2012 IEEE 36th International Conference on Computer Software and Applications

Software Product Line Engineering for Developing Self-adaptive Systems: Towards the Domain Requirements

Liwei Shen, Xin Peng, Wenyun Zhao
School of Computer Science
Fudan University
Shanghai, P.R.China
{shenliwei, pengxin, wyzhao}@fudan.edu.cn

***Abstract*— Self-adaptive systems are now facing the anticipation of mass customization. Therefore, the Software Product Line (SPL) engineering for developing Self-Adaptive systems (SPL4SA) can be an effective way. At the first sight, SPL4SA is the straightforward combination of the two methodologies of SPL engineering and self-adaptive systems. However, the direct and unsystematic combination will bring difficulty in the domain requirements analysis and in the customization process. In this paper, in order to give a solution to the practical problems, we propose a domain requirements meta-model in SPL4SA. It is described with different point of views and the variability binding constraints inside are emphasized. Based on it, a guidance is concluded to support the consistent customization towards the domain model. In addition, an experimental study about a web-based business product line involving self-adaptation capability is conducted to evaluate the model.**

Keywords- self-adaptative system; software product line; domain requirements

I. INTRODUCTION

Over time, self-adaptation has become an essential capability required by software systems in the areas of pervasive computing, high-performance and distributed systems, high-dependable systems, and online service systems. These self-adaptive (or autonomous) systems are expected to dynamically adapt their parameters, structures and/or behaviors at runtime in response to environment and/or requirements changes. And these systems usually have four aspects of self-management capability, i.e. self-configuration, self-optimization, self-healing and self-protection [8][17]. Typically, self-adaptive systems have the MAPE (Monitor, Analyze, Plan, Execute) [8][17] control loop, and we assume that they conform to the principle of separation of concern (SOC) [2], i.e. adaptation implementation is encapsulated and separated from business logic.

If a series of similar software systems in a specific business domain share a large portion of requirements, software product line (SPL) [1][3] is obviously an effective way to achieve improvements in cost, quality, productivity and time to market. SPL is a methodology which supports mass customization by developing application products from a common set of core assets in a prescribed way [1]. If self-adaptation is a built-in capability of those application products, the SPL should doubtless involve the self-adaptation implementation in the core assets and mass customization, thus making the problem of SPL engineering for developing Self-Adaptive systems (SPL4SA).

SPL4SA is to develop a series of similar self-adaptive products in the same business domain. At the first sight, SPL4SA is the straightforward combination of the two methodologies of SPL engineering and self-adaptive systems: implement self-adaptive systems according to the MAPE structure and the principle of separation between adaptation logic and business logic; achieve mass customization by deriving self-adaptive products in application engineering from the core assets developed in domain engineering. However, as will be discussed in Section 2, the direct and unsystematic combination will bring difficulty in the requirements analysis of the domain and in the customization process for deriving products. This makes it a new engineering problem that needs to be specially studied.

In this paper, we aim to highlight the practical significance of SPL4SA, and investigate its special engineering concerns towards the domain requirements beyond the straightforward combination of the two related paradigms. To this end, we propose a domain requirements meta-model in SPL4SA. It is described with different points of views and the variability binding constraints inside are emphasized. Based on it, a guidance is concluded to support the consistent customization towards the domain model. The contribution is practiced and evaluated with an experimental study about a web-based business product line involving self-adaptation capability.

The remainder of this paper is organized as follows. Section 2 presents the background and research problems with comparison to related work. Section 3 presents the domain requirements meta-model and its constituents. Then in Section 4 the experimental study is described in detail. Section 5 gives out some discussions towards the research problems based on the practice experience. Finally the conclusion and the future work reside in Section 6.

II. BACKGROUND AND RESEARCH PROBLEMS

In this section, we first highlight the practical significance of SPL4SA with an example, and then analyze

0730-3157/12 $26.00 © 2012 IEEE
DOI 10.1109/COMPSAC.2012.40

the research problems embodied. Related work, especially DSPL, is also summarized and compared to clarify the meaning of SPL4SA.

A. SPL4SA: an Example of PHOM Systems

The Publishing Houses Order Management (PHOM) systems are typical domain-specific applications that are required by a series of publishing houses customers, e.g. more than ten houses in Shanghai. These systems are self-adaptive ones supporting business-oriented self-configuration, self-optimization and self-healing. For example, the systems are able to store the orders in local files on condition that database is not available. On the other hand, they share quite similar requirements in both business logic and adaptation logic. For instance, some systems may recommend the clients about the popular books while the others may not. Furthermore, the adaptation logics towards the network changing may vary since the network facility is of difference. These characteristics make it natural to develop and maintain the systems as a software product line, as is called SPL4SA in this paper.

