An Extended Feature Modelling Based Component Model for Performance Evaluation

Zhongjie Wang, Xiaofei Xu, and Dechen Zhan
School of Computer Science and Technology, Harbin Institute of Technology, China

Abstract: *Traditional component modelling methods focus emphatically on the precise and formal descriptions of business semantics, while usually cannot support to conveniently evaluate the reuse performance of components. Our main concern is to present a new component model for measuring performance after components being designed and before practically reused. The proposed model uses the Extended Feature Modelling method (Ext-FM) to express business semantics and uses variation point to express component's reuse mechanism. Some example metrics for component reusability are addressed briefly to validate the effectiveness of this model.*

Keywords: *Component model*, *extended feature modelling*, *variation point*, *reusability metrics*.

Received August 28, 2005; accepted April 13, 2006

1. Introduction

From 1990s *component-related technologies* [3, 15] have been considered as one of the most pivotal technologies to realize *software reuse*, and there has been a consensus both in academe and in industry that, component reusability influences the performance of software reuse rigorously [11]. Currently, most of popular metrics for component reusability [10, 17] usually evaluate component performance by analyzing practical accumulated reuse data after components have been reused for some periods. However, it is quite indispensable to evaluate component reusability before they are practically reused, to find those designs that are not suited to reuse and modify these deficiencies so as to improve reusability as far as possible for better software reuse [19].

Pre-reuse reusability should be acquired by analyzing *component model* which describes component structure and semantics, and considered as the foundation of analyzing and evaluating components' properties and behaviours [4]. It establishes a common theoretical and application platform for basic component-based development [13] activities, such as storage, query, adaptation and composition, etc. Currently, there exist numerous component models, which can be classified into three types according to their purposes [8], i. e., component description and classification models, such as REBOOT [12], JBCL [7]; component specification and composition models, such as 3C [16], Wright [1], JBCOM [14]; and component implementation models, such as DCOM [9], CORBA CCM [2], EJB [14].

Present research on component models tends to *two opposite extremes*, research on component syntax structure and implementation techniques in programming language level without considering business semantics, and research on component semantics using high formal approaches (e. g., formal languages) while breaking away from business background in reality. Neither approaches pay much attention to the mappings between component models and domain business models, and there are some limitations that make it difficult for these component models to support reusability evaluation expediently [19].

The objective of this paper is to present a new component model to support reusability evaluation in a simple and precise way. The rest of this paper is arranged as follows. In section 2, we present extended feature modelling method (Ext-FM), and based on traditional feature modelling, we put forward the concept of Feature Dependency (FD) and show five types of FDs. In section 3, a new component model based on Ext-FM is shown, and we emphatically discuss how to express variable semantics in this model by variation points. In section 4, the way of how to evaluate pre-reuse reusability with the help of the new model, including several metrics and the corresponding evaluation methods, are briefly discussed. Finally, comparisons between our model and other models, and the conclusion are shown in section 5.

2. Extended Feature Modelling (Ext-FM)

Feature-oriented methods have been widely applied in the field of software reuse, in which, features are used to capture the *commodities and differences* among related business systems in a given business domain [5, 6]. The characteristics, such as hierarchical, extendable

and multi-dimensional [4], make feature more suitable to express the variations in business domain space than other technologies.

After a domain's feature models are created, they could be reused in all the business systems in this domain. If we can construct business components according to domain feature models, these identified components could also be reused when constructing various applications in the corresponding domain.

In fact, there exist an essential semantics consistency between component's functions and domain features, and a component can be considered as a sub feature space of a business domain's global feature space, i. e., the component model and domain business model are in a uniform semantics space. If we use the same feature space to describe component semantics, a direct mapping between domain model and component model is easily obtained [4]. In this section, we firstly introduce some basic concepts in traditional feature modelling, and emphatically present the idea of FD and five types of FD.

2.1. Feature and Feature Space

Feature is an ontology that is used to describe the knowledge of external world, and is represented as "*terms*" or "*concepts*" used to describe the services supplied by a specific business domain [4]. Features are hierarchical, i. e., there exist hierarchy structures between parent features and child features. According to this property, related features can form a multi-layer feature space, denoted as $\Omega = <F, D>$, in which F is feature set, and D is the set of feature dependencies between features in F.

Ω is usually represented as the form of *feature tree*, in which there is one and only one root feature f_{root}, and two directly connected features in the tree are parent and child feature, respectively. We use child (f), parent (f), ancestor (f), descendant (f) and sibling (f) to denote f's child feature set, parent feature set, ancestor feature set, descendant feature set and sibling feature set, respectively.

In a feature tree, leaf feature (without child features, or child (f) = $\varnothing$) are called atomic features, and non-leaf features (child (f) $\neq \varnothing$) are complex features. There exists *composition* relationship, or "*whole-part association*", between parent and child features, which make feature space appear in the form of a hierarchical tree.

In a specific business domain, features could be business processes, business activities, business objects, attributes and operations, etc. A feature item is an instance of a specific feature, and it describes the feature's one possible value under a given business environment [19]. Let *dom* (f) denotes the set of all feature items of feature f, and is called the "*domain*" of f. $\forall \tau \in dom\ (f)$, τ is called a value of f. For a specific business domain, *dom* (f) is a finite set.

A feature is the abstraction of all its feature items, and there exists a "*generalization-instantiation*" relationship between feature and its items. We use feature items to describe the variability of feature itself.

