Collaborative Feature Modeling: A Voting Based Approach with Divergence Tolerance and Consensus Facilitation

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

Xin Zhou
China Research Laboratory
IBM
Beijing, China
zhouxin@cn.ibm.com

***Abstract*—Feature models provide an effective way to organize and reuse requirements in specific software domains. Many feature modeling 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 to the quality of constructed feature models. However, few of these approaches provide explicit mechanisms or methods to support such collaboration. In this paper, we propose a collaborative feature modeling approach, an approach that aims to explicitly support collaboration among stakeholders. The basic idea of this approach is to introduce the voting operations in a feature model's construction - that is, besides adding new elements to a feature model as normal, stakeholders can now vote *yes* or *no* on any existing element in the feature model to express their viewpoints to this element. The main characteristic of this approach is that it not only tolerates conflicting viewpoints from different stakeholders, but also facilitates the emergence of consensus among them. A case study on the video playing software domain is introduced to show the feasibility of our approach.**

Keywords-feature model; collaboration; domain analysis

I. 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 [13]. 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 modeling 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 to the quality of constructed feature models. For example, in FODA [13] and FORM [14] 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 [10], 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 existing feature modeling approaches provide explicit mechanisms or methods to support 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 quality of constructed feature models cannot be well controlled and guaranteed. Furthermore, even though domain analysts use general collaboration tools (such as gIBIS [3] or Wikipedia) to enhance collaboration among stakeholders, the effectiveness of collaboration will still be hindered by the concept-gap between feature models and these general tools. On the one hand, stakeholders have to share knowledge and resolve conflicts through general concepts in these tools; on the other hand, they need to transform acquired knowledge and feedback into particular concepts in 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 propose a collaborative feature modeling approach, which aims to explicitly support collaboration among stakeholders. The basic idea of this approach is to introduce the voting operation in a feature

model's construction - that is, besides adding new elements to a feature model as normal, stakeholders can now vote *yes* or *no* on any existing element in the feature model to express their viewpoints to this element. Following this idea, we propose an extended meta-model of feature models to explicitly introduce the voting operation into feature models, and define a process and a set of guidelines to construct feature models in a collaborative way. A case study on the video playing software domain is conducted to verify the feasibility of this approach. The results of this case study show that our approach provides a simple but effective way to tolerate conflicting viewpoints from different stakeholders as well as facilitating emergence of consensus among them.

The reminder of this paper is organized as follows. Section II introduces some preliminaries of our approach. Section III and IV present the static and dynamic aspects (i.e. the conceptual framework and the modeling process) of our approach, respectively. A case study is introduced in Section V to show the feasibility of our approach, and related work is discussed in Section VI. Finally, Section VII concludes this paper with a brief summary and future work.

II. PRELIMINARIES

In this section, we introduce three core concepts of CSCW systems, and a meta-model of traditional feature models. They serve as bases of the *collaborative* and the *feature modeling* aspect in our approach, respectively.

A. Core Concepts of CSCW Systems

The field of Computer Supported Cooperative Work (CSCW) was initiated in the mid-80s with the purpose of studying how technology could support group work [11]. Over the years, researchers [6][8][9][17][19] have identified several central concepts. Although technologies have changed tremendously, these concepts still shape the design of modern CSCW systems. Among these concepts, the most important three are production, awareness, and coordination.

Production refers to *objects* produced and shared in a CSCW system, and *operations* applied on these objects [17]. It can be represented as the conceptual framework [9] of a CSCW system. The implementation of this concept in our approach will be described in Section III.A and III.B.

Awareness means that every CSCW system should provide necessary information to show the context of individual's work [6]. For example, a collaborative writing tool should display the position of coauthors' edit cursor, to help authors identify their work context, i.e. their editing boundaries. The support for awareness in our approach will be described in Section III.C.

Coordination relates to mediating and meshing individual work in a shared workspace [19]. While *production* defines activities available to collaborators, the term *coordination* focuses on the organization of these activities between them [9] in order to reduce possibilities of conflict or repetition. The coordination mechanism in our approach will be explained in Section IV.C.

B. A Meta-Model of Feature Models

Fig. 1 shows a meta-model of traditional feature models. Generally, a feature model consists of a set of features and relationships between them. There are two types of relationships, namely *refinements* and *constraints*.

