Tracing Software Product Line Variability – From Problem to Solution Space

KATHRIN BERG, JUDITH BISHOP
University of Pretoria
and
DIRK MUTHIG
Fraunhofer IESE

The management of variability plays an important role in successful software product line engineering. There is a need for a universal variability management approach to be consistent and scalable; it should provide traceability between variations at different levels of abstraction and across various generic development artifacts; and there should be a means for visualizing variability. Focusing specifically on the aspect of traceability in the context of such an approach, we define a conceptual variability model that captures variability information in a third dimension, and allows a 1-to-1 mapping of variability between the problem space and the solution space. Decision models, of which the feature model is most popular, are commonly used for, amongst others, managing traceability of variation. These, however, usually reside in a two dimensional space. We analyze the feature model in a small case study with regards to our conceptual variability model, and present our findings.

Categories and Subject Descriptors: D.2 [**Software**]: Software Engineering – *Reusable Software*
General Terms: Management, Documentation
Additional Key Words and Phrases: software product line engineering, variability management, traceability

1. INTRODUCTION

Software product line engineering is an approach that develops and maintains families of products while taking advantage of their common aspects and predicted variabilities [Weiss and Lai 1999]. A product line infrastructure is developed based on the commonality and variability that exists between the products in a family. Commonality defines those characteristics that are the same between products, whereas variability implies the opposite, i.e. the differences between the products.

"Software variability is the ability of a software system or artifact to be changed, customized or configured for use in a particular context. A high degree of variability allows the use of software in a broader range of contexts, i.e. the software is more reusable" [Bosch 2004]. Software, specifically in software product lines, needs to support increasing amounts of variability, so that the complexity of managing the amount of variability becomes a main concern that needs to be addressed.

Managing variations at different levels of abstraction and across all generic development artifacts[1] is a daunting task, especially when the systems supporting various products are very large, as is common in an industrial setting. Traceability will improve the understanding of system variability, as well as support its maintenance and evolution. With large systems the necessity to trace variability from the problem space to the solution space is evident. Approaches for dealing with this complexity of variability need to be clearly established.

In [Berg and Muthig 2005], it has been recognized that a unified approach for variability management has yet to be defined. There are four essential characteristics that such an approach must possess. These are consistency, scalability, traceability and a means for visualization. Therefore, only a *consistent* approach that is *scalable*, provides *traceability* between points of variation at different levels of abstraction and throughout all development phases, as well as provides a means for *visualizing* variability will be appropriate for the management of variation in a product line. This paper focuses on one of the four aspects, namely traceability, in the context of such a unified variability management approach.

[1] A generic artifact is a product line artifact that contains variant elements representing product line variability.

Author Addresses:
K. Berg, Department of Computer Science, University of Pretoria, 0002, South Africa; kberg@cs.up.ac.za.
J. Bishop, Department of Computer Science, University of Pretoria, 0002, South Africa; jbishop@cs.up.ac.za.
D. Muthig, Fraunhofer Institute Experimental Software Engineering (IESE), Sauerwiesen 6, D-67661 Kaiserslautern, Germany; Dirk.Muthig@iese.fraunhofer.de.

