CoFM: An Environment for Collaborative Feature Modeling

Li Yi, Wei Zhang, Haiyan Zhao, Zhi Jin, Hong Mei
Institute of Software, School of EECS, Peking University Beijing, China
Key Laboratory of High Confidence Software Technology, Ministry of Education of China

{yili07, zhangw, zhhy, zhijin}@sei.pku.edu.cn, meih@pku.edu.cn

ABSTRACT

Feature models provide an effective way to organize and reuse requirements in specific software domains. Many feature-oriented domain analysis approaches have been proposed to guide the construction of feature models, and in most of these approaches, collaboration among stakeholders is considered to be one of the key factors in the construction of feature models. However, few of these approaches provide explicit mechanisms or methods to support efficient collaboration among stakeholders. In this paper, we present an environment developed for collaborative feature modeling (CoFM) that explicitly supports stakeholders' collaboration in feature-oriented domain analysis. The basic idea of supporting the collaborative work is a mechanism called centralized broadcasting of modeling operations – that is, each modeling operation performed by a user is sent to a central feature model, and then the operation will be broadcasted to other users to make it visible. We develop a tool to implement CoFM, and conduct a series of case studies to explore strengths and weaknesses of it. The observations made from the case studies show that CoFM improves the efficiency of feature modeling, and it is not only suitable for feature-oriented domain analysis, but also feature-based requirements elicitation.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specification – *methodologies, tools*

General Terms

Design, Experimentation, Human Factors

Keywords

Feature Model; Domain Analysis; Requirements Elicitation; Collaboration

1. INTRODUCTION

In software reuse, feature models provide an effective way to organize and reuse requirements in specific software domains. The concept of feature models was first introduced in the FODA method [7]. The idea of feature models is to encapsulate requirements into a set of features and dependencies among features, and then to reuse these encapsulated requirements by selecting a subset of features from a feature model, while maintaining dependencies among features. To take full advantage of feature models in software reuse, a basic problem is how to construct high quality feature models that capture sufficient commonality and variability requirements in specific software domains.

Many feature-oriented domain analysis approaches have been proposed to guide the construction of feature models, and in most of these approaches, collaboration among stakeholders is considered to be one of the key factors in the construction of feature models. For example, in FODA [7] and FORM [8] methods, during a feature model's construction, domain analysts (i.e. constructors of feature models) should make intensive communications with users and domain experts to gather the necessary information about the domain under consideration and resolve possible conflicts among these stakeholders, and after the feature model is constructed, the model then should be reviewed by domain experts, users, requirements analysts and other related stakeholders, to ensure the quality of the feature model. In FeatuRSEB [6], domain experts are important information sources for domain analysts in the whole domain analysis cycle – context modeling and scoping, domain use case modeling, and domain feature modeling. The reason for collaboration is obvious: most real software domains are complex and often evolve frequently, thus it is usually impossible for only one or a few persons to obtain a comprehensive understanding of a domain without sharing knowledge with others.

However, few of these approaches provide explicit mechanisms or methods to support efficient collaboration among stakeholders. One consequence of this problem is that, in feature models' construction, the way to collaborate among stakeholders and the effectiveness of collaboration will depend much on domain analysts' personal experience, and thus the efficiency of collaboration and feature models' construction is often unsatisfied. Furthermore, even if domain analysts use general collaboration tools (e.g. wikis) to enhance collaboration among stakeholders, the effectiveness of feature models' construction will still be hindered by the concept-gap between feature models and these general tools. Therefore, domain analysts have to switch their work context frequently: on the one hand, stakeholders have to share knowledge and resolve conflicts of stakeholders in the collaboration tools; on the other hand, they need to transform acquired knowledge and feedback into feature models. Such a transformation process imposes considerable work load on domain analysts and other stakeholders, and makes feature models' construction a time-consuming and error-prone task. We

believe that this situation can be improved by integrating collaboration support into feature modeling activities through a more systematic manner.

In this paper, we present an environment developed for collaborative feature modeling (CoFM) that explicitly supports stakeholders' collaboration in feature-oriented domain analysis. The basic idea of supporting the collaborative work is a mechanism called centralized broadcasting of modeling operations – that is, each modeling operation performed by a user is sent to a central feature model, and then the operation will be broadcasted to other users to make it visible. We provide two types of operations: creating operations (add a new element to the feature model) and voting operations (vote *yes* or *no* on any existing elements to express viewpoints on this element). Following this idea, we propose an extended meta-model of feature models to explicitly introduce these operations into feature models, and define a process to construct feature models in a collaborative way. We develop a tool to implement the concepts and the process, and conduct a series of case studies to explore strengths and weaknesses of CoFM. The observations made from the case studies show that CoFM improves the efficiency of feature modeling, and it is not only suitable for feature-oriented domain analysis, but also feature-based requirements elicitation (feature request).

The reminder of this paper is organized as follows. Section 2 introduces some preliminaries on traditional feature models. Section 3 and 4 present the conceptual framework and the modeling process of CoFM, respectively. A brief introduction of the implementation of CoFM is presented in Section 5. A series of case studies and observations from the cases are introduced in Section 6. Related work is discussed in Section 7. Finally, Section 8 concludes this paper with a brief summary and future work.

2. PRELIMINARIES: THE FEATURE MODEL

In this section, we introduce a meta-model of traditional feature models. It serves as the basis of the extended feature models constructed in CoFM..

Figure 1 shows the meta-model of traditional feature models. Generally, a feature model consists of a set of features and relationships between them. The concept of *feature* can be understood in two aspects: intension and extension [12]. In
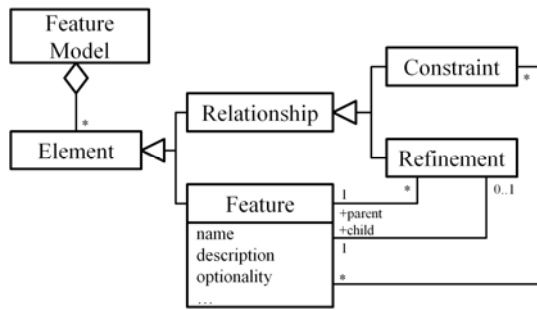
intension, a feature denotes a cohesive set of individual

Figure 1. A meta-model of feature models

requirements. In *extension*, a feature describes a software characteristic having sufficient user/customer value.

