A feature oriented approach to mapping from domain requirements to product line architecture

Chongxiang Zhu, Yuqin Lee, Wenyun Zhao , Jingzhou Zhang

Software Engineering Lab, Computer Science and Technology Department, Fudan University,Shanghai,200433,China,
cxzhu@fudan.edu.cn; li_yuqin@yahoo.com.cn;
wyzhao@fudan.edu.cn; zjz@ssc.stn.sh.cn

Abstract. *Domain requirements dependencies have very strong influence on all development processes of member products in a software product line(SPL), especially product line architecture. There are some feature oriented approaches to managing requirements dependencies in software product lines. However, few of them deal with mapping from requirements to product line architecture. This paper presents an approach to analyzing domain requirements dependencies' influence on product line architecture. Not only a feature dependencies classification is defined, but also mapping rules from requirements to features and mapping rules from features to architecture are developed to decrease inconsistencies between assets and increase reuse in a product line. This approach gives out a formal method to validate whether consistencies between product assets are coincident with those between requirements based on mapping rules. A case study for spot and futures transaction domain is described to illustrate this approach.*

Keywords: domain requirements, feature dependencies, mapping rules, product line architecture

1. Introduction

One of the effective approaches to realizing large scale software reuse is product line engineering(PLE). Its goal is to support the systematic development of a set of similar software systems by understanding and controlling their common and distinguishing characteristics[1]. Adopting a proper architecture which comprises a map that guides developers when new product instances are derived[17] is one of the keys to achieve successful PLE activities, for the architecture not only reflects requirements dependencies, but also support realizing requirements commonality and variability [18].

Many researchers have recognized that individual requirements are seldom independent of each other, and various kinds of dependency exist among them [2,3, 4, 5, 6, 7, 8, 9]. A software product line deals with a set of member products, so more dependencies exist among requirements of SPL than that of an individual application. Dependencies must be well managed during analysis phase and along with system evolvement, otherwise can lead to conflicts and inconsistencies .

A feature is a set of tight-related requirements from stakeholders' viewpoints. Feature dependencies reflect requirements dependencies. There are some feature oriented approaches to managing requirements dependencies in software product lines. However, few of them deal with mapping from requirements to product line architecture.

The goal of our work is to analyze domain requirements dependencies' influence on product line architecture. Not only a feature dependencies classification is defined, but also mapping rules from requirements to features and mapping rules from features to architecture are developed to validate whether consistencies between product assets are coincident with those between requirements. The approach supports easy generation of product line architecture from domain requirements, and keeps consistencies between product line assets along with those between domain requirements.

The paper is organized as follows: Section 2 discusses related work. Section 3 defines classification of features dependencies in a SPL, Section 4 proposes mapping rules from domain requirements to features and from features to product line architecture to decrease inconsistencies. Based on mapping rules discussed in section 4, section 5 validates consistencies between product line assets coincident with those between requirements. Section 6 illustrates and analyses our approach by an example. Section 7 draws a conclusion and some suggestions for future work.

2. Related work

There are some approaches dealing with feature dependencies and exploiting feature dependencies' influence on product line architecture. Most of them focus on feature dependencies modeling, analysis or management. Only few of them deal with mapping from requirements to features and/or to assets.

J. Savolainen, et al. [18] presented a set of rules that map the selection constraint values of requirements to

the selection constraint values of features which in turn map on to the selection constraint values of architectural assets. They assumed that leaf node features below root features are either mandatory or variable and that these map directly on to mandatory or variable assets. But actually, features and assets have also n*m relation, i.e. multiple features may depend on the same asset, and a feature may depends on multiple assets.

Hassan Gomaa, Michael E. Shin [10] proposed a multiple-view meta-model for software product lines to describe how each view relates semantically to other views. The meta-model depicts life cycle phases, views within each phase, and meta-classes within each view. The relationship between the meta-classes in the different views was described. Consistency checking rules were defined based on the relationships among the meta-classes in the meta-model. The approach dealt with feature dependencies.