Proceedings of SAICSIT 2005, Pages 182 –191

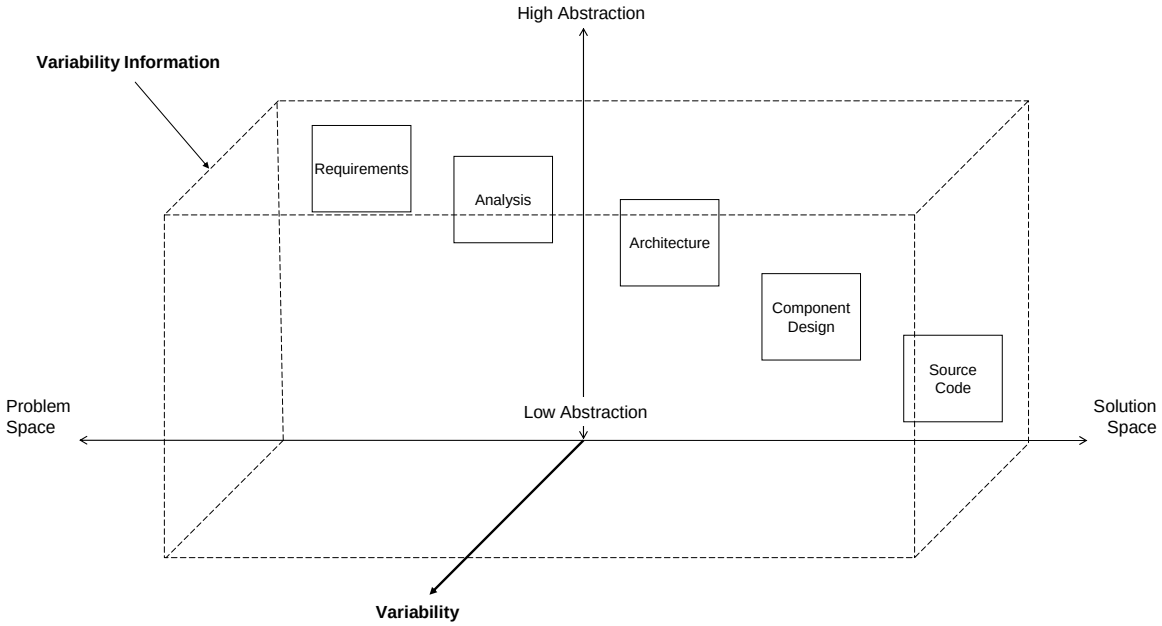

Figure 1. *Three dimensions for software product line engineering*

In literature it is acknowledged that there is a need for establishing general traceability among all artifacts in both problem and solution space, which includes variability information in software product lines. Many tools and approaches for handling different facets of traceability exist [von Knethen 2001]. However, even though tool support is a necessity, current tools still allow human decision making only in a limited way to interfere with the static development processes inherently supported by the tools. Moreover, experience has shown that it is practically infeasible to provide traceability for every single aspect consistently throughout development phases and over time.

For this reason, and the earlier mentioned importance of explicitly managing variability in software product lines, we limit our focus on tracing variability only. Since variability distinguishes single system development from system family development and determines, to a great extent, the success of a software product line, it is more beneficial to establish 100% traceability for variation between generic artifacts, than 80 % traceability between all artifacts.

We define a conceptual variability model that addresses traceability of variations at different levels of abstraction and across various generic development artifacts, while adhering to the specified requirements of a unified approach. The model is concerned with explicitly describing variability information[2] in the third dimension and thus also providing a 1-to-1 mapping of variation between problem and solution space.

Software engineering for single systems has thus far been done in two dimensions. One dimension represents the phases of development and the other, levels of abstraction. All development artifacts can be placed somewhere in these dimensions. With the addition of variability in software product line engineering, we add a third dimension that explicitly captures variability information between product line members. It is the management of variability in this dimension that our conceptual model deals with. See Figure 1.

Simply identifying and modeling variabilities among the products in a product line does not define what features are associated with what products, as well as what dependencies and interrelationships exist among variabilities. This information is captured in what is called a decision model. Essentially a decision model consists of decisions that relate user visible options to specific system features and finally to variation points within generic artifacts.

We consider feature modeling, a common decision modeling approach known in literature to be used for the management of variabilities in software product lines. The feature model has been recognized by Riebisch [2003, 2004] to be the model that closes the gap between the problem space and the solution space by providing means for tracing variability. With the use of a small library system product line case study, we critically analyze the feature modeling approach with regards to our conceptual variability model for traceability, and compare the results to determine whether they coincide with the requirements set for a unified approach.

The main contribution of this paper is aimed at establishing a conceptual variability model for tracing variability in software product lines firstly, by explicitly modeling variability information in a third dimension separate to that of

[2] Variability information refers to points of variation, their relationships and dependencies.

other development artifacts, and secondly, by satisfying the requirements for a unified variability management approach.

Section 2 of this paper defines the requirements for a unified variability management approach. In Section 3 the concepts of traceability between development spaces is described, followed by the conceptual variability model for supporting traceability of variation in the third dimension. Section 4 briefly describes the feature modeling approach with regards to tracing variability in a library system product line, followed by an analysis and comparison of the model with regards to our conceptual variability model. Section 5 concludes the paper and describes future work.

2. A UNIFIED APPROACH FOR VARIABILITY MANAGEMENT

To harvest the full benefits of software product lines, variability needs to be managed in an appropriate and consistent way across all software development phases. That is, independent of the level of abstraction or the product-specific context, variation should be easily managed in a unified approach. According to Beuche et al. [2004], methods for supporting variability management need to consider the following:

- Models expressing commonality and variability to support variability management need to be simple, yet universal;
- Variability must be manageable at all levels of abstraction;
- The introduction of new variability expression techniques should be easily possible.

A unified approach needs to possess four important characteristics when being considered for managing variability. The approach needs to be consistent, scalable, provide traceability between points of variation at different levels of abstraction and throughout all development phases, and provide a means to visualize the variability.

- *Consistency*: Standardization can prevent confusion and the incorrect usage of an approach. Variability should be handled the same way at different levels of abstraction and across development phases. A consistent approach reduces the possibility of errors that might occur when using different methods for managing variability at different abstraction levels.
- *Scalability*: Whether dealing with only a single component or a large complex system, variability must be easily manageable. It is not sufficient for a method to be successful in managing variability on a small scale, but becomes too complex to handle on a larger scale. A method should not only be scaleable upwards, but also downwards. That is, it should be economically useful when applying product line principals to only a small system or sub-system without extraordinary effort and overhead.
- *Traceability*: Variability at different levels of abstraction and across development phases are associated with each other and need to be linked to simplify evolution and maintenance of a software product line. These multi-dimensional relationships need to be appropriately managed.
- *Visualization*: The visualization of variability and its dependencies between products in a product line promotes understandability of variation and provides an overview thereof.

As shown in [Berg and Muthig 2005], an approach satisfying all of the above requirements has not yet been clearly defined. This paper only concentrates on the aspect of traceability in the context of such a unified approach.