There are two types of relationships, namely *refinements* and *constraints*. The *refinement* relationships organize features with different levels of abstraction or granularities into a hierarchical structure. In this hierarchical structure, each feature is either *mandatory* or *optional* (i.e. the *optionality* of a feature). If a feature is selected, its mandatory children features must be selected as well, while its optional children can either be removed or selected.

The *constraint* relationships describe dependencies between features. There are two basic kinds of constraints: *requires* and *excludes* [7][8]. Given two features *A* and *B*, *A requires B* means that if *A* is selected, *B* must be selected as well. *A excludes B* means that at most one of them can be selected at the same time. More types of constraints can be found in [12].

3. THE CONCEPTUAL FRAMEWORK

In this section, we introduce a kind of extended feature models (EFMs) to explicitly support collaboration at the meta-model level. Based on a meta-model of EFMs, we further clarify two types of operations on EFMs, and introduce three types of views for EFMs.

3.1 The Meta-Model of Extended Feature Models (EFMs)

Figure 2 shows the meta-model of extended feature models. An EFM is composed of a set of *vote-able elements*. The top-level elements are *features* and *relationships*. Each feature relates to a set of second-level vote-able elements, that is, a feature has one vote-able *optionality* attribute, and may have one or more vote-able *names* and *descriptions*. The result of vote on each element is recorded as its *supporters* and *opponents;* both are a set of *stakeholders.*

This meta-model defines two basic kinds of modeling operations that can be used by stakeholders in a feature model's construction. That is, stakeholders can either *create* new elements in a feature model or *vote* on existing ones. Other kinds of modeling operation
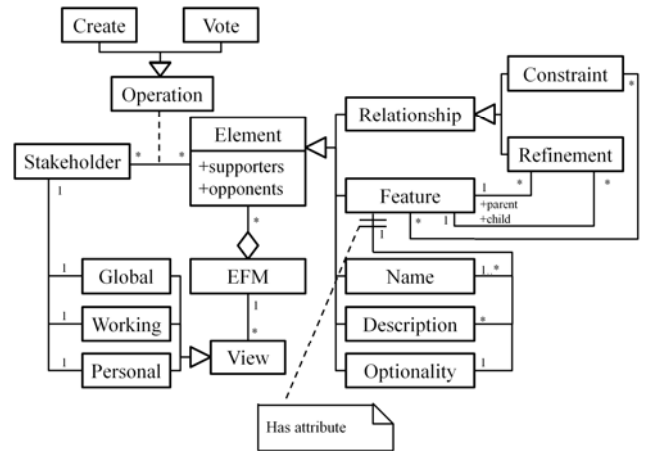
(such as *modification* or *deletion*) are implemented through the

Figure 2. The meta-model of EFMs

composition of the two basic kinds of operations. (See Section 3.2 for details.)

This meta-model also defines three types of *views* for EFMs, namely *global view, working view* and *personal view*. Each view contains specific information to serve different purpose. In a feature model's construction, each stakeholder relates to one global view, one working view and at least one personal view. These views are automatically generated from EFMs. (See Section 3.3 for details.)

3.2 Operations on EFMs
In this sub-section, we first introduce the two types of operations provided in CoFM, namely the creating operations and the voting operations. We then define a set of *automatic voting inference rules* to prevent inconsistent votes. Based on these rules, we discuss how to simulate other operations via the provided operations..

3.2.1 The creating and voting operations
The function of creating operations is to add new elements to a feature model. The creating operations in our approach are more general than in other feature modeling approaches [6][7]. For example, in our approach, a stakeholder can create several names for one feature, while in other approaches the stakeholder can only assign exactly one name to a feature.

There are restrictions on the creating operations to ensure that no duplicate elements are created (Table 1). Feature names are special because they are considered as identifiers of features, so no duplicate names are allowed within or between features. Different names of a feature are treated as aliases of each other. For other attributes, we only forbid duplicates within a feature, for example, the same description cannot be created twice for a feature. Relationships must be established among existing features and no duplicates are allowed in an EFM.

The function of voting operations is to allow stakeholders to express their viewpoints on elements in a feature model. For an element in an EFM, a *yes* vote means that the voter thinks this element is valid for the EFM, while a *no* vote shows the opposition. No restrictions are imposed on the voting operations; however, we introduce a mechanism called automatic voting inference to ensure the consistency of the votes (Section 3.2.2).

Table 1. Restrictions on the creating operations

Target	Restrictions
Name	The name has not appeared in any of the features in current EFM.
Description, Optionality	No duplicate has been created in current feature.
Feature	A name of the feature must be created simultaneously.
Relationship among Features $\{F_1, F_2, …, F_N\}$	The features $\{F_1, F_2, …, F_N\}$ must exist in current EFM and this relationship does not exist.

3.2.2 Automatic voting inference
Semantics of different votes from the same stakeholder may be inconsistent. For example, if there is a feature *A* and a constraint *A requires B,* what if a stakeholder votes *no* on the feature but

votes *yes* on the constraint? Based on the semantics of *yes* and *no* votes, the stakeholder supports a constraint that involves an invalid feature. In this case, there is an inconsistency between two votes from the same stakeholder.

Our solution to such problem is *automatic voting inference*. That is, whenever an operation is performed, inferred voting operations will be performed to related elements automatically. To define which the related elements are, we need to take a closer look at relations between the validity of different elements in an EFM. For clarity, we first define a predicate *Valid as*:

$$Valid\ (Element\ E) = \begin{cases} True, if\ E\ is\ valid\ in\ the\ EFM, \\ False, else. \end{cases}$$

Then the relations between the validity of different elements can be defined as follows:

$Valid\ (Relationship\ R\ involving\ Feature\ F) \rightarrow Valid\ (F),$ (1)

$Valid\ (Attribute\ A\ of\ Feature\ F) \rightarrow Valid\ (F).$ (2)

Beside, according to the semantics of votes, we can deduce that

$Yes\ (Element\ E) \leftrightarrow Valid\ (E),$ (3)

Where the predicate *Yes* is defined as,

$$Yes\ (Element\ E) = \begin{cases} True, if\ vote\ yes\ on\ E, \\ False, if\ vote\ no\ on\ E. \end{cases}$$

According to (1), (2) and (3), we can determine the inferred voting operations on related elements of an operation, for each type of element the operation being performed on, as defined in the following *voting inference rules* (*VIRs*).

