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

Li Yi, Wei Zhang, Haiyan Zhao, Hong Mei
Institute of Software
School of EECS, Peking University
Beijing, China
{yili07, zhangw, zhhy}@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 [1]. 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 [1] and FORM [2] 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 [3], 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 often impossible for only one or a few persons to obtain a comprehensive understanding of a domain without sharing knowledge with others.

However, few of the 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.

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. Following this idea, we propose an extended meta-model of feature models to explicitly introduce the voting operations into feature models, and define a process and a set of guidelines to construct feature models in a collaborative way. A tool prototype is also developed to carry out this collaborative feature modeling approach efficiently. To verify the feasibility of this approach, a case study on the video playing software domain is conducted. 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 the emergence of consensus among stakeholders.

The reminder of this paper is organized as follows. In Section 2, we introduce some preliminaries. The static and

dynamic aspects of our approach, i.e. the conceptual framework and the modeling process are explained in Section 3 and 4, respectively. A case study is introduced in Section 5 to show the feasibility of our approach, and we explore some related work in Section 6. Finally, Section 7 concludes this paper with a brief summary and future work.

II. PRELIMINARIES

In this section, we introduce some concepts of CSCW systems and a meta-model of feature models. They serve as bases of the "collaborative" aspect 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 [13]. Over the years, CSCW researchers [14][15][16][17][18] have identified some central concepts, which still shape the design of modern CSCW systems, although technologies have changed tremendously. The most important concepts are production, awareness, and coordination.

Production refers to objects produced and shared in a CSCW system, and to operations on these objects [18]. It can be viewed as an ontological model [16] or a conceptual framework of a system. Implementation of this concept in our system will be described in Section 3.

Awareness means systems must provide information about the context of individual work [15]. For example, collaborative writing tools often show the position of coauthors' cursor, which helps authors identify their editing boundaries as the context of work. Section 3 will describe our support for awareness through *views*.

Coordination is related to mediating and meshing individual work in a shared workspace [17]. While "production" defines activities available to collaborators, the term "coordination" concerns the organization of these activities between collaborators [16] in order to reduce possibilities of confliction or repetition. Section 4 will give details about the coordination mechanism in our system.

B. A Meta-Model of Feature Models

Fig. 1 Figure 1. shows a meta-model of feature models based on our previous work [19]. A feature model consists of a set of features and relationships between them. There are two types of relationships, namely *refinements* and *constraints*.

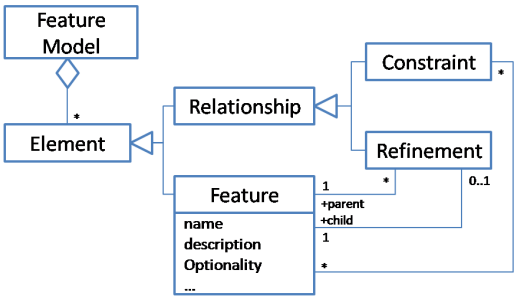

Figure 1. A meta-model of feature models.

The concept of *feature* can be defined in two aspects: intension and extension [19]. In intension, a feature is a cohesive set of individual requirements. In extension, a feature is a user/customer-visible capability of a software system.

The *refinement* relationships organize features into a hierarchical structure, according to different levels of abstraction and various granularities of the features. In this hierarchical structure, each feature can be mandatory or optional. If a feature has been bound, its mandatory children must be bound as well, while its optional children can be removed or bound.

The *constraint* relationships describe dependencies between features. There are two kinds of binary constraints: *requires* and *excludes* [1][19]. If feature A requires feature B and A has been bound, it means B must be bound as well. If two features exclude each other, it means at most one of them can be bound. More types of constraints can be found in [19].

III. THE CONCEPTUAL FRAMEWORK

In this section, we explain the extended feature model (EFM) which will be collaboratively constructed. We first introduce the meta-model of EFMs as an overview. Then we give more details about the extensions we made.