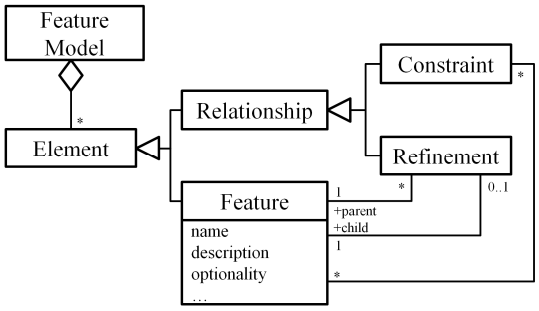

Figure 1. A meta-model of feature models.

The concept of *feature* can be understood in two aspects: intension and extension [21]. In *intension*, a feature denotes a cohesive set of individual requirements. In *extension*, a feature describes a software characteristic having sufficient user/customer value.

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* attribute 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* [13][21]. 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 [21].

III. THE CONCEPTUAL FRAMEWORK

In this section, we present 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 then introduce three types of views for EFMs.

A. The Meta-Model of Extended Feature Models (EFMs)

Fig. 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 *supporter* and *opponent;* 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 composition of the two basic kinds of operations. (See Section III.B 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 III.C for details.)

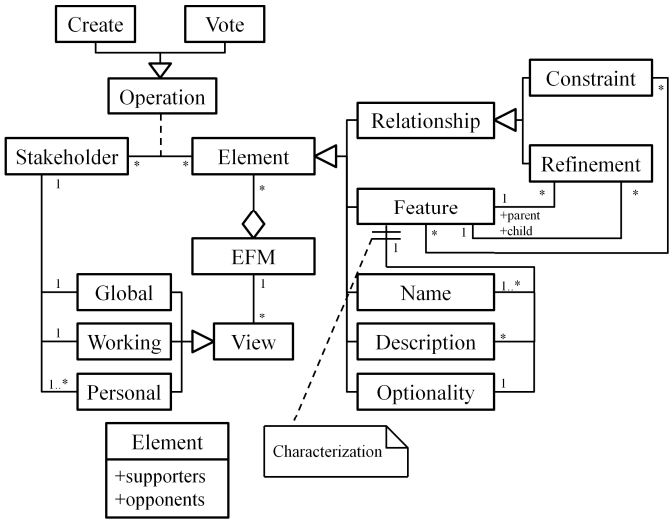

Figure 2. The meta-model of extended feature models.

B. Operations on EFMs

In a feature model's construction, we provide two basic kinds of modeling operation, namely *creating* and *voting*. Based on the two kinds of operation, we further define a set of *restrictions* and *automatic voting propagation rules* to prevent invalid or incorrect operations.

TABLE I. RESTRICTIONS ON THE CREATING OPERATIONS

Operation Target	Restrictions
Name	When creating a name for a feature, the name should not have appeared in any of the features in the current EFM.
Description	When creating a description for a feature, the same description should not have appeared in the current feature.
Feature	When creating a feature, a name for the feature should also be created simultaneously.
Refinement (*X refines Y*)	When creating an *X refines Y* refinement, the *X* and *Y* are two existing features in current EFM and this refinement does not exist.
Binary Constraint (*X requires Y*, *X excludes Y*)	When creating an *X requires Y* or an *X excludes Y* constraint, the *X* and *Y* are two existing features in current EFM and this constraint does not exist.
Optionality	The optionality attribute has predefined values (either *mandatory* or *optional*), and only the voting operations can be applied to it.

1) The creating operations and restrictions on them

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 [10][13][21]. 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.

Restrictions on the creating operations are to ensure that no duplicate elements are created. The specific restrictions on each kind of creating operation are described in Table I. In EFMs, a feature may have several names proposed by one or more stakeholders. We treat the different names of a feature as aliases of each other, and no duplicate names are allowed in or between features. Unlike names, descriptions are not identifiers of features, so we only forbid duplicate descriptions in the same feature. For relationships, we only list refinements and binary constraints here for simplicity. Restrictions on group and complex constraints defined in [21] are similar. The optionality attribute of a feature has two predefined values: mandatory or optional. "Create a new optionality" is meaningless and therefore disallowed.