VIR-1:

$$Yes\ (Relationship\ R) \rightarrow \bigwedge_{F\ involved\ in\ R} Yes\ (Feature\ F).$$

VIR-2:

$$\neg Yes\ (Feature\ F) \rightarrow \bigwedge_{R\ involves\ F} \neg Yes\ (Relationship\ R).$$

VIR-1 and *VIR-2* can be derived from each other. The two *VIRs* ensure the consistency between the votes on relationships and the votes on features involved in the relationships from the same stakeholder. For example, if a stakeholder votes *yes* on the *A requires B* constraint, then two *yes* votes will be performed automatically on feature *A* and *B,* respectively. If the stakeholder votes *no* on feature *A*, then a *no* vote will be performed on the *A requires B* constraint, while feature *B* will not be affected.

Similarly, the next *VIRs* define the voting propagation between features and their attributes.

VIR-3:

$Yes\ (Attribute\ A\ of\ Feature\ F) \rightarrow Yes\ (F),$

VIR-4:

$$\neg Yes\ (Feature\ F) \rightarrow \bigwedge_{A\ belongs\ to\ F} \neg Yes\ (Attribute\ A),$$

The four VIRs above clarify the consistent relations between voting operations. Furthermore, there are also such relations between the creating and voting operations. The semantics of the creating operations is to make currently non-existing elements exist in an EFM, while the voting operations express support or opposition of the validity of existing elements. Thus we can say that if a stakeholder creates an element, s/he must also support for its validity. Therefore we can derive the following *creating-imposed voting inference rule* (*VIR-C*).

VIR-C:

Create element E → Yes (E).

All VIRs are transitive; however no actual transition happens when performing a voting operation, according to VIR-1 to VIR-4. Transitions only take effect when performing creating operations. For example, when a stakeholder creates a relationship, VIR-C will be applied, which in turn, makes VIR-1 take effect. For a voting operation, however, it is possible to apply multiple VIRs for the operation. For example, VIR-2 and VIR-4 take effect when voting *no* on a feature.

In addition, we define an extra rule to deal with repetitive votes. Since we have not imposed any restriction on the voting operation, a stakeholder may vote on the same element for many times. Furthermore, multiple operations submitted by a stakeholder may cause several inferred votes on the same element as well. Thus it is highly possible that repetitive voting could happen. We handle this problem with the following rule.

Repetitive Vote Rule (RVR):

If there are multiple votes on an element from the same stakeholder, no matter these votes are submitted explicitly or are inferred, only the last vote counts.

With the help of the rules, we can ensure that the votes from a stakeholder are consistent. Furthermore, the rules allow us to simulate other modeling operations via the creating and voting operations, as described in the next sub-section.

3.2.3 Simulating other modeling operations

Most feature modeling approaches provide operations for *creating*, *deleting*, *modifying* and *moving* elements in feature models. In CoFM, by contrast, we purposely provide only one of these operations (the creating operation) plus the voting operation, to prevent stakeholders from editing and overwriting others' work directly, thus different opinions can be expressed in a feature model equally.

However, we do allow stakeholders to simulate the three kinds of non-provided operation, *if and only if they can reach a consensus*. Since modifying and moving operations are combination of creating and deleting operations, we will focus on simulating the deleting operation. The basic idea comes from the fact that deletion is opposite to creation. According to the discussion of *VIR-C*, if we apply *VIR-C* from the opposite direction, we get:

$$\neg Yes\ (Element\ E) \rightarrow Delete\ E. \qquad (4)$$

Equation (4) means that if a stakeholder votes *no* on an element, his intention is to oppose the validity of the element, which conforms to the semantics of deleting operations. Moreover, equation (4) works perfectly with the *VIR-2 and VIR-4*, that is,

when a feature is deleted, its attributes and involved relationships should also be deleted. This is a fairly reasonable result.

However, in collaborative work, there will be multiple voters on the same element. Therefore, we define that an element cannot be deleted, until all votes on it are *no*, which means consensus has to be achieved, as shown below:

Vote-Simulating Deletion:

Delete element E =$_{def}$ All votes on E are no.

One property of the simulated deletion is that every supporter of an element has the chance to express his opinion on the deletion of this element. Specifically, *VIR-3* ensures that each creator or supporter of any attribute of a feature must also be a supporter of the feature. Therefore, a contribution to any part of a feature gives the stakeholder a chance to express his/her opinion on the deletion of the feature. The benefit of the property is that it eliminates the risk of deliberate deletion (so-called *vandalism*).

Another property is that, the deletion of an element does not require all stakeholders vote *no* on the element, but only *all stakeholders who have voted on it*. In other words, it only needs consensus among those who are interested in the element. This property reduces unnecessary workload on stakeholders who have no interest in an element. Besides, it makes simulation of *undo* operations possible: if an element's creator is the only stakeholder works on it, the creator can undo his creation by voting *no* on it.

As long as we simulate the deleting operations, the modifying and moving operations are trivial. To modify an element, stakeholders delete it first, and then create a new one of the same type. To move a feature, stakeholders need to modify the refinements that involve the feature as a child.

3.3 Views for EFMs

The creating and voting operations introduced in the previous sub-section allow conflicting opinions to coexist in an EFM. In this sub-section, we discuss how to represent an EFM to stakeholders with the aim of helping them understand their current context of work. The basic idea is that a stakeholder's context of work should be built from his own viewpoints on an EFM. Besides, the whole picture of the EFM, and different stakeholders' understanding of the EFM should be provided as supplements.

