Configuration Complexity: A Layered based Configuration Repository Architecture for conflicts identification

GJCST Computing Classification
F 2.2, I 4.7

Uzma Afzal[1] Dr. Syed Irfan Hyder[2]

***Abstract*-An unstable product configures if conflicts (features\ requirements\decision) identify in late phases of software product configuration. This late discovery of conflicts makes the configuration process more complex. We proposed layered base complexity by capturing conflicts at the time of their generation.**

Keywords-Configuration, Feature Conflicts, Configuration complexity

I. Introduction

Today, configuration management (CM) is more important than ever because customers want new designs of products of higher quality at lower prices. Efficient CM can shorten the product life cycle, minimize production cost, and guarantee product quality [1]. Market competitiveness forces product vendors to build flexible products that not only support a specific customer's need but also a group of customers having similar requirements domain. A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy specific needs of a particular market or mission, and that are developed from a common set of core assets in a prescribed way, according to the definition used by the Software Engineering Institute (SEI) [2]. Product configuration has proven to be an effective means to implement mass customization [3]. Through a configuration process, product modules or components are selected and assembled according to customer requirements [u2] into 4. Product configuration is a collaborative process and Deriving a product from a product line is a complex task requiring the involvement of many heterogeneous stakeholders. Taking their different roles and needs into account is essential to exploit the possible benefits of product lines. Numerous stakeholders need to be supported in understanding the variability provided by the product line. Integration of processes and people is critical. Many critical failures of today's major systems are the consequence of inadequate management and control over an integrated set of components [5]. Abstraction and instantiation are two steps to realize product configuration. So-called abstraction is to elicit a product model from all products, and use a product model, a configuration rule base, and a part instance base to represent all products. Instantiation is according to a customer's demands to confirm the value of every component in the product model tree, and the process of confirmation is based on the product model, configuration rules, and part instance base [6].

About-[1]Department of computer science, Federal Urdu University of Arts Science & Technology (e-mail-u.afzalkhan@gmail.com)
About-[2] PAF-Karachi Institute of Engineering & Technology

II. SPL Configuration

Software product lines, typically separating two key areas:

- Domain engineering
- Application engineering.

Figure 1 shows the relationship between domain engineering and application engineering. During domain engineering, the variability and commonalities of the product line's reusable core assets such as requirements, architectural elements, or solution components are captured in variability models. A significant body of research is available on modeling approaches and notations for this purpose. During application engineering, concrete products are derived from the product line by selecting, configuring, integrating, and deploying the core assets. Compared to the vast amount of research results on building product lines, few approaches and tools are available for product derivation [7].

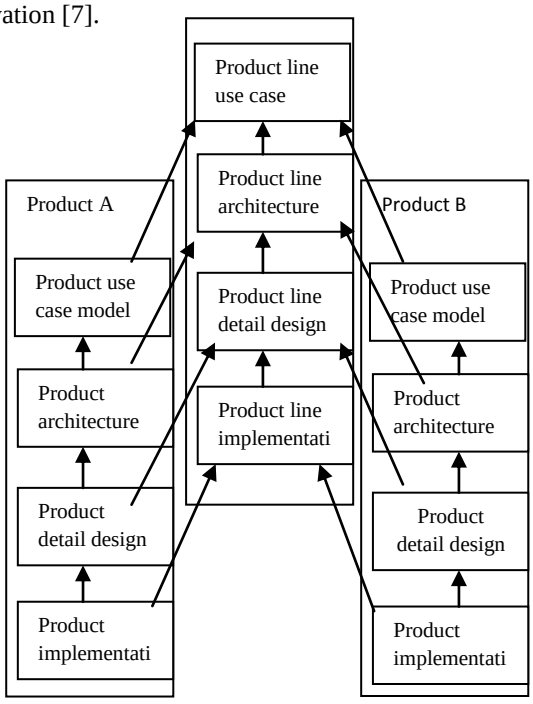

Figure 1 : Modeling Dependencies [8]

Configuration of SPL is a collaborative process and its usual steps are:

- Organization selects the product that meets its business objectives.
- Configuration team starts working.
- Configuration manger splits product into configuration units (configuration repository is single and shared).
- Each configuration unit is assigned to single/multiple developer(s).
- Configuration units are re-assembled into a single product.