The instantiation of a feature f is the process of choosing a proper feature item from dom (f) for f to satisfy a specific semantics context, denoted as τ_R (f). Similarly, by instantiating a feature vector $Y = (f_1, f_2, \ldots, f_n)$, we can get an instance of Y, denoted as $\tau(Y) = (\tau(f_1), \tau(f_2), \ldots, \tau(f_n))$, in which $\tau(f_i) \in$ dom (f_i), $1 \leq I \leq n$. If we instantiate each feature $f_1, f_2, \ldots, f_n$ in Ω, we can get Ω's one instance, denoted as t (Ω). All the instances of Ω form Ω's instance set $T(\Omega)$. It is easy to know that $T(\Omega) \subseteq dom\ (f_1) \times dom\ (f_2) \times \ldots \times dom\ (f_n)$, and $\forall t \in T(\Omega)$, $t = (\tau_1, \tau_2, \ldots, \tau_n)$, in which $\tau_I \in dom\ (f_i)$ is the projection of t on f_i, also denoted as $t\ [f_i]$. t's projection on feature set X is denoted as $t\ [X]$.

2.2. Feature Dependency

In feature space $\Omega = <F, D>$, D is the dependency set between features in F. Feature dependencies can be classified into five types, as follows:

- *Whole-Part Association (WPA)*: It is the simplest FD, which depicts the *fixed composition relationship* between child and parent features. WPA explicitly behaves as the parent-child structure between features.
- *Feature Integrity Dependency (FID)*: It depicts the *variable* composition relationship between parent feature's items and child features themselves. It ensures parent feature's semantics integrity according to four selection strategies to choose specific child features for each feature item of the parent feature. FID is the exclusive one type of FD in traditional feature modelling.
- *Feature Value Dependency (FVD)*.
- *Feature Multi-Value Dependency (FMVD)*: FVD and FMVD both describe constraints between different features' instances.
- *Feature Semantics Dependency (FSD)*: It is not related to feature instantiation, but depicts the semantics constraints between features themselves.

Definition 1: Each instance (or value) of a feature f can be denoted by instances of child (f)'s one subset. $\forall \tau \in dom\ (f)$, there $\exists\ Y \subseteq$ child (f), $Y = \{f_1, f_2, \ldots, f_n\}$, and $\forall f_i \in Y$, there must exist at least one feature item $\tau_I \in dom\ (f_i)$ which makes that τ can be uniquely determined by $\tau_1, \tau_2, \ldots, \tau_n$, then Y is the essential sub-feature set of τ, denoted as $Y =$ es_set(τ).

Definition 2: There exists an FID between a feature f and a feature set Y, if and only if $Y \subseteq$ child (f), and for each feature item τ of f, it uniquely determines a subset

Y' of Y, and Y' is a subset of es_set (τ), i. e., $Y' \subseteq$ es_set (τ). The FID can be denoted as $f| \rightarrow Y$.

Actually, FID defines whether f's each child feature would be selected as an essential part of f's instance when f is instantiated. $f| \rightarrow Y$ can be classified into four detailed types, i. e., *mandatory FID*, *optional FID*, *single-selection FID* and *multiple-selection FID*, denoted as $f|^M \rightarrow g$, $f|^O \rightarrow g$, $f|^S \rightarrow Y$, $f|^T \rightarrow Y$, respectively.

- *Mandatory FID*: $f|^M \rightarrow g \Leftrightarrow \forall \tau \in dom(f)$, $g \in$ es_set (τ) must be true, i. e., g is always an indispensable part of f.
- *Optional FID*: $f|^O \rightarrow g \Leftrightarrow \exists P, Q \subseteq dom(f)$, $P \cup Q = dom(f)$, $P \cap Q = \varnothing$, which makes that $\forall \tau \in P$, there is $g \in$ es_set(τ), and $\forall \tau \in Q$, there is $g \notin$ es_set (τ). That is to say, only when f is instantiated to items in P, g is a mandatory part of f, and when f is instantiated to items in Q, g is not necessary at all.
- *SingleS election FID*: $f|^S \rightarrow Y \Leftrightarrow$ (1) $|Y| \leq |dom(f)|$; (2) there exists a partition $\{P_1, P_2, \ldots, P_n\}$ of $dom(f)$, $n = |Y|$, $P_i \cap P_j = \varnothing$, $\cup_{I=1\ldots n} P_i = dom(f)$, and $\forall P_i$, there exists one and only one $f_i \in Y$ which satisfies that $\forall \tau \in P_i, f_i \in$ es_set (τ), and $\forall f_i' \in Y \setminus \{f_i\}, f_i' \notin$ es_set (τ). This shows that for each feature item τ of f, there is only one feature in Y to be contained in τ.
- *Multiple-Selection FID*: $f|^T \rightarrow Y \Leftrightarrow$ (1). There exists a partition $\{P_1, P_2, \ldots, P_n\}$ of $dom(f)$, $P_i \cap P_j = \varnothing$, $\cup_{I=1\ldots n} P_i = dom(f)$; $\forall P_i, \exists Y' \subseteq Y$ which makes $\forall \tau \in P_i$, $\forall f_j \in Y', f_j \in$ es_set (τ), and $\forall f_j' \in Y \setminus Y', f_j' \notin$ es_set (τ). This shows that for each feature item τ of f, there are several but not all features in Y to be contained in τ.