Follow this ides, we provide three types of EFM views, namely *working view*, *global view* and *personal view*. For each EFM, every stakeholder has a working view as the main view, as well as a global view and a personal view as supplements. All the views are generated automatically.

3.3.1 Working view

Working views act as main workspaces of stakeholders. For a stakeholder, the corresponding working view is generated according to the following rule.

WV (EFM m, Stakeholder s) = {Element e | e ∈ m ∧ s has not voted NO on m}.

According to the rule, a working view of a stakeholder consists of elements on which the stakeholder vote *yes*, and elements created by others but the stakeholder has not shown any opinions on. Invalid elements in a stakeholder's point of view are eliminated

(by voting *no* on them) in the stakeholder's working view, and it may help the stakeholder focus on their work.

3.3.2 Global view
A global view reveals the whole picture of an EFM. It is composed of all undeleted elements in an EFM.

$$GV \ (EFM \ m, Any \ stakeholder) = \{Element \ e \mid e \in m \wedge At \ least$$
$$one \ YES \ vote \ on \ e\}.$$

The rule *VIR-C* introduced in previous section attaches an initial *yes* vote on each elements thus ensures that a global view can be generated correctly even if no stakeholders have explicitly voted *yes* on any element.

Stakeholders may switch to the global view to avoid information missing. If a stakeholder has voted *no* on an element, the element will be invisible in his working view since then. Therefore, he might miss useful information about the element without using the global view. For example, a stakeholder may vote *no* on a feature in the early stage of the construction. Since then, most stakeholders have expressed opposite opinions on the feature. This possibly means that the first stakeholder needs to reconsider his/her decision. By utilizing the global view frequently, the stakeholder would not miss information like this.

3.3.3 Personal view
A personal view is used to represent a specific stakeholder's own perspective of an EFM. This view consists of all the elements supported by a stakeholder.

$$PV \ (EFM \ m, Stakeholder \ s) = \{Element \ e \mid e \in m \wedge s \ has$$
$$voted \ YES \ on \ e\}.$$

Different personal views may show different perspectives and interests in a domain. For example, end users may concern more about the service-level features, while programmers might be interested in the implementation-level features and constraints between them. Thus it is possible that the personal views of end users depict services provided by the software in a domain, and the personal views of programmers show more details about the techniques used in the domain.

4. THE PROCESS
In this section, we introduce the process of collaborative feature modeling, based on the conceptual framework discussed before. We first give an overview of the process, and then present the conflict resolution and the coordination mechanisms in detail, respectively. Finally, we give some guidelines for stakeholders to carry out the process.

4.1 An Overview of the Process
Figure 3 shows the process of collaborative construction of an EFM. In this process, there are two kinds of activities, namely *stakeholder activities* (performed by stakeholders) and *supporting activities* (performed automatically), as shown in Table 2.

According to the process, a stakeholder goes through the following steps iteratively.

Table 2. Main activities in CoFM

| Activity	Description
Stakeholder Activities	
Submit Operations	Submit creating and voting operations.
Resolve Conflicts	Resolve conflicts in their working views and personal views.
Communicate	Communicate with other stakeholders via comments and/or discussion pages.
Supporting Activities	
Infer Votes	Calculate inferred votes after operations are sent to the central EFM.
Coordinate and Apply Changes	Coordinate changes made by stakeholders, and then apply them to the EFM.
Update Views	Update global, working and personal views.
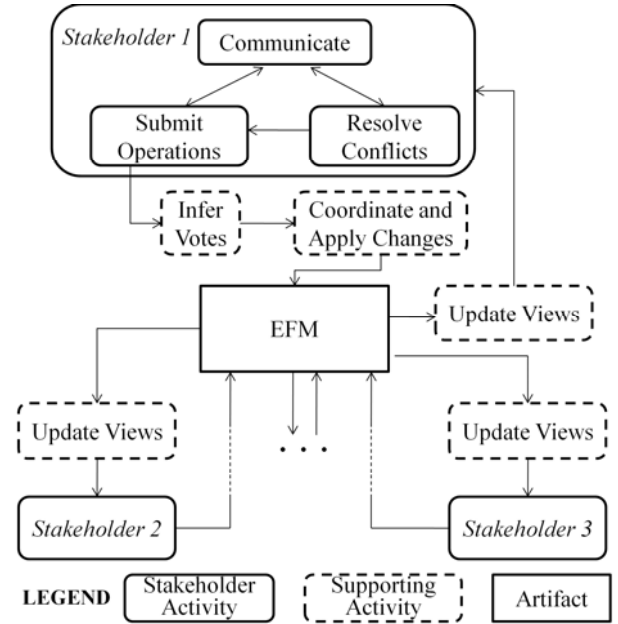
Figure 3. The process of collaborative feature modeling

Update Views: When stakeholders start to work, views are automatically generated for them. Whenever a stakeholder has successfully updated the central EFM, all stakeholders' views are updated as well. Particularly, update of the working view and the personal view includes a detection of conflicts.

Resolve Conflicts: Conflicts in a stakeholders working and personal views indicate unfinished work for him/her. The stakeholder can resolve these conflicts via the creating and voting operations, and they need to communicate with others during the resolution (Section 4.2).

Submitting Operations: Whenever a stakeholder performs an operation, the operation (without inferred votes) is submitted to the central EFM immediately.

Infer Votes: Inferred votes are computed after an operation has submitted. Then the original operation and inferred votes are ready to update the EFM.

Coordinate and Apply Changes: In the context of collaborative work, changes submitted by multiple stakeholders need to be coordinated. If the original operation and its inferred votes are still valid after the coordination, they will be applied to the EFM and in turn, broadcast to all stakeholders (including the submitter). Then views of all stakeholders will be updated (Section 4.3).

4.2 Conflict Resolution

It has been observed that conflicts are common in collaborative work, because of divergence between stakeholders. In CoFM, only the divergence in the working-view-level and the personal-view-level, not the EFM-level, can lead to conflicts. In other words, we don't force stakeholders to eliminate any divergence in an EFM; however, a stakeholder's working view is his/her individual workspace and the personal view is his/her perspective of the feature model, therefore his work is considered unfinished if there are still conflicts in them.