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

Fig. 2 shows the meta-model of extended feature models. An EFM is composed of many vote-able *elements*. The top-level elements are *features* and *relationships*. A feature may have more than one *name*, and each name is a vote-able element. Similarly, a feature may have many vote-able *descriptions*. However, a feature has exactly one *optionality* attribute which is an element as well. The result of vote on every element is recorded as its *supporters* and *opponents*, both of which are a set of *stakeholders*.

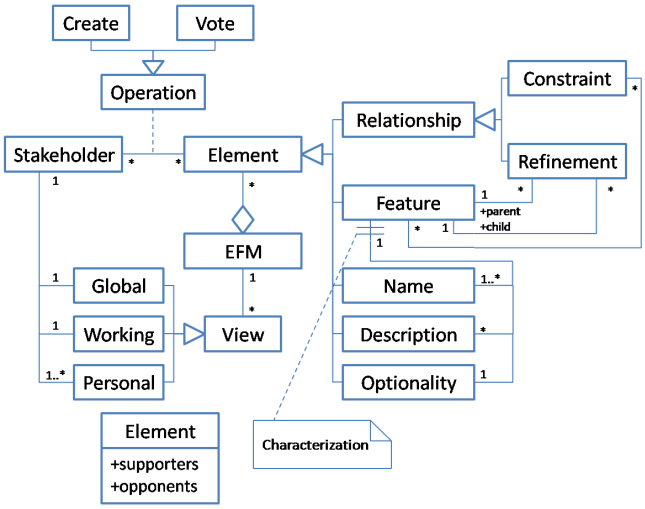

Figure 2. The meta-model of extended feature models.

Stakeholders work on EFMs with the *operations* we provide. They can either *create* new elements in EFMs or *vote* on existing ones. Other ordinary modeling operations,

such as modification or deletion, are implemented through the two basic operations. (See Section III-B for details.)

We provide three types of *views* for each EFM, namely *global view*, *working view* and *personal view*. Each stakeholder has one global view, one working view and at least one personal view for each EFM he involved. The views are derived from EFMs by generating rules. (See Section III-C for details.)

B. Operations on EFMs

We provide two operations for stakeholders, namely creating and voting. Stakeholders can create new elements and add them to an EFM, and they can vote yes or no on any existing element. Besides, we incorporate Vote Propagation Rules according to the semantic of creating and voting.

1) Creating and restrictions

Although creating is an ordinary modeling operation, it is less restricted in EFMs than in feature models introduced in Section II. For example, stakeholders can create several names for a single feature in an EFM, while they can only give exactly one name for each feature in ordinary feature models. In fact, the only restriction on creation is that no duplicate elements can be created (except for Optionality), as described in Table I.

TABLE I. RESTRICTIONS ON CREATION

Element	Restriction
Name	The name has not been used in *any of the features* in current EFM.
Description	The description has not been used in current feature.
Feature	A name must be created successfully when creating a feature.
Refinement (X refines Y)	X and Y are existing features in current EFM and no "X refines Y" existed.
Constraint (X requires Y)	X and Y are existing features in current EFM and no "X requires Y" existed.
Constraint (X excludes Y)	X and Y are existing features in current EFM and no "X excludes Y" or "Y excludes X" existed.
Optionality	Its values are predefined (*mandatory*, *optional*), thus no creation is allowed.

In EFMs, a feature may have many names because terminologies used by stakeholders may be different. We treat the names of the same feature as aliases of each other, and no duplicate names are allowed in one 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 in Table I for simplicity. The restrictions on group and complex constraints defined in [19] are similar.

Finally, the optionality of features has predefined values: mandatory or optional. "Creating a new optionality" is meaningless and therefore disallowed.

2) Voting and automatic vote propagation

Stakeholders vote yes or no on an existing element to express their viewpoints on the element. First of all, we want to describe the semantic of vote in terms of *existence*:

- *A "Yes" vote shows support for existence of the element in current EFM, while "No" shows opposition.* (If type of the voted element is not "Optionality".)
- *A "Yes" vote shows support for treating current feature as an optional one, while "No" treats it as a mandatory one.* (If type of the voted element is "Optionality".)

The semantic of voting on "Optionality" is special because the optionality is creating-forbidden and actually a Boolean variable.