B. Static Variability and Dynamic Variability

Variability analysis and design is the key for SPL engineering to achieve the comprehensive reuse-based application development. On the other hand, self-adaptive systems also depend on variability in their structures and behaviors to support runtime adaptations in response to the changes in the requirements and environments.

The variability is categorized as the static one and the dynamic one according to its binding time [13]. The former refers to the binding on the variation points (VPs) prior to the usage of the system, while the latter implies the binding that occurs during runtime [14]. It is noticeable that traditional SPL mainly takes the static VPs as the core concerns in a domain, while the self-adaptive system principle focuses on the dynamic VPs.

SPL4SA covers the two principles and their related VP styles. Firstly, a SPL4SA is a specific kind of SPL that the individual self-adaptive system is derived by binding the static VPs. Secondly, each self-adaptive system can modify its structure or behavior through binding the dynamic VPs when the system is running.

The SOC (Separation of Concern) [2] principle implies to decouple the adaptation logic from the business requirement when developing a self-adaptive system. Thus in SPL4SA, variability resides in the both kinds of logics. The static variability in business logic distinguishes the derived applications after product delivery. The dynamic variability in business logic then implies the transferable configurations within an application. The same is true for the adaptation logic. However, in our paper, we do not take the dynamic variability in the adaptation logic into consideration. Thus each generated application should have its fixed adaptation behavior.

C. SPL4SA: Research Problems

Intuitively, we can apply the domain engineering from SPL to establish the reusable platform covering the core assets in the business requirements as well as in the adaptation logics. Variability should be emphasized in the platform to denote the difference between the separate products (static variability in the business requirements and in the MAPE constructors) and between the timeline snapshots of the runtime application (dynamic variability in the business requirements). Finally, mass customization is supported through the application engineering.

However, the direct and unsystematic combination will bring difficulty in the requirement analysis of the domain and in the customization process for deriving products.

First, they do not provide a uniform view to explicitly contain the business requirements and the adaptation logics. Since domain model and MAPE structure are separately emphasized in traditional SPL and self-adaptive systems, it may be a tough job to merge them together.

Second, they have no efficient method to explicitly express the variability in the Plan which is a major constituent of the MAPE control loop. Plan is composed of a set of reconfiguration rules taking effect according to the changing environment. In order to make the rules machine-readable and the variability embodied easy to identify, the Plan should be reasonably organized.

Third, the relationships between the variability of the elements in the both logics have not been considered. If they are missing, it will lead to the inconsistent customization results when deriving the products. For example, when the dynamic variability does not persist in the final product due to the stable environment, it will probably not inform the adaptation logic to remove the corresponding adaptation rules since there are no constraints between them to imply the influence from a specific binding strategy.

SPL4SA is anticipated to be a systematic paradigm for developing a set of related self-adaptive systems in a consistent way. Therefore, in order to solve the problems, we have proposed a domain requirements model in SPL4SA which will be introduced in Section 3.

D. Related Work

Compared with SPL4SA, a relevant but quite different paradigm is Dynamic Software Product Line (DSPL) which adopts SPL techniques for developing individual self-adaptive systems. Hallsteinsen [4] defined DSPL as a product line that produces software capable of adapting to fluctuations in user needs and available resources. It is accomplished by reconfiguring a product dynamically to produce new products within the product line during execution [12]. The dynamically derivation is thus analogized as the reconfiguration of the adaptive system. Other works including [5][6][7] all contribute in this kind of DSPL from the requirements analysis to the architectural design and the implementation. Rather than aiming at a single self-adaptive system and its runtime configurations,

SPL4SA involves a group of related applications where the variability exist between applications as well as inside single ones.

There are also works concentrating on the flexible binding time, where a variation point can be bound at different times [18][19]. The implementation technique for the variation points is the focus which contributes to make an asset more reusable. Although SPL4SA contains VPs with different time and the dynamic VPs may also be decided at delivery time, it is barely the result of the applied constraints upon the binding of the variation points. In addition, we do not emphasize the variability implementation technique issues in this paper.

III. DOMAIN REQUIREMENTS MODEL IN SPL4SA

In this section, we will first introduce the domain requirements meta-model in SPL4SA. The variability binding constraints are emphasized later. Besides, a guidance for the domain model customization is also provided.

A. The Domain Requirement Meta-model in SPL4SA

Since the model should include the system behavior as well as the knowledge when the system and how the adaptation should be performed [15], the domain model is considered to be described through different points of views following the SOC principle: business requirements, context and adaptation logic. Figure 1 depicts the meta-model.