In traditional feature modelling methodology [4, 5, 6], features are classified into *mandatory feature*, *optional feature* and *alternative features* according to whether a feature is included in its parent feature, just corresponding to g and Y in $f|^M \rightarrow g, f|^O \rightarrow g, f|^S \rightarrow Y$ and $f|^T \rightarrow Y$. But it does not explicitly associate child features with parent feature's items. We improved this shortcoming. Therefore, FID actually can be considered as the dependencies between the "*type*" of parent feature and the "*value*" of its child features, and can be called "*type-value*" dependencies, which describes what child features constitute the essential sub-features of each feature item of their parent feature.

Definition 3: Two feature sets X, Y are subsets of F in Ω. For every two instances t_1, t_2 of Ω, if $t_1[X] = t_2[X]$ always leads to $t_1[Y] = t_2[Y]$, then we call Y "feature value dependent" on X, denoted as $X \rightarrow Y$. $X \rightarrow Y$ means that one instance of X uniquely determines one instance of Y.

Definition 4: Three feature sets X, Y and Z are subsets of F in Ω, and $Z = F - X - Y$. FMVD $X \rightarrow\rightarrow Y$ exists, when and only when for arbitrary instance t in $T(\Omega)$, t's each unique projection on (X, Z) corresponds to a set of Y's instances, which are determined by X's value and not related to Z's value at all.

FVD and FMVD are consistent in essence, and they both depicts the dependencies between instances of two feature sets, called "*value-value*" dependency, i. e., one instance of a feature set unique-value or multiple-value determined instance(s) of another feature set. They usually appear between parent/child features or sibling features.

Similar to functional dependency in relational model and in database normalization, FVD and FMVD also have the characteristics of *reflexivity*, *augmentation*, *transitivity*, *pseudo transitivity*, *union* and *decomposition*, etc. According to Armstrong Axiom, we can get a feature set X's closure on FD set D, denoted as $X+$, which contains all the features that directly or indirectly dependent on features in X.

Definition 5: FSD refers to the semantics association between features. According to different feature types, FSD can also have multiple types, such as *temporal constraints* between business operation features, *association/ generalization/ composition* dependencies between business object features. Generally speaking, we use predicate $P(X)$ to denote that features contained in X should satisfy the constraints of P. The concrete expression of P is determined by its concrete type, and different types of FSD have different constraint intensity.

3. An Ext-FM Based Component Semantics Model

As pointed out before, a business component defines a sub-space of one specific business domain's feature space, so it can be denoted as $C <cid, f_{root}, F, D, PS, RS>$, in which:

- *cid* is the unique identity of C.
- f_{root} is the root feature of C's feature space, and is the ancestor of all other features in this space.
- F is the set of all features containing in component feature space except f_{root}, and satisfies $\forall f \in F$, $|dom(f)| \geq 1$.
- D is the set of feature dependencies between features in $\{f_{root}\} \cup F$.
- PS is the set of features that C provides to other components.
- RS is the set of features that C required from other components.

Features in PS are provided to be used in other components by *providing* interfaces, and features in RS are obtained from other components by *required*

interfaces. The component model should also have the following constraints:

1. $F \subseteq descendant\ (f_{root})$.
2. $\forall f \in F - RS$, if $child\ (f) \neq \varnothing$, then $\forall g \in child\ (f)$, there must be $g \in F - RS$ or $g \in RS$.
3. $\forall f \in RS$, $parent\ (f) \notin RS$.
4. $PS \neq \varnothing$.
5. $PS, RS \subseteq \{f_{root}\} \cup F$.

In Figure 1, we show a simple example of Ext-FM based component, which includes 8 features in F, 1 feature in PS and 4 features in RS. Component reusability can be represented by feature's variability, which has two different styles as following:

1. *Feature's Variability*: Only in some special business circumstances, one feature is a required constituent of component feature space, and in other one it is unnecessary.
2. *Feature Value's Variability*: One feature can be instantiated as an arbitrary feature item contained in the feature's domain.

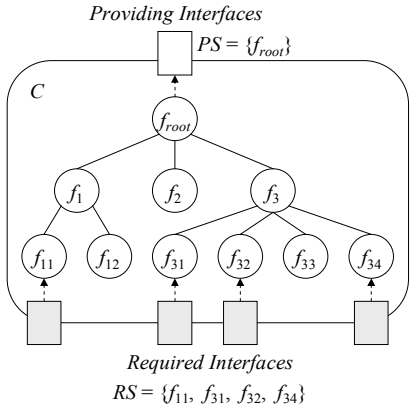


Figure 1. An example of Ext-FM based component.

In fact, feature's variability can be transformed to feature value's variability. $\forall f$ which is a non-mandatory feature, by adding one null feature item $I_\varnothing$ into *dom (f)*, f can now be regarded as a mandatory feature. If f should not be chosen in some circumstances, f can be considered to be instantiated as $I_\varnothing$.

According to the number of instances, features contained in one component can be classified into two types:

1. Fixed Feature: Which satisfies $|dom(f)| = 1$ and $dom(f) \neq \{I_\varnothing\}$;
2. Variable Feature: Which satisfies $|dom(f)| > 1$.

For variable feature f, if $I_\varnothing \in dom\ (f)$, then f is called "optional variable feature". Every variable feature in component C can also be called a "*variation point*" of C.