III. SPL Configuration Management

Software product configuration is the process of selecting components from the existing repository and their assembling with the objective of timely, cost effective product delivery. It is, an integral part of any software development activity, takes on a special significance in software product line context. This is due to the special property of software product line, in which the core assets are shared by all products. There are more member products in one product family than in conventional software systems. Hence, in product line, there are much more number of products, assets, and components that needs to be configuration managed. To reduce the working load and the complication of configuration management, it is important to select the right artifacts under configuration management [9].

It also involves identifying the configuration of software (i.e. selected software work products and their description) at given point in time, systematically controlling changes to the configuration through out the software development life cycle [10]. As a result of configuration process, configuration model are produced containing a list of desired product feature [11].

IV. SPL Feature Model

Features are key distinctive characteristics of a product [12]. A feature design provides a graphical tree like notation that shows the hierarchical organization of features [12]. A feature model is commonly used to guide the configuration process since it breaks down the variabilities and commonalities of product line into a hierarchy of feature as shown in figure 2. Additionally feature model encompass constraints that prevents the derivation of inconsisted product specification i.e. product containing incompatible feature [13].

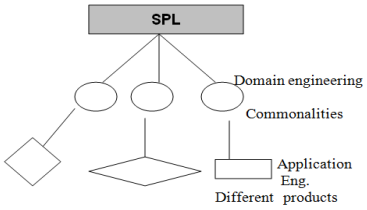

Figure 2 : SPL from domain to application engineering

A feature model allows for inclusion and exclusion of various features and variants so that a valid feature configuration is produced. A feature model also guides product configuration and can be used to validate a particular configuration for conformance [14]. Feature model provide the base for the configuration of whole system. Normally feature model develops in beginning of the development / configuration process. However, Change pervades the entire software life cycle. Requirements change when developers improve their understanding of the application domain. [9]. These changes affect the feature model and its consistency. An invalid feature model leads to an invalid product configuration or it can be said that only consistent and valid feature model gives a successfully configured product Additionally, In global environment, the software configuration becomes critical due to the characteristic of distributed development (physical distance, cultural differences, trust, communication and other factors [10].

V. SPL Configuration Issues

As shown in figure 3, a large-scale product configures from a centralized, shared repository and divides into different modules to make configuration process less complex. Enabling collaborative product configuration brings new and challenging problems such as the proper coordination of configuration decision [13].because a typical software development team consists of multiple developers who work together on closely related sets of common artifacts [15].

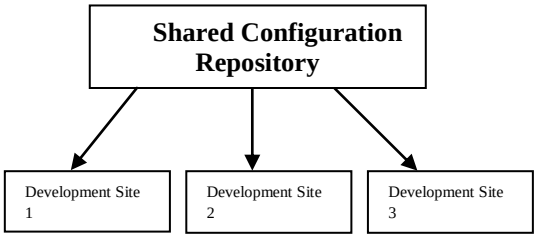

Figure 3 : Configuration from Multiple Sites

The main cause of the system design problems lay with the adhoc way in which large and distributed systems are built, where individual make their own decisions about configuration and life cycle[16] from a configuration and life cycle management perspective failure and recovery was usually inconsistently detected and handled[16].

In an ideal scenario either configuration is collaborative or not, feature model plays an important role in configuration and provides a base and work like a blue print for whole configuration process, only modeled features are configured in final product. Unfortunately, we are not living in an ideal environment in which every thing is according to our desire. Real /practical environment is quite different and it is very clear that the root cause of major configuration issues is the configuration of the products in an ad hoc way where each individual take his own configuration decisions. This late discovery of conflicts makes the configuration process more

complex and strongly affects the cost, efforts and schedule of the product.

A. *Is It Really A Problem?*

Distributed configuration management is intended to support the activities of project that is configured from multiple sites [16]. Multiple developers from multiple sites configure a product from a product family. SPL variant can not be constructed arbitrarily e.g. a car can not have both ABS and Standard braking software controller. A key step in building a SPL is therefore creating a model of the SPL variability and the constraints on variant configuration [18] however a model is an abstract representation of actual implementation.