3. CONCEPTUAL VARIABILITY MODEL FOR TRACEABILITY

Traceability is the link describing a relationship or dependency between two artifacts developed during the various phases of software engineering. Being able to capture this traceability is a challenging task. Moreover, with the introduction of variability in product lines, the relationships and dependencies between generic artifacts become even more intricate, and therefore even harder to capture and manage. Well established traceability improves the comprehension of the product line infrastructure and its product-members' development, and provides support for their evolution and maintenance [Sametinger and Riebisch 2002].

The concept of our variability model is based on some of the ideas of multi-dimensional separation of concerns, where software artifacts are modeled and implemented by permitting separation of overlapping concerns along multiple dimensions of composition and decomposition [Tarr et al. 1999]. Our model is intended to encapsulate all and only the *variability* concerns across the development space. It separates the variability information out of generic artifacts, and thereby facilitates traceability.

3.1 The Concepts

The terms *problem space* and *solution space* have been previously introduced by Czarnecki and Eisenecker [2000]. The problem and solution space together represent the development phases of software product line engineering, as in the context of this paper. See Figure 2. The problem space generally refers to systems' specifications established during the domain analysis and requirements engineering phases, whereas the solution space refers to the concrete systems created during the architecture, design and implementation phases. All the developed artifacts collectively form the product line infrastructure.

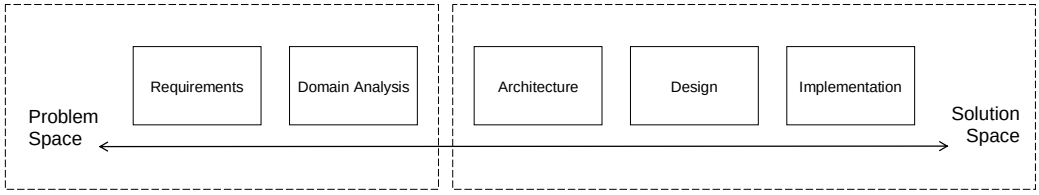

Figure 2. *The Problem and Solution Space*

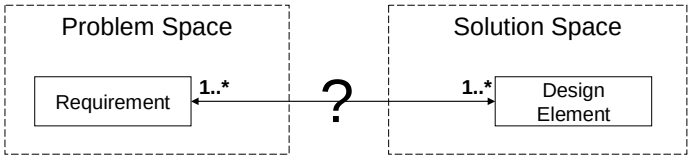

Figure 3. *n-to-n Traceability between Problem and Solution Space*

Traditionally a decision as to whether a certain characteristic is included in a product or not was made in the problem space during scoping activities and the software product was designed accordingly. However, with the development of a product line, these decisions need to be delayed in the development process to the point where it is most beneficial to an organization [van Gurp et al. 2000]. Consequently, variability must not only be considered in the problem space, but also in the solution space, and therefore needs to be managed at each development phase, from the initial requirements to the final implementation [Myllymäki 2002].

Variability is realized in all generic artifacts as *variation points*. According to Jacobson et al [1997], "a variation point identifies one or more locations at which the variation will occur". Czarnecki and Eisenecker [2000] state that "variation points allow us to provide alternative implementations of functional or non-functional features". Variation points may thus appear in various generic artifacts and at any level of abstraction. The relationships and dependencies between variation points need to be explicitly captured and appropriately managed in order to reap the benefits of traceability, as mentioned above.

Ideally, traceability should provide a link for relationships and dependencies between all variation points in the development artifacts, from the problem space to the solution space [Sochos et al. 2004]. Such a link would imply a 1-to-1 mapping. Currently, the opposite is true. As one moves between the problem space and the solution space there is usually an n-to-n mapping between, for example, variations specified in the requirements and variations modeled in the design. To exemplify, a variation point in the requirements specification may be realized by a number of variation points scattered across various components. At the same time, a single component might realize a number of variable requirements. See Figure 3. In Tarr et al. [1999], such relationships are described as scattering and tangling.

3.2 The Model

To achieve a 1-to-1 mapping between variability represented in the problem space and the solution space, it is necessary to introduce a conceptual variability model that explicitly captures and structures the individual variation points' information for each generic artifact, i.e. their relationships and dependencies, and thus provides the appropriate mapping between all variation points in the two dimensional space. The model does not represent yet another level of abstraction in the development process, but rather a third dimension for variability, covering the whole development space. It explicitly and uniformly captures and structures the variation points of each artifact and traces them to their appropriate dependent or related variation points in other artifacts. See Figure 4.

The structure of development artifacts are determined by structures useful for capturing system information, such as requirements, architectural views, or design information. Variation points, representing variability, are woven into these structures given by single system software development approaches. Hence, the room for capturing relationships among variation points, even in a single two dimensional artifact, is limited. This is not the case when working in the third dimension. There, all variation points are located in one space. They are kept in a structure that is optimized for capturing relationships and constraints among variation points, or decisions, without a direct link to a development artifact's variation point.