All the fixed features in component C constitute C's fixed part *fix_part* (C), and all the variable features constitute C's variable part *var_part*(C). By choosing one specific feature item for each variable feature in C, C is instantiated and a set of "component instances" is obtained. Component's instantiation process can actually be considered as the process of variable features' instantiation, and is usually carried out before the component is practically reused. The set of all the instances of component C is denoted as instance (C), and for $\forall f \in var\_part\ (C)$, $\forall t \in$ instance (C), denote $\rho\ (f, t)$ as the feature item that f is instantiated to in t.

A component model has two parts: The specification and the implementation. It is necessary for a reusable component to supply its implementation besides its specification form to make up of the fully executable component, so as to be reused in practical applications. Because fixed part of one component must be reused, during component design phase, the fixed part must be implemented as source code form. For every variation point, because it contains multiple feature items, these feature items are not always necessary to be implemented during design phase by component designers, and can be deferred until reuse phase by application developers. Denote f's feature item τ's implementation as *impl* (f, τ), and if τ has not yet been implemented during design phase, then *impl* (f, τ) = $\varnothing$.

In conclusion, the basic process of component reuse can be divided into three phase:

1. According to constraints of each feature dependency in D, instantiate every variable feature f, i. e., choose one feature item τ from *dom* (f).
2. If *impl* (f,τ) = $\varnothing$, implement τ using proper programming languages.
3. Construct an integrated software system by composition of all the reused components.

In Figure 2, we present an example of feature-oriented component, which contains two fixed features f_1, f_2 and three variable features f_3, f_4, f_5, with their domain $dom\ (f_1) = \{\tau_{31}, \tau_{32}, I_\varnothing\}$, $dom\ (f_2) = \{\tau_{41}, \tau_{41}\}$, $dom\ (f_3) = \{\tau_{51}, I_\varnothing\}$. There exist some FDs between its features, e. g., $\{f_3\} \rightarrow \{f_4\}$, $\{f_3\} \rightarrow \{f_5\}$.

4. Ext-FM Based Metrics for Component Reuse Performance

In this section, we present several key metrics for component reusability evaluation based on Ext-FM based component model.

As mentioned in [11], *component reusability* is defined as the synthesis of two characteristics: *Usefulness* and *usability*. Usefulness is the extents to which a reusable component will *often* be needed, and can be evaluated by the reuse scope or reuse frequency of functions supplied by the component. Usability assesses the extent to which a component is *easy* to use, regardless of its functionality, and can be

evaluated by reuse cost, reuse efficiency, etc. Here we present several key metrics for both *usability* and *usefulness* to illustrate how our Ext-FM based component model could support component performance evaluation. These metrics are:

- Granularity.
- Reuse Frequency.
- Reuse Cost.
- Reuse Efficiency.
- Cohesion.
- Coupling.

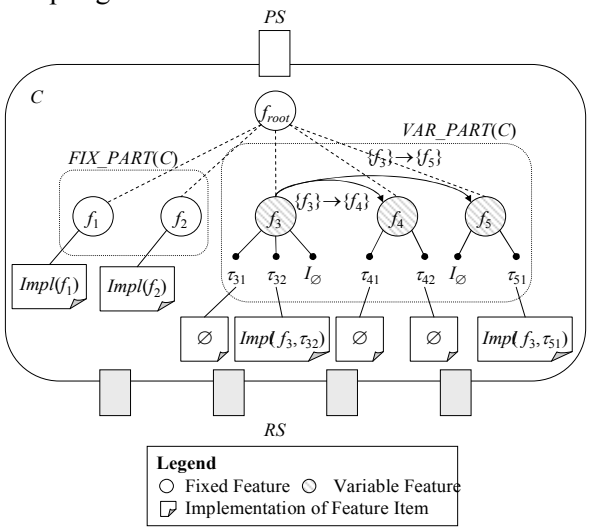


Figure 2. Variation point mechanism for Ext-FM based component model.

4.1. Granularity

Definition 6: A feature *f*'s granularity is defined as the sum of granularity of all its child features, i. e.,

$$G(f)=\begin{cases}\sum_{f_i\in \text{child}(f)} G(f_i), & \text{child}(f)\neq\varnothing\\ 1, & \text{child}(f)=\varnothing\end{cases}$$

For a component *C*, because it can be regarded as a sub space of domain feature space, its granularity can be measured by the granularity of the root feature contained in *C*, i. e., $G(C)=G(f_{root})$.

4.2. Reuse Frequency

Reuse frequency *CF* (*C*) is defined as the chance of a component *C* could be frequently used at different situations, and have two metrics: Absolute frequency CF_A (*C*) and relative frequency CF_R (*C*). CF_A (*C*) can be measured by the number of instances of *C*'s feature space, denoted as CF_A (*C*) = |*T*(*C*)|. A larger CF_A (*C*) means that *C* could be reused in more circumstances, and it has larger usability.

CF_R (*C*) describes the chance that *C* could be reused in the global feature space Ω, and can de measured by the ratio of the number of instances of *C*'s feature space and the number of Ω's instances, i. e., $CF_R(C)=|T(C)|/T(\Omega)$. The larger $CF_R(C)$ is, the more chances *C*'s feature set could be reused during the process of constructing Ω, so *C* has larger reusability.

4.3. Reuse Cost

According to component reuse process in section 3.2, component reuse cost *CRC* (*C*) can be decomposed into three parts: *Instantiation cost CI* (*C*), *implementation cost CP* (*C*) and *composition cost CC* (*C*), and we have *CRC* (*C*) = *CI* (*C*) + *CP* (*C*) + *CC* (*C*).