In the meta-model, the involved elements in each viewpoint are all considered in terms of **features** which are identified separately, i.e. business feature, context feature and adaptation feature. Therefore, separate feature models [10][11] can be used to express the commonalities and variabilities in SPL4SA from different perspectives.

If a feature is considered as a variation point, its variability is classified from three dimensions. It can be an optional feature (ifOptinal=true), an alternative feature with variants (ifAlternative=true) or a variable feature (ifVariable=true). In particular, the last one represents the variability inside a feature, usually realized as an attribute with changeable values. A feature's variability can also cover more than one dimension. Furthermore, each dimension should be specified whether its binding time is static or dynamic.

The feature model of business requirements denotes the system behavior which varies between the separate products as well as between the timeline snapshots of a runtime application. The feature diagram follows the traditional tree-like style. Meanwhile, the static as well as the dynamic VPs are accommodated and they should be depicted separately and clearly.

The feature model of context offers a formal representation for the physical or business environments towards a set of self-adaptive systems. The diagram is also organized as a tree style. In the model, the context features can be optional or alternative and all of them should be bound statically.

The feature model of adaptation logic contains the variability in the MAPE control loop. An adaptation feature is further divided into sub-features including monitor, analyzer, plan and execution. In addition, the reconfiguration rules in the plan part are described through ECA (*On* Event *If* Condition *Do* Action) [9] representation. Since condition and action constitute the main semantic of a rule, a condition as well as an action is decomposed explicitly. From the meta-model, we can see associations between the adaptation feature constituents. They conform to the control loop and are performed in sequence.

It should be further noticed that the variability in the adaptation logic feature model resides in monitors and plans, and all of the VPs are bound statically. The former implies the monitors along with their alternative implementation choices. As for the latter, the variability related to these features is threefold. First, the rules themselves can be alternative. Second, the variations in the condition usually exhibit as the comparable parameters to be set, thus represented as a variable feature. Third, the action feature may be alternative that it can have a set of selectable choices. In addition, it can also be variable due to the definable parameters contained in the action description.

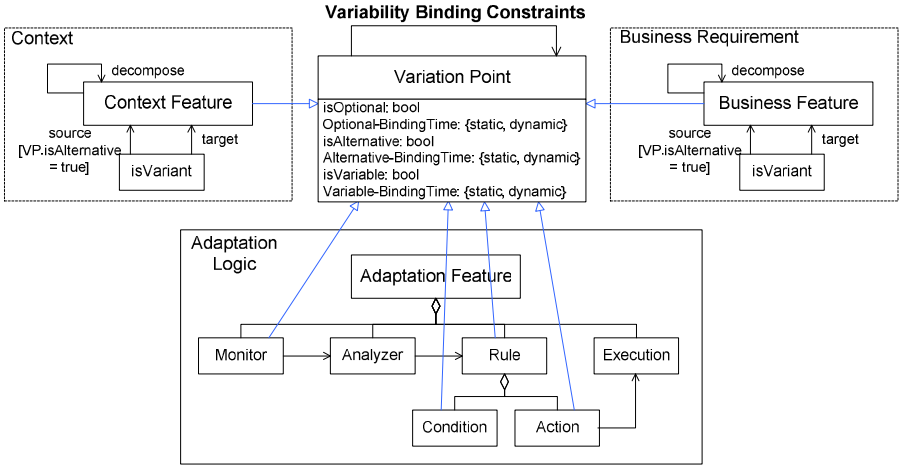

Figure 1. The domain requirements meta-model in SPL4SA

B. Variability Binding Constraints

The constraints denote the dependency between the binding statuses of the two variation points. They help to ensure the consistency in the customization phase.

Before listing the constraints, we have to classify two terms in the customization process towards the variation points i.e. *bind* and *persist*. *bind* is the decision operation made to the static or dynamic VPs before system delivery. *persist,* on the other hand, is specific to the dynamic VPs to denote whether they still have the capability of dynamically binding in the derived products. Certainly, if a dynamic VP is not persisted, its dynamism disappears in the derived product. In addition, we uniquely adopt *require* as the constraint type [16], since it can give out a uniform semantic between the bindings of the variation points. It means the depender cannot be bound/persisted if the dependee is not bound/persisted.

The constraints are then listed as follows. They are all identified by the prefix *Ct*.

Ct0: structure dependency or configuration *require* between the business VPs

Some constraints come from the structure of the feature model which is implicit. For example, if a parent optional feature is removed from the application, its children features with dynamic variability cannot be persisted any more. Other constraints are defined by the configuration dependencies between two features especially as *require*. It implies that an optional feature cannot be bound if its required feature has not been bound yet.

Ct1: bind(static business VP) *require* bind(context VP)

Binding of the static business variation points may depend on the specific context where an application resides. Thus, this kind of constraint is regarded as an important binding rationale for the static VPs.