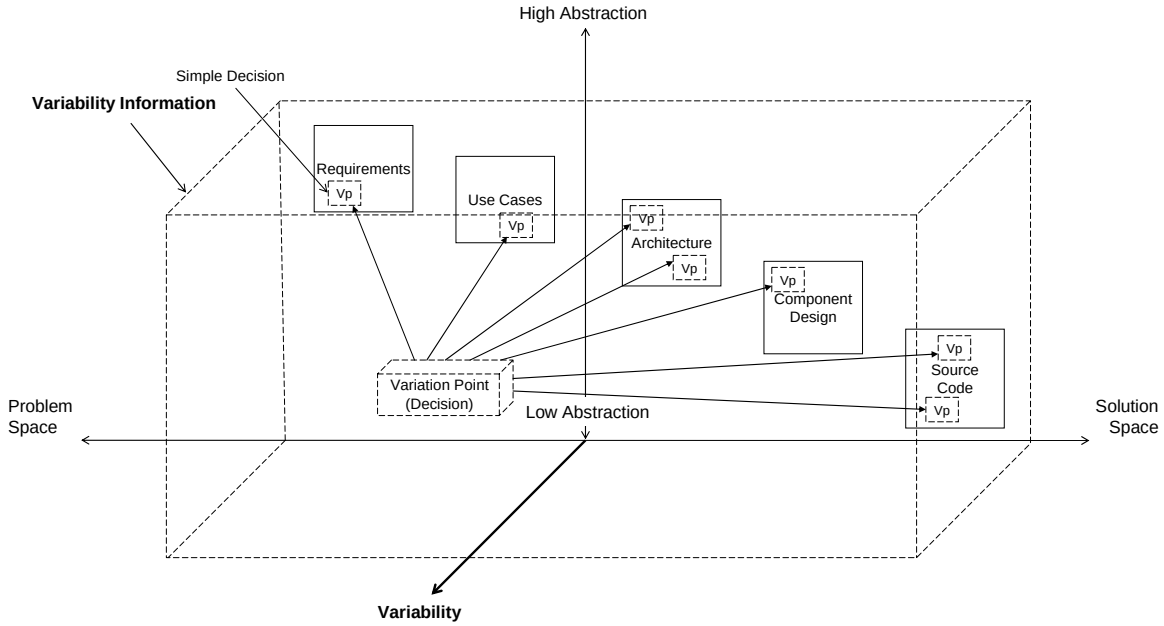

Figure 4. *Conceptual Model for Traceability*

Every variation point is related to a decision [Muthig and Atkinson 2002]. *Decisions* are variation points that constrain other variation points and that can explicitly be related to a domain concept. Hence, decisions are meant to structure and document variation points, especially their inter-relationships and dependencies. A decision defines a question that captures the essence of the related variability in the sense that answering this question corresponds to resolving the decision, which is a variation point. A *simple decision* is a variation point identifying only one location in a generic artifact. It is a decision that is directly related to a generic artifact and that does not constrain other variation points.

In [Muthig 2002], a metamodel for product line artifacts is described. Here, a variation point, i.e. an optional requirement or class, is defined as both a variant model element and a decision. Hence, if you create a variant model element in the two dimensional development space, this model element also has a third dimension in the variability direction. Because the model element cannot be separated from the directly connected simple decision, both together being the variation point, information of these elements are created in the variability dimension by nature.

Looking from the variability dimension, only at the bottom, in the two dimensional space of development, decisions are simple decisions that are part of a generic artifact. All other decisions in the variability dimension are not generic artifacts, but are connected with simple decisions via constraints. Therefore, every decision in the conceptual variability model is grounded in the development space. Figure 7 illustrates an example of our model.

By not restricting the model to a specific development space and capturing individual variability information for each generic artifact, the model conforms to the requirements of a unified variability management approach. Since variability information is uniformly captured for individual generic artifacts across all development phases; whether it is for a single stand-alone component or for the whole product line, the model makes provision for consistency and scalability; it also provides all information necessary for visualizing variability with regards to their relationships and dependencies.

4. ANALYSIS OF FEATURE MODEL SUPPORT FOR TRACING VARIABILITY

The feature model is currently used by some product line engineers for the management of variability, and amongst others traceability of variation. The library system product line is used in a small case study to evaluate the use of feature modeling in tracing variability between the problem and the solution space. The following sections briefly describes and analyzes the use of the feature model for traceability support in a library system product line, followed by an comparison of the model with regards to the conceptual variability model for traceability, as defined in section 3.

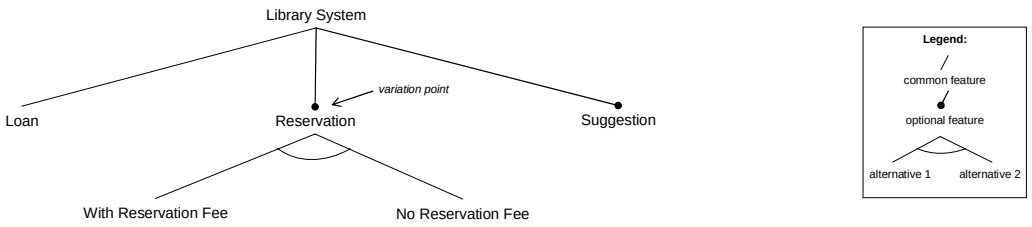

Figure 5. *Extract of a Library System Product Line Feature Model*

4.1 The Feature Model

Feature modeling, originating from the Feature-Oriented Domain Analysis (FODA) method [Kang et al. 1990], is commonly used in literature for the management of variability in software product lines. As part of the Domain Analysis method, feature models are used to describe and hierarchically structure common and variable features for product line-members. Features represent product capabilities and characteristics that are important to the user[3]. A feature indicating variability corresponds to a variation point. See Figure 5.