In a distributed configuration environment there must be some collaborative mechanism to keep configuration synchronized. For software product configuration management tool support for collaboration on model is therefore crucial [19]. Traditional SCM have support this task for textual artifacts such as source code on the granularity of files and textual lines. They do not work well for graph like models [19]. However SPL product configuration is a decision making process in which group of stake holders chose features for a product [20] and in our collaborative scenario involvement of multiple stakeholders is a basement of product configuration, different configuration units are assigned to different developers that create problem when each individual takes his own configuration decisions (for e.g. feature selection) without going in detail. Integration of the asynchronous efforts of engineers who may be adhering to different configuration management procedures and practice is one of the critical issues [17]. There is a lot of techniques to describe features are existed but common to all of these notation is that they still require maintainers to identify and understand the interaction among features in systems [21].

VI. Proposed Solution

An unstable product configures if conflicts are not captured or captured in the late phases of software product configuration so an approach is required to capture these conflicts in earlier stage.

To solve the problem we proposed a Layered based configuration repository (shared) architecture to reduce the configuration complexity by capturing conflicts (Requirements conflicts, features conflicts, decision conflicts) at earlier stage.

We separate the features from the usual configuration repository and proposed a layered based architecture for feature repository and provide facility to exchange information between layers on a common infrastructure to avoid feature\requirement\decision conflicts of collaborative configuration. The service of proposed shared repository does not merely concern storing data but the mechanism for conflicts detection.

A. *Architecture Of Proposed Repository*

We proposed architecture of the configuration repository that is shared between multiple developers and suggest the storage of configuration data in layer format. Our repository consists on two main layers and one intermediate communication layer.

Layers are listed below.

- Product domain layer [PDL]
- Intermediate control layer [ICL]
- Product Application layer [PAL]

PDL and PAL will communicate via ICL. Product domain layer is also divided into two parts that are features layer and constraints layer.ICL plays an important role in conflicts identification because no feature will be added to the application layer until or unless Product Application Layer talk to Product domain Layer through Intermediate control Layer.

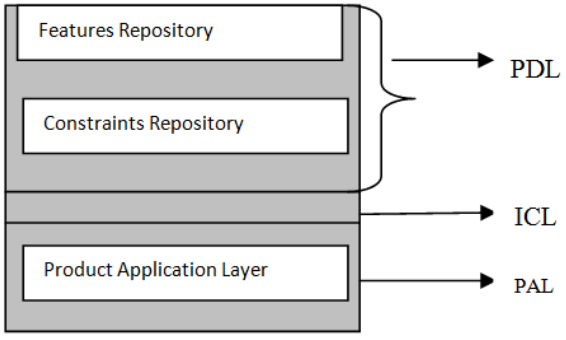

Figure 4: shared repository used by multiple developers from multiple sites

B. *Product domain layer*

It is the very first layer of Configuration repository and store features and constraints, related to the Product domain.

Features Repository sub layer: Features repository is the base of the product domain layer. Features are key distinctive characteristics of a product [12]. A feature design provides a graphical tree like notation that shows the hierarchical organization of features [I2]. A unique identifier is assigned to each feature (naming convention can be used for ease). All features that stored here are the part of the domain of product line or they can be said the core features of product. Different types of features are stored in the repository figure 5 describes the two classifications that are: Independent/dependent and mandatory/variable [22].

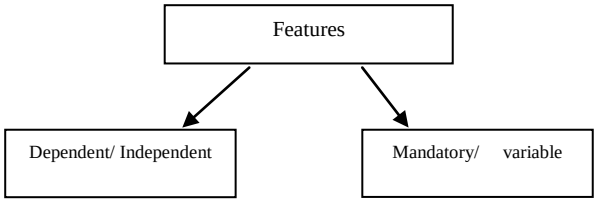

Figure 5: features classification

Independent Features: Because they are not depended on any other feature for their configuration and will not affect the any other component configuration and do not evolve any type of conflicts so only independent feature constraints that apply on them with feature identification tag are stored.