Ct2: bind(dynamic business VP) *require* bind(context VP)

Under a specific context situation, some dynamic business VPs do not persist its original dynamism in the derived application. Thus, this kind of constraint is usually applied to remove the dynamic VPs or its variants from the generated product. At times it can be regarded to bind an alternative dynamic VP before runtime so that it cannot be rebound anymore.

Ct3: bind(rule) *require* persist(dynamic business VP)

This constraint implies that a rule should be kept in the generated application if its related dynamic VP still has the capability of dynamically binding. Otherwise, it will never take effect and ought to be removed. If so, the contained conditions, rules and executions are discarded cascade. In addition, the analyzers and monitors which server uniquely for the rule are also required to be removed.

Ct4: bind(action VP) *require* persist(dynamic business VP)

Each action has its corresponding operation target. The target may be a dynamic optional feature or a variant belonging to a dynamic alternative feature. Therefore, the action will be meaningless and should not be bound when the target VP is not persisted in the derived application.

Ct5: bind(monitor VP) *require* bind(context VP)

This kind of constraint is similar with Ct1 that some of the monitor VPs can be determined.

C. Customization Guidance

In traditional SPL, usually the static business VPs are taken as the customization objects. However in SPL4SA, both the static and dynamic business VPs may be decided by the context binding strategy and the VPs in the adaptation logic may be influence by the persistence of the related dynamic VPs. Therefore, we conclude a customization guidance for SPL4SA based on the variability binding constraints introduced as follows.

1) Users bind the context VPs. All the variation points can be determined in a certain application context.

2) Apply Ct1 to bind the related static business VPs based on the results from step1.

3) If there remains unbound static business VPs, bind them through Ct0 or decided by users. Otherwise, skip this step.

4) Apply the structure constraint in Ct0 to remove the related features. For example, they can be a group of mandatory features or dynamic VPs in the branch of an optional feature that has been removed.

5) Apply Ct2 to remove the dynamic VPs or its variants based on the results from step 2. This step involves pre-binding a dynamic alternative feature before system delivery.

6) Apply Ct3 to remove the rules that do not take effect based on the results from step5. In addition, remove the contained conditions, actions and executions, or the related analyzers and monitors if necessary.

7) Apply Ct4 to bind the related action VPs in the remaining rules based on the results from step6.

8) Apply Ct5 to bind the related monitor VPs based on the results from step2.

9) Users bind the undetermined VPs in the rules. For example, they have to set the variable attributes within the conditions or the actions.

10) Users bind the undetermined VPs in the monitors.

IV. Experimental Study of the PHOM Domain

In this section, we will depict the domain requirements model for PHOM. The variability binding constraints are highlighted and analyzed. Based on them, we also perform a customization towards the domain model.

A. The Domain Requirements Model of PHOM

The domain requirements model for PHOM domain is constructed in an ad-hoc way and is depicted in Figure 2. As introduced in Section 3, this model is mainly composed of the three parts: business requirements, context and adaptation logic. Feature models locate in all the parts and the legends are illustrated in the right of the figure. In particular, a variable feature is expressed as a feature with