The feature model, in Figure 5, shows that a library system has, amongst others, a loan feature, and an optional reservation and suggestion feature. The reservation feature has two alternative sub-features, either with or without a reservation fee. For a detailed description of our library system product line case study, see [Bayer et al. 2001].

In [Riebisch 2003], feature models are said to be the artifact that closes the gap that exists between the problem space and the solution space by providing traceability between other development artifacts. By using features to abstract the many requirements of a large product family to a higher level, the complexity of variability is somewhat reduced and an n-to-1 mapping between requirements and features is achieved. However, the additional use of features in the problem space does not affect the n-to-n relationship that remains between variation points of artifacts in the problem and those in the solution space, as illustrated by Figure 6.

4.2 Analysis

When using the feature modeling approach to model variability during the development of the library system product line, requirements are abstracted to a higher level, thereby providing a better overview of the product line features. The relationship that exists between the single variable requirement and the variable feature representing it needs to be established and captured. Furthermore, since the feature model is supposed to provide traceability between problem and solution space, the variable features must also be traceable to the individual variation points in the artifacts of the solution space. Here, only an n-to-n relationship can be established.

Having to create and maintain these relationships between the feature model and other product line artifacts results in the same problem that was established for single-system development; it is not different from general traceability. Since the feature model resides in the two dimensional space, and can be seen as simply another product line artifact, all that it contributes to, is additional effort to created and maintain general traceability links between artifacts.

Many feature modeling approaches are used to represent variability in the problem space, i.e. at the requirements level [Kang et al. 1998], or in the solution space, i.e. at the architecture [Weiler 2003] or source code level [Czarnecki and Eisenecker 2000]. These approaches often do not address traceability between artifacts across problem and solution space. Furthermore, there are a few feature modeling approaches that address traceability between the two development spaces, i.e. features in the problem space are traced to design elements in the solution space by using additional models and tools [Beuche et al. 2004, Sochos et al. 2004]. These, however, usually deal with traceability between features and a specific artifact only, and not between other generic development artifacts as well.

Figure 7 illustrates how the conceptual variability model would improve the traceability of variation from problem to solution space. The *Reservation* variation point in the third dimension contains the variability information, that is the relationships and dependencies, for all the variation points in the various generic artifacts spread across the development space. Therefore, by resolving the variation point in the third dimension, i.e. by simply answering the question of whether a specific Library System instantiation needs to support the reservation feature or not, the individual generic artifacts can be automatically configured where necessary. It is now also possible to trace exactly where and which artifact will be impacted when making changes to a single specific artifact, or when evolving the product line as a whole.

[3] The user represents any interested stakeholder or external system.