Component instantiation process is as follows: According to FIDs, specify whether each variable feature should be contained in the instance; then according to FVDs/FMVDs, select one specific feature item for each chosen variable features. Therefore, instantiation cost can be approximately denoted as:

$$CI(C)=C_D\times|D|+\sum_{f\in VARIABLE\_PART(C)}\left(RF_R(f,C)\times C_F\right)$$

In which C_D is the unit cost to deal with each FD, and C_F is the unit cost to choose proper feature item for one variable feature. Because only those chosen features are required to be instantiated, we add a coefficient RF_R *(f, C)* to denote the probability that feature *f* could be chosen in *C*'s instance, and it is also called the *relative reuse frequency* of *f* relative to *C*, and

$$RF_R(f,C)=|T(f)|/|T(C)|,\ 0<RF_R\ (f,C)\le 1.$$

Implementation cost refers to the cost that, after s component is instantiated, if some chosen feature items of some chosen variable features are not yet implemented (coded), then programmers should implement them. Because we will not know in advance which feature items could be chosen, we also calculate CP (*C*) approximatively by:

$$CP(C)=C_P\times\left(\sum_{f\in VARIABLE\_PART(C)}|NON\_IMPL\_SET(f)|\right)$$

in which

$$Non\_impl\_set\ (f)=\{\tau\,|\,\tau\in dom(f),\ impl(\tau)=\varnothing\}$$

and C_P is the average unit implementation cost for a feature item.

Composition cost CC (*C*) refers to the cost that *C* composes with other components to form the whole domain feature space. Component composition is the process of creating interface connection between *C* and other components' features, i. e.,

1. Create connections between *C*'s required features *RS* and other components' providing features *PS*.
2. According to the FVDs/FMVDs between *C* and other components, create associations between instances of *C* and other components. Therefore,

composition cost can be calculated by $CC(C) = C_B \times |RS(C)| + C_M \times |outer\_FD(C)|$, in which C_B, C_M are unit cost for interface connection and unit cost for FVD/FMVD matching, respectively. $Outer\_FD(C) = \{X \rightarrow Y | \exists f \in X \text{ and } f \in F(C), \exists g \in Y \text{ and } g \notin F(C)\}$.

4.4. Reuse Efficiency

Reuse efficiency refers to the efficiency when we use the component to construct feature space of a specific business. It is closely related to the size of a component's feature space, the larger the size is, and the higher the component's reuse efficiency is.

We could calculate reuse efficiency by the ratio of *C*'s feature space in the whole domain feature space, i. e.,

$$CRE(C) = |F(C)| \Big/ |F(\Omega)|$$

Because it is difficult to specify Ω's scope, we could use *C*'s implementation absolute granularity to indirectly represent *C*'s reuse efficiency, i. e., $CRE(C) = NAG(C)$. The larger *C*'s granularity is, the more contributions to construct the whole domain feature space *C* has, so the higher *C*'s reuse efficiency is.

4.5. Cohesion

Cohesion and coupling are two key metrics used to evaluate component performance in literatures. In our component model, five types of FDs are the main reasons leading to cohesion in component and coupling between components. Here we use the concept of *Feature Dependency Density* (FDD) to evaluate cohesion and coupling.

Definition 7: A feature *f*'s inner FDD depicts the intensity (or, semantics closeness) of FDs between *f* and its child features, and between *f*'s child features, denoted as:

$$Inner\_FDD(f) = \alpha_1 \times FID\_Ds(f) + \alpha_2 \times FVD\_Ds(f) + \alpha_3 \times FSD\_Ds(f) + \alpha_4 \times child\_Ds(f).$$

If $child(f) = \varnothing$, then $inner\_FDD(f) = 1$. In this definition:

- *FID_Ds (f)*: The FDD between *f* and its child features caused by FIDs, and we could calculate it by the average of each child feature's reuse frequency relative to *f*, denoted as:

$$FID\_Ds(f) = \frac{\sum_{g \in child(f)} RF_R(g, f)}{n}, \quad n = |child(f)|$$

A child feature is more frequently contained in *f*, the more cohesion between this child feature and *f* has.

- *FVD_Ds (f)*: The FDD between *f*'s all child features caused by FVD/FMVDs, and we could get it by calculating the ratio of the number of FVD/FMVDs existing between features of child *(f)* and the possible largest number of FVD/FMVDs between child *(f)*. The larger the number of FVD/FMVDs in child *(f)*, the higher cohesion *f* has. Because FVD/FMVD can be represented as $X \rightarrow Y$ or $X \rightarrow\rightarrow Y$, so there will exist FVD/FMVDs with the number of at most $C_n^1 + C_n^2 + \ldots + C_n^{n-1} = 2^n - 2$, therefore:

$$FVD\_Ds(f) = \frac{\left|\{X \rightarrow (\rightarrow) Y | X, Y \subset child(f)\}\right|}{2^n - 2}$$

- *FSD_Ds (f)*: The FDD between *f*'s child features caused by FSD, and it has similar measurement with *FVD_Ds (f)*, denoted as:

$$FVD\_Ds(f) = \frac{\sum_{P(X),\ X \subseteq child(f)} Complexity(P(X))}{2^n - n - 1}$$

- *Child_Ds (f)*: The average of the *inner FDD* of *f*'s all child features, i. e.,

$$child\_Ds(f) = \frac{\sum_{g \in child(f)} inner\_FDD(g)}{n}$$

- $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ are the coefficient for the above four FDD, and $\sum_{i=1}^{4} \alpha_i = 1$.

