Variability Management in Software Development using FeatureIDE : A Case Study

Asif Irshad Khan[1], Md. Mottahir Alam[2] and Wajdi Al Jedaibi[1]
[1]Department of Computer Science, FCIT, King AbdulAziz University, Jeddah, Saudi Arabia.
[2]Department of Electrical and Computer Engineering, King AbdulAziz University, Jeddah, Saudi Arabia.

Abstract— Software Product Line (SPL) Engineering is a widely used strategy for the efficient development of family of software products that have common as well as variable features. In this approach, software artifacts such as requirements specification, system architecture and design, components, etc are reused across the family of a product line with/without some adaptations. SPL helps in producing quality software products at a relatively shorten time to market as well as reduced development cost through the systematic reuse of software artifacts. This paper discusses the variability concept in software product family using feature based modeling. A case study is conducted to explore all valid combinations of features in order to generate a set of unique products in a family using a variability management FeatureIDE tool. Further, the paper highlights the cross-cutting concern in Variability.

Index Terms—Software Product Line, Variability , Software Architecture, Feature-Oriented Software Development, FeatureIDE

1 INTRODUCTION

IF we look in the past during 90's we were using software application using distributed system in conjunction with component based model like COM, DCOM, CORBA etc, to build single product, but with the present market needs this approach seems to be phased out because of software product families demand, their narrow scope, high development cost, platform/environment specific last but not least a client specific.

Software Product Line (SPL) is increasingly manifests the attention of software development organizations because can addressed software variability and helps in production cost reduction, quality of service (QoS) and speedy released schedule [1]. Most of the software companies who adopted software product line already realized that software product line has capabilities to fulfill the current hunger for mass customization.

Applying software product line in building Software product families using sets of interrelated systems from common assets yield remarkable improvements in customer satisfaction, productivity, and improved time to market with high quality product [2].

Adopting SPL required proper management and implementation techniques as in realistic product lines, variability proliferates, and it is often a cross-cutting concern. Hence, systematic management of the variability between the products is must to avail the benefits of product line engineering [3].

Variability in software is the ability of a system to be efficiently extended, changed, customized or configured for use in a particular context [4], while Product Line Variability describes the variation or differences between the similar systems that belong to a product line in terms of properties and qualities [5].

Numerous languages are developed for efficient and effective implementation of modeling and reasoning in software variability. Most of these languages either lack a solid conceptual basis and/or a thorough formal semantics.

If there are ten different systems, we needed to have 10 different copies. With the advent of web based application the older model became obsolete

- *Dr. Asif I. Khan and Dr. Wajdi Al Jedaibi are working as faculty members in the Department of Computer Science, Faculty of Computing and Information Technology, King AbdulAziz University, Jeddah, Saudi Arabia.*
  E-mail: aikhan@kau.edu.sa , waljedaibi@kau.edu.sa
- *Md. Mottahir Alam is working as a faculty member in the Department of Electrical and Computer Engineering, Faculty of Engineering, King AbdulAziz University, Jeddah, Saudi Arabia.*
  E-mail: amalam@kau.edu.sa.

as application was hosted in at one server and single application is able to serve thousands.

Still the growing cost of development is the nightmare for application consumer. At one side where in the big organization phasing out the legacy system needs hefty amount at the same time Product based company releasing the newer version of a single product almost every year. Equipping with newer software requires training of new software, development cost, maintenance and evaluation cost.
Software Industries are broadly divided into three ITES Information Technology Enabled Services, IT Information Technology services and product Based industries.

Variability in software helped in minimizing the cost and effort and maximizing the efficiency. Like a product based company designs the product which is easy to customize which can sell before ITES with minimum customization. In such way product developed once can be reutilize with minor changes.
Further, Products that incorporate variability are helpful for various purposes for example multiple user segments can be addressed, categorization of price as per the products is possible, portability support for different hardware platforms and operating systems, provides different sets of products customization as per the customer requirement and needs and cover different market areas and market structure[6].

Software variability modeling supports development and reuses of several software artifacts. Variability helps runtime deployment of new version release of already deployed systems. Software variability control commonalities and differences between different component as component differs in the way they communicate and interact, further variability facilitates reuse of software artifacts in multiple products in an organized manner.

The paper is organized as follows: Section I Describes Introduction of the paper, Sections II Discuss Variability in Software product lines, Section III Discuss Feature Modeling, Section IV Case Study of Feature modeling, Section V Highlighted Cross-cutting concern in Variability, Section VI Discusses challenges related to Variability in Software Product Line, Section VII Conclusion and Future Work.