Dependent Features: because they are dependent on other features for their configuration or the configuration of any other dependent feature can affect them so applied constraint with feature identification tag and dependent feature tag are stored in the repository.
Mandatory Features: must be presented in all member products of Software Product Line. Mandatory features illustrate product family commonality [22]. They are stored with a mandatory tag and part of the all variants of any SPL product.
Variable Features: not necessarily appear in all member products in a SPL. Variable features illustrate product family variability [22].
Constraints Repository: It is the second sub layer of the product domain layer that contains all the constraints apply on features. How they stored in repository is dependent on their nature (Uni feature Constraint and multi feature constraint).
These listed constraints are taken from [23] and modified accordingly but it is not the limit other constraints can also be added to the repository.
Mandatory: A feature or a product P requires a feature F.
Optional: The existence of F in P is optional.
Or: In a feature or a product P, there is F1 or F2 or F3... or Fn.
Alternative: if (P > 0) then sum (F1, F2, Fn) in {1..1} else F1 = 0, F2 = 0, ..Fn = 0.
Implies: if (P > 0) then f > 0. That is, if there is a Feature P in a product, then there must be at least a feature F there.
Excludes: if (P > 0) then F = 0. C cannot exist in a product P.

C. Product Application Layer

It is the second layer of proposed layered repository. This layer contains a reference tag for each derived product of the product family, uniquely identified by a Product identifier. As the configuration is moved on and features are configured their unique ids are linked with the product identifier tag by exchanging information from the product domain layer via intermediate layer.

D. Intermediate control layer

It is a middle layer that is used for communication between the two main layers. Both layers talk to each other or exchange information via this communication layer. At the time of product derivation no feature will be added to the PAL until or unless PDL communicate to PAL and find a positive response that the feature addition will not create any feature conflict.

VII. Layers Communication Mechanism

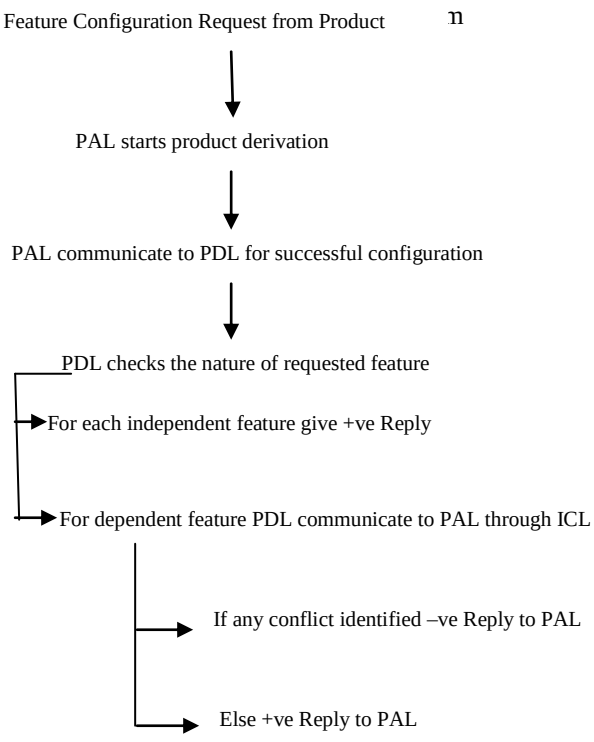

Figure 6 :communication mechanism

VIII. Prototype & Results

A tool named "Product Configuration Tool" is developed to support the proposed architecture (conflict identification interface is shown in figure 7). An interface is related to each layer of the configuration repository.
Tool has two views.

- For the population of configuration repository
- For the product derivation

Business pattern data of an ERP system is used to validate the repository architecture and its supportive tool. We mapped the business pattern to our proposed schema and then plugged it to the Product Configuration Tool and setup a test environment figure 9 shows a sample of test case. Figure 8 shows the graphical representation of obtained results that proves our thesis statement.