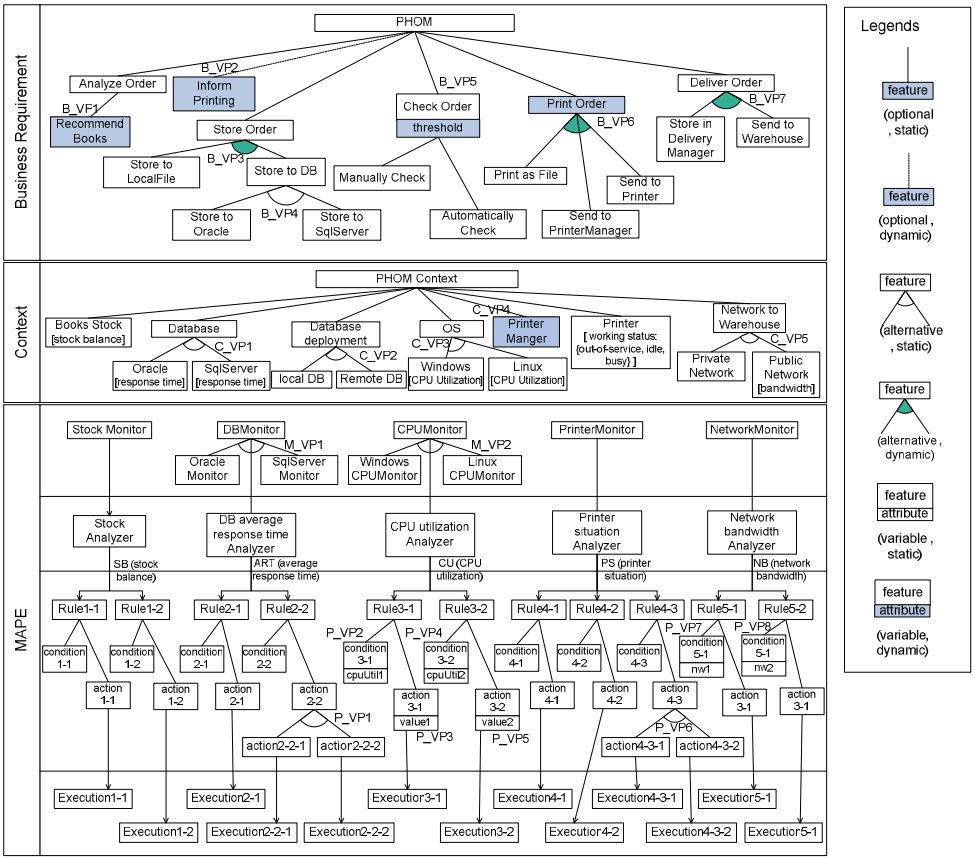

Figure 2. Domain requirements model for PHOM

attributes. The attribute can be in a continuous range and it can be assigned any concrete value inside this range. It can also be selected from a discrete value set which is predefined.

Due to the page limitation, the domain requirement model is simplified but covers the main business functions, involved contexts as well as adaptation capabilities the systems embody. In the diagram, each variation point is marked as a sequence VP number with specific prefix according to the part it resides.

In the feature diagram of the business requirements, the structure of the software behavior with the domain is explicitly exhibited. However, we need to give additional information to record the underlying semantics of the variation points in detail which can be seen in Table 1. The VP type column denotes the variation dimensions a feature involves. It is represented by a pair including the variability

TABLE 1. SUMMARY OF THE BUSINESS VPS IN THE DOMAIN MODEL

B_VPn	VP name	VP type	Description
B_VP1	Recommend Books	(optional, static)	It is included in a derived application when there is a need to inform clients of the popular books.
B_VP2	Inform Printing	(optional, dynamic)	It is bound to inform the printing house for more books if the stock balance is at a low level.
B_VP3	Store Order	(alternative, dynamic)	It is replaced by one of its variants during runtime according to the DB response time.
B_VP4	Store to DB	(alternative, static)	It is replaced by one of its variants at delivery time according to the database involved.
B_VP5	Check Order	(variable, dynamic)	It selects the order checking method by an if-else branching through the comparison between the order amount and a threshold. The threshold can be reset during runtime according to the CPU usage situation.
B_VP6	Print Order	(optional, static), (alternative, dynamic)	It is included in a derived application when there is a need to print orders. The print operation can be replaced by one of its variants during runtime according to the printer status.
B_VP7	Deliver Order	(alternative, dynamic)	It is replaced by one of its variants according to the network availability.

type as well as the binding-time specification, i.e. static or dynamic. In particular, we can see that B_VP6 (*Print Order*) covers two dimensions of optional and alternative. It means the alternative feature can be bound to a variant at runtime if it is an included feature after delivery. On the other hand, the description column gives out the semantic for the VPs about their binding rationale. Although it is an informal representation, it indirectly indicates the context elements and the reconfiguration policies that should be investigated in the domain model.

The context feature diagram indicates the different combination on the context elements of the applications in the PHOM domain. In instance, a specific publishing house may be equipped with a processing server adopting *Windows OS* (bind C_VP3) as well as a database server installed *Oracle* (bind C_VP1). Besides them, there also exist context features with attribute placed in bracket. However, they are not variable features thus the attributes cannot be defined by the users or the system.

All the variation points in the adaptation are static-binding. In the monitor part, there are two alternative VPs denoting the specific monitors to be used in a certain context. In the plan part, we identified eleven rules some of which have variability. The rules whose names contain the same number before the hyphen are regarded as the rules which concern the same context changing and influence the same dynamic VP's binding. Table 2 lists the rules' constituents where the condition and action parts are highlighted.

In the table, variable features are expressed as the boldfaced parameters with braces. On the other hand, the alternative actions within a rule are separately listed and the named choices are also embodied in braces. The last column denotes the dynamic VP which is the operation target of the related rules. The adaptation model diagram together with the additional table offers users an intelligible method to comprehend the adaptation capability in the domain.