2 VARIABILITY IN SOFTWARE PRODUCT LINES

Software industry is observing and reporting increasingly Software systems ability to vary behavior during their lifecycles, this vary in behavior may be variability in hardware to variability in software systems. For example, Airplanes manufacturers often use different engine control software to construct numerous versions of the same physical engine for different Airplanes models. Second, to achieve economically feasibility, software industry delay design stage to the latest phase in the software deployment lifecycle, as they find it is very expensive to reverse design decisions once they are already taken [7].

Reusability in software engineering is defined as the process of using existing resources [Components, libraries, architecture, code, analysis models, configuration management plans or artifacts] to develop a new computer application. Software reusability helps in developing and maintaining computer application as software is not build from scratch and resources used to build the software are previously used and tested in other similar domain.

Variability by definition means "ability to adjust or likely to change or vary". In Software Engineering variability is by and large understood as capability of a software system or artifact to efficiently extended, configured and customized using an organized structured manner for a particular context [8].
Delaying design decisions until last stages of the development process is possible with the inclusion of Software variability. However, variability has to be maintained (cost of maintaining variant feature) which results in extra costs associated with these delayed decisions. For instance, implementation and testing is required for several version release of a of certain system features [7].

Software product line engineering is employed as one of the systematic software reuse where product line deals with domain specific components [9]. For the last several years Software product line engineering rapidly emerging as an important software development paradigm and promising growing concept in the field of software engineering.
A software product line is a set of products sharing common architecture and features, which also have a capability to deals with product specific features. Software product line is

considered as the next logical step for reusing software components and architecture over a number of different applications in various domains. Some of the benefits an organization can avail by applying software product line are shorter development time with high end of efficiency, improved time to delivery, Easier reuse, maintenance, integration and achieve mass customization.
An example of a software product family can be considered as different versions Microsoft Office (Student/Home Edition, Small Business, Standard, Professional and Ultimate). Variations include the kind upgrade support (Can or can't upgrade to it from earlier versions of Office or new release), applications and features and it supports, licensing agreements, and product price. A software product family is intra organizational, and reuse of core software components within the family [10].

For building large, efficient, high-quality SPF systems it is very important to pay a lot of attention to the general software architecture, explicit architecture yield high quality and Complex products, while quality, diversity, faster time to market , Lower cost maintenance can be achieved by software components. To manage diversity, it is very important to have a generic architecture with explicit variation points for SPF [10].
Applying software product line in building software application brings some risks which are very important to carefully address if a company want to adopt software product line.
For an organization software product line approach is a new technical strategy and it requires staff that are sound skilled as well as both expert technical and organizational management engineers.
Adoption of this new approach is challenging as it concern with the employee's roles and responsibilities. Success of software product line approach can be affected if there are resistances within the adopting organization [9]. Oversight of Sound technical and monitoring to the development effort is a key for successful software product line approach [2].

2.1 VARIABILITY TYPES IN SOFTWARE PRODUCT LINE

: the author [11] highlighted the following different types of software product line variability:

External Variability: This type of variability includes all product line variability that is visible to the customer. It thus includes variability that is essential for defining a product line application. External Variability is typically defined in the requirements , payment methods for point of sale (POS) terminal is an example of an external variability as customers decide for each payment method whether they need it or not (Pay with credit card or Pay with cash).

Internal Variability: Internal Variability includes all variability of the product line which is not visible to the customer. Nevertheless, internal variability is additionally required for deriving a product line application.
Typically documented in other development artifacts, often introduced due to technical reasons, Example for Internal Variability: credit card authorization mode, this variability is invisible to the customer, based on the customer's network infrastructure (Integrated Services Digital Network (ISDN), Digital subscriber line (DSL) modem, Virtual Private Network (VPN)), the installer of the POS terminal chooses one of the three alternatives.

Mandatory Variability: Variability that must be bound in any valid product line application. (More precisely, a variation point that has to be bound) Examples: for each car, a particular engine has to be chosen, for each POS terminal at least one method of payment must be offered.

Optional Variability: Variability that can additionally be bound in an application. (More precisely, a variation point that can but must not be bound) Examples: An air-conditioning system can be included in a car (with a choice between automatic and manual adjustments) another example multi-media ring-tones for a cell phone (with a choice between MP3 and MIDI tones).
Deciding between external and internal variability is influenced by business strategy, marketing and complexity of the software product family.

3. FEATURE MODELING

Feature models are nowadays a popular formalism for representing variability in an SPL [12]. A feature is a prominent or distinctive user-visible aspect, quality, or characteristic of a software system[13].