There are no restrictions when stakeholders are voting on elements. However, votes of the same stakeholder may be inconsistent. For example, if there is a feature "A" and a constraint "A requires B", what does it mean if a stakeholder vote no on the feature but vote yes on the constraint? Based on the semantic of vote, the stakeholder supports a constraint which involves a feature considered nonexistent. Such votes are inconsistent and meaningless.

Our solution to such problems is to propagate the vote to corresponding elements automatically, whenever a voting operation is submitted. First of all, we want to take a closer look at the existence of different elements in an EFM. For clarity's sake, we define the predicate *Exist* as,

$$Exist(Element\ E) = \begin{cases} True, & E\ exists\ in\ the\ EFM \\ False, & else. \end{cases}$$

Thus, the existence of elements has the following relationships.

$$Exist(Relationship\ R\ involving\ feature\ F) \rightarrow Exist(F). \quad (1)$$

$$Exist(Attribute\ A\ of\ feature\ F) \rightarrow Exist(F). \quad (2)$$

According to the semantic of voting operation, we immediately know that, (we will discuss optionality later,)

$$Yes(Element\ E\ not\ Optionality) \leftrightarrow Exist(E). \quad (3)$$

Where the predicate *Yes* is defined as,

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

Therefore, according to (1), (2) and (3), we define the following Automatic Vote 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. They ensure consistent votes on features and relationships from a stakeholder. For example, if there is a feature "A" and a relationship "A requires B", and if a stakeholder votes yes to the relationship, a yes vote to both feature A and B will be propagated. If the stakeholder votes no on A, then the relationship will be propagated a no vote but B will not be affected.

Similarly, the next two rules define the vote propagation between features and their attributes, except for the optionality.

PR-3:

$$Yes(Attribute\ A\ of\ Feature\ F) \rightarrow Yes(F), where\ A\ typeof\ Name\ or\ Description.$$

PR-4:

$$\neg Yes(Feature\ F) \rightarrow \bigwedge_{A\ belongs\ \ to\ F} \neg Yes(Attribute\ A), where\ A\ typeof\ Name\ or\ Description.$$

Now we want to consider the optionality attribute. According to previous discussions, the semantic of vote on optionality is whether the current feature is mandatory or not, while the semantic of vote on a feature is whether the feature should exist or not.

Given these semantics, we can say that *a feature must exist before its optionality is considered*. Therefore, we define a pair of rules as below:

PR-5: Vote on optionality of feature F $\rightarrow Yes(F)$.

PR-6: $\neg Yes(Feature\ F)$ *→ Remove the voter's vote on optionality of F.*

The six PRs above deal with the propagation between votes. However, there is a connection between creating and voting operations, according to their semantics. The semantic of creating is to make a currently nonexistent element exist in an EFM, while the voting expresses support or opposition of existence of the element. (Because no creation is allowed for the optionality, we omit the semantic of vote on optionality here.) We know that if a stakeholder creates an element, he obviously supports for its existence. Therefore we have the following Creating-imposed Vote Propagation Rule (PR-CV).

PR-CV: Create element E → Yes(E), where type of E is not the Optionality.

All PRs are *transitive*, although transitions only take effect when stakeholders are creating something -- no actual transition happens when submitting voting operations, due to the definition of PR-1 to PR-6. For example, when stakeholders create a relationship, PR-CV applies, which in turn, makes PR-1 take effect. Besides, it is possible to apply multiple PRs to one voting operation. For example, PR-2, 4 and 6 apply to voting NO on features.

In addition to the propagation rules, we have 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, and we handle them with the Repetitive Vote Rule (RVR).

RVR: If there are multiple votes on an element from the same stakeholder, no matter the votes are submitted by voting operations or propagated, only the last vote counts.

Although there are only two operations available for stakeholders, we can implement other ordinary modeling operations by the two, with the help of the rules, as described in the next sub-section.

3) Implementing other ordinary modeling operations

Traditional feature modeling approaches provide operations for creating, deleting, modifying and moving elements in feature models. We call these four operations as *ordinary modeling operations*. As mentioned before, we purposely provide only one ordinary modeling operation – the creating – plus the voting operation, to prevent stakeholders from editing and overwriting others' work directly, thus to satisfy the divergence tolerance property.