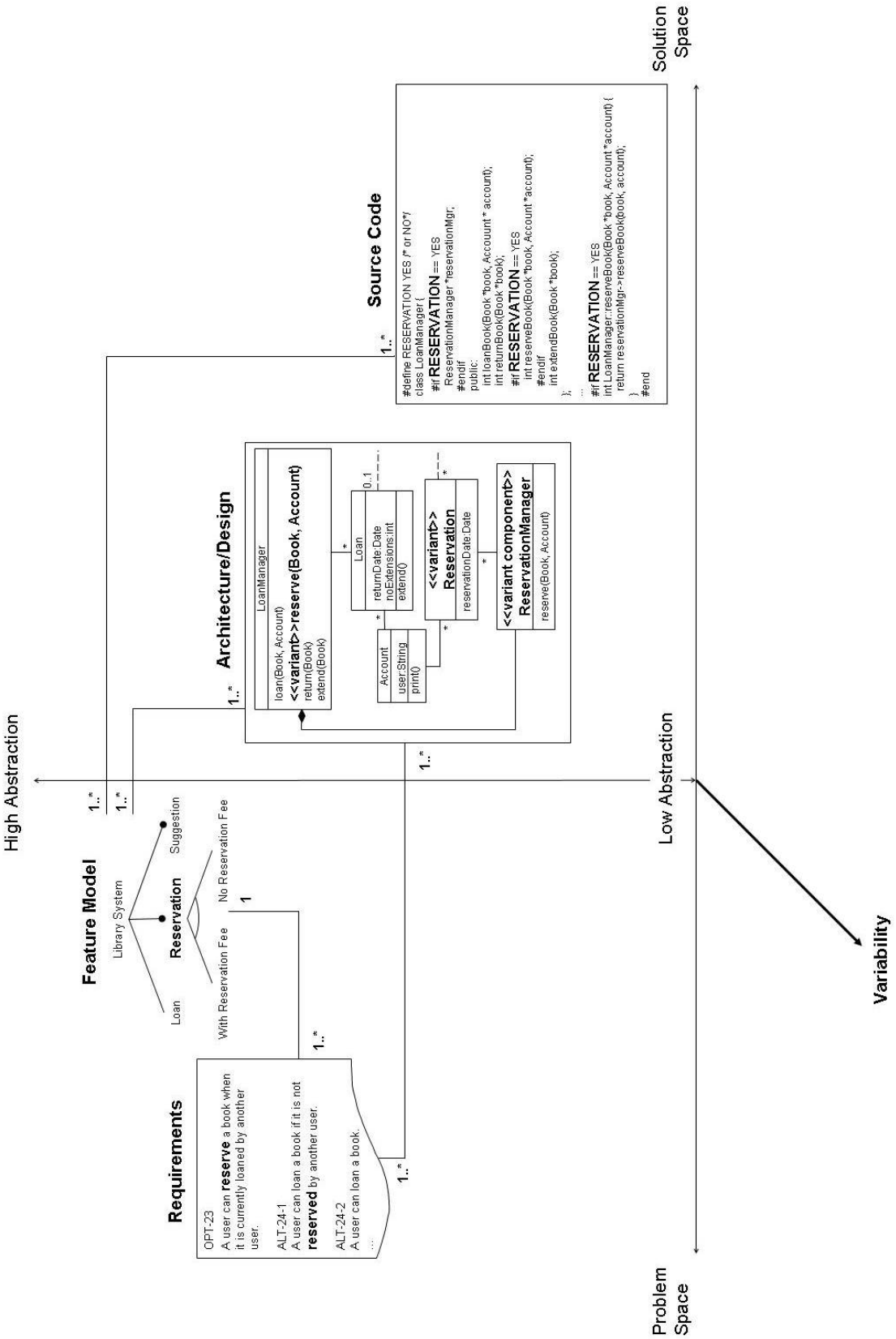
Solution Space
Source Code
#define RESERVATION YES /* or NO*/
class LoanManager {
#if RESERVATION == YES
ReservationManager *reservationMgr;
#endif
public:
int loanBook(Book *book, Accouunt * account);
int returnBook(Book *book);
#if RESERVATION == YES
int reserveBook(Book *book, Account *account);
#endif
int extendBook(Book *book);
};
...
#if RESERVATION == YES
int LoanManager::reserveBook(Book *book, Account *account) {
return reservationMgr->reserveBook(book, account);
}
#end
Architecture/Design
LoanManager
loan(Book, Account)
<<variant>>reserve(Book, Account)
return(Book)
extend(Book)
Loan
returnDate:Date
noExtensions:int
extend()
Account
user:String
print()
<<variant>>
Reservation
reservationDate:Date
<<variant component>>
ReservationManager
reserve(Book, Account)
1..*
0..1
High Abstraction
Feature Model
Library System
Loan
Reservation
Suggestion
With Reservation Fee
No Reservation Fee
1
Requirements
OPT-23
A user can reserve a book when it is currently loaned by another user.
ALT-24-1
A user can loan a book if it is not reserved by another user.
ALT-24-2
A user can loan a book.
...
Low Abstraction
Problem Space
Variability