It can also be defined as an increment in functionality provided by one or more members of a product line [14].
The graphical representation of a feature model is called as feature diagram, Feature modeling is evolved around the notions of mandatory and option feature and sub-feature of a product, these hierarchy relations helps the selection process of different product in the product family.
Feature diagram is represented using the concept of a directed tree, The Root node in the directed tree illustrate a concept and other nodes represents features of the concept.

The subfeatures of a feature can either be optional or mandatory, or can be in an Alternative-group or an Or-group[15]. Feature diagrams plays a vital role in the feature modeling. Feature models helps to express variability in software product lines by reporting features and their applicable combinations[15].

FeatureIDE [16] is an Eclipse based plug-in that provides integrated Development Environment (IDE) supports for all phases of feature-oriented software development from domain analysis, domain implementation, requirements analysis, to the software generation[15] [16].

To explain the very basic concept of software product line (SPL), we take an example of human family, we consider (wife- husband , son and daughter) are the features of a family. A family consist of mandatory feature wife and husband and optional feature son and daughter. Fig. 1, shows feature model of this software product line (SPL).

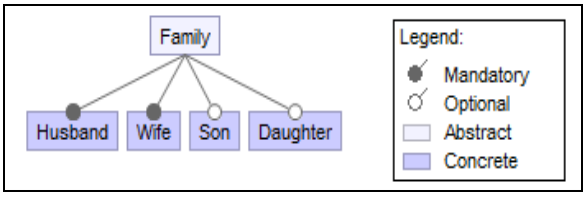

Fig.1

Java source of the mandatory and optional features are given in table 1. from the table it is very clear that, son and daughter optional feature extends husband and wife mandatory feature. The husband and wife feature defines a single Java file, Family. The optional features son and daughter extend Family.

TABLE 1: SHOWS JAVA SOURCE FOR FAMILY SPL.

Husband & Wife	Son	Daughter

class Family {
String wife, husband;
public Family(String wife, String husband) {
this.wife = wife;
this.husband = husband;
}
public void print() {
System.out.println(wife +" & "+ husband);
}
 	
/*****************/
public static void main(String[] args) {
Family myfmily = new Family("Bob","Alice");
myfmily.print();
}
}
	class Family {
	String son;
	Family(String son) {
	this.son = son;
	}
	public void print() {
	
	System.out.println(this.son);
	}
	
	/*****************/
	public static void main(String[] args) {
	Family myfmily = new Family("Moody");
	myfmily.print();
	}
	}
		class Family {
		String daughter;
		Family(String daughter) {
		this. daughter = daughter;
		}
		public void print() {
		System.out.println(this. daughter);
		}
		
		/*****************/
		
		public static void main(String[] args) {
		Family myfmily = new Family("Lily");
		myfmily.print();
		}
		}

In the given SPL example, there are 4 achievable products (programs). The table 2 lists each configuration/program and the possible output after the program get executed:

TABLE 2: SHOWS CONFIGURATION AND ITS OUTPUT

Configuration	Production (product)
Husband & Wife	Bob & Alice
Husband & Wife + Son	Bob & Alice Moody
Husband & Wife + Daughter	Bob & Alice Lily
Husband & Wife + Son + Daughter	Bob & Alice Moody Lily

in SPL configuration is nothing but selection and de-selection of the features from the feature model. Fig. 2 below shows the selection of the Husband-Wife and Son features, so, there are two possible configuration in this selection as shown in Fig. 2.