2) The voting operations and automatic voting propagation

The function of voting operations is to allow stakeholders to express their viewpoints on elements in a feature model. The semantics of different voting operations is described as follows:

- For an element in an EFM (except for the optionality elements), a *yes* vote means that the voter thinks this element is valid for the EFM, while a *no* vote shows the opposition.
- For an optionality element in an EFM, a *yes* vote means that the voter thinks the related feature is an *optional* feature, while a *no* vote means *mandatory*.

There are no restrictions on the voting operations. However, 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 votes, the stakeholder supports a constraint that involves an invalid feature. In this case, there is an inconsistency between the votes from the stakeholder.

Our solution to such kind of problem is called *automatic voting propagation*. That is, whenever a voting operation is submitted, deduced voting operations will be propagated to related elements automatically. To decide the related elements of an element being voted, we take a closer look at the relations between validity of 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 validity of elements can be defined as follows:

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

and

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

According to the semantics of votes, we can deduce the following property (we will discuss *optionality* later),

$$Yes(Element\ E\ not\ Optionality) \leftrightarrow Valid(E), \quad (3)$$

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 decide the deduced voting operations of a submitted voting operation for each type of element being voted, as defined in the following *automatic voting propagation rules* (*PRs*).

- *PR-1:*

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

- *PR-2:*

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

PR-1 and *PR-2* can be easily derived from each other. The two *PRs* 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 propagated automatically to feature *A* and *B,* respectively. If the stakeholder votes *no* on feature *A*, then a *no* vote will be propagated to the *A requires B* constraint, while feature *B* will not be affected.

Similarly, the next *PRs* define the voting propagation between features and their names/descriptions.

- *PR-3:*
  Yes(Attribute A of Feature F) →Yes(F),
- *PR-4:*

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

where A is a Name or Description.

Now we deal with the voting propagation involving the *optionality* attribute. According to previous discussions, voting on optionality is to decide whether the related feature is mandatory or not, while voting on a feature is to decide whether the feature is valid or not. Based on the semantics above, we can deduce that *a feature must be valid before its optionality can be considered*. Therefore, we define the following *PRs:*

- *PR-5:*
  Vote on the optionality of feature F→Yes(F).
- *PR-6:*
  ¬Yes(Feature F) → Remove the voter's vote on the optionality of F.

The six *PRs* above clarify the consistent relations between votes. 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. (Since no creation is allowed for the optionality attribute, we omit it here.) 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 propagation rule* (*PR-CV*).

- *PR-CV:*
  Create element E → Yes(E).

All *PRs* are transitive, but transitions only take effect on the creating operations. That is, according to the definition of *PR-1* to *PR-6*, no actual transition happens when submitting a voting operation. For example, when a stakeholder creates a relationship, *PR-CV* will be applied, which in turn, makes *PR-1* take effect. For a voting operation, however, it is also possible to apply multiple *PRs* to it. For example, PR-2, 4 and 6 take effect when submitting a *no* vote on a feature.

In addition to the seven *PRs* above, 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, operations submitted by the stakeholder may cause several propagated votes on the same element as well. Thus it is highly possible that repetitive voting could happen. We handle this kind of 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 propagated, only the last vote counts.

With the help of the eight rules, stakeholders can simulate other modeling operations via the creating and voting operations, as described in the next sub-section.

3) Simulating other modeling operations

Most feature modeling approaches provide operations for *creating*, *deleting*, *modifying* and *moving* elements in feature models. As mentioned before, we purposely provide only one of these four kinds of operation (the creating operations) plus the voting operations, 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 consensus*. Since modifying and moving operations are combination of creating and deleting operations, we focus on how to simulate the deleting operations first. The basic idea comes from the fact that deletion is opposite to creation. According to the discussion of *PR-CV*, we can apply *PR-CV* again but from opposite direction, as shown below. (Again, since there are no deleting operations for the optionality attribute, we omit it here.)

PR-CV Inversed (PR-CVI):
¬Yes(Element E) → Delete E.

PR-CVI 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, *PR-CVI* works perfectly with the *PR-2, PR-4,* and *PR-6*, 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 below:

- *Vote-Simulating Deletion:*
  Delete element E $=_{def}$ *All votes on E are no.*

One property of these rules is that, an element cannot be

deleted if its creator doesn't vote *no* on it. This property derives from *PR-CV,* which ensures an initial *yes* vote on each element from its creator. This property 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* vote *no*. 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 the only stakeholder works on an element is the creator of it.