Cohesion of component C depicts the degree of semantics closeness of *C*'s all features, which can be measured by inner FDD of *C*'s foot feature, i. e.,

$$Cohesion(C) = inner\_FDD(f_{root})$$

We could calculate the root feature's cohesion from leaf features and recursively top wards until to f_{root}. We will not present the detailed calculation here.

4.6. Coupling

Definition 8: A feature *f*'s outer FDD depicts the intensity of FDs between *f* and other features, and can be calculated by:

$$Outer\_FDD(f) = \beta_1 \times WPA\_Ds(f) + \beta_2 \times FID\_Ds(f) + \beta_3 \times FVD\_Ds(f) + \beta_4 \times FSD\_Ds(f)$$

In which:

- *WPA_Ds* (*f*) is the FDD between *f* and its child features caused by WPA, i. e., the degree that *f* depends on outer features to realize its own functions, measured by $WPA\_Ds(f) = 1 - \frac{1}{n}$, $n = child(f)$.
- The meanings of *FID_Ds*(*f*) is same as the *FID_Ds*(*f*) in Definition 15.
- *FVD_Ds*(*f*) is the FDD between *f* and other features caused by FVD/FMVD, and can be calculated by:

$$FVD\_Ds(f) = 1 - \frac{1}{\left|\{X \rightarrow (\rightarrow) Y | f \in X, f \notin Y\}\right|}$$

- *FSD_Ds* (*f*) is the FDD between *f* and other features caused by FSD, and can be calculated by:

$$FSD\_Ds\ (f) = 1 - \frac{\left|\left\{\mathrm{P}(X) \middle| f \in \mathrm{P}(X)\right\}\right|}{\sum_{\mathrm{P}(X) \in FSDs(f)} Complexity\left(\mathrm{P}(X)\right)}$$

- $\beta_1, \beta_2, \beta_3, \beta_4$ are the coefficient of the above four FDD, and $\sum_{i=1}^{4} \beta_i = 1$.

Component *C*'s coupling depicts the closeness degree of features in *C* and in other components. We could calculate it by the average of *C*'s all the features' outer FDD, denoted as:

$$Coupling\ (C) = \frac{1}{\left|F(C)\right|} \sum_{f \in F(C)} outer\_FDD(f)$$

5. Experiments and Comparisons with Related Works

We have applied the model and metrics in practical design and implementation of a component-based Enterprise Resource Planning (ERP) system, named NERP, during which the approach presented in this paper has shown its effectiveness significantly.

Taking a sub-system "procurement management system" in ERP as an example, we have identified 25 components from its business models, listed in Table 1. Due to limited space, here we only present the structure of a component C_{21} in Figure 3, and Table 2 shows the meanings of each feature/feature item contained in C_{21} (for more details about other components, please see [18] for reference). Using the metrics in this paper, we get the statistical data (performance) of each component, shown in Table 3.

The following are some comparisons between our Ext-FM based component model and other component models in literatures:

1. 3C, Wright, JBCOM and other component models are much closer to formal semantics level, and they lack of precise descriptions on semantics in problem domain, which lead to a gap between component models and domain business models. Our model adopt extended feature modelling as a tool to describe component semantics and to create direct semantics mapping between component models and domain models, i. e., enterprise's business models and component models are uniformly represented by the form of feature space, and a component is considered as a sub space of domain model, therefore, realizing consistency between the two model levels.
2. Many component models usually use highly formal way to describe semantics, such as *Z*, predicate logic, etc, to support the automation of component-based software reasoning, validation, and evolution, etc. This way leads to more highly complexity and much poorer readability and understandability. Our model adopts semi-formal method, i. e., feature modelling, which is easier to be used and understood, and can be integrated with activities in domain analysis phase of software reuse.
3. Compared with the feature-oriented component modelling in literatures, our model mainly supports evaluation on component performance, therefore, we ignore some complex and unnecessary information, and pay more attention to component's "content", i. e., business semantics, and emphasize on component semantics and semantics dependencies, which makes modelling easier.
4. Traditional feature modelling methods have limitations, which primarily behaves that the way to depict semantics variability is very simplex, and cannot express various variable semantics completely. Our model extends traditional feature modelling and gets another three types of FD (FVD, FMVD, FSD) besides FID.

The most significant advantage of our model is that, most of component models are difficult to support evaluating component performance, while our model can do that gracefully and conveniently based on the analysis of component feature space, just as several example metrics presented in section 4.

Table 1. Component set for purchasing domain process.

Component ID	Component Name
C_1	Acquiring Procurement Requirements
C_2	Operating on Procurement Requirements
C_3	Planning Procurement
C_4	Auditing Procurement Plans
C_5	Adjusting Procurement Plans
C_6	Managing Supplier Information
C_7	Managing Evaluation Strategies
C_8	Evaluating Suppliers
C_9	Public Bidding Management for Procurement
C_{10}	Quotation Management
C_{11}	Supplier Selection for Specific Plans
C_{12}	Creating New Procurement Orders
C_{13}	Allocating Procurement Orders
C_{14}	Modifying Procurement Orders
C_{15}	Querying Procurement Orders
C_{16}	Auditing Procurement Orders
C_{17}	Updating Procurement Orders
C_{18}	Cancelling Procurement Orders
C_{19}	Managing Product Arrival Plans
C_{20}	Monitoring the Products on the way
C_{21}	Product Arrival Informing
C_{22}	Product Testing Management
C_{23}	Inventory In
C_{24}	Reimbursing Products
C_{25}	Account Payable Management