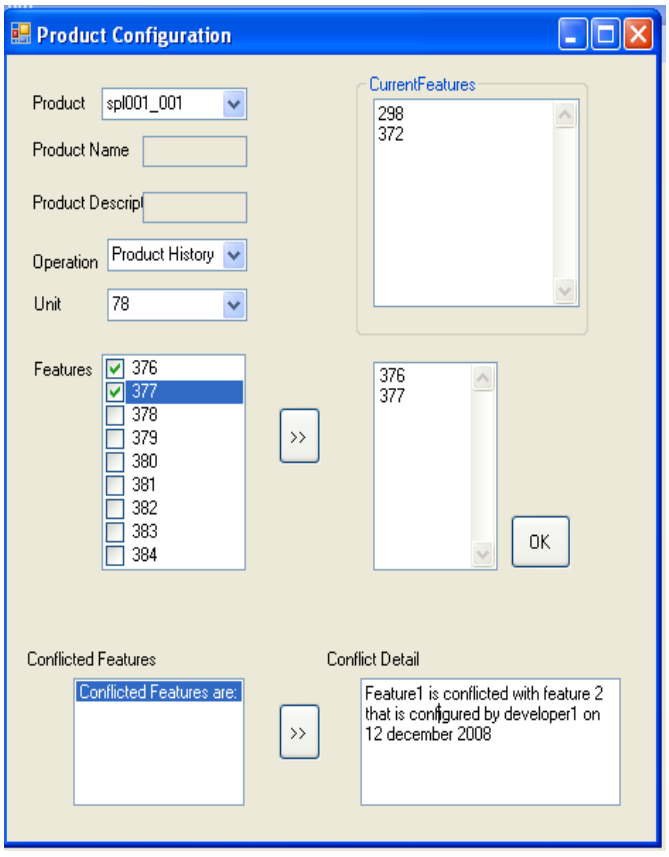

Figure 7 conflict identification view

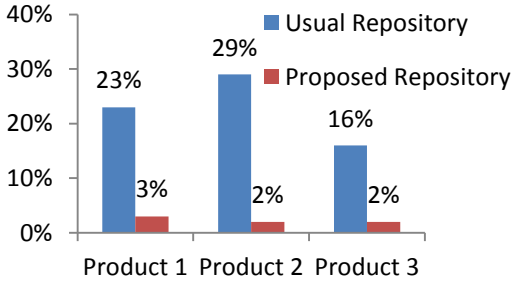

Figure 8: Graphical comparison of existing and proposed repository architecture

Test Case No.	TC_01	Test Status	Pass

Purpose & Scope

The purpose of this test is to make sure that conflicted are identified to maintain product stability.

Test strategy: Black Box Testing	Testing Methodology: Manual

Test Script & Results

Testing data:
Configuration Repository: BP_2_For_Development [ERP Repository]
Software Product Line: SPL_01 [Pharma]
Derived Product: SPL_001_001
Product History: 298,15425,15463,17279,624
Request of Feature: 502
Constraint Apply: 298 OR 502

Test Script:
1. Launch PCT application
2. From the Product derivation menu click "product derivation"
3. Select Product Id from testing data".
4. Check the history of product to make sure that 298 is added.
5. Select "Add Feature from combo".
6. Select the feature 502 from the automatic generated feature list box.
7. Press OK.

Expected Result:
Product will not add to the product and a message is displayed to show the conflict detail.

Exception & Corrective Action

No.

Comments & Conclusion

Actual Result:
As per ER
Conflicts are identified.

Figure 9: Sample test case

IX. Conclusion & future work

An unstable product configures if conflicts are not identified or identified in the late phases of software product configuration so an approach is required to capture these conflicts in earlier stage. We proposed a Layered based configuration repository (shared) architecture that reduces the configuration complexity by capturing conflicts (Requirements conflicts, features conflicts, decision conflicts) at earlier stage to reduce the configuration complexity.

A "Product Configuration Tool" (PCT) is developed to support the proposed architecture. PCT has two views one for the population of configuration repository and other for the product derivation. Business pattern data of an ERP system is used to validate the repository architecture and its supportive tool.

Future directions include the integration of architecture with existing feature analysis tools and Extend the interface to visualize the model and Enable Architecture to support distributed repository.