However, we do allow stakeholders to simulate the three non-provided operations in EFMs, *if and only if they can reach consensus*. Since the modification and moving are combination of creation and deletion, we focus on implementing the deletion operation.

The basic idea comes from semantics of creation, voting and deletion. In fact, deletion is opposite to creation. According to the discussion of PR-CV in the previous sub-section, it seems we can apply PR-CV again but from another direction, as shown below. (Again, since there is no deletion for the optionality attribute, we omit voting on optionality here.)

Voting-Deduced Deletion (VDD): ¬Yes(Element E) → Delete(E), where type of E is not the Optionality.

VDD means if a stakeholder votes no to an element, his intention is to delete the element from the EFM, which conforms with the semantic of voting no – opposing the existence of the element. Moreover, VDD works perfectly with the PRs. If we combine the semantic of VDD with PR-2, PR-4 and PR-6, we get that when a feature is deleted, its attributes and involved relationships are also deleted. This looks fairly reasonable.

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

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

It is worth noticing that the PR-CV ensures an initial "yes" vote on every element, from its creator. Therefore an element can be deleted only if its creator permits it by voting no on it. It again reflects the divergence tolerance characteristic of our approach – every viewpoint of the domain should be and can be reserved.

Another property is the deletion does not require *all stakeholders* vote no on the element, only *all voters on the element* are needed. In other words, it only needs consensus between stakeholders who interest in the element. Therefore it can reduce unnecessary work load on stakeholders. Furthermore, this makes the simulation of undo possible, if the only stakeholder works on an element is the creator of it. See Table II for an example.

In addition, the simulation of deleting features seems a little bit tricky, since features are not "atomic" elements – they are composed of elements of the type *name, description and optionality*. Is it possible that delete a feature could destroy others' contributions to its name, description or optionality? For example, what if a being deleted feature

contains a name whose supporter is not one of the voters of the feature? In fact, this cannot happen, thanks to the vote propagation rules PR-3 and PR-5. These rules ensure that every creator or supporter of any name and description of a feature, and every voter of its optionality, *must* be a supporter of the feature as well. Therefore, every contributor of a feature has the chance to express his opinion on the existence of the feature.

Examples of deletion can be found in Table II.

TABLE II. EXAMPLES OF DELETION, MODIFICATION AND MOVING

(Content: undo, deletion, modification, moving. List votes on elements after each operation. For modification, two situations: everyone create new, or someone create new.)

As long as we implement deletion with creating and voting, the modification is easy to understand. Therefore we only give the implementation and valid elements of it, and show some examples in Table II.

Modification

- Implementation: All voters of an element vote no on it, and then some of them create new elements *of the same type*.
- Valid Elements: Name, Description, Feature and Relationship

The last ordinary modeling operation, moving features, is trivial. It is actually the modification of the *refinement* relationships. Examples can be found in Table II as well.

C. Views for EFMs

The creating and voting operations introduced in the previous sub-section allow different opinions to coexist in the same EFM. Now we want to discuss how to represent the EFM to stakeholders, with the aim of help them understand current context of work.

Our basic idea is a stakeholder's context of work is built on his own viewpoints on an EFM, thus the context is easy to understand for him. Besides, the whole picture of the EFM, and different stakeholders' understanding of the EFM should be provided.

Therefore, we provide three types of EFM views, 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 are automatically generated.

1) Working view

Working views act as the main working space of stakeholders. We generate the working view of an EFM for a specific stakeholder according to the following rule.

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

Thus, the working context of a stakeholder consists of,

- Elements he supported, either created by others or himself, and
- Elements created by others and he has not shown any opinions.

Since the voting on optionality has different semantic, it always resides in the features in working views.

It is worth noticing that there might be three kinds of abnormal features in a working view, which only exist in EFMs,

- Unnamed Feature: the name of the feature is missing because the stakeholder has voted "No" on all names of it.
- Isolated Feature: the feature has no parent because the stakeholder has voted "No" on all refinements that contain this feature as a child.
- Multi-Positioned Feature: a feature has more than one parent which might be created by different stakeholders, and the stakeholder has not expressed his viewpoints on them.