There are four types of conflict in a stakeholder's working view.

Unidentifiable Feature (UF): the stakeholder denies all existing names of a feature without giving a new one, which makes the feature unidentifiable in his working/personal view.

Non-positioned Feature (NPF): the stakeholder denies all existing positions of a feature without giving a new one. (A feature must either be positioned as a root feature, or as a child of another feature.)
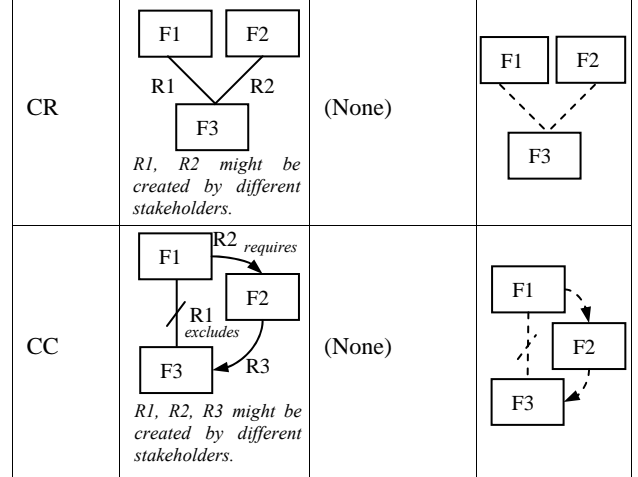
Conflicting Refinements (CR): multiple refinements existing in a working/personal view are considered conflicting if they involve the same feature as the child but different features as the parent.

Conflicting Constraints (CC): multiple constraints existing in a working/personal view are considered conflicting if there are two involved features *A* and *B,* where both *A excludes B* and *A requires B* can be deduced from these constraints.

Table 3 gives some examples of conflicts in a stakeholder's working view.

Table 3. Examples of conflicts in a working/personal view

Type	The Stakeholder's		
	Initial View	Operations	Resulting View
UF	*(feature)* F Aliases: $A_1..A_n$	Vote *no* on the name *F, $A_1, .., A_n$*	*(Unnamed)*
NPF (1)	F1 *(Root)* R: F2 refines F1 F2	Vote *no* on R	F1 *(Root)* F2
NPF (2)	F1 *(Root)* F2	Vote *no* on *F1*	F2

CR	F1 F2 R1 R2 F3 *R1, R2 might be created by different stakeholders.*	(None)	F1 F2 F3
CC	F1 R2 *requires* R1 *excludes* F2 F3 R3 *R1, R2, R3 might be created by different stakeholders.*	(None)	F1 F2 F3

For an unidentifiable feature, the stakeholder should either create a suitable name, or reconsider existing names.

For a non-positioned feature, the stakeholder should create a refinement involving it as a child, or make it a root feature explicitly, or reconsider existing refinements.

For conflicting refinements, the stakeholder should vote on these refinements to accept exactly one of them, or even denies all and create a new one.

For conflicting constraints, the stakeholder should vote no on some of them until the conflict has resolved, or even denies all.

The conflicts in a working view actually define the minimal amount of work that a stakeholder must do to obtain a usable feature model that conforms to traditional feature models. For example, a newly joined stakeholder only needs to resolve conflicting refinements and conflicting constraints to transform his working view into a usable feature model, which has a traditional hierarchical structure. In addition, there is a bonus for him: aliases and descriptions of features contributed by different stakeholders can help him deepen the understanding of the current domain.

4.3 Coordination

In the context of collaborative work, when multiple stakeholders are working on the same set of elements at the same time, their changes of the elements may need to be coordinated. The need for coordination is rooted in the *update delay problem*: update of the EFM originated by one stakeholder cannot immediately become visible to others. According to types of the update (*create* or *vote*), there are four possible situations, as illustrated in Figure 4.

Duplicate creation happens when a stakeholder (S2) creates an element *E* before a previous creation of the same element has became visible (i.e. *update EFM*) to him. Therefore the creation of the same element happens twice.

Conflicting aliases emerge when two stakeholders try to create the same alias on different features at the same time. Since different features cannot have the same name or alias, the situation needs to be coordinated.

Unreachable vote is a vote on a nonexistent element. In Figure 4, if the *no* vote on *E* submitted by *S1* leads to the deletion of *E*, then *S2*'s *yes* vote is an unreachable vote.
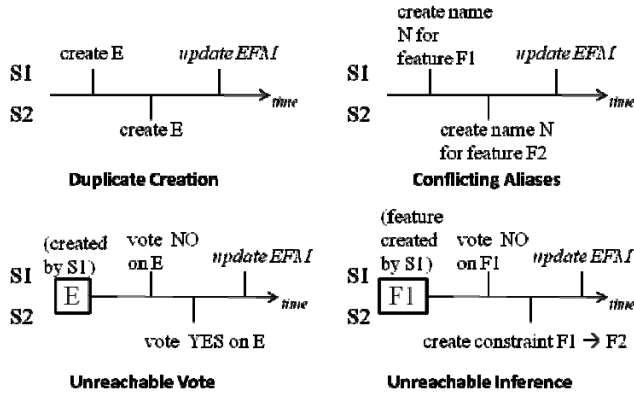
Figure 4. Examples of situations needing coordination

Unreachable inference is similar to unreachable votes. In Figure 4, if the *no* vote on *F1* leads to the deletion of *F1*, and if *S2* creates a constraint involving *F1* before the deletion becomes visible to him, then the inference of *yes* vote on F1 (according to the rule *VIR-1*) is unreachable.

Coordination of these situations follows a *serialized update strategy*, that is, all update applies to the EFM in the same order of their submission. Furthermore, if the update being applied is no longer valid, it fails and takes no effect on the EFM, and its submitter will be informed the failure.

For duplicate creations, the first creating operation adds a new element to the EFM, and the latter is converted to a *yes* vote. In Figure 4, *S1* creates the element *E*, and *S2* has a *yes* vote on *E*.