Figure 6. *Traceability using Feature Modeling (adapted from [Riebisch 2004])*

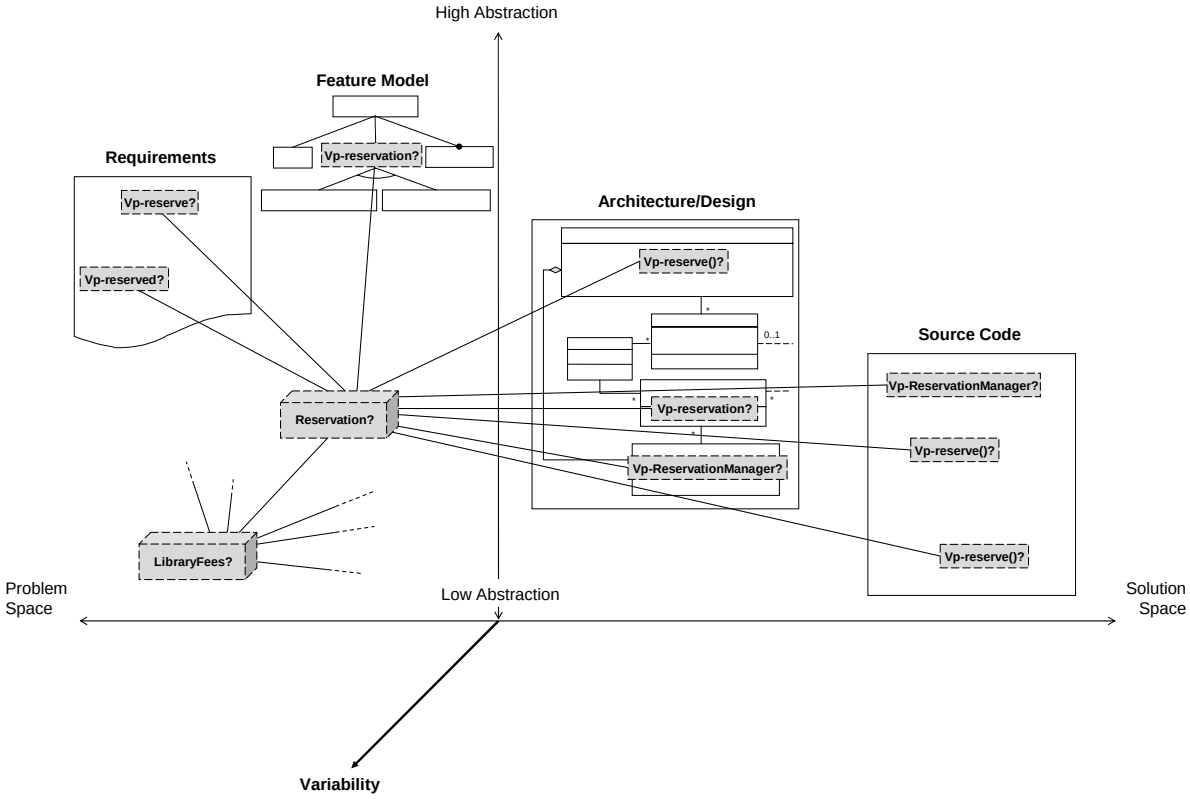

Figure 7. *Example of Conceptual Variability Model for Library System Product Line*

4.3 Comparison

The hierarchical tree-like structure of variable features in the feature model provides an excellent means for visualizing variability, and is therefore generally used for representing and communicating variation to the users of a product line. However, numerous differing and inconsistent feature modeling approaches exist, and a consistent method has yet to be defined and standardized.

When applying feature modeling techniques to illustrate variable features and their inter-dependencies of a number of large product-members, the feature model rapidly increases in size to such an extent that it becomes too complex to manage and impossible to keep an overview of. Scalability-issues are dealt with by modeling certain features, important to specific users, in separate diagrams and, sometimes, at different levels of abstraction.

Although several approaches use feature modeling for representing variability at different levels of abstraction, it is not clear how the features in different feature models and at different levels of abstraction are related to one another. As such, feature-modeling approaches that address the traceability of variation across development spaces have not been clearly defined. For this reason, the feature model does not provide sufficient support for traceability of variation in the context of a unified approach.

Even though the conceptual variability model does not explicitly provide a means for visualizing variability in a tree-like structure as the feature model does, it captures all the necessary information to generate user-specific views of variability in either a table format or a hierarchical tree-like structure.

The model consistently captures variability information at any level of abstraction for individual product line artifacts. Usually, depending on the development methodology used, the system as a whole can be seen as a single product line artifact [Atkinson et al. 2001]. Thus, whether dealing with a single component or with the complete system, the variability model is structured in such a way that it provides scalability and consistency.

Traceability is dealt with by capturing variability information explicitly and modeling the dependencies and relationships separate from other development artifacts. A great amount of information needs to be captured and it would seem infeasible to do so, however tool support [Kruse 2004] is a necessity for successful variability modeling. The sheer size of the variability information and the human interaction needed, poses a number of challenges that still need to be addressed when considering the capturing of this information. It would be worthwhile investigating whether

automating this process is at all feasible. The conceptual variability model provides a global approach for consistently managing variation at all levels of abstraction, from the problem space to the solution space.

Table 1. *Comparison between the Feature Model and Conceptual Variability Model*

	Feature Model	**Conceptual Variability Model**
Consistency	-	+
Scalability	-	+
Traceability	-	+
Visualization	+	+

Table 1 shows a comparison between the feature model and the conceptual variability model for traceability with regards to a unified approach.

5. CONCLUSIONS AND FUTURE WORK

Variability, and its management, is the distinguishing factor between conventional software engineering and software product line engineering. Traceability of variability in a software product line is *crucial* for its success; it is needed everywhere. For this reason, a sound and systematic approach is required.

This work proposes a *unified* approach for successful variability management. Focusing on the aspect of traceability of variation, which is a requirement of such an approach, a conceptual variability model is defined that captures variability information in the third dimension, and not in the two dimensional space where development artifacts reside.

The feature model, thus far used in literature as an approach for managing and tracing variability amongst others, is compared to the conceptual variability model. It has both its strengths and weaknesses when used for the management of variability. The analysis results have shown that even though the feature model provides an excellent means for visualizing variability at individual levels of abstraction, it does not improve the traceability between generic artifacts across development spaces. The feature model is only yet another development artifact, at a different level of abstraction in the two dimensional space. Furthermore, additional traceability between its model elements and other model elements needs to be established. Therefore, it is insufficient to use it as a means for tracing variability from the problem to the solution space. Moreover, it does not satisfy all the requirements of a unified approach.

The conceptual variability model, although not having specific visualization mechanisms, provides traceability between the problem and solution space, as well as satisfies the requirements of a unified approach. It was also established that the feature model, with its strong visualization capabilities for variation, compliments the conceptual variability model by providing appropriate views of the variability information being captured.