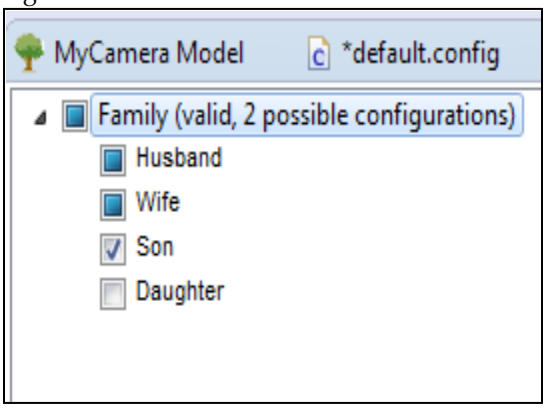

Fig. 2 shows selection of features in SPL

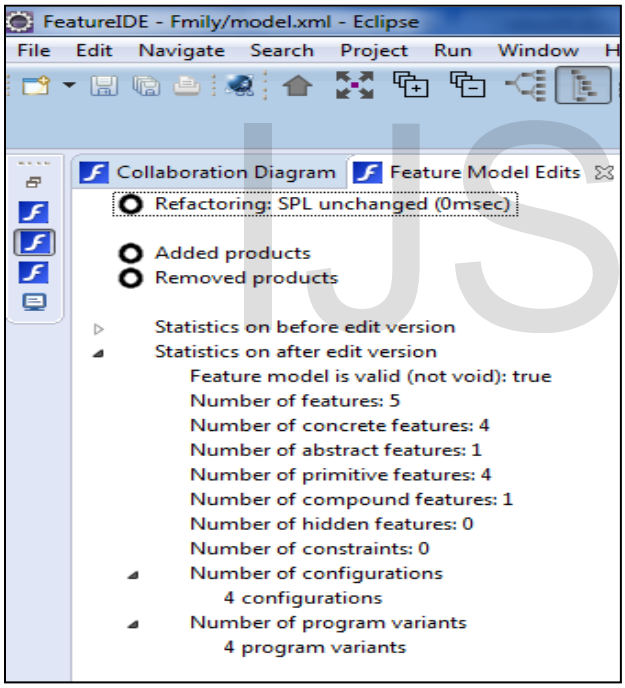

Fig. 3 shows statistics of family in SPL

The Fig. 4 below shows the selection of the Husband-Wife, Daughter and Son features, so, there is only one possible configuration in this selection as shown in Fig. 3.

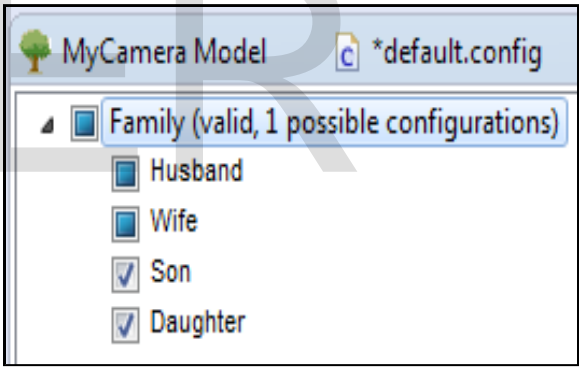

Fig. 4 shows selection of features in SPL

4 CASE STUDY:

In this section, we try to explore in detail about feature modeling with the help of a camera feature model case study. Now a days, customers have the advantages of selecting available features in a camera as per their requirement by selecting different variants in a given product line of camera. Features such as Body type selection, screen selection, flash selection, eye auto focus, face recognition, and tracking are customizable.

Domain analysis is first done, domain and their dependencies is constructed using a Feature model tool (FeatureIDE), a feature model can be constructed graphically by adding and removing features in a FeatureIDE graphical editor[16].

Fig. 5 shows a feature model of a system, called Camera . Nodes represent features and edges show relationships among features. A single root node,

Camera, represents the specific domain being modeled.
Connections between a feature and its group of subfeatures can be and-, or-, and alternative groups [17] .
The sub-features of and-groups can be either mandatory or optional. Optional features are represented with an empty circle, such as Remote Control, Wireless etc in Fig. 5.
These features may or may not be part of a product. On the other hand, mandatory features, such as Body Type, Image Sensor, and Connectivity etc are represented by filled circles and are part of all products in that specific SPL. Further, there are alternative features which may be exclusive (XOR) or not exclusive (OR).

Example of exclusive feature is Body Type as shown in the Fig 5. representing the Feature Diagram of the Camera. A feature is exclusive implies that only one sub-feature can be selected from the alternatives. Thus, Compact, UltraCompact, and LargeSensorCompact are alternative features for Body Type. Examples of OR features in the Fig. 5 are Articulated LCD, Screen size, Touch Screen, and Live View which allows the selection of more than one option for a product.

Apart from features and their relationships, a feature model can also include composition rules [18]. A composition rule refers to additional cross-tree constraints to restrict feature combinations [12]. It helps in validating a combination of unrelated features which cannot be expressed otherwise. A cross-tree constraint is a propositional formula over the set of features and usually shown below the feature diagram [16].
.