K. Lee and K. C. Kang [11] extended the feature modeling to analyze feature dependencies that are useful in the design of reusable and adaptable product line components, and presented design guidelines based on the extended model. They gave out six types of feature dependencies which have significant influences on the design of product line assets. They emphasized on solutions to hide variable features from their usage client features.

H. Ye and H. Liu [12] gave out a matrix-based approach to modeling feature dependencies in a scalable way. Three hierarchical relationships and three non-hierarchical relationships had been identified, but the dependencies in each class are not representative.

W. Zhang, H. Mei et al [9] identified static, dynamic and specification level feature dependencies. They emphasized on feature dependencies modeling and interaction between different dependencies types. Dependencies at specification level can be combined to dynamic dependency classes.

We propose a feature dependencies classification by identifying them both in static and dynamic way. Mapping rules from requirements to features and from features to architecture are developed to decrease inconsistencies between assets and increase reuse in a product line. Meanwhile we validate whether consistencies between product line assets are coincident with those between requirements based on mapping rules.

3.Classification of feature dependencies

3.1 Feature variability

Features in a software product line may be classified to two types: mandatory and variable features.

Mandatory features are those which must be present in all member products in a software product line.

Variable features are those which may not present in all member products in a software product line.

Mandatory features illustrate product family commonality, and *Variable* features illustrate product family variability. Compared with commonality, variability is much more complex and takes more attention of research.

According to variable characteristics, variable features can be classified to the following types:

***Alternative*:** If one and only one feature have to be chosen for a member product from a feature set, the features of the set are called alternative features. If we use cardinality to describe it, the cardinality of the set is 1. The term cardinality refers to the number of basic members which can be chosen from a set.

***Mutually exclusive*:** If at most one feature can be chosen for a member product from a feature set, the features of the set are called mutually exclusive features. The cardinality of mutually exclusive set is 0..1.

***Multiple alternatives*:** If at least one feature has to be chosen for a member product from a feature set, the features of the set are called multiple alternative features. The cardinality of multiple alternatives set is 1..*.

***Optional*:** If a feature may or may not be chosen for a member product, this feature is called optional feature. The cardinality of optional features set is 0..*.

The variability types imply relations among the features in a set. For example, features in one alternative set imply that the features have alternative relation, and more than one feature can not be chosen in a member product.

Feature variability will be used as selection constraints of features to produce SPL. Feature variability is denoted as:

SCf= {mandatory, alternative, mutually exclusive, multiple alternatives, optional}.

3.2 Feature dependencies

The dependencies (dep) among features can be classified to static and dynamic dependencies.

The static dependencies reflect hierarchical feature relations and static constraints among features in the same level. Static dependencies include decomposition, generalization, and static constraints. Static constraints include required and excluded, which reflect dependencies between peer features, especially different variants in one variable point. Decomposition and generalization reflect dependencies between parent and child features.

In addition to static dependencies, there are several dynamic dependencies reflecting operational relations among features. Dynamic dependencies include serial, collateral, synergetic and change.

According to our classification method, feature dependencies are denoted as the following in a formal way.

Dep_f= {decomposition, generalization, static constraints, serial, collateral, synergetic, change }
Static constraints= {excluded, required }
Change= {state change, behave change, data change, code change }

Define 1: Dependencies between two features other than excluded are transitive. It means that two conjoint direct dependencies can be united to produce an implied dependency. All these dependencies other than excluded are called *requisite* dependencies. For instance, feature A has a serial dependency with feature B, and feature B has a change dependency with feature C, we can know that feature A has implied *requisite* dependency with feature C. This dependency may be repeated to find out implied dependencies between two disconnected features.

4. Formal representation of mapping rules

Requirements and features have n*m relation, i.e. multiple requirements may depend on the same feature, and a requirement may depend on multiple features. dependencies between requirements and those between features are not identical. Anyway we may assume that requirements have same dependencies kinds as features. So variability and dependencies of features discussed above are applicable for requirements. Features and assets have also k*l relation The relationship between selection constraints of domain requirements, features and software product line assets is illustrated in figure1.