The last thing about deleting operations is the simulation of deleting features. It seems a little bit tricky, since features are not *atomic* elements – they are composed of name, description, and optionality elements. Is it possible that deleting a feature destroys other stakeholders' contribution to its name, description or optionality? (For example, a feature being deleted contains a name whose supporter is not one of the voters of the feature.) In fact, this kind of situation cannot happen according to *PR-3* and *PR-5*. The two rules ensure that each creator or supporter of any name or description of a feature, and each voter of the feature's optionality, must also be a supporter of the feature. Therefore, every contributor of a feature has the chance to express his opinion on the deletion of the feature.

As long as we simulate the deleting operations, the modifying and moving operations are trivial. To modify an element (except for the optionality), 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.

C. 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.

In our approach, we provide three types of EFM view, namely *working view*, *global view* and *personal view*. For each EFM, every stakeholder has exactly one working view as the main view, as well as one global view and several personal views as supplements. All the views can be generated automatically.

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.

$WV(EFM\ m, Stakeholder\ s) = \{Element\ e \mid e \in m \wedge s\ has\ not\ voted\ no\ on\ m\}$.

From the above rule, we can observe that 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.

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 *PR-CV* introduced in previous section attaches an initial *yes* vote on each elements thus ensures that a global view can be generated correctly even if the stakeholders have not explicitly voted *yes* on an element.

3) Personal view

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

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

A stakeholder can switch to personal views of other stakeholders to get the snapshot of a specific stakeholder's perspective of the domain.

IV. 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.

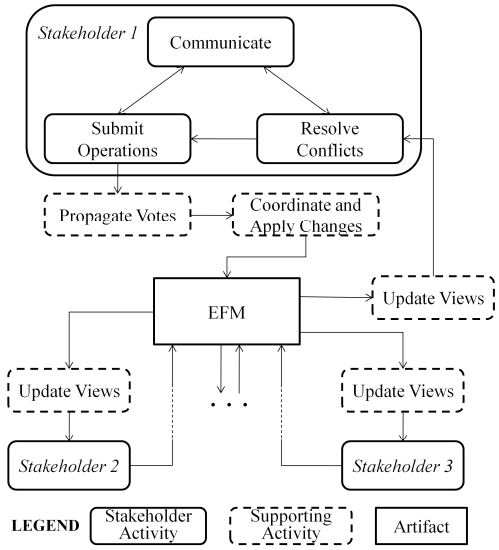

Figure 3. The process of collaborative feature modeling.

A. An Overview of the Process

Fig. 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 II.

TABLE II. ACTIVITIES

Activity	Description
Stakeholder Activities	
Submit Operations	Submit creating and voting operations.
Resolve Conflicts	Resolve conflicts in the working view.
Communicate	Communicate with other stakeholders via comments and/or discussion pages.
Supporting Activities	
Propagate Votes	Propagate votes after operations are submitted.
Coordinate and Apply Changes	Coordinate changes made by stakeholders, and then apply them to the EFM.
Update Views	Update global, working and personal views.

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

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

Resolve Conflicts: When the views are updated, stakeholders should first focus on the conflicts detected in their working views and resolve them via the creating and/or voting operations, and they need to communicate with others during the resolution. (See Section IV.B.)

Submitting Operations: In addition to conflict resolution, stakeholders submit operations to construct the EFM, and communicate with others as well as switch between the views during the construction.

Propagate Votes: Propagated votes are computed after an operation has submitted. Then the original operation and propagated votes are ready to update the shared EFM.

Coordinate and Apply Changes: In the context of collaborative work, changes submitted by multiple stakeholders need to be coordinated. After the coordination, if the original changes are still valid, they are applied to the EFM and in turn, cause the update of views of all stakeholders; otherwise the changes are neglected and its submitter will be informed. (See Section IV.C.)

B. Conflict Resolution

It has been observed that conflicts are common in collaborative work, because of divergence between stakeholders. However, not all divergences lead to conflicts, only significant ones do [7]. Particularly, in our approach, only the divergence in the working-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. In contrast, a stakeholder's working view is his individual workspace, and his work is considered unfinished if there are still conflicts in it.

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 view. (Names are identifiers of features.)
- *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 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 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.

Since working views are generated through stakeholders' operations, these conflicts emerge from their operations as well. Table III gives some examples of conflicts in a stakeholder's working view.