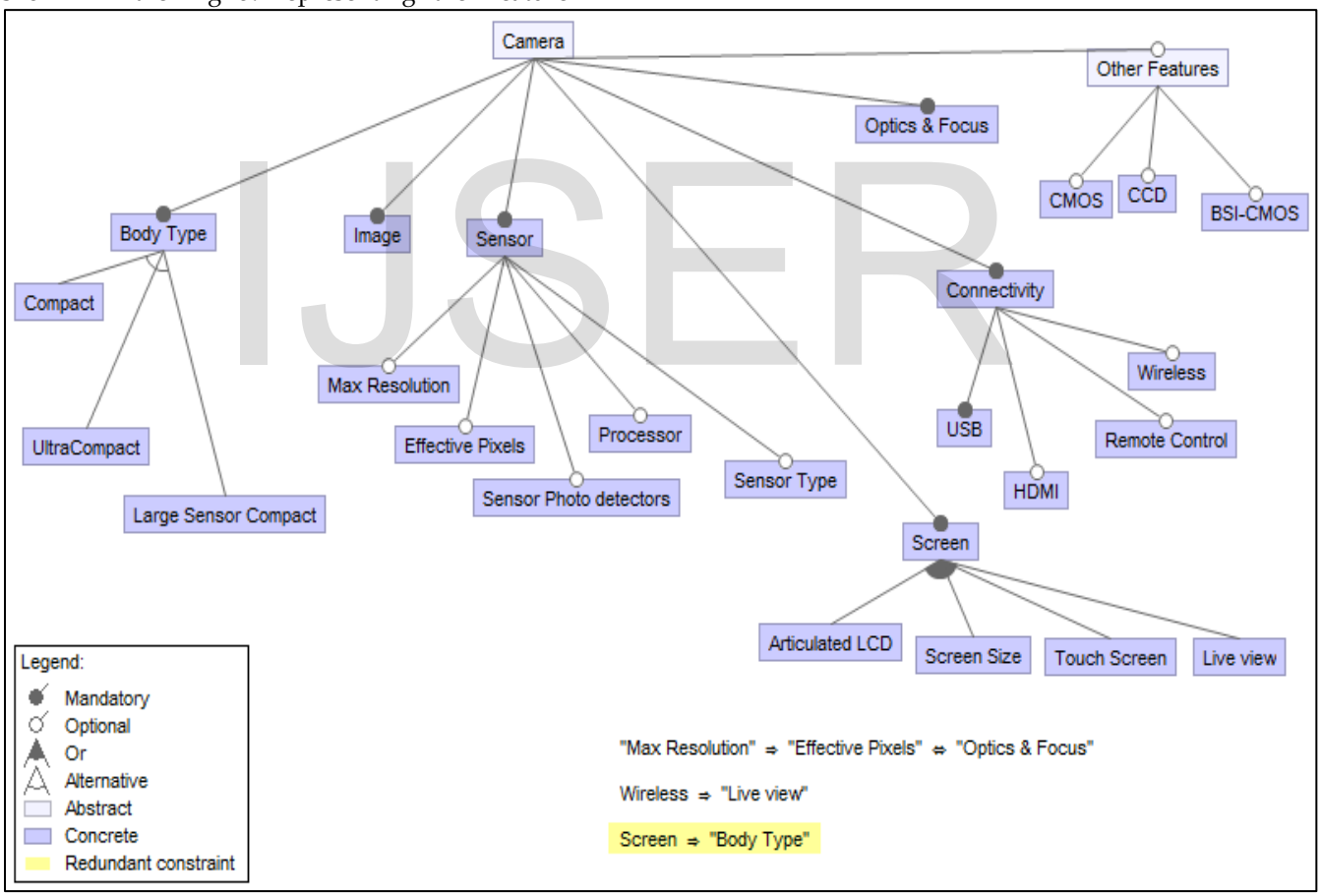

Fig. 5: Feature-diagram example of a camera product

In brief, a Feature model can be summarized as follows: the selection of a feature implies the selection of its parent feature. Also, if a feature is selected, all mandatory sub-features of an and-group must be selected. For optional feature, at least one sub-feature must be selected while in case

of alternative feature, exactly one sub-feature is to be selected line.

Finally, all cross-tree constraints must be fulfilled. A configuration is a subset of all unique and valid combination of features defined in the feature model. A configuration is valid only if the combination of features is allowed by the feature model.

FeatureIDE also support requirements analysis by a configuration editor. domain implementation is also supported by FeatureIDE using SPL implementation tools , The editor gets the feature model from domain analysis as input and offers configuration choices[16].

Configuration editor as shown in Fig. 6 shows an invalid configuration and some highlighted features. Selecting one of the green features results in a valid configuration.

On the right side, the advanced configuration editor is shown, in which features can be eliminated to reduce the remaining configuration options.