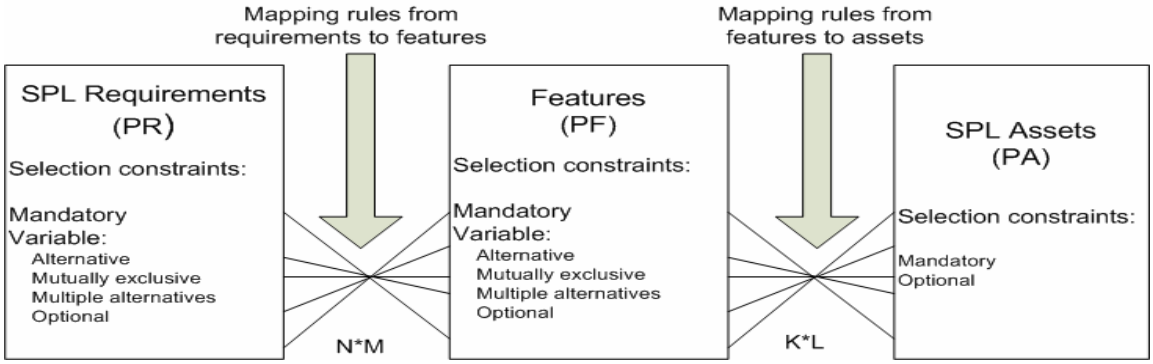

Figure1**: Relationship between selection constraints of domain requirements, features and SPL assets**

4.1 Formal representation of products, requirements, features, assets and selection constraints

We organize software product line as PL, which produces several products. The number of products in software product line is v.

1) $PL \equiv \{P_i \mid 1<i<v\}$

Product line PL realizes domain requirements. Domain requirements are refined to product line requirements PR. The number of product line requirements is m. Requirements define (def) software product line. Each requirement has a unique number.

2) $PR \equiv \{R_i \mid 1<i<m \wedge def(PL, R_i)\}$

3) $\forall R_i \in PR, R_j \in PR \mid (i=j) \vee (R_i \neq R_j) \wedge (i \neq j)$

Product line requirements specify features PF. The number of product line features is n. Each feature represents a unique feature in software product line.

4) $PF \equiv \{F_i \mid 1<i<n \wedge specify(PR, F_i)\}$

5) $\forall F_i \in PF, F_j \in PF \mid (i=j) \vee (F_i \neq F_j) \wedge (i \neq j)$

Product line features specify product line assets PA. The number of product line assets is u. Each asset has a unique number.

6) $PA \equiv \{A_i \mid 1<i<u \wedge specify(PF, A_i)\}$

7) $\forall A_i \in PA, A_j \in PA \mid (i=j) \vee (A_i \neq A_j) \wedge (i \neq j)$

Based on discussion on section 3, every requirement has a selection constraint SC_r.

8) $SC_r \in$ {*mandatory, alternative, mutual exclusive, multiple alternatives, optional*}

Every feature has also a selection constraint SC_f.

9) $SC_f \in$ {*mandatory, alternative, mutual exclusive, multiple alternatives, optional*}

Every asset has a selection constraint SC_a.

10) $SC_a \in$ {*mandatory, optional*}

4.2 Mapping rules from requirements to features

If features are one by one mapping of requirements, dependencies between requirements are the same as those between features. Features variability will be gained according to the following mapping rules. Features consistencies will inherit requirements consistencies.

Mapping rule 1: (mandatory requirement to mandatory feature) If a feature is specified by a mandatory requirement, the variability (selection constraints) of the feature is mandatory.

11) $\exists F_i \in PF, \exists R_j \in PR$, specify $(R_j, F_i) \wedge SC_r(R_j) = mandatory \Rightarrow SC_f(F_i) = mandatory$

Mapping rule 2: (variable requirements to optional feature) If a feature is specified by a variable requirement, the variability of the feature is optional.