For conflicting aliases, the first alias is kept, and the latter one is neglected. In Figure 4, only *F1* will be assigned the alias *N,* and *S2* will be informed his creating operation has failed.

For unreachable votes, they are no longer valid on a nonexistent element and are neglected. In Figure 4, the element *E* will be deleted, and *S2* will be informed that his/her voting operation has failed.

For unreachable inferences, they are neglected as well as the operations which cause the inference. In Figure 4, the feature *F1* will be deleted, and *S2* will be informed that his/her creating operation has failed.
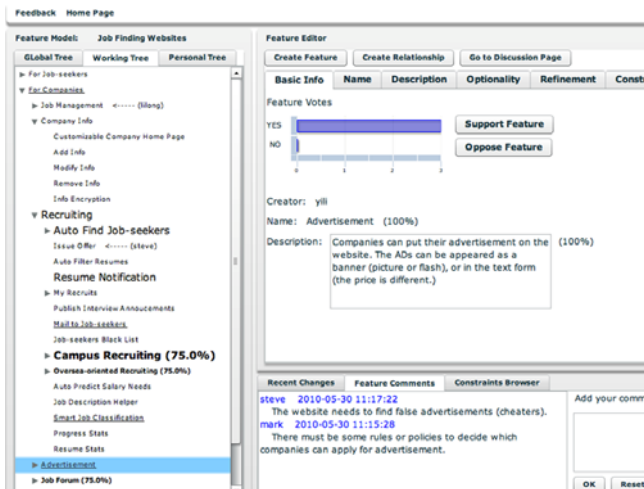

Figure 5. Screenshot of the CoFM tool

5. THE IMPLEMENTATION

We have developed a system (with a typical C/S architecture) to implement all the concepts and processes presented in previous sections. After the first version of the system has been developed, we conduct a few case studies. According to feedbacks collected in these case studies, we develop the second version of the system, in which most existing functions are improved and new functions are added. We use the second version of the system for the rest cases.

Figure 5 is a screenshot of the modeling page of our current system. The feature browser on the left side displays the EFM as global, working and personal views. Details of each feature are shown in the right-top panel. Other information such as the constraints, comments and recent changes can be viewed in the right-bottom panels.

Because our system is a kind of CSCW (Computer Supported Cooperative Work) systems, we developed functions to implement two of the central concepts of CSCW systems: awareness and communication [4][5].

Awareness means that a CSCW system should provide necessary information about activities of other people in the shared workspace, which provides the context for individual's work [4]. We provide four kinds of awareness information. In the use of our system, we find that the most frequently used UI region is the feature browser, so we decide to display all these information in the feature browser, with different UI elements.

- *Editing locations of other people*. From this information, users can know who is working on the feature model currently and which feature the others are focusing on. (People's names are marked next to the being edited features.)

- *Controversies in current feature model*. We use bolded fonts to indicate controversial features (controversies in a feature or its attributes,), and we use different colors for each type of conflicts described in Table 3. Controversial and conflicting constraints are highlighted in the constraints-overview.

- *Authorship*. Features created by current user are underlined in the tree views.

- *Unread recent changes*. The font size of features containing unread recent changes is larger than others. Besides, users can view the full change history in a specific panel.

The communication support includes comments for each feature and a discussion page for each feature model.

6. CASE STUDIES

6.1 Overview of the cases

The cases conducted in CoFM can be divided into two categories: domain analysis cases and requirements elicitation cases. Table 4 shows the basic information about these cases.

In the domain analysis cases, five or six participants construct a domain feature model for a familiar domain collaboratively. They spend a couple of hours working simultaneously at first, and then

they may work freely (at any time they like, and often at different time) in the next few days.

In the requirements elicitation cases, CoFM is used as an environment for organizing and discussing requirements of ongoing projects. The requirements are expressed in terms of requested features, which are common in real world's open source projects [10]. In the beginning, members of a project create a few top-level features, which serve as feature/requirements categories. In the next few weeks, stakeholders create new features as their requirements, followed by further refinement and discussion of details about the requirements.

The participants of these cases have already learned the basic concepts of feature models and have some knowledge of requirements engineering, and all of them have educational background in computer science, so their learning curves of CoFM are very short. However, some of them have little or no experience on domain analysis and requirements elicitation (they can be perceived as stakeholders like *users* or *clients*), and some of them are more experienced (they can be perceived as *domain analysts* or *requirements analysts*). We think the knowledge structure of our participants and real world projects' stakeholders are similar.

Table 4. Basic information about the cases

Name	Description	Number of Features	Number of Participants
Domain Analysis Cases			
Music Player	The domain feature model of music playing software such as iTunes.	158	5 (including 1 experienced)
Job Finding Website	This domain feature model is based on the top 3 job finding websites in China.	113	6 (including 2 experienced)
Requirements Elicitation Cases			
CoFM	Feature request and feedback for CoFM itself.	32	6 (including 1 project member)
RFTM	These are tools of other ongoing projects in our lab.	32	4 (including 1 project member)
CRE		40	8 (including 2 project members)

6.2 Observations
In this sub-section, we wish to present some observations of the uses, strengths and weaknesses of CoFM.

6.2.1 Efficiency of feature modeling is improved
One observation is that the efficiency of feature modeling is improved. There are three reasons for such improvement.

The first reason is the parallelism of individual work during the construction of feature models. We have observed that the participants always focus on a certain part of a feature model (i.e. a certain sub-tree in the tree view) during a period of time. With the display of others' editing location, the participants tend to choose a different part from others. This leads to parallel creating and voting operations in different parts of a feature model. Especially, this phenomenon can be most clearly observed in the early stage of feature modeling, when participants are focusing on creating descendant features for high-level features.

The second reason is that in most situations, adjusting relationships among a group of features does not disturb the work on these features, and vice versa. For example, in the Job Finding Website Domain case, there is a period of time when three different parent features are linked to a feature *Browse Company Info* (i.e. three conflicting refinements). When two participants are discussing and resolving these conflicting refinements, another participant continues his work on writing description and creating child features for *Browse Company Info* without any trouble. This often happens in the later stage of feature modeling, when there exist a lot of earlier created features, and one or two participants (often the most experienced ones) are focusing on adjusting refinements and adding constraints, while others are trying to improve names and descriptions of existing features and find missing features.