Illustrative examples are shown in Table III. (包括这种情况：我 vote no 给父特征，导致它的子特征在我这边是孤立的。)

TABLE III. ABNORMAL FEATURES IN EFMS

The abnormal features can be easily detected and fixed. For the first two categories, the stakeholder only needs to create a new name or a new refinement, or just changes his opinions on the existing names or refinements after discussing with other stakeholders. For the third category, he needs to make choice by voting on the refinements, or even opposes all of them and creates a new one.

2) Global view

Global views reveal the whole picture of an EFM. It is composed of all undeleted elements in an EFM.

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

The rule PR-CV introduced in previous section attaches an initial Yes vote on elements, thus ensures global views always correctly generated even if the stakeholders have not explicitly voted yes.

3) Personal view

Personal views dedicate to represent stakeholders' own EFMs, by put the element supported by them together.

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

In a word, the private view of a stakeholder consists of elements he supported, either created by him or by other stakeholders.

Stakeholders can switch to personal views of any others, thus get the snapshot of a specific stakeholder's viewpoint.

An example of the EFM and its views is shown in Fig.3.(Use a small part of Video Playing Software in case study.)

Figure 3. An example of the EFM and its views.

IV. THE PROCESS

In this section, we introduce the process of collaborative feature modeling, based on the conceptual framework discussed before. We first describe the process and activities in it, and then we give some guidelines for stakeholders. Finally, we introduce the coordination mechanism which handles the situation when multiple stakeholders are working on the same set of elements simultaneously.

A. *Activities and the Process*

In this sub-section, we first introduce an overview or the process, and then give more details about the activities in it.

Fig.4 shows the process of collaborative construction of an EFM.

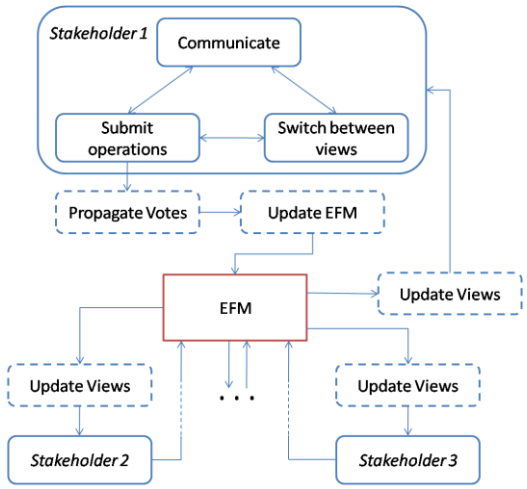

Figure 4. The process of collaborative feature modeling.

There are two categories of activities in it,

- Stakeholder Activity: activities initiated by stakeholders,
- Supporting Activity: activities initiated by the system automatically.

Activities of each category are described in Table IV.

TABLE IV. ACTIVITIES

Activity	Description
Stakeholder Activities	
Submit Operations	Submit creating and voting operations.
Switch Between Views	Switch between global, working, and personal views.
Communicate	Communicate with other stakeholders via comments and/or discussion pages.
Supporting Activities	
Propagate Votes	Propagate votes after operations are submitted.
Update EFMs	Update EFMs with submitted operations and propagated votes.
Update Views	Update global, working and personal views.

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

Update views: When a stakeholder starts to work, views are automatically generated for him. Whenever any stakeholder successfully updates the shared EFM, all stakeholders' views will be updated. Particularly, every update of the working view includes a detection of abnormal features (defined in Section III-C).

Work on the EFM: There is no specific order of these activities for a stakeholder. The stakeholder should use his working view as the main view, although he can switch to any view at any time as he like. The stakeholder can submit operations in any of the views. He may communicate with others during his work.

Propagate votes and update the EFM: Propagated votes are computed after an operation has successfully submitted. Then the result of the operation and propagated votes are updated into the shared EFM, and in turn, updated into all views attached to the EFM including the submitter's.