TABLE III. EXAMPLES OF CONFLICTS IN A WORKING VIEW (WV)

Type	The Stakeholder's		
	Initial WV	**Operations**	**Resulting WV**
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* F2 R1 *excludes* 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.

C. Coordination

In the context of collaborative work, it is highly possible that multiple stakeholders are working on the same set of elements at the same time, and their changes of the elements 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 Fig. 4.

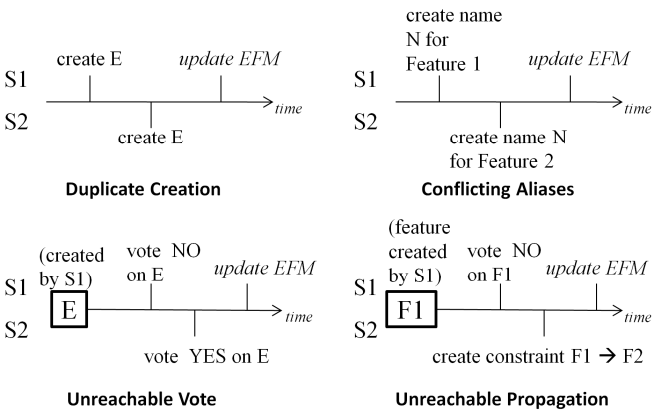

Figure 4. Examples of situations needing coordination.

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 Fig. 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.

Unreachable propagation is similar to unreachable votes. In Fig. 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 propagation of *yes* vote on F1 (according to the rule *PR-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 Fig. 4, *S1* will create the element *E*, and *S2* will have a *yes* vote on *E*.
- For conflicting aliases, the first alias is kept, and the latter one is neglected. In Fig. 4, only *Feature 1* 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 Fig. 4, the element *E* will be deleted, and *S2* will be informed that his/her voting operation has failed.
- For unreachable propagations, they are neglected as well as the operations which cause the propagations. In Fig. 4, the feature *F1* will be deleted, and *S2* will be informed that his/her creating operation has failed.

D. Guidelines for Stakeholders

In this sub-section, we propose a set of guidelines that help stakeholders effectively utilize the operations and views to improve the collaboration. Guidelines proposed by traditional feature modeling approaches [10][13] such as feature identification guidelines, are still fully applicable to stakeholders' work.

- *Perform conflict resolution as a start.* As we discussed before, conflicts provide a starting point for stakeholders' work, and should be the first priority for stakeholders.
- *Focus on divergences.* Stakeholders should always pay attention to divergences, especially those they have not voted on, even if the divergence doesn't lead to conflicts. Divergence often reflects insufficient understanding of a domain. By discussing on divergences, stakeholders can improve their understanding of the domain.
- *Show support explicitly by voting yes.* Although *PR-CV* ensures an initial *yes* vote on each element from its creator, we recommend other stakeholders explicitly vote *yes* when they agree the validity of the elements. Thus the creator can observe that his/her contribution has been reviewed and accepted by others.
- *Use personal views to track different perspectives of a domain.* Stakeholders can switch to anyone's personal view to get a snapshot of a specific perspective of the domain. For example, users may concern more about 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 users and programmers depict different perspectives of a domain.

- *Use global views 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 global views frequently, the stakeholder would not miss information like this.

V. A CASE STUDY

In this section, we introduce a case study to demonstrate the feasibility of our approach. In the study, an EFM of the video playing software domain is collaboratively constructed by stakeholders from different backgrounds. For simplicity, only three stakeholders are involved. Two of them are frequent users (*U1* and *U2*), and the other one (Programmer *P*) has experience in developing video playing software.

U1 is the first contributor, as shown in Table IV.

TABLE IV. THE EFM AT THE BEGINNING (PART)

Elements in the EFM	Votes of Stakeholders
	U1
Features	
Play Control (F1), Play/Pause (F2), Stop (F3)	$\bigcirc_C$
Relationships	
(F2, F3) refines F1	$\bigcirc_C$

$\bigcirc_C$: YES votes initiated when creating (PR-CV).

After that, another user *U2* joins and makes some changes on the EFM, as Table V shows.