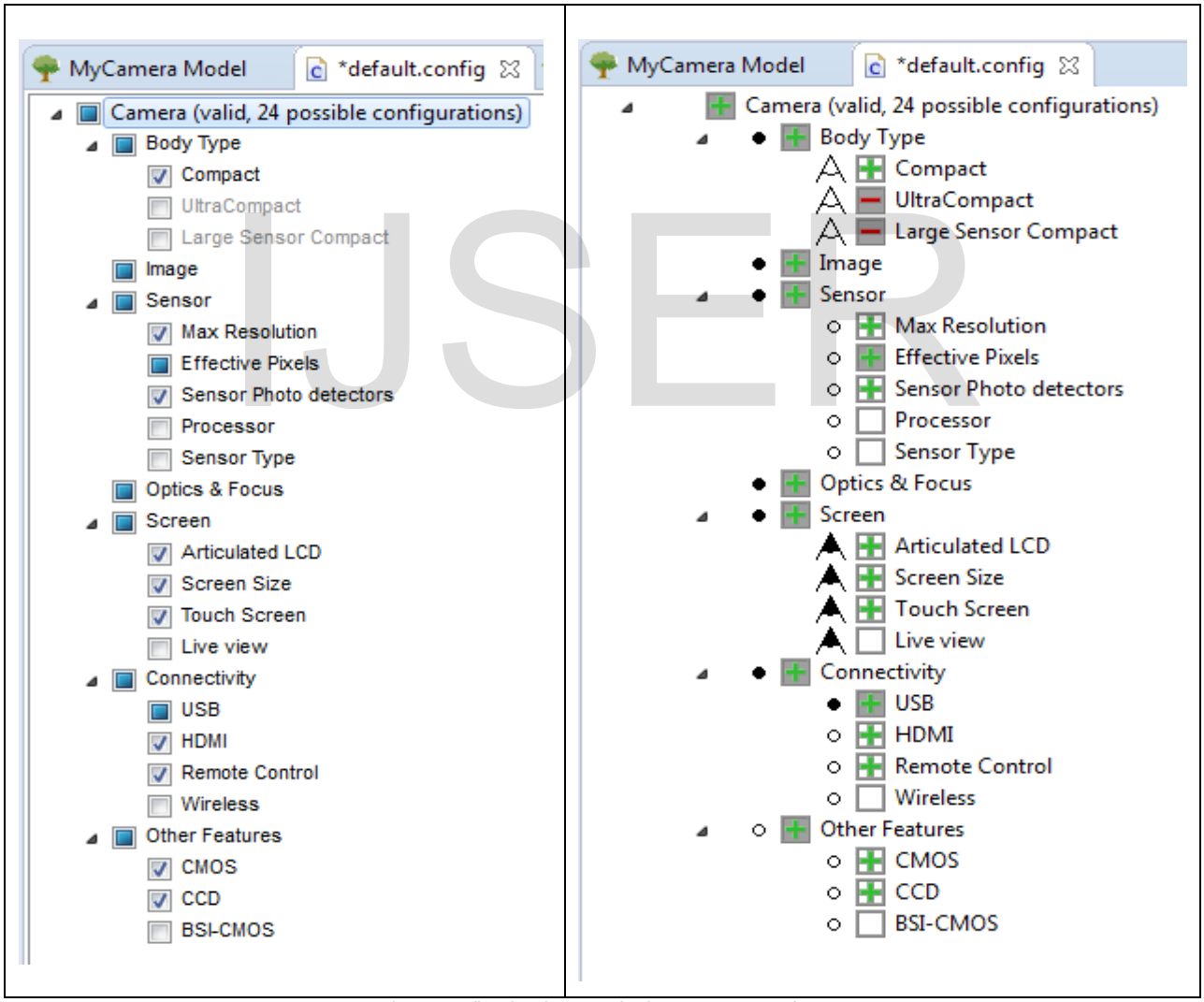

Fig 6: Configuring features in the camera product

5 CROSS-CUTTING CONCERN IN VARIABILITY

The term crosscutting concerns refers to common functionality every software application needs like authentication, authorization, caching, exception management, logging and instrumentation, validation etc. but these are not necessarily must have components. And depending on the environment and exposure these things needs to be handled. Following are few examples to illustrate the concept further [19]:

Authentication: to verify who you are - i.e. Is the user really who he/she represents himself to be? Now you may or may not need this step depending on scope of the application. But if it is User Interface (UI) based most likely you should have one. But if it intranet based running in inside firewall i.e. only for internal user, does not do any update but view only and application takes care of volume i.e. it restricts or deny request which may hang system/network etc. And there are applications which do not have UI and to communicate to the application is to write some code/interface then you may not need it.
In one application back-end service I worked which mainly does read only service and run inside corporate network inside firewall- did not have any authentication. Now the authentication used to be part of applications and still it is for much new and old application. But now it is changing, often organization are using common tool/app may be third party like site minder, or using common LAN ID (LDAP etc.) which restrict outside user. So the authentication part is outside of the application.