B. Variability Binding Constraints in the PHOM Domain

In our practice, the variability binding constraints are depicted in Table 3

TABLE 3. SUMMARY OF THE VARIABILITY BINDING CONSTRAINTS

Ct0	There is no explicit configuration dependency in the business feature model.
Ct1	bind(B_VP4.Store to Oracle) require bind(C_VP1.Oracle) bind(B_VP4.Store to SqlServer) require bind(C_VP1.SqlServer)
Ct2	bind(B_VP3.Store to LocalFile) require bind(C_VP2.Remote DB) bind(B_VP6.Send to PrintManager) require bind(C_VP4) bind(B_VP7.Store in DeliveryManager) require bind(C_VP5.Pulbic Network)
Ct3	The constraints can be included from Table 2. e.g. bind(Rule2-1) require persist(B_VP3) bind(Rule2-2) require persist(B_VP3)
Ct4	bind(P_VP1.action2-2-1) require persist(B_VP3.B_VP4.Store to Oracle) bind(P_VP1.action2-2-2) require persist(B_VP3.B_VP4.Store to SqlServer) bind(P_VP6.action4-3-1) require persist(B_VP7.Send to PrinterManager) bind(P_VP6.action4-3-2) require !persist(B_VP7.Send to PrinterManager)
Ct5	bind(M_VP1.Oracle Monitor) require bind(C_VP1.Oracle) bind(M_VP1.SqlServer Monitor) require bind(C_VP1.SqlServer) bind(M_VP2.Windows CPU Monitor) require bind(C_VP3.Windows) bind(M_VP2.Linux CPU Monitor) require bind(C_VP3.Linux)

Next we analyze some of the scenarios belonging to the different constraint categories.

For example of Ct1, if the application adopts Oracle as its database, the variant Store to Oracle in B_VP4 can be chosen. Another constraint in this category indicates the opposite situation. It is notable that these two constraints do not conflict because there is an implicit mutually exclusive relationship between the variants of the same feature.

For example of Ct2, if the database is not remotely deployed (C_VP2 binds *Local DB*), the variant *Store to LocalFile* of B_VP3 will never be chosen at runtime since the specified context ensures the highly accessibility to the

TABLE 2. THE ECA RULES' DESCRIPTIONS

Rule	Condition	Action	Target
Rule1-1	SB <= 200	bind B_VP2	B_VP2
Rule1-2	SB > 200	unbind B_VP2	
Rule2-1	ART > 3	bind variant *Store to LocalFile* in B_VP3	B_VP3
Rule2-2	ART < 1	**action2-2-1**: {bind variant B_VP4.*Store to Oracle* in B_VP3}	
		action2-2-2: {bind variant B_VP4.*Store to SqlServer* in B_VP3}	
Rule3-1	CU > {**cpuUtil1**}	reset *threshold* in B_VP5 to {**value1**}	B_VP5
Rule3-2	CU < {**cpuUtil2**}	reset *threshold* in B_VP5 to {**value2**}	
Rule4-1	PS = out-of-service	bind variant *Print as File* in B_VP7	B_VP7
Rule4-2	PS = idle	bind variant *Send to Printer* in B_VP7	
Rule4-3	PS = busy	**action4-3-1:** {bind variant *Send to PrinterManager* in B_VP7}	
		action4-3-2: {bind variant *Print as File* in B_VP7}	
Rule5-1	NB >= {**nw1**}	bind variant *Send to Warehouse* in B_VP8	B_VP8
Rule5-2	NB < {**nw2**}	bind variant *Store in Delivery Manager* in B_VP8	

database (ART<1). Thus there is no reason to persist the variant. In addition, following the instance beforehand, B_VP3 has lost its dynamism because it can only be specified as the concrete *Store to DB* feature which has been decided at delivery time.

As for Ct3, if B_VP3 does not persist its dynamism after performing Ct2, the involved rules Rule2-1 and Rule2-2 can be removed from the adaptation logic because there is no need to perform the adaptation towards B_VP3.

For example of Ct4, action2-2-1 can be bound only if the concrete variant B_VP4.*Store to Oracle* of B_VP3 is reserved. In particular, the fourth constraint in this category means the binding of action4-3-2 is possible when the variant B_VP7.*Send to PrinterManager* is not included (we use ! to denote the removal of the variant).

As for Ct5, the variant *Oracle Monitor* of M_VP1 is chosen when the system adapts *Oracle* as its database.

C. A Customization Example

We have conducted a practical customization according to a specific application requirement following the guidance. The steps along with the binding strategy results can be seen in Table 4.

