privmsgs jforum_privmsgs_text	Mvnforummessage	N/A
Rank	jforum_ranks	Mvnforumrank	jrf_rank

4.2. Implementation

We implement a tool for data access semantics elicitation, concept pruning/merging and feature analysis. First, the tool parses SQL statements captured by SQL tracing to elicit data access semantics for each method as defined in Table 2. We use the Java-based SQL parser Zql [11] to implement SQL parsing. Besides, the tool also converts application-specific tables and fields to the entities and properties defined in the domain data model.

Second, the tool generates ConExp-compatible CEX files encoding the formal context as the input to ConExp to obtain the original concept lattice.

Third, the tool implements the merging/pruning algorithm presented in Table 4 to produce candidate features. We overwrite some methods in the class conexp.frontend.ContextDocument to implement concept merging and pruning based on the ConExp objects for the concept lattice data structure, including conexp.core.Lattice and conexp.core.LatticeElement.

Forth, the tool provides the intents and extents of each candidate feature for evaluation and naming. After all the domain features are determined, the tool then reconstructs the feature structure and identifies feature variations on the rules defined in 3.5.

4.3. Results

In this case study, we have 658 objects (methods) and 132 attributes (data access semantics) from the three applications. After the concept analysis on the mixed formal contexts, 3280 concepts are generated to form the original concept lattice. After concept merging and pruning, 162 concept clusters are produced as candidate features. In feature evaluation and naming, 94 out of the 162 candidate features are accepted as domain features and named by the analysts. Among all the domain features, 16 of them are identified as composite features, and the others are identified as primitive features.

We find most of the meaningless candidate features (61.8%) consist of concepts with specific intents but without specific extents. These meaningless concepts are usually located in positions near the top of the concept lattice, denoting that they have only a small set of intents but often have a big set of extents. Actually, most of them only have incomplete data access semantics like *select* accesses, e.g. there is a meaningless concept with only the intent of "*Forum.name+select*" but shared by many objects.

Part of the recovered feature structure before structure reconstruction is depicted in Figure 3. It is not well structured due to the quite different functional hierarchies in different applications. Therefore, feature structure reconstruction is conducted to form a common and well-structured feature hierarchy. Then after variability evaluation, we obtain the final domain feature model recovered from the three forum applications as presented in Figure 4 (part). Out of all the 94 domain features, 33 are identified as optional features, 2 as OR features, and 1 as alternative feature.

From the recovered feature model, we can see that most of the basic forum functions like category management, rank management, thread list, post/reply are common in these forum applications. The differences (variations) are largely embodied in some additional forum functions, e.g. "delete non-activated members", "lock forum", "report threatening content to admin", etc. From the feature model, we can see that most of the variations are optional features, and alternative/OR features are not so many. "delete single user" is an example of alternative features. Its two variants are "physical delete" and "logic delete". The former means to directly delete the user from the database, while the latter is to mark the user as deleted without database deletion.

Based on the validation of the analysts, the recovered feature model well reflects the business of the forum domain and most of the identified feature variations are correct (92.5%). The problems existing in the recovered feature model largely fall into two types: the feature hierarchies in some parts of the feature model and the validity of variation type qualification. An example of the former is the sub-feature structure for "user function". From Figure 4, we can see that intermediate features between "user function" and its sub-features are missing, making an obscure feature hierarchy. As for the validity of variation type qualification, although most of the variation types are correct (86.2%), there still exist some features that are misclassified in variability evaluation. After analyzing the causes, we find that most of the problems root in incompleteness data access semantics due to the limitations on the coverage of dynamic analysis and the data model mappings. The limitations will be discussed in the next subsection.