The third reason is that participants are often inspired by others' work. All participants have admitted that "by looking at what is being created by the others, I can always think of a related feature immediately." For example, just a few seconds after a participant creates the feature *Browse Company Info*, another participant creates two features –*Search Company* and *Publish Company Info*. Furthermore, most participants agree that understanding a proper named and described feature is easier than creating a new one. They said, "Although I have to understand others' work, it is still easier to review an existing element than to think of a new one by myself, therefore my efficiency is improved." Therefore, individuals can work more efficiently in the collaboration.

6.2.2 Support for awareness is critical
We believe that support for awareness is crucial to the collaborative work. The participants have reported that the display of others' editing location is the most useful function for their work, because they can find which part of the feature model is idle and needs attention, thus they can try their best to work in parallel.

The awareness of controversial elements is also useful for participants' work. With the highlighting of controversial elements, participants can quickly find and resolve divergences that might cause potential problems in further collaboration, for example, conflicting refinements of a top-level feature.

In addition, participants have positive comments for awareness information about "their own creations" and "recent changes", in terms of improving their efficiency. Highlighting their own creations helps them find if there are any controversies in these elements, and it is impossible for them to remember the creator of every element. Highlighting recently changed elements (especially features) helps them from two aspects. First, they always interest in the most recently changes in a feature model, especially when they leave the feature model for a long period of time. In the contrast, some participants also want to know the least recently changed features, which may indicate a problematic feature, such as a feature with a vague name and no descriptions thus no one can understand it.

6.2.3 Feature models are useful for feature-based requirements elicitation (feature request)

Because feature models provide an effective way to organize requirements, we find that CoFM is not only suitable for domain analysis, but also useful for feature request. Feature request via online forums is one of the most common forms of requirements elicitation in the real world's open source projects. However, feature request in many projects lacks organization – requests of the same feature are scattered over many topics in the forum, which has negative impact on the efficiency of elicitation [10].

The situation has improved when we use CoFM as the platform of feature request. In CoFM, the discussions about a feature request are well organized. The requested feature is put into a proper location in the feature model, so its relation with other functions (features) can be clearly viewed through refinements and constraints. Then stakeholders discuss details about the feature by editing its descriptions and adding comments. In many cases, stakeholders (usually the developers) refine the requested feature to many descendant features. Specifically, we have observed a pattern in confirming certain requirements of the feature, that is, the developers create a set of child features as choices, and then the requester votes yes or no on the child features as the answer. For example, in our use of CoFM as feature request platform for itself, a user requests a feature *Export the Feature Model as File*, and then the developer creates child features relating to the file format such as *TXT, RTF, XML,* and *HTML*, and the user votes on them to confirm his requirements. By using CoFM, feature request is performed in a clear and structured way.

6.2.4 The need of customizable attributes of features

One of the participants' complaints about the first version of CoFM is that the expressive power of feature models. For example, when using CoFM for feature request, participants want to express the priority, importance and ease of implementation of the requested features, and they can only write them in features' descriptions. As a result, they asked for adding such attributes to the features in the feature model. It makes us realize that we need to allow users of CoFM adding attributes to features in a certain feature model during the construction of the feature model. We design a mechanism called *customizable attributes of features*. All features have default attributes, i.e. *name, description* and *optionality*. Besides, users can add attributes to all features in a certain feature model, in one of the predefined types: single-line *string*, multi-lines *text*, *enumeration*, and *number*. (We may define more types in the future.) They can create new values or vote on existing values of any attribute in a feature as before. We have observed that every feature model constructed in the new version of CoFM, no matter used for domain analysis or feature request, have a few customized attributes. This shows the ability of customization and the flexibility brought by it has very positive influence on the use of CoFM.

6.2.5 The need of outlining

When using CoFM for domain analysis, we have found that if an experienced domain analyst has created a few top-level features in advance, then there will be fewer conflicts in further construction, and therefore it will be more efficient. We call it the *outlining* of feature modeling, because top-level features can be considered as categories of features. We observe this by comparing the construction of Music Player Domain and Job Finding Website Domain feature models.

In the case of Music Player Domain, the initial 30 or 40 features are created by a participant with no experience in domain analysis. We found that the structure of these features can be almost directly mapped onto the structure of menu items of the participant's favorite music playing software. For example, the top-level features are *Play Control*, *Media Library*, *Audio Control* and *Lyrics*, which are the same of top-level menu items. However, this kind of structure may be helpful for using the software, but is not proper for a feature model. For example, *Play Control* refers to the basic and most frequently used features in the software (with a play control panel on its UI), so it has child features like *Play Local Files* and *Play from a URL*. However, from a domain analyst's perspective, these two features may be renamed to *Local File* and *Online Audio Stream*, and they should be child features of a top-level feature named *Formats to Support*. Such structural mismatch between the software and the feature model causes a lot of conflicts in refinement relationships when more experienced participants join the collaboration. In fact, the participants decide to stop creating new features unless they can reach a consensus on the top one or two levels of features and put the rest of initial features in right position.

With the experience of the Music Player Domain case, we decide to ask a more experienced participant to create the top one or two levels of features as a start, for the Job Finding Website Domain feature model. The participant spends about an hour on analyzing three most popular job finding websites, and then creates the top two levels of features in five minutes. The top-level feature *For Job Seekers* has child features like *Personal Info, Search for Jobs* and *Resume*. Another top-level feature *For Companies* has child features like *Company Info, Job Management, Search for Job Seekers, Interview Management* and *Advertising*. As a result, there are few conflicts of refinement relationships during the further construction, and feedback shows that the initial outline-features help participants find new features in a more systematic way.

According to the observations above, we believe that the outlining, especially performed by experienced domain analysts, can stimulate thinking and improve the efficiency of collaborative feature modeling.