12) $\exists F_i \in PF, \exists R_j \in PR$, specify $(R_j, F_i) \wedge ((SC_r(R_j) = \text{alternative}) \vee (SC_r(R_j) = mutual\ exclusive) \vee (SC_r(R_j) = multiple\ alternatives) \vee (SC_r(R_j) = optional)) \Rightarrow SC_f(Fi) = optional$

If features are one to more mapping of requirements, it means that several features are specified by one requirement. Dependencies between specified features and other features inherit dependencies between the requirement and other requirements. Dependencies between specified features are inquired, because these features have to realize the requirement together. Features variability will be gained according to the following mapping rules.

Mapping rule 3: (mandatory requirement to mandatory features) If some features are specified by a mandatory requirement, the variability of the features is mandatory.

13) $\exists PF_k \subseteq PF, \exists R_j \in PR, \forall F_i \in PF_k$, specify $(R_j, F_i) \wedge SC_r(R_j) = mandatory \Rightarrow \forall F_i \in PF_k, SC_f(F_i) = mandatory$

Mapping rule 4: (variable requirements to optional features) If some features are specified by a variable requirement, the variability of the features is optional.

14) $\exists PF_k \subseteq PF, \exists R_j \in PR, \forall F_i \in PF_k$, specify $(R_j, F_i) \wedge ((SC_r(R_j) = alternative) \vee (SC_r(R_j) = mutual\ exclusive) \vee (SC_r(R_j) = multiple\ alternatives) \vee (SC_r(R_j) = optional)) \Rightarrow SC_f(F_i) = optional$

Mapping rule 5: (mutually exclusive requirements to mutually exclusive features) Requirements in a mutually exclusive set should better specify different features, and the variability of the features is *mutually exclusive*. Otherwise features will have redundancy, and lead to redundancy assets. For example, if two mutually exclusive requirements specify a feature, whether which one requirement is selected, the feature being specified by two mutually exclusive requirements has to be selected.

15) $\exists R_i, R_j \in PR, \exists F_i, F_j \in PF, SC_r(R_i) = SC_r(R_j) = mutually\ exclusive \wedge$ specify $(R_i, F_i) \wedge$ specify $(R_j, F_j) \Rightarrow (i \neq j) \wedge SC_f(F_i) = SC_f(F_j) = mutually\ exclusive$

Mapping rule 6: (optional requirement to mandatory feature) If a feature is specified by an optional requirement with which a mandatory requirement has **requisite** dependency, the variability of the feature is mandatory.

16) $\exists F_i \in PF, \exists R_j \in PR$, specify $(R_j, F_i) \wedge \exists R_k \in PR$, required$(R_k, R_j) \wedge SC_r(R_j) = optional \wedge SC_r(R_k) = mandatory \Rightarrow SC_f(F_i) = mandatory$

Mapping rule 7: (mandatory requirement or all optional requirements to a mandatory feature) If a feature is specified by all optional requirements, or by requirements among which at least one is mandatory, the variability of the feature is mandatory.

17) $\exists F_i \in PF, \exists PR_k \subseteq PR$, specify $(PR_k, F_i) \wedge ((\forall R_l \in PR, SC_r(R_l) = optional \Rightarrow R_l \in PR_k) \vee (\exists R_l \in PR_k, SC_r(R_l) = mandatory)) \Rightarrow SC_f(F_i) = mandatory$

Mapping rule 8: (optional requirement to optional feature) If any requirement in the set of requirements specifying a feature is optional, the variability of the feature is optional.

18) $\exists F_i \in PF, \exists PR_k \subseteq PR$, specify $(PR_k, F_i) \wedge (\forall R_l \in PR_k, SC_r(R_l) = optional) \Rightarrow SC_f(F_i) = optional$

After these mapping rules are applied on requirements, variability of features will not be the same as that of specified requirements.