TABLE V. THE EFM AFTER U2 JOINS (PART)

Elements in the EFM	Votes of Stakeholders	
	U1	***U2***
Features		
Play Control (F1), Play/Pause (F2), Stop (F3)	$\bigcirc_C$	○
Basic Control (F4), Advanced Control (F5), Repeat (F6), Jump (F7); Streaming Media (F8), Online Video Clip (F9), Online TV (F10)		$\bigcirc_C$
Relationships		
(F2, F3) refines F1	$\bigcirc_C$	X
(F2, F3) refines F4; (F6, F7) refines F5; (F4, F5) refines F1; (F9, F10) refines F8		$\bigcirc_C$

○: YES votes submitted by voting operations.

X: NO votes submitted by voting operations.

Now conflicts emerge from *U1*'s working view, i.e. conflicting refinements *(F2, F3) refines F1/F4*. (When *U2* created *(F2, F3) refines F4,* he denied the refinement created by U1 as well; otherwise he would get a conflict.)

After reconsidering the refinements, *U1* resolves the conflict by voting *no* on *(F2, F3) refines F1*. In addition, the *Advanced Control* created by *U2* inspires *U1* to add more features relating to it. However, *U1* opposes the validity of *Online TV* because he always uses specialized software provided by special service providers like BBC, to watch *online TV* programs. Therefore he believes it belongs to a different domain. Table VI shows the changes made by *U1*.

TABLE VI. THE EFM UPDATED BY U1 (PART)

Elements in the EFM	Votes of Stakeholders	
	U1	***U2***
Features		
Play Control (F1), Play/Pause (F2), Stop (F3)	$\bigcirc_C$	○
Basic Control (F4), Advanced Control (F5), Repeat (F6), Jump (F7); Streaming Media (F8), Online Video Clip (F9)	$\bigcirc_P$	$\bigcirc_C$
Online TV (F10)	X	$\bigcirc_C$
Slow (F11), Fast (F12)	$\bigcirc_C$	
Relationships		
~~(F2, F3) refines F1~~	X	X
(F2, F3) refines F4; (F6, F7) refines F5; (F4, F5) refines F1	○	$\bigcirc_C$
F10 refines F8	X_p	$\bigcirc_C$
F9 refines F8	○	$\bigcirc_C$
(F11, F12) refines F5	$\bigcirc_C$	

$\bigcirc_P$: YES votes propagated according to PR-1 to PR-6.

X_p: NO votes propagated according to PR-1 to PR-6.

Although it is just a very small part of the EFM, it still reflects important characteristics of our approach:

- *Divergence Tolerance*: (Temporary) divergences on the domain can be tolerated and coexist in the EFM for further negotiation. Some divergences eventually transform into consensus (e.g. *(F2, F3) refines F1/F4*); some might not and are considered as an indicator of domain variability (e.g. *Online TV*).
- *Consensus Facilitation*: By reserving and highlighting divergences and conflicts in working views, stakeholders are attracted to focus on these elements, which further leads to the resolution of them and facilitates the achievement of consensus.

In addition, the quality of the feature model is improved when more stakeholders join the construction. This can be demonstrated through the construction of video format/codec related features, before and after the programmer *P* joins.

Before the programmer joins, *U1* and *U2* have created some features about the video file formats. However, they confused video format with codec due to lack of technical expertise, as Table VII shows. (A video format can store data of different codec, e.g. MP4 is a video format which can hold

data of the codec MPEG-4 ASP, MPEG-4 AVC or VC-1.)

TABLE VII. FORMAT/CODEC BEFORE P JOINS (PART)

Elements in the EFM	Votes of Stakeholders	
	U1	*U2*
Features		
Video Format (F1), AVI (F2), MP4 (F3)	○C	○
MKV (F4), WMV (F5)		○C
Names (Aliases)		
F3: MPEG-4 (*confuse MP4 with MPEG-4*)		○C
Relationships		
(F2, F3) refines F1	○C	
(F4, F5) refines F1		○C

After *P* joins, he clears up the confusion about format and codec, and adds constraints between them, which improve the quality of the EFM. (See Table VIII.)

During the construction, the two users *U1* and *U2* tend to focus on service-level features and contribute little to constraints. In contrast, the programmer *P* is able to add more implementation-level features and explore the constraints between existing features. Therefore the EFM becomes more comprehensive and useful with the help of collaboration among stakeholders.

Fig. 5 shows the EFM, in which the consensus and divergence as well as features in different personal views are marked. We can see that the personal views depict different perspectives of the domain. For example, when talking about *Streaming Media*, there's no difference between *online TV* and *online video clip* for the programmer *P,* only *Protocol* matters. In contrast, the two users care about the difference between TV programs and video clips.