B. *Some guidelines for stakeholders*

Use abnormal features as a start. When a stakeholder begins to work, the first thing should be done is to fix abnormal features, if any, in his working view. Because these features reveal unfinished work of the stakeholder, for each type:

- The u*nnamed feature* emerges when the stakeholder has denied all the names of the feature without creating a new one for it. The stakeholder should either give it a suitable new name, or reconsider existing names.
- The *isolated feature* is similar to unnamed features, but its parent is missing. The stakeholder should create a new refinement relationship involving this feature as a child, or make it a root feature, or reconsider existing refinements linked to it.
- *The multi-positioned feature* means stakeholders have not achieved consensus on the parent of the feature, and the stakeholder has not fully express his opinion. The stakeholder should vote on related refinements to accept exactly one of them, or even vote no on all and create a new refinement.
- *(Conflicting constraints, detected use our previous work)*

Focus on divergence. Stakeholders should always pay attention to divergence, especially those they have not voted on. Divergence often indicates uncommon knowledge of a domain. By discussing and expressing opinions on them, stakeholders can improve their understanding of the domain.

Show support explicitly by voting yes. Although the vote propagation rule PR-CV ensures an initial Yes vote on elements from their creators, we recommend other stakeholders explicitly vote yes when they agree the existence of the elements. Thus creators can tell that their contributions have been reviewed 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 high-level features, while programmers might interest in 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. For example, a stakeholder may vote no on a feature in early stage of the

work. Since then, most stakeholders have expressed opposite opinions on the feature. This possibly means the first stakeholder needs to reconsider his decision. By using global views frequently, stakeholders would not miss information like this.

C. Coordination in EFMs

When multiple stakeholders are working on the same EFM, there might be conflicts which cannot happen in individual work, and they to be coordinated. The conflicts are caused by an *update delay problem*: update of the EFM originated by one stakeholder cannot immediately become visible to others, as illustrated in Fig. 5.

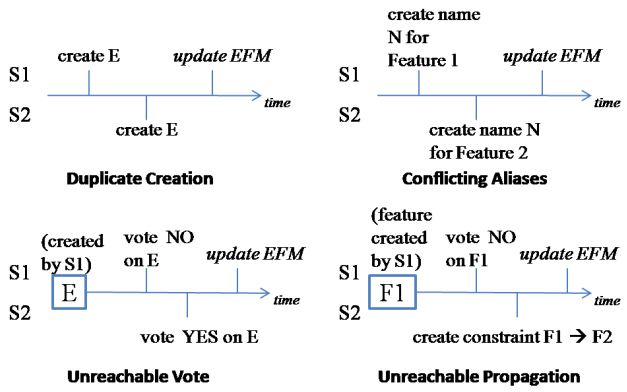

Figure 5. Examples of problems needing coordination.

Duplicate Creation happens when a stakeholder (S2) creates an element E before a previous creation of E by S1 has been updated into the EFM and becomes visible to S2. 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 previous situation causes a potential conflict.

Unreachable vote is the vote on an inexistent element. In Fig. 5, 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. In Fig. 5, if the No vote on F1 leads to the deletion of it, and if S2 creates a constraint involving F1 before the deletion becomes visible to him, then the propagation of Yes vote on F1 by S2 (according to rule PR-1) is unreachable.

Coordination of these situations follows a *serialized update strategy*, that is, all update applies on the EFM in the order of their submitting. Furthermore, if the being applied update is no longer valid, it fails and takes no effect on the EFM.

For duplicate creations, the first creation adds a new element to the EFM, and the later creations are converted to yes votes. In Fig. 5, 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 later ones are neglected. In Fig. 5, only Feature 1 will be assigned an alias N.

For unreachable votes, they are no longer valid on an inexistent element and are neglected. In Fig. 5, the element E will be deleted, and S2 will be informed his vote has failed.

For unreachable propagations, they are neglected, as well as the operations which cause the propagations. In Fig. 5, the feature F1 will be deleted, and S2 will be informed his creation of the constraint has failed.