Authorization: Is the process or a method by which a computer system verifies about level of access an authenticated user should gain. So in UI based application user accessibility of system may be different based on the assigned privileges, some function may be disabled/invisible to user. Now, this component is also used to be integral part of application.
Trend has changed how to implement those as separate entity and pluggable. For example in some project the entitlement is done centrally and other application plug into this service. Other entitlement is to data e.g. in a financial company different user will have different access level to a fund. This may be one row in database table but will have different access. This access level can be implemented in each application or subscribe to a service specific for that purpose.

Caching: Depending on the application performance/scalability application needs to cache data. Caching usually means caching within the application but that concept is slowly diminishing. There are different caching methodologies that can be used. For example caching for limited time (allotted time) or caching as per availability of memory, First in First out (FIFO)/ Last in First out (LIFO), or based on frequency of use.
Other implementation is similar to virtual memory which does not limit to available memory. Caching technique used to be spread across the application but now mostly centralized caching is used which are more configurable and manageable. Compartmentalize caching is done using different service based on category of data.
For example instead of storing all reference data in the application itself it uses other reference data service which does the caching. Modern architectures are done to use easily pluggable caching service. Refreshing caching is done on demand or by some automated process, or sometime event base e.g. when data is changed at source some event can be generated to refresh caching.

Communication: is a mainly deal with the communication between components across layers and tiers. Deployment scenarios play an important role and the application must support the communication mechanism, many enterprise application uses message queue to communicate between different functionality and application. For example to remove caching and reload data can be done using MQ. Other scenario is communication with other application/layer is to make it transparent to calling part and making it configurable.

Exception management: another crosscutting concern responsible for security and reliability. Increasingly getting customized based on application. One popular exception handling is bubble up i.e. logged and transformed as necessary before passing them to the next layer. Failure should not leave the application in an unstable state/failed state, and that exceptions do not allow the application to reveal sensitive information or process details. Exception logging can be done as

simple text out or store in database or some time creating application specific format to recover/replay.

Logging and instrumentation: logging may be required for different purpose, one common use is debugging application exception, but based on scope of application it can have other use e.g. a financial application may required to maintain the signature of each stage of transaction for audit or legal purpose. Logging is implemented some time to use trace listener so as to configure at runtime. New application mostly use centralized specialized component for logging some time services are used for logging.

Validation: Helps in maintaining reliability as well as usability of an application. Lack of validation in an application results several issues such as data redundancy and inconsistencies, violations found in business rules and unacceptable usability experience. In addition, security issues such as cross-site scripting attacks, SQL injection attacks; buffer overflows etc. can be vulnerable to application. Organization often uses already built in component or service which proved to do good amount of validation. There are rule based validations which can be configured without doing code change.

6 CHALLENGES RELATED TO VARIABILITY IN SOFTWARE PRODUCT LINE

Variability in software product line comes up with broad range of challenges both technical and non-technical; following are some of such challenges highlighted by [20]

- How to model variability
- How to handle system complexity
- How to handle product line architecture and documentation issues.

Variability assessment means process of finding how, when and where variability should evolve, variability assessment analyses the product family artifacts to find the mismatch among them [21] , also, it helps to know how far the product family be able to support a new product. Important issues related to variability assessment are listed by [21]

- First issue is the lack of explicit methodological guidance resulted in unpredictable outcomes and waiting of efforts.
- Second issue is the lack of availability of expert and timely pressure resulted as mostly assessments are done for needs of immediate problems.
- Third issue as per author is generalization over a number of features.
- If obsolete variability are not discarded from the core functionality, it leads to complexity of the product family but predictability and traceability reduced.
- Another issue that is highlighted by the authors is the lack of techniques for addressing variability at all levels of abstraction, for cases where there is a major change in the product family artifacts at component and architecture level.
- In case of variability mismatch occurrence there is a lack of alternative solutions.

7 CONCLUSION AND FUTURE WORK

Software Product Line is an emerging methodology to develop a family of software products by reusing sets of artifacts. In this paper we used an open source tool, FeatureIDE, to conduct a case study to analyze different aspects of variability management in SPL.

We found that FeatureIDE is one of the best available tool that can be integrated with the development process and it supports different languages for SPL implementation. Further, we discussed cross-cutting concerns and challenges related to variability in software product line.

In future we will extend our study to perform qualitative analysis on different available variability management tools.