TABLE 4. STEP RESULTS OF THE CUSTOMIZATION PROCESS

Step 1	bind C_VP1.Oracle bind C_VP2.local DB bind C_VP3.Linux unbind C_VP4 bind C_VP5.Public Network
Step 2	bind B_VP4.Store to Oracle
Step 3	unbind B_VP1 bind B_VP6
Step 4	null
Step 5	remove B_VP3.Store to LocalFile (bind B_VP3.B_VP4.Store to Oracle) remove B_VP6.Send to Manager
Step 6	remove Rule2-1, condition2-1, action2-1, execution2-1 remove Rule2-2, condition2-2, action2-2, action2-2-1, action2-2-2, Execution2-2-1, execution2-2-2 remove DB average response time Analyzer, DB Monitor
Step 7	bind P_VP6.action4-3-2
Step 8	bind M_VP2.Linux CPU Monitor
Step 9	set P_PV2.cpuUtil1 = 70 set P_PV3.vaule1 = 1000 set P_PV4.cpuUtil2 = 40 set P_PV5.value2 = 500 set P_PV7.nw1 = 80 set P_PV8.nw2 = 50
Step 10	null

In the process, the binding strategies upon the context VPs in Step 1 are given by users. It then derives the customization on the other parts based on the constraints' definition through to Sep 8. In particular, the binding in Step 3 still relies on the users. In Step 6, two rules as well as the related MAPE features are integrally removed. As for Step 9 and 10, the binding is left for the users over again. At last, the binding decision from the ten steps is applied on the domain model to generate an application requirement model.

V. DISCUSSION

The SPL4SA paradigm introduced in our paper seems to touch the area rarely focused. However, the anticipation of developing a set of similar self-adaptive systems in a systematic way drove our effort in the research.

As discussed in section 2, we pose three research problems. Thus based on the case study, we evaluate the SPL4SA mechanism towards the domain requirements which is considered as a solution to these problems.

Firstly, we have established a uniform domain requirement model which follows the domain analysis activity. Reference to the SOC principle, the domain model is described through different points of views. Although the multi-view representation is not a new idea, SPL4SA is constituted with the parts which are thought to be indispensable. The domain meta-model introduced in section 3 outlines the main constituents which are business requirements, adaptation logics and context. The first part focuses on the systems' behavior which is anticipated by the clients. The second part provides solutions to the adaptation implementation which is a concern behind-the-screen. And the third part is the objective reflection to the system environment. Variability is emphasized in these constituents. On the other hand, we also recognize the importance of the variability binding constraints. It offers benefits in realizing the domain and guaranteeing the consistency of the customization.

Secondly, in order to merge the adaptation logic in a formal requirement representation, the meta-model includes the mechanism to formally model the variability of the adaptation logic, especially the reconfiguration rules in the plan part. Although the problem how to explicitly model adaptation and how to compose it with feature diagram has been given attention [13], we extend it with the goal to highlight the variability it contains. In adaptation feature model, all the elements involved in the adaptation logics are considered as a specific kind of feature. Furthermore, the diagram is decomposed into the different constituents according to the MAPE constructors. The formal modeling for adaptation logic brings benefits. On one hand, the changeable parts in the adaptation control loops can be directly located rather than analyzing the informal rules represented in natural language. On the other hand, with the same representation as the business feature model, the customization upon the VPs can be easily performed.

Thirdly, we have paid great attention to ensure the consistent customization for SPL4SA. We think this as a notable research concern since it is a tough experience to perform the customization without the support of variability binding constraints. For example, if Ct3 is not identified, there will be a situation that Rule2-1 is remained but its operation target B_VP3 is not dynamic anymore. Although the reserved rule will not take effect since its condition will

never be reached in this case, it reduced the understandability to the users and developers. Think again if Ct4 constraints are not discovered, then which action to choose in P_VP1 may be error-prone because the customization rationale is missing. As a result, it will be a fatal error towards the generated product if the variant is wrongly chosen. However, the constraints help to avoid the inconsistency. They are also regarded as the major constituent of the domain requirement model. The knowledge can be collected when modeling the domain requirements, then it is organized and expressed in propositional logics. The customization guidance is summarized from the practice. The step sequence is fixed in order to assure the correctness, integrality and to eliminate the redundant operations.

Limitations are also obvious on the effort of applying the method. The domain knowledge including the business requirements and the adaptation rules are identified ad hoc. The work of discovering variability further depends on the stakeholders' understanding towards the whole domain. On the other hand, the method introduced has not involved the design and implementation aspects of the SPL4SA framework.

VI. CONCULSION AND FURTHER WORK

When self-adaptive systems face the anticipation of mass customization, the SPL engineering for developing Self-Adaptive systems (SPL4SA) could be an effective way. SPL4SA aims to develop a series of similar self-adaptive products in the same business domain. At first sight, it can be combined of the two methodologies of SPL engineering and self-adaptive systems in a straightforward way. However, the direct and unsystematic combination will bring difficulty in the requirement analysis of the domain and in the customization for deriving products (Section 2).