6.2.6 A short summary

The observations made from the case studies show that feature models are suitable for being constructed in collaborative way described in this paper. With adequate awareness and communication support, the efficiency of feature models' construction can be increased. In addition, experienced domain analysts always act as a key role in the construction of feature models. Furthermore, we have found that feature models are powerful tools for feature request, which is the most common form of requirements elicitation in real world's open source projects.

7. RELATED WORK

In FODA [7], FORM [8], FeatuRSEB [6], and FOPLE [9], feature models are constructed through the commonality and variability analysis to a set of applications in a domain. The construction task is performed by domain analysts, with the assistance of other

stakeholders such as users, domain experts and requirements analysts. However, despite its inherent need of intensive collaboration, none of these approaches has explicitly incorporated mechanisms or techniques to support collaboration among stakeholders in the construction of feature models.

Many generic collaboration tools allow users to express their viewpoints through a voting mechanism which is similar to ours. For example, the *gIBIS* tool [3] proposed a framework for people to solve complex problems in a structured way, that is, solve a problem by identifying *issues* (what the problem is), submitting *positions* (possible solutions to issues), and providing *arguments* (reasons to support or object to positions). However, although these tools allow different or even conflicting opinions to be expressed, they can only tolerate temporary divergences; all divergences must be eliminated in the end of work. By contrast, feature models used in CoFM can tolerate permanent divergences; the divergences are treated as indicators of domain variability. Furthermore, in these tools, conflicts between stakeholders need to be manually identified, while in feature models, conflicts are well-defined and can be automatically detected.

In the research of requirements engineering, several methods have been proposed to support collaboration among stakeholders. The EasyWinWin system [1] provides an environment for stakeholders to propose and discuss requirements. The first step in EasyWinWin is to establish categories of the requirements as an outline, and it serves as a basis for later steps, e.g. a completeness checklist for final requirements. The observations from CoFM show that the outlining is also a key step for feature modeling, and it can be seamlessly integrated into current feature modeling process – all the users have to do is to construct the top-level features first, and these features may involve with the whole feature model during the later construction.

Noor et al. [11] proposed a collaborative product line planning (CoPLP) approach, in which the desired products are discussed in terms of features and the process of collaboration is similar to EasyWinWin. However, CoPLP did not use feature models to express and organize the features, therefore there is a lack of explicit modeling of constraints between the features and verification of the products. CoFM can also be used as a platform for feature-based requirements elicitation, and constraints between the features can be explicitly expressed.

In addition, Laurent and Cleland-Huang [10] have pointed out that feature request is a common form of requirements elicitation in open source projects, and a major problem is that the requested features are often lack organization. In their research [2], data mining technologies are incorporated to help organize the request and discussion of features. In CoFM, feature models can also guide the feature request towards a clear and structured process.

8. CONCLUSIONS AND FUTURE WORK

This paper presents a collaborative feature modeling environment (CoFM) to improve the efficiency of feature models' construction.

In this environment, stakeholders can create new elements and vote *yes* or *no* on existing elements to express their opinions on the elements. Following this idea, we extend the meta-model of traditional feature models by introducing *creating* and *voting* operations, as well as three types of views for stakeholders. Besides, we define a process for constructing feature models in a collaborative way. We develop a tool and conduct a series of case studies to explore strengths and weaknesses of CoFM. The observations show that CoFM improves the efficiency of feature modeling, and it is not only suitable for feature-oriented domain analysis, but also feature-based requirements elicitation.

In the future, we want to conduct more case studies with larger scale and more people, and improve CoFM according to the feedback. In addition, we plan to incorporate more techniques into our feature modeling approach, such as algorithms for calculating confidence or weights of users' operations, and mechanisms for helping users find relationships between features.

9. REFERENCES

[1] Boehm, B., Grunbacher, P., Briggs, R. O. 2001. Developing groupware for requirements negotiation: Lessons learned. *IEEE Software*. 18, 3, 2001, 46-55.

[2] Castro-Herrera, C., Cleland-Huang, J., Mobasher, B. Enhancing stakeholder profiles to improve recommendations in online requirements elicitation. In *Proceedings of Intl. Conf. on Requirements Engineering*. RE '09. 37-46.

[3] Conklin, J., Begeman, M. gIBIS: A hypertext tool for exploratory policy discussion. *ACM Trans. On Information Systems (TOIS)*. 6, 1988, 303-331.

[4] Dourish, P., Bellotti, V. Awareness and coordination in shared workspaces. In *Proc. of the 1992 ACM Conf. on CSCW*. 107-114.

[5] Ellis, C. A., Gibbs, S. J. Rein, G. Groupware: Some Issues and Experiences. *Communications of ACM*, 34, 1991, 39-58.

[6] Griss, M. L., Favaro, J., d'Alessandro, M. Integrating Feature Modeling with the RSEB. In *Proc. Of the Fifth Intl. Conf. on Software Reuse.* ICSR '98. 76-85.

[7] Kang, K. C., Cohen, C., Hess, J., Nowak, W., Peterson, S. *Feature-Oriented Domain Analysis (FODA) Feasibility Study.* Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.

[8] Kang, K. C., Kim, S., Lee, J., Kim, K., Kim, G. J., Shin, E. FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. *Annals of Software Engineering*. 5, 1998, 143-168.

[9] Kang, K. C., Lee, J., Donohoe, P. Feature-Oriented Product Line Engineering. *IEEE Software*. 19, 4, 2002, 58-65. DOI= 10.1109/MS.2002.1020288.

[10] Laurent, P., Cleland-Huang, J. Lessons Learned from Open Source Projects for Facilitating Online Requirements Processes. In *Proc. Of Working Conf. on RE: Foundations for Software Quality.* REFSQ '09.

[11] Noor, M. A., Rabiser, R., Grunbacher, P. Agile product line planning: A collaborative approach and a case study. J. Syst. Software. 2007. DOI=10.1016/j.jss.2007.10.028

[12] Zhang, W., Mei, H., Zhao, H. Y. A feature-oriented approach to modeling requirements dependencies. In *Proc. of the 13th IEEE Intl. Conf. on Requirments Engineering.* RE '05. 273-282