4.3 Mapping rules from features to SPL assets

Features are refined to features tree and dependencies graph. We only realize leaf features to product line assets. Parents can be composed by leaf features. We can not select features freely to produce assets, because dependencies exist among features.

Features and assets have also k*l relation, i.e. multiple features may depend on the same asset, and a feature may depends on multiple assets.

Mapping rule 9: (mandatory features to mandatory asset) If at least one feature in the set of features specifying an asset is mandatory, the variability of the asset is mandatory.

19) $\exists A_i \in PA, \exists PF_k \subseteq PF$, specify $(PF_k, A_i) \wedge (\exists F_j \in PF_k, SC_f(F_j) = mandatory) \Rightarrow SC_a(A_i) = mandatory$

Mapping rule 10: (optional features to mandatory asset) If an asset is specified by an optional feature with which a mandatory feature has ***requisite*** dependency, the variability of the asset is mandatory.

20) $\exists A_i \in PA, \exists F_j \in PF$, specify $(F_j, A_i) \wedge \exists F_k \in PF$, required$(F_k, F_j) \wedge SC_f(F_j) = optional \wedge SC_f(F_k) = mandatory \Rightarrow SC_a(A_i) = mandatory$

Mapping rule 11: (optional features to optional asset) If any feature in the set of features specifying an asset is optional, the variability of the asset is optional.

21) $\exists A_i \in PA, \exists PF_k \subseteq PF$, specify $(PF_k, A_i) \wedge (\forall F_l \in PF_k, SC_f(F_l) = optional) \Rightarrow SC_a(A_i) = optional$

Mapping rule 12: (dependency based constraints) If two features have excluded dependency, the assets are specified by those two features should better be

separated and inherit excluded dependency. Similarly, if feature A has inquire dependency with feature B, the assets are specified by those two features may be gather together to produce an integrated asset. The later is not compelling.

22) $\exists F_i, F_j \in PF, \exists A_i, A_j \in PA$,excluded$(F_i, F_j) \wedge$ specify $(F_i, A_i) \wedge$ specify $(F_j, A_j) \Rightarrow (i \neq j)$

After these mapping rules are applied from features to assets, variability of specified assets will not be the same as that of features.

Variability change will guide design of software product line architecture and components. Those mandatory assets changed from optional requirements should be carefully regarded as SPL commonality too.

5. Validating consistencies between product line assets coincident with those between domain requirements

In large software product line, inconsistency detection and handling is time consuming. After inconsistency detection and handling, we assume that consistency of requirements is satisfied. Then we map requirements to SPL assets. From domain requirements to features, then to SPL assets, three different hierarchies create a need for consistency support. The mapping rules proposed above guarantee consistencies inheritance.

5.1 Validating consistencies between requirements coincident with those between mapped features

If features are one by one mapping of requirements based on mapping rules 1 and 2, variability of features is the same as that of requirements specifying them. Dependencies between features are the same as those between requirements. So features consistencies (consist) will inherit requirements consistencies.

23) $\exists PR_k \subseteq PR$, $\forall R_i, R_j \in PR_k$, $\exists F_i, F_j \in PF$, specify (R_i, F_i), specify $(R_j, F_j) \Rightarrow (SC_f(F_i) = SC_r(R_i)) \wedge (SC_f(F_j) = SC_r(R_j)) \wedge (dep(R_i, R_j) = dep(F_i, F_j)) \Rightarrow consist(R_i, R_j) = consist(F_i, F_j)$

If features are one to more mapping of requirements based on mapping rules 3 and 4, it means that several features are specified by one requirement. Those features have same variability as that of requirement specifying them. Dependencies between specified features and other features inherit dependencies between the requirement and other requirements. Dependencies between specified features are "inquire", because these features have to realize the requirement together. Consistencies between specified features and other features will inherit requirements consistencies.