In this paper, we concentrate on the domain requirements of a SPL4SA and propose a domain requirements meta-model to formally express them. An instantiated model includes the parts of business requirements, context and adaptation logic. Furthermore, it contains a set of variability binding constraints along with a guidance to ensure the consistent customization towards the domain model. Finally, a web-based business product line involving self-adaptation capability is conducted.

The mechanism towards domain requirements is performed manually till now, so the supporting tools are anticipated to ease the human effort. Therefore, the tool providing the domain modeling as well as the consistent customization is planned. On the other hand, the case study we performed maybe simple. There are no complex adaptation logics which operate on more than one dynamic variation point. In addition, we have not met the conflict situations occur between the adaptation rules. Under this circumstance, the constraints may be more complex than those we classified. We plan to take this issue for our future work.

ACKNOWLEDGMENT

This work is supported by National Natural Science Foundation of China under Grant No. 90818009, National High Technology Development 863 Program of China under Grant No. 2012AA011202 and No. 2011AA010101.

REFERENCES

[1] P.Clements and L.Northrop, "Software Product Lines: Practices and Patterns", Addison-Wesley, 2002.

[2] P.K.McKinley, S.M.Sadjadi, E.P.Kasten and B.H.Cheng, "Composing Adaptive Software", IEEE Computer, 37(7):56-64, 2004.

[3] K.Pohl, G.Böckle, and F.V.Linden, "Software Product Line Engineering - Foundations, Principles, and Techniques", Springer Verlag, Heidelberg, 2005.

[4] S.Hallsteinsen, M.Hinchey, S.Park and K.Schmid, "Dynamic Software Product Line", Computer, 41(4): 93-95, 2008.

[5] N.Bencomo, P.Sawyer, G.Blair and P.Grace, "Dynamically Adaptive Systems are Product Lines too: Using Model-Driven Techniques to Capture Dynamic Variability of Adaptive Systems", in International Workshop on Dynamic Software Product Lines, 2008.

[6] S.Hallsteinsen, E.Stav, A.Solberg and J.Floch, "Using Product Line Techniques to Build Adaptive Systems", in International Software Product Line Conference, 2006.

[7] P.Trinidad, A.Ruiz-Cortes, J.Pena, D.Benavides, "Mapping Feature Models onto Component Models to Build Dynamic Software Product Lines", in International Workshop on Dynamic Software Product Lines, 2007.

[8] IBM Autonomic Computing Architecture Team, "An architectural blueprint for autonomic computing", Technical Report, IBM, 2006.

[9] K.R.Dittrich, S.Gatziu and A.Geppert, "The Active Database Management System Manifesto: A Rulebase of ADBMS Features", in SIGMOD Records ,1996.

[10] K.C.Kang, SG.Cohen, JA.Hess, WE.Novak and AS.Peterson "Feature-oriented domain analysis feasibility study", Technical report, Software Engineering Institute, Carnegie Mellon University, 1990.

[11] K.C.Kang et al. FORM: A feature-oriented reuse method with domain-specific architecture. Annals of Software Engineering, 1998, V5: 143-168.

[12] K.C.Kang, V.Sugumaran and S.Park, "Software Product Line Engineering: Overview and Future Directions", Applied Software Product Line Engineering, 2009.

[13] A.Classen, A.Hubaux, F.Saneny, E.Truyeny, J.Vallejos, P.Costanza, W.D.Meuter, P.Heymans, and W.Joosen, "Modelling variability in self-adaptive systems: Towards a research agenda", in Workshop on Modularization, Composition, and Generative Techniques for Product Line Engineering held as part of GPCE08, 2008.

[14] K.Schmid and H.Eichelberger, "Model-Based Implementation of Meta-Variability Constructs: A Case Study using Aspects", in International Workshop Variability Modelling of Software-intensive Systems, 2008.

[15] M.Acher, P.Collet, et al. Modeling Context and Dynamic Adaptations with Feature Models. In 4th International Workshop Models@run.time at Models 2009 (MRT'09), page 10, October 2009.

[16] K.Lee and KC.Kang, "Feature Dependency Analysis for Product Line Component Design", in International Conference on Software Reuse, 2004.

[17] M.Huebscher and J.McCann, "A survey of autonomic computing—degrees, models, and applications", ACM Computing Surveys, 2008

[18] V.Chakravarthy, J.Regehr, and E.Eide, "Edicts: implementing features with flexible binding times", in International Conference on Aspectoriented Software Development, 2008.

[19] E.Dolstra, G.Florijn and E.Visser, "Timeline variability: The variability of binding time of variation points", in Workshop on Software Variability Management, 2003.