V. A Case Study

(EFM: Video Playing Software;

Stakeholders: Users, Programmers;

Consensus Facilitate: 在展示 Working View 的时候突出这个特点，即某些特征有不同的精化策略，Highlight 这些地方，使得人们优先关注这些分歧，很快就精化策略达成一致。

Divergence Tolerance：有人建了一个“音乐播放”的特征，这个特征在现有产品里确实存在（暴风影音），其他人觉得不合适。第一个人会继续往下精化一些自己的东西，比如支持的格式等，其他人不受影响。

Personal Views： User 比较关注上层的特征，Programmer 在底层建了很多技术细节的特征，这样在 Personal View 里可以看到两人对于领域的不同视角。其中，programmer 的 view 里是多棵树，每棵树的根是一个由 User 提出的高层特征）

Global View：因为等于 EFM，所以在 Personal View 之前先展示一下。

)

VI. Related Work

In FODA [1], FORM [2], FeatuRSEB [3], and FOPLE [4], feature models are constructed through a systematic analysis of commonality and variability in a family of systems across a domain. Such task is performed by domain analysts, with the assistance of end users, domain experts and requirements analysts. However, despite its inherent need of collaboration, none of the methods has explicitly incorporated mechanisms or techniques for supporting collaboration in feature model constructions.

For other models, such as ontologies, there exist some methods and tools supporting collaborative model construction. The Ontolingua Server [11] provides a web server which supports collaborative ontology construction by geographically distributed users. The Ontolingua Server uses an access control mechanism that is typical in most multi-user file systems: read and write access to an ontology is controlled by its owner. Another example is OntoEdit [12], which forces users to lock before editing to avoid overwriting in a shared ontology. Like most collaboration tools, both tools emphasize the importance of achieving consensus on ontologies and neglect divergence between users, or consider it harmful.

In the research of requirements engineering, researchers have already realized the importance of developing tools which explicitly support collaboration. The CREWS project [5][6] proposed a set of methods and tools for collaboratively eliciting and validating requirements with scenarios. The OCPI [7] adopted online forums to facilitate the

requirements elicitation process. The IOTC [8] provided a collaboration medium for stakeholders to express their requirements by combining a special software collaboration tool with discussion forums, mailing lists and wikis. In addition, Rashid [9] and Damian [10] discussed the importance of user participation and the need of collaboration tools in requirements management.

VII. CONCLUSIONS AND FUTURE WORK

REFERENCES

[1] 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.

[2] 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.

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

[4] 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.

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

[6] C. Rolland, C. Souveyet, C. B. Achour, "Guiding Goal Modeling Using Scenarios," IEEE Trans. on Software Engineering, Dec. 1998, pp. 1055–1071.

[7] C. Castro-Herrera, J. Cleland-Huang, B. Mobasher, "Enhancing stakeholder profiles to improve recommendations in online requirements elicitation," in the 17th IEEE Intl. Conference on Requirements Engineering (RE 09), Sep. 2009, pp. 37–46.

[8] R. Recio, C. Salzberg, J. Palm, C. Machuca, "Leveraging collaborative technologies in the IO requirements process," in the 16th IEEE Intl. Conf. on Requirements Engineering (RE 08), Sep. 2008, pp. 283–288.

[9] A. Rashid, "OpenProposal: towards collaborative end-user participation in requirements management by usage of visual requirement specifications," in the 15th IEEE Intl. Conf. on Requirements Engineering (RE 07), Oct. 2007, pp. 371–374.

[10] D. Damian, S. Marczak, I. Kwan, "Collaboration patterns and the impact of distance on awareness in requirements-centred social networks," in the 15th IEEE Intl. Conf. on Requirements Engineering (RE 07), Oct 2007, pp. 59–68.

[11] A. Farquhar, R. Fikes, J. Rice, "The Ontolingua Server: a Tool for Collaborative Ontology Construction," Intl. Journal of Human Computer Studies, 1997, vol. 46, no. 6, pp. 707–728.

[12] Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, D. Wenke, "OntoEdit: Collaborative Ontology Development for the Semantic Web," Lecture Notes in Computer Science, 2002, pp. 221–235.

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

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

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

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

[17] 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.

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

[19] 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.