Every specified feature has the same relation with the specifying requirement. According to23),

24) $\exists R_i \in PR$, $\exists PF_k \subseteq PF$, specify $(R_i, PF_k) \Rightarrow \forall F_j \in PF_k$, $SC_f(F_j) = SC_r(R_i) \wedge (dep(R_i, PR) = dep(F_i, PF) \Rightarrow consist(R_i, PR) = consist(F_i, PF) \wedge (\forall F_l, F_m \in PF_k, dep(F_l, F_m) = \textit{required})$

dep(Ri, PR) represents dependencies between Ri and other requirements in PR.

dep(Fi, PF) represents dependencies between Fi and other features in PF.

consist(Ri, PR) represents consistencies between Ri and other requirements in PR.

consist(Fi, PF) represents consistencies between Fi and other features in PF.

For mapping rule 5, based on principle of 23), features consistencies will inherit requirements consistencies.

For mapping rule 6, we assume F_i and PF are inconsistent, and feature specified by R_k is F_k. F_k and PF has inconsistency too, then R_k and PR have inconsistency. This result contradicts with precondition of requirements consistencies. So F_i and PF are consistent.

For mapping rule 7 and 9, because all requirements in PR_k specifying the feature are consistent with PR, the specified feature is consistent with PF, in the view of every requirement in PR_k.

From above, mapping rules from requirements to features guarantee features consistencies are the same as requirements consistencies. If requirements are consistent, features being specified using those rules are consistent too.

5.2 Validating consistencies between features coincident with those between mapped assets

For mapping rules 9 and 11, because all features in PF_k specifying the asset Ai are consistent with PF, the specified asset Ai is consistent with PA, in the view of every feature in PF_k.

For mapping rule 10, we assume A_i, and PA are inconsistent, and asset specified by F_k is A_k. A_k and PA are inconsistent too, then F_k and PF are inconsistent. This result contradicts with precondition of features consistencies. So Ai and PA are consistent.

For mapping rule 12, based on principle of 23), assets consistencies will inherit features consistencies.

From above, mapping rules from features to assets guarantee assets consistencies are the same as features consistencies. If features are consistent, assets being specified using those rules are consistent too.

We assume domain requirements consistencies have been analyzed. Based on mapping rules discussed above, assets with consistencies are produced.

6. Case study

We take the spot and futures transaction product line as example. After analysis of domain requirements, we get requirements model and requirements dependencies model. Requirements model in figure 2 shows requirements model of the example.

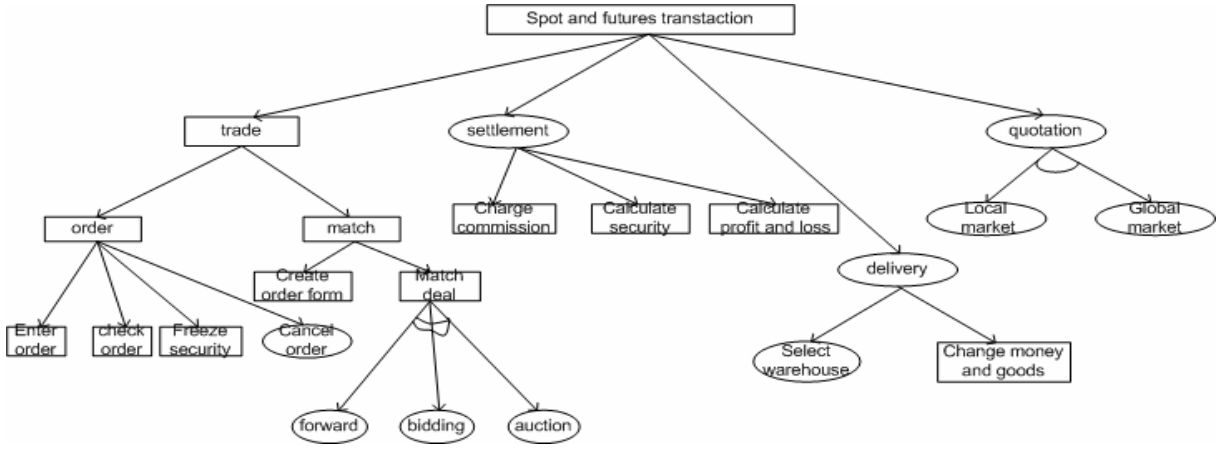

Figure 2**. Requirements model of example SPL**

In the above figure, we use rectangles to represent mandatory requirements, ellipse to represent variable requirements. Single arc is used to represent alternative set, double arc is used to represent multiple alternatives.