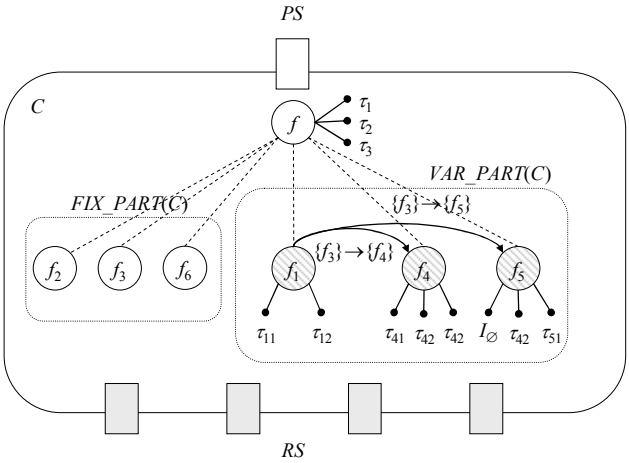


Figure 3. Structure of the component "product arrival informing".

Table 2. Feature, feature item and feature dependencies in Figure 3.

Features	Meanings	Feature Items	Meanings	Feature Dependencies
f	Product Arrival Informing	τ_1	Product Arrival without Order	$f\|^M \rightarrow \{f_1, f_4, f_5, f_6\}$
		τ_2	Product Arrival with Order	$f\|^O \rightarrow \{f_2, f_3\}$
		τ_3	Import Product Arrival	$f_4 \rightarrow f_5$
f_1	Product Arrival Bill Management	τ_{11}	Domestic	ExeOrder(f_1, f_2, f_3)
		τ_{12}	Import	ExeOrder(f_1, f_4)
f_2	Auditing Imported Product Arribal Form			ExeOrder(f_1, f_5)
f_3	Customs Declaration for Importing Products			ExeOrder(f_4, f_6)
f_4	Inform to Measure	τ_{41}	Measure by Sampling	ExeOrder(f_5, f_6)
		τ_{42}	Measure All	
		τ_{43}	Measure None	
f_5	Inform to Quality Checking	τ_{51}	Check by Sampling	
		τ_{52}	Check All	
		τ_{53}	Check None	
f_6	Inform to Inventory In			

5. Conclusion

In this paper, we presented a new component semantics model based on extended feature modelling, to solve the problem that current component models cannot support us to evaluate component reuse performance precisely and easily.

This model has been used in the design and development of several Enterprise Resource Planning (ERP) systems, and the practical experiences have proved its validity and effectiveness on variability representation and performance evaluation. We are sure that this component model will be a useful supplement for the research and practice of component-based software reuse.

Table 3. Component metrics for purchasing domain process.

C	$G(C)$	$CF_R(C)$	$CRE(C)$	$CI(C)$	$CP(C)$	$CC(C)$	$CRC(C)$	$Cohesion(C)$	$Coupling(C)$
C_1	1	0.64	0.014	1.27	3.84	5.72	10.83	1.00	0.14
C_2	3	0.64	0.043	3.83	7.40	11.03	22.26	0.75	0.28
C_3	6	0.82	0.086	7.67	12.23	18.23	38.13	0.68	0.47
C_4	2	1.00	0.028	2.55	3.29	4.90	10.74	0.94	0.12
C_5	4	0.73	0.057	5.11	5.21	7.76	18.08	0.85	0.20
C_6	2	1.00	0.028	2.55	14.77	22.02	39.34	0.91	0.57
C_7	2	0.57	0.028	2.55	3.82	5.69	12.06	0.82	0.14
C_8	1	0.66	0.014	1.27	6.92	10.32	18.51	1.00	0.27
C_9	3	0.51	0.043	3.83	5.85	8.72	18.4	0.81	0.22
C_{10}	1	0.53	0.014	1.27	5.85	8.72	15.84	1.00	0.22
C_{11}	2	0.94	0.028	2.55	9.56	14.25	26.36	0.79	0.37
C_{12}	1	1.00	0.014	1.27	15.33	22.86	39.46	1.00	0.59
C_{13}	1	0.67	0.014	1.27	2.39	3.56	7.22	1.00	0.09
C_{14}	1	0.72	0.014	1.27	5.81	8.66	15.74	1.00	0.22
C_{15}	2	1.00	0.028	2.55	10.04	14.97	27.56	0.96	0.39
C_{16}	1	1.00	0.014	1.27	2.85	4.25	8.37	1.00	0.11
C_{17}	1	1.00	0.014	1.27	6.08	9.06	16.41	1.00	0.23
C_{18}	2	0.62	0.028	2.55	4.20	6.26	13.01	0.74	0.16
C_{19}	3	0.58	0.043	3.83	3.97	5.92	13.72	0.76	0.15
C_{20}	1	0.57	0.014	1.27	1.99	2.96	6.22	1.00	0.07
C_{21}	6	0.81	0.086	7.67	7.78	11.60	27.05	0.77	0.30
C_{22}	4	0.75	0.057	5.11	9.22	13.75	28.08	0.68	0.35
C_{23}	5	0.71	0.072	6.39	10.70	15.95	33.04	0.64	0.41
C_{24}	5	0.68	0.072	6.39	3.26	4.86	14.51	0.83	0.12
C_{25}	9	0.84	0.130	11.51	10.55	15.73	37.79	0.76	0.41

Acknowledgement

Research works in this paper are fully supported by the Specialized Research Fund for the Doctoral Program of Higher Education (SRFDP) in China (Grant No. 20030213027) and the Natural Science Foundation of China (Grant No. 60573086).