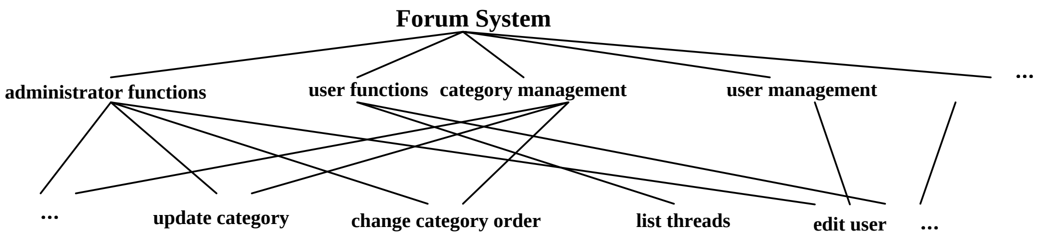

Figure 3. Part of the recovered feature structure before reconstruction

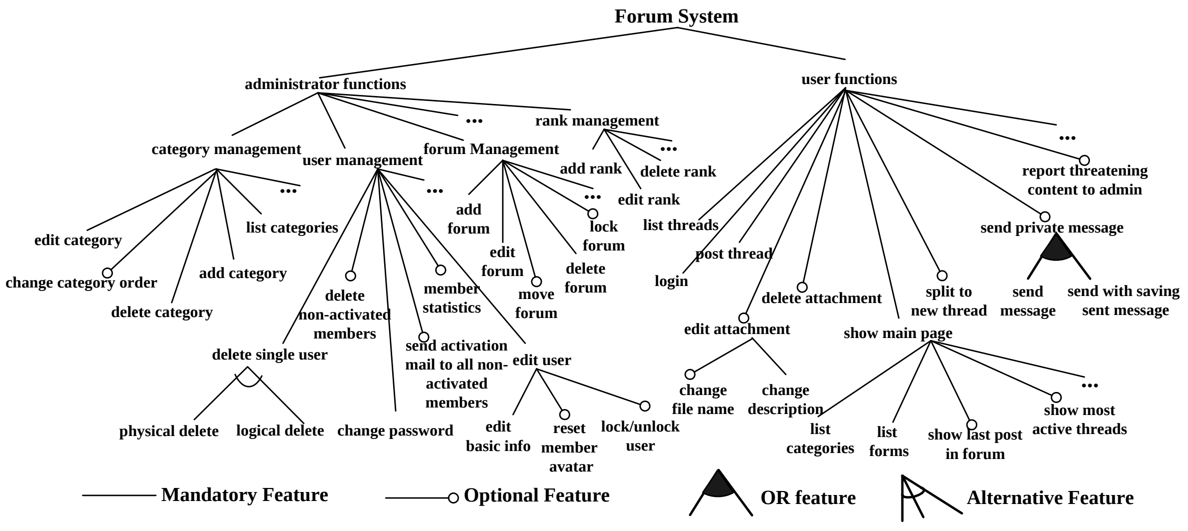

Figure 4. Part of the final recovered domain feature model

4.4. Discussion

4.4.1 Correctness and Completeness

In our method, the correctness and completeness of the recovered domain feature model are largely influenced by three factors: the number of reference applications and their representativeness; the coverage of the dynamic analysis for SQL tracing; and the quality of the data schema mappings.

The first problem can be resolved by carefully choosing reference applications for feature model recovery. It is worth noting that applications developed in different programming language can be employed together for feature model recovery, since the analysis basis of our method is the data access semantics with data model mappings. The only thing that should be considered in addition is the SQL tracing implementation for different languages. The number of the chosen applications influences the variability evaluation greatly. Therefore, involving more applications in the reverse analysis is always better.

In our case study, although we carefully design a series of test cases based on the user/administrator documents, there are still some missing data access semantics. For example, “delete all the posts in a board” is a common feature among the three forum applications, but it is identified as a variation due to the missing of a *delete* access for an application. In that application, the *delete* statement will be executed only if the board is not empty. This is a technical detail that is not included in the documents, so it is neglected in our test cases. From the case study, we find this kind of missing data access semantics may influence both the feature structure and the variability evaluation.