X. References

1) I. Choi and S. Bae: An Architecture for Active Product Configuration Management in Industrial Virtual Enterprises, Int J Adv Manuf Technol (2001) 18:133–139 2001 Springer-Verlag London Limited
2) Paul Clements and Linda Northrop: *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2001.
3) Chunjing Zhou & Zhihang Lin & Chuntao Liu: Customer-driven product configuration optimization
4) for assemble-to-order manufacturing enterprises, Int J Adv Manuf Technol (2008) 38:185–194 DOI 0.1007/s00170-007-1089-6
   Zhang Jinsong · Wang Qifu · Wan Li · Zhong Yifang: Configuration-oriented product modelling and knowledge management formmade-to-order manufacturing enterprises, DOI 10.1007/s00170-003-1871-z Int J Adv Manuf Technol (2005) 25: 41–52
6) S R Moor, J Gunne-Braden and K J Gleen: Enterprise configuration management — controlling integration complexity, BT Technol J Vol 15 No 3 July 1997
7) F. S. Zeng . Y. Jin: Study on product configuration based on product model, Int J Adv Manuf Technol ,DOI 10.1007/s00170-006-0500-z
8) Mag. Rick Rabiser: A User-Centered Approach to Product Configuration in Software Product Line Engineering. Christian Doppler Laboratory for Automated Software Engineering.
9) John D. McGregor:Software Product Lines, JOURNAL OF OBJECT TECHNOLOGY
10) Liguo Yu1 and Srini Ramaswamy :" A Configuration Management Model for Software Product Line"
11) Leonardo Pilatti,, Jorge, Luis Nicolas Audy, Rafael Prikladnicki: Software Configuration Management over a Global Software Development Environment: Lessons Learned from a Case Study, GSD'06, May 23, 2006, Shanghai, China.
12) [Marcilio Mendonca1, Thiago Tonelli Bartolomei2, Donald Cowan: Decision-Making Coordination in Collaborative Product Configuration, SAC'08, March 16-20, 2008, Fortaleza, Ceará, Brazil.
13) LUO Daizhong, DIAO Shanhui: Feature Dependency Modeling for Software Product Line, International Conference on Computer Engineering and Technology,2009
14) Marcilio Mendonca, Krzysztof Czarnecki : Towards a Framework for Collaborative and Coordinated Product Configuration, OOPSLA'06 October 22-26, 2006, Portland, Oregon, USA.
15) Goetz Botterweck1, Steffen Thiel1, Ciarán Cawley1, Daren Nestor1, André Preußner2: Visual Configuration in Automotive Software Product Lines, Annual IEEE International Computer Software and Applications Conference,2008
16) Anita Sarma, David Redmiles and André van der Hoek :"Empirical Evidence of the Benefits of Workspace Awareness in Software Configuration Management, SIGSOFT 2008/FSE-16, November 9--15, 2008. Atlanta, Georgia, USA
17) Patrick Goldsack, Julio Guijarro, Steve Loughran, Alistair Coles, Andrew Farrell, Antonio Lain, Paul Murray, Peter Toft: The SmartFrog Configuration Management Framework, HP Laboratories
18) Andre van der hoek, Dennis Heimbigner and Alexenser l.Wolf: A generic peer to peer repository for distributed configuration management, ,Proceeding of ICSE-18
19) J.White and D. C. Schmidt , D. Benavides and P. Trinidad and A. Ruiz–Cortés: Automated Diagnosis of Product-line Configuration Errors in Feature Models, 12th International Software Product Line Conference,IEEE 2008.
20) Maximilian Koegel, Jonas Helming, Stephan Seyboth: Operation-based conflict detection and resolution, CVSM'09, May 17, 2009, Vancouver, Canada
21) Marcilio Mendonca1, Thiago Tonelli Bartolomei2, Donald Cowan1: Decision-Making Coordination in Collaborative Product Configuration , SAC'08, March 16-20, 2008, Fortaleza, Ceará, Brazil.
22) Maryam Shiri, Jameleddine Hassine, Juergen Rilling : Feature Interaction Analysis: A Maintenance Perspective, , *ASE'07*, November 5-9, 2007, Atlanta, Georgia, USA.
23) Yuqin Lee and Wenyun Zhao : A feature oriented approach to managing domain requirements dependencies in software product lines, proceeding of the first international multi symposium on computer and computational science(IMSCC06)
24) Cheng Thao1 Ethan V. Munson1 Tien N. Nguyen2,:Software Configuration Management for Product Derivation in Software Product Families, 15th Annual IEEE International Conference and Workshop on the Engineering