After using mapping rules on requirements and specified features, we get features and assets illustrated in table 1. According to consistent rules, we know that assets in the above table are consistent.

Table 1**: Features transformed from requirements and assets transformed from features of example SPL**

Requirements			Map	Features			Map	Assets		
ID	**Requirement**	**V**	**Rules**	**ID**	**Feature**	**V**	**Rules**	**ID**	**Asset**	**V**
0	Spot and futures transaction	M		0	Spot and futures transaction	M		0	Spot and futures transaction	M
1	Trade	M	1	1	Trade	M	9	1	Trade	M
1.1	Order	M	7	1.1	Order	M	9	1.1	Order	M
1.1.1	Enter order	M	1	1.1.1	Enter order	M	9	1.1.1	Process order	M
1.1.2	Check order	M	1	1.1.2	Check order	M				
1.1.3	Freeze security	M	1	1.1.3	Freeze security	M				
1.1.4	Cancel order	O	2	1.1.4	Cancel order	O	11	1.1.2	Cancel order	O
1.2	Match	M	1	1.2	Match	M	9	1.2	Match	M
1.2.1	Create order form	M	1	1.2.1	Create order form	M	9	1.2.1	Create order form	M
1.2.2	Match deal	M	1	1.2.2	Match deal	M	9	1.2.2	Match deal	M
1.2.2.1	Forward	MA	2	1.2.2.1	Forward	O	11	1.2.2.1	Forward	O
1.2.2.2	Bidding	MA	2	1.2.2.2	Bidding	O	11	1.2.2.2	Bidding	O
1.2.2.3	Auction	MA	2	1.2.2.3	Auction	O	11	1.2.2.3	Auction	O
2	Settlement	O	2	2	Settlement	O	11	2	Settlement	O
2.1	Charge commission	M	1,	2.1	Process settlement	M	9	2.1	Process settlement	M
2.2	Calculate security	M	7							
2.3	Calculate profit and loss	M								
3	delivery	O	2	3	delivery	O	11	3	delivery	O
3.1	Select warehouse	O	2	3.1	Select warehouse	O	11	3.1	Select warehouse	O
3.2	Change money and goods	M	1	3.2	Change money and goods	M	9	3.2	Change money and goods	M
4	Quotation	O	2	4	Quotation	O	11	4	Quotation	O
4.1	Local market	ME	5	4.1	Local market	O	11,12	4.1	Local market	O
4.2	Global market	ME	5	4.2	Global market	O	11,12	4.2	Global market	O

M: mandatory; O: optional; A: alternative; MA: multiple alternatives; ME: mutually exclusive. V: Variability.

7. Conclusion and future work

The proposed approach defines a feature dependencies classification. These dependencies can also be applied to requirements. Mapping rules from requirements to features and mapping rules from features to architecture are developed, and we validate consistencies between product assets coincident with those between requirements based on mapping rules.

The approach is more effective than other approaches. It provides reasonable dependency classifying method. Based on classified variability and dependencies, effective mapping rules are developed. This approach supports easy generation of product line architecture from domain requirements, and keeps consistencies between product line assets along with those between domain requirements. The approach may decrease inconsistency analysis cost and increase reuse in SPL. A case of spot and futures transaction product line is used to illustrate and validate the approach.

Based on classified feature dependencies and mapping rules, we have studied influence in a very abstract level. We will do future research on how different feature dependencies influence architecture implementation and product releasing in a SPL

8. Acknowledgement

This work is supported by the National Natural Science Foundation of China under Grant No. 60473061; the National High Technology Development 863 Program of China under Grant No.2005AA113120.