References

[1] Allen R. and Garlan D., "A Formal Basis for Architectural Connection," *ACM Transactions on Software Engineering and Methodology*, vol. 6, no. 3, pp. 213-249, 1997.

[2] Ben Natan R., *CORBA_A Guide to the Common Object Request Broker Architecture*, McGraw-Hill, 1995.

[3] Jacobson I., Griss M., and Jonsson P., *Software Reuse: Architecture, Process and Organization for Business Success*, Addison-Wesley, 1997.

[4] Jia Y., "The Evolutionary Component-Based Software Reuse Approach," *PhD Dissertation*, Graduation School of Chinese Academy of Sciences, 2002.

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

[6] Kang K. C., Kim S., Lee J., Kim K., Shin E., and Huh M., "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures," *Annals of Software Engineering*, vol. 5, no. 1, pp. 143-168, 1998.

[7] Li K., Guo L., Mei H., and Yang F., "An Overview of JB (JadeBird) Component Library

System JBCL," *in Proceedings of the 24th International Conference on Technology of Object-Oriented Languages*, IEEE Computer SocietyPress, Los Alamitos, USA, pp. 206-213, 1997.

[8] Mei H., "A Component Model for Perspective Management of Enterprise Software Reuse," *Annals of Software Engineering*, vol. 11, no. 1, pp. 219-236, 2001.

[9] Microsoft, "Distributed Component Object Model Protocol COM/1.0," available at: http://www.microsoft.com/library, 1996.

[10] Mili A., Fowler S., Gottumukkala R., and Zhang L., "An Integrated Cost Model for Software Reuse," *in Proceedings of the 22nd International Conference on Software Engineering*, Limerick, Ireland, ACM Press, pp. 157-166, June 2000.

[11] Mili H., Mili A., Yacoub S., and Addy E., *Reuse-Based Software Engineering: Techniques, Organization, and Controls*, John Wiley and Sons, 2002.

[12] Sindre G., Conradi R., and Karlsson E. A., "The REBOOT Approach to Software Reuse," *Journal of Software and Systems*, vol. 33, no. 3, pp. 201-212, 1995.

[13] Sparling M., "Lessons Learned through Six Years of Component-Based Development," *Communications of the ACM*, vol. 43, no. 10, pp. 47-53, 2000.

[14] Sun Corporation, "Enterprise JavaBeans Specifications Version 1.1," available at: http://java.sun.com/products/ejb/docs.html, 1998.

[15] Szyperski C., *Component software: Beyond Object-Oriented Programming*, Addison-Wesley, 1998.

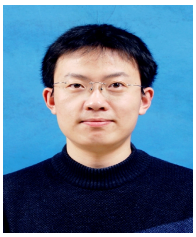
[16] Tracz W., "Implementation Working Group Summary," *in Proceedings of Reuse in Practice Workshop*, IDA Document D-754, Pittsburgh, PA, pp. 10-19, 1990.

[17] Vitharana P., Jain H., and Zahedi F., "Strategy-Based Design of Reusable Business Components," *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, vol. 34, no. 4, pp. 460-474, 2004.

[18] Wang Z. J., "Optimization Technology for Reconfiguration and Reuse of Enterprise Software and Applications," *PhD Dissertation*, Harbin Institute of Technology, 2005.

[19] Wang Z. J., Xu X. F., and Zhan D. C., "A Component Optimization Design Method Based on Variation Point Decomposition," *in Proceedings of the 3rd ACIS International Conference on Software Engineering, Research, Management and Applications*, Michigan, USA, pp. 399-406, 2005.

[20] Wu Q., Chang J., Mei H., and Yang F., "JBCDL: An Object-Oriented Component Description Language," *in Proceedings of 24th International Conference on Technology of Object-Oriented Languages*, Los Alamitos, CA, USA, IEEE Computer Society Press, pp. 198-205, 1997.



Zhongjie Wang is a lecture in computer application technology in School of Computer Science and Technology at Harbin Institute of Technology (HIT), China. His research interests include software engineering, software reuse, software reconfiguration, software component related techniques.

Xiaofei Xu is a professor and dean of School of Computer Science and Technology at Harbin Institute of Technology (HIT), China. He is a member of the CIMS Subject Expert Committee of Chinese 863 National High-Tech Program; standing member of Council of China Computer Society; senior member the US Society of Manufacturing Engineer (SME); member of German Society of Operational Research; director of Council of Heilongjiang Province Computer Society; member of Editorial of Journal of CIMS and Journal of Harbin Institute of Technology (New Series). His research interests include computer integrated manufacturing system (CIMS), database systems, supply chain management, agile virtual enterprises, management and decision information system, and knowledge engineering. He has published more than 200 academic papers and 3 books. He has implemented over 20 projects from Key Projects of National 863 Hi-tech Project, National Science Foundation, Ministry/Province Science Foundation, Province Outstanding Youth Foundation Project and international cooperative Project.

Dechen Zhan is a professor in School of Computer Science and Technology at Harbin Institute of Technology (HIT), China. His research interests include computer integrated manufacturing system, enterprise resource planning, decision support systems, software reuse and reconfiguration.