REFERENCES

[1] F. Ahmed, L. F. Capretz, The software product line architecture: An empirical investigation of key process activities, Information and Software Technology, v.50 , (October, 2008.), n.11, p.1098-1113.

[2] a framework for software product line practice, version 5.0, software engineering institute, carnegie mellon university, web url: http://www.sei.cmu.edu/productlines/frame_report/introduction.htm

[3] M. Voelter, Handling Variability, Version 2.0, December 11, 2009, Hillside Europe.

[4] M. Svahnberg, J. V. Gurp, and J. Bosch. , A Taxonomy of Variability Realization Techniques. Software Practice and Experience 35 (2005), no. 8, pages 705-754.

[5] A. Metzger, P. Heymans, K. Pohl, P.Y. Schobbens and G. Saval, Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and

automated analysis, In: Sutcliffe, A., ed.: Proceedings 15th IEEE International Conference on Requirements Engineering, 15-19 October 2007, New Delhi, India, IEEE Computer Society, 2007.

[6] T. Asikainen, A conceptual modeling approach to software variability, Doctoral Dissertation, Helsinki University of Technology, Faculty of Information and Natural Sciences, Department of Computer Science and Engineering, 2008.

[7] A. Lapouchnian, Exploiting Requirements Variability for Software Customization and Adaptation. PhD. Thesis, Department of Computer Science University of Toronto. 2011.

[8] M. Galster and P. Avgeriou, Handling Variability in Software Architecture: Problems and Implications, Proceedings of the Ninth Working IEEE/IFIP Conference on Software Architecture, 2011.

[9] p. Kuvaja, j. Similä and h. Hanhela, software engineering techniques, lecture notes in computer science, 2011, volume 4980/2011, 143-157.

[10] r. V. Ommering and j. Bosch, building reliable component-based software systems , components in product-line architectures, (2004), artech house boston ,london.

[11] K. Pohl , Paluno, The Ruhr Institute for Software Technology , Universität Duisburg-Essen , web url: http://www.cse.msu.edu/~cse435/Lectures/2010-Lectures/Notes/Lecture11b- Product-Line-Variability-Pohl-notes.pdf

[12] Czarnecki, K., Helsen, S., and Eisenecker, U. "Formalizing Cardinality-based Feature Models and Their Specialization".Software Process: Improvement and Practice, vol. 10, issue 1,2005, pp. 7-29

[13]Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., Peterson, A. S., 1990. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Tech. Rep. CMU/SEI-90-TR-21, Software Engineering Institute.

[14] Batory, D., Sarvela, J. N., and Rauschmayer, A. Scaling step-wise refinement. In ICSE '03: Proceedings of the 25th International Conference on Software Engineering (Washington, DC, USA, 2003), IEEE Computer Society, pp. 187–197.

[15] A Quick Tutorial on FeatureHouse and FeatureIDE, http://www.cs.utexas.edu/users/dsb/cs392f/Videos/FeatureHouse/FHTutorial.htm

[16] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich.FeatureIDE: An Extensible Framework for Feature-Oriented Software Development. Science of Computer Programming, 79(0):70-85, January 2014.

[17]Batory, D., 2005. Feature Models, Grammars, and Propositional Formulas. In: Proc. Int'l Software Product Line Conference (SPLC). Springer, Berlin, Heidelberg, New York, London, pp. 7-20.

[18] T. Thüm, C. Kästner, S. Erdweg, N. Siegmund, Abstract features in feature modeling, Proc. Int'l Software Product Line Conference, SPLC, IEEE, Washington, DC, USA (2011), pp. 191–200.

[19] J.D. Meier, A. Homer, D. Hill, J. Taylor, P. Bansode, L. Wall, R. Boucher Jr and A. Bogawat, Business Layer Guidelines, patterns & practices Application Architecture Guide 2.0.

[20] L. Chen, M. A. Babar , N. Ali, Variability management in software product lines: a systematic review, Proceedings of the 13th International Software Product Line Conference, August 24-28, 2009, San Francisco, California.

[21] S. Deelstra, M. Sinnema, J. Bosch, Variability assessment in software product families , in Journal of Information and Software Technology, 51 (2009), pp. 195–218.

[22] A. I. Khan and et. al., "A Comprehensive Study of Commonly Practiced Heavy and Light Weight Software Methodologies", IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 4, No 2, July 2011, ISSN (Online): 1694-0814, www.IJCSI.org.

[23] A. I. Khan and et. al., "An Improved Model for Component Based Software Development", Software Engineering, Vol. 2 No. 4, 2012, pp. 138-146. doi: 10.5923/j.se.20120204.07