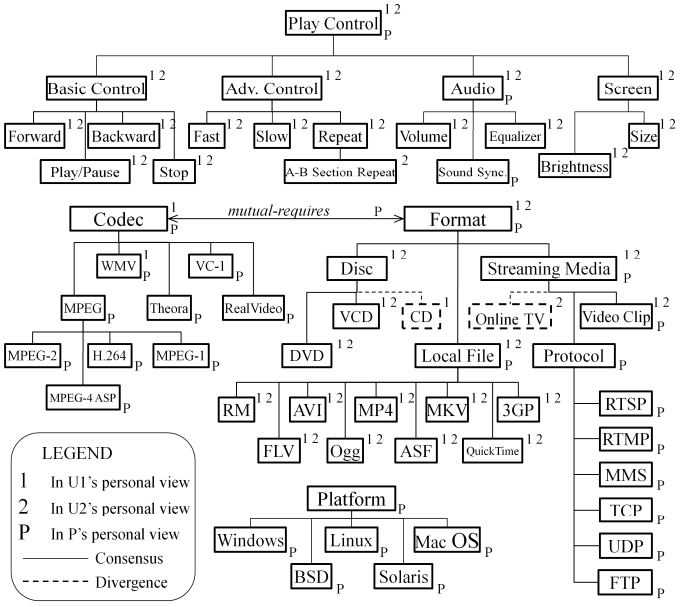

Figure 5. The EFM of video playing software domain.

TABLE VIII. FORMAT/CODEC AFTER P JOINS (PART)

Elements in the EFM	Votes of Stakeholders		
	U1	*U2*	*P*
Features			
Video Format (F1), AVI (F2), MP4 (F3)	○C	○	○P
MKV (F4), WMV (F5)		○C	○P
Video Codec (F6), MPEG-4 AVC (F7), MPEG-4 ASP (F8)			○C
Names (Aliases)			
F3: MPEG-4		○C	X
F4: Matroska; F7: H.264			○C
Relationships			
(F2, F3) refines F1	○C		○
F4 refines F1		○C	○
F5 refines F1		○C	X
(F7, F8) refines F6			○C
F6 mutual-requires F1 *(formats and codec must work together to deliver video data)*			○C

VI. RELATED WORK

In FODA [13], FORM [14], FeatuRSEB [10], and FOPLE [15], 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.

There are some CSCW tools in which the collaboration mechanism is similar to our voting-based approach. In the *gIBIS* tool [3] and the *WinWin* system [1], stakeholders solve problems by proposing possible solutions and associating supportive or opposing argument with existing solutions. In *Synoptic* [7], stakeholders can rate possible solutions in five levels (from *totally satisfy* to *totally frustrated*). 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. In contrast, in our approach, feature models can tolerate permanent divergences, which are treated as indicators of domain variability. Furthermore, in these tools, conflicts between stakeholders need to be manually identified, while in our approach, conflicts are well-defined and automatically detected.

In the research of requirements engineering, several approaches have been proposed to support collaborative requirements elicitation [2][5][12][18] and requirements management [4][16][20]. These approaches incorporate traditional collaboration techniques such as online forums, emails, and wiki-like tools. Some of them [5][20] deeply

integrate requirements with communication data (e.g. emails sent and received by stakeholders) for further analysis. For example, in [5], if a stakeholder's communicate data is linked with too many requirements, the stakeholder is considered as a possible bottleneck of the collaboration. A problem of these approaches is that they focus on maintain consensus and so do not allow conflicts to be expressed. This may lead to neglect of requirements which seem unimportant in the present but have potential impact in the future.

VII. Conclusions and Future Work

This paper proposed a collaborative feature modeling approach to improving feature models' construction with explicit support of collaboration among stakeholders. In this approach, stakeholders can not only create new elements in a feature model as normal, but also can vote *yes* or *no* on existing elements to express their opinions on the elements. For this purpose, 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 and a set of guidelines to construct feature models in a collaborative way. In this process, conflicts resolution and coordination mechanisms are incorporated to guide stakeholders' work. A case study of video playing software domain shows the divergence tolerance and the consensus facilitation characteristics of our approach, and also demonstrates that when more stakeholders join the collaboration, it is more likely to improve the quality of constructed feature models.