Data schema mappings are established on the analysts’ understanding on the data schemas. According to our experience, it is easy to define a set of good-enough mappings, since most of the basic business entities and their properties are obvious. However, it is not easy to establish a complete set of mappings due to some special and usual data items used in single systems. These missing data mappings may influence the completeness of data access semantics also. Fortunately, they are usually related to some minor functions, so their influences are often small.

4.4.2 Comparison with Manual Recovery

Compared with manual feature model recovery from existing systems, our method requires less effort. The major costs of our method are related to data schema mapping and feature naming. Data schema mapping involves the creation of a common ER data model and mapping all data schema to it. Besides manual mapping, the automatic schema matching techniques [20] in the database area can also be used to help automate the schema mapping process. In our method, the analysts name the recovered features with the hints of the intents and extents of the features. The extents contain rich lexical information of related methods, while the intents contain the combination of a set of data access semantics. The feature evaluation and naming is a subjective process that requires comprehensive business experience in the domain.

Our method has great advantages in more comprehensive and complete feature identification, since it identifies features from existing representative reference applications. It can identify implicit features that are often neglected in manual recovery. For example, in the manual feature analysis of our case study, the analysts neglected the feature “*logic delete*”, but it was captured by our automatic method.

However, manually recovered feature model usually has a clearer feature hierarchy. In Figure 4 we can see that the hierarchy of the sub-features of *UserFunctions* is not clear. Therefore, based on the initial feature model automatically recovered, the domain analysts usually should do some further analysis to obtain more superior feature model.

5. Related Work

Most of recent researches on requirement model recovery are targeted to single systems. Yu et al. [3] propose a method for recovering goal models from both structured and unstructured legacy code. Antoniol et al. [4][5] propose an approach to feature identification and comparison in large multi-threaded object-oriented programs using static and dynamic data, knowledge-based filtering, and probabilistic ranking.

Many techniques such as clone techniques [12], information retrieval [13-16], dynamic execution trace analysis [7] and cohesive OO nature of a class [17] may be adopted for domain feature model analysis. They all have their advantages in some cases and shortcomings in other cases especially to heterogeneous systems.

Most of recently reported feature location methods show advances in applying information retrieval (IR) for recovering traceability between code and documentation. Zhao et al. [16] proposed a static non-interactive approach to feature location. Poshyvanyk et al. [15] propose to combine LSI-based analysis and formal concept analysis (FCA). For domain analysis on heterogeneous systems, national language style key words contain in documents and source code can be considered as the natural bridge to build connections between heterogeneous systems. But it has two main problems: difficult to consolidate terminology and chaos of vocabulary in use cased by diversity of semantics. The second problem especially results low precision and inaccurate for variability analysis on different systems. For instance, the word *Group* (in JGossip) stands for the meaning of a category of forum may be misunderstood as group of users and cause inaccuracy. IR techniques are excessively dependent on terminology of different systems. Moreover, in real world products, terminologies are more different in a thousand and one ways. They are harder to be mapped than database tables.

Basit et al. [12] identified and unify structural clones across the systems. It shows structural clone detection becomes an important element of re-engineering a legacy system for reuse into Product Lines. Structural clone is effective for software product line system families originate from the same code base that is modified in various ways for each specific product. But it is not easy to apply it on discrepant implementation systems because these systems usually have different architecture and design solutions.

Many recent researches [9][18][19] use database reverse engineering techniques to help software reverse engineering tasks. Our approach is inspired by using database information to help program comprehension.

6. Conclusion and Future Work

In this paper, we propose to use data access semantics with a common data schema mapping to consistently characterize the program units from different applications. Based on the data access semantics, we present a method for domain feature model recovery using formal concept analysis and further concept pruning/merging, structure reconstruction and variability analysis. The case study shows that data access semantics can effectively characterize business functions and provide a consistent analysis basis for different applications.