9. References

[1] Mikyeong Moon, Keunhyuk Yeom, "An Approach to Developing Domain Requirements as a Core Asset Based on Commonality and Variability Analysis in a Product Line", IEEE transactions on software engineering, vol. 31, no. 7, July 2005, pp551-569.

[2] . Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, and J. Natt och Dag, "An Industrial Survey of Requirements Interdependencies in Software Product Release Planning", In Proceedings of Fifth IEEE International Symposium on Requirements Engineering, IEEE Computer Society, 2001, pp. 84-91.

[3] A.G. Dahlstedt, A. Persson, "Requirements Interdependencies–Moulding the State of Research into a Research Agenda", In Proceedings of Ninth International Workshop on Requirements Engineering: Foundation for Software Quality, Klagenfurt/Velden, Austria, June 2003, pp. 55-64.

[4] S. Ferber, J. Haag, J. Savolainen, "Feature Interaction and Dependencies: Modeling Features for Reengineering a Legacy Product Line", The Second Software Product Line Conference 2002, LNCS 2379, August 2002, pp. 235–256.

[5] J. Giesen, A. Volker, "Requirements Interdependencies and Stakeholders Preferences", In Proceedings of IEEE Joint International Conference on Requirements Engineering, Sep 2002, pp. 206-209.

[6] J. Karlsson, S. Olsson, and K. Ryan, "Improved Practical Support for Large-scale Requirements Prioritizing", Requirements Engineering Journal, Vol. 2, No. 1, 1997, pp. 51-60.

[7] K. Lee, K.C. Kang, "Feature Dependency Analysis for Product Line Component Design", The Third Software Product Line Conference 2004, LNCS 3107, Aug 2004, pp. 69–256.

[8] B. Ramesh, M. Jarke, "Toward Reference Models for Requirements Traceability", IEEE Transactions on Software Engineering, Vol. 27, No.1, January 2001, pp. 58-93.

[9] Wei Zhang, Hong Mei, Haiyan Zhao, "A Feature-Oriented Approach to Modeling Requirements Dependencies", Proceedings of the 2005 13th IEEE International Conference on Requirements Engineering (RE'05),2005.

[10] Hassan Gomaa, Michael E. Shin, "A multiple-View Meta-modeling Approach for Variability Management in Software Product Lines", ICSR 2004, LNCS 3107, pp274-185, 2004

[11] Kwanwoo Lee, Kyo C. Kang, "Feature Dependency Analysis for Product Line Component Design", ICSR 2004, LNCS 3107, pp69-85, 2004

[12] H. Ye and H. Liu, "Approach to modeling feature variability and dependencies in software product lines", IEE Proc.-Softw., Vol. 152, No. 3, June 2005, pp101-109.

[13] J. Giesen, A. Volker, "Requirements Interdependencies and Stakeholders Preferences", In Proceedings of IEEE Joint International Conference on Requirements Engineering, Sep 2002, pp. 206-209.

[14] A. von Knethen, B. Paech, F. Kiedaisch, and F. Houdek, "Systematic Requirements Recycling through Abstraction and Traceability", In Proceedings of IEEE Joint International Conference on Requirements Engineering, Sep 2002, pp. 273-281.

[15] Martin S. Feather, Steven L. Cornford, Mark Gibbel, "Scalable mechanisms for requirements interaction management", 2000, IEEE.

[16] J. Andersson and J. Bosch，"Development and use of dynamic product-line architectures", IEE Proc.-Software., Vol. 152, No. 1, February 2005, pp11-28.

[17] Coplien, J.O.: 'Multi-paradigm design'. PhD thesis, Vrije Universiteit, Brussels, 2000

[18] J. Savolainen, L. Oliver, et al, "Transitioning from product line requirements to product line architecture", Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05), 2005 IEEE