Our future work will focus on providing practical tool support for our approach, and apply it to a suitable environment, such as SaaS product development and maintenance (where the number of stakeholders is large and the collaboration is more intensive) to verify our approach's scalability and usability.

Acknowledgment

This work is supported by the National Basic Research Program of China (973) under Grant No. 2009CB320703, the Science Fund for Creative Research Groups of China under Grant No. 60821003.

References

[1] B. Boehm, P. Bose, E. Horowitz, M. J. Lee, "Software Requirements As Negotiated Win Conditions," Proc. of Intl. Conf. Requirements Engineering (RE 94), 1994, pp. 74–83.

[2] C. Castro-Herrera, J. Cleland-Huang, B. Mobasher, "Enhancing stakeholder profiles to improve recommendations in online requirements elicitation," Intl. Conf. on Requirements Engineering (RE 09), Sep. 2009, pp. 37–46.

[3] J. Conklin, M. Begeman, "gIBIS: A Hypertext Tool for Exploratory Policy Discussion," ACM Trans. On Information Systems (TOIS), 1988, vol. 6, pp. 303–331.

[4] D. Damian, S. Marczak, I. Kwan, "Collaboration patterns and the impact of distance on awareness in requirements-centred social networks," Intl. Conf. on Requirements Engineering (RE 07), Oct 2007, pp. 59–68.

[5] B. Decker, E. Ras, J. Rech, P. Jaubert, M. Rieth, "Wiki-Based Stakeholder Participation in Requirements Engineering," IEEE Software, 2007, vol. 24, no. 2, pp. 28–35.

[6] P. Dourish, V. Bellotti, "Awareness and Coordination in Shared Workspaces," Proc. of the 1992 ACM Conf. on CSCW, pp. 107–114.

[7] S. Easterbrook, "Handling Conflicts Between Domain Descriptions with Computer-Supported Negotiation," Knowledge Acquisition: An International Journal, 1991, vol. 3, 255–289.

[8] C. A. Ellis, S. J. Gibbs, G. Rein, "Groupware: Some Issues and Experiences," Communications of ACM, 1991, vol. 34, pp. 39–58.

[9] C. Ellis, J. Wainer, "A Conceptual Model of Groupware," Proc. of the 1994 ACM Conf. on CSCW, pp. 79–88.

[10] M. L. Griss, J.Favaro, M. d'Alessandro, "Integrating Feature Modeling with the RSEB," Fifth Intl. Conf. on Software Reuse (ICSR 98), IEEE Computer Society, Jun. 1998, pp. 76–85.

[11] J. Grudin, "Computer-supported cooperative work: history and focus," Computer, 1994, vol. 27, no. 5, pp. 19–26.

[12] P. Haumer, K. Pohl, K. Weidenhaupt, "Requirements Elicitation and Validation with Real World Scenes," IEEE Trans. on Software Engineering, Dec. 1998, pp. 1036–1054.

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

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

[15] K. C. Kang, J. Lee, P. Donohoe, "Feature-Oriented Product Line Engineering," IEEE Software, vol. 19, no. 4, July/Aug. 2002, pp. 58–65, doi:10.1109/MS.2002.1020288.

[16] M. Lang, J. Duggan, "A Tool to Support Collaborative Software Requirements Management," Journal of Requirements Engineering, 2001, vol. 6, part 3, pp. 161–172.

[17] Y. Laurillau, L. Nigay, "Clover Architecture for Groupware," Proc. of the 2002 ACM Conf. on CSCW, pp. 236–245.

[18] R. Recio, C. Salzberg, J. Palm, C. Machuca, "Leveraging collaborative technologies in the IO requirements process," Intl. Conf. on Requirements Engineering (RE 08), Sep. 2008, pp. 283–288.

[19] K. Schmidt, C. Simone, "Coordination Mechanisms: Towards a Conceptual Foundation of CSCW System Design,", CSCW Journal, 1996, vol. 5, no. 2–3, pp. 155–200.

[20] V. Shiha, B. Sengupta, S. Chandra, "Enabling Collaboration in Distributed Requirements Management," IEEE Software, 2006, vol. 23, no. 5, pp. 52–61.

[21] W. Zhang, H. Mei, H. Zhao, "A feature-oriented approach to modeling requirements dependencies," in Proc. of the 13th IEEE Intl. Conf. on Requirments Engineering (RE 05), 2005, pp. 273–282.