In our future work, we will explore more analysis methods like static analysis, information retrieval, clone detection to recover domain requirements and designs with commonality/variability analysis, and compare their effects and applicability through more case studies.

Acknowledgments. This work is supported by National Natural Science Foundation of China under Grant No. 60703092, National High Technology Development 863 Program of China under Grant No. 2007AA01Z125, and Shanghai Leading Academic Discipline Project under Grant No. B114.

7 References

[1] K. C. Kang, S. G. Cohen, J. A. Hess, W.E. Novak, A.S. Peterson. Feature Oriented Domain Analysis (FODA) Feasibility Study. *Technical report CMU/SEI-90-TR-21*, Software Engineering Institute, Carnegie Mellon University

[2] E. Chikofsky and J.H. Cross II. Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software*, 7(1):13-17, 1990.

[3] Y.Y. Yu, Y.Q. Wang, J. Mylopoulos, S. Liaskos, A. Lapouchnian, J.C.S do Prado Leite. Reverse Engineering Goal Models from Legacy Code. In *RE’05*, pages 363- 372, 2008.

[4] G. Antoniol, Y.G. Gueheneuc. Feature Identification: A Novel Approach and a Case Study.In *ICSM'05*, pages 357- 366, 2005.

[5] G. Antoniol, Y. Gael. Feature Identification: An Epidemiological Metaphor. *IEEE Trans. on Soft. Eng.*, 32(9): 627-641, 2006.

[6] B. Ganter, R. Wille. Formal Concept Analysis: Mathematical Foundations. Springer-Verlag, 1996.

[7] T. Eisenbarth, R. Koschke and D. Simon. Locating Features in Source Code. *IEEE Trans. on Soft. Eng.*, 29(3): 210-224, 2003.

[8] Concept Explorer. http://sourceforge.net/projects /conexp.

[9] C. Gould, Z. Su, and P. Devanbu. Static checking of dynamically generated queries in database. Applications. In *ICSE’04*, pages 645 – 654, 2004.

[10] A. Cleve, J.L. Hainaut, Dynamic Analysis of SQL Statements for Data-Intensive Applications Reverse Engineering. In *WCRE’08*, pages 192 – 196, 2008.

[11] Java SQL Parser ZQL : http://www.gibello.com /code/zql.

[12] H.A. Basit, and S. Jarzabek, A Data Mining Approach for Detecting Higher-level Clones in Software. *IEEE Trans. on Soft. Eng.*, to appear.

[13] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, A. Rummler. An Exploratory Study of Information Retrieval Techniques in Domain Analysis. In *SPLC2008*, pages 67-76, 2008.

[14] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic. An information retrieval approach to concept location in source code. In *WCRE’04*, pages 214-223, 2004.

[15] D. Poshyvanyk and A. Marcus. Combining formal concept analysis with information retrieval for concept location in source code. In *ICPC’07*, pages 37-48, 2007.

[16] W. Zhao, L. Zhang, Y. Liu, J. Sun, F. Yang . SNIAFL: Towards a Static Noninteractive Approach to Feature Location. *ACM Trans. on Soft. Eng. and Methodology*, 15(2): 195–226, 2006.

[17] U. Dekely, Y. Gil .Revealing Class Structure with Concept Lattices. In *WCRE’03*, pages 353-363, 2003.

[18] D. Grosso, C. D. Penta, M. de Guzman, I. G. Rodriguez. An approach for mining services in database oriented applications. In *CSMR’07*, pages 287 – 296, 2007.

[19] M. Alalfi, J.R. Cordy and T.R. Dean. SQL2XMI: Reverse Engineering of UML-ER Diagrams from Relational Database Schemas. In *WCRE’08*, pages 187-191, 2008.

[20] E.Rahm and P.A.Bernstein. A survey of appro-aches to automatic schema matching. *VLDB Journal*,10(4): 334-350,200