In future work, we look towards combining the strengths of the two approaches and extend the conceptual variability model with user-defined views of product line variability, such as with the feature model. We will extend an existing tool that can capture variability information represented in the third dimension. The tool will allow specific variability information to be selected and extracted, so that a hierarchically structured representation can be generated in the two dimensional space, whilst still being managed in the three dimensional space.

The variability model presented in this paper needs to be thoroughly evaluated in order to determine its value in large industry-related product lines. We will now start to apply and assess our model so that it can be seamlessly adopted into a software product line development lifecycle.

6. ACKNOWLEDGMENTS

This work was partially funded by the South African Research Foundation Grant No.2196 and the University of Pretoria Postgraduate Study Abroad Bursary Programme.

REFERENCES

ATKINSON, C., BAYER, J., BUNSE, C., KAMSTIES, E., LAITENBERGER, O., LAQUA, R., MUTHIG, D., PAECH, B., WÜST, J. AND ZETTEL, J. 2001. *Component-based Product Line Engineering with UML. (Component Software Series).* Addison-Wesley, London.

BAYER, J., MUTHIG, D. AND GOEPFERT, B. 2001. The Library Systems Product Line: A KobrA Case Study. Technical Report IESE-Report No. 024.01/E, Fraunhofer Institute for Experimental Software Engineering (IESE).

BERG, K. AND MUTHIG, D. 2005. A Critical Analysis of Using Feature Models for Variability Management. Submitted to SPLC-Europe 2005.

BEUCHE, D., PAPAJEWSKI, H. AND SCHRÖDER-PREIKSCHAT, W. 2004. Variability management with feature models. *Science of Computer Programming*. Vol. 53, No. 3, 333-352.

BOSCH, J. 2004. Preface. In *Proceedings of the 2nd Groningen Workshop on Software Variability Management: Software Product Families and Populations*. SVM2004, Groningen, The Netherlands, December 2004.

CZARNECKI, K. AND EISENECKER, U.W. 2000. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley.

JACOBSON, I., GRISS, M., AND JONSSON, P. 1997. *Software Reuse – Architecture, Process, and Organization for Business Success*. Addison-Wesley.

KANG, K., COHEN, S., HESS, J., NOVAK, W. AND PETERSON, A. 1990. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh.

KANG, K.C., KIM, S., LEE, J., KIM, K., SHIN, E. AND HUH, M. 1998. FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. *Annals of Software Engineering 5*, 143-168.

KRUSE, T. 2004. Managing Decision Model Constraints in Product Line Engineering. Diploma Thesis, Fraunhofer Institute Experimental Software Engineering.

MUTHIG, D. 2002. *A Lightweight Approach Facilitating an Evolutionary Transition Towards Software Product Lines*. PhD Thesis, Fraunhofer IRB Verlag, Stuttgart.

MUTHIG, D. AND ATKINSON, C. 2002. Model-Driven Product Line Architectures. In *Proceedings of the 2nd International Software Product Line Conference*, SPLC 2, San Diego, CA, USA, August 2002, G.J. CHASTEK, Ed. Lecture Notes in Computer Science, Vol. 2379, Springer-Verlag Berlin Heidelberg, 110-129.

MYLLYMÄKI, T. 2002. Variability Management in Software Product Lines. Technical Report 30, Institute of Software Systems, Tampere University of Technology.

RIEBISCH, M. 2003. Towards a More Precise Definition of Feature Models. In *Proceedings of Modelling Variability for Object-Oriented Product Lines ECOOP Workshop*, Darmstadt, Germany, July 2003, M. RIEBISCH, J. O. COPLIEN AND D, STREITFERDT, Eds. BooksOnDemand Publ. Co., Norderstedt, 64-76.

RIEBISCH, M. 2004. Supporting Evolutionary Development by Feature Models and Traceability Links. In *Proceedings of the 11th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, ECBS 2004, Brno, Czech Republic, May 2004, IEEE Computer Society Press, 370-377.

SAMETINGER, J. AND RIEBISCH, M. 2002. Evolution Support by Homogenously Documenting Patterns, Aspects and Traces. In *Proceedings of the 6th European Conference on Software Maintenance and Reengineering*, Budapest, Hungary, March 2002, IEEE Computer Society Press, 134-140.

SOCHOS, P., PHILIPPOW, I. AND RIEBISCH, M. 2004. Feature-Oriented Development of Software Product Lines: Mapping Feature Models to the Architecture. In *Proceedings of the 5th Annual International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World*, NODe 2004, Erfurt, Germany, September 2004, Lecture Notes in Computer Science, Vol. 3263. Springer-Verlag Berlin Heidelberg, 138-152.

TARR, P., OSSHER, H., HARRISON, W. AND SUTTON JR., S. M. 1999. N Degrees of Separation: Multi-Dimensional Separation of Concerns. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE 1999, Los Angeles, USA, May 1999, IEEE Computer Society Press, 107-199.

VAN GURP, J., BOSCH, J. AND SVAHNBERG, M. 2000. Managing Variability in Software Product Lines. In *Proceedings of IEEE/IFIP Conference on Software Architecture*, WICSA 2001, Amsterdam, The Netherlands, August 2001, IEEE Computer Society 2001, 45-54.

VON KNETHEN, A. 2001. *Change-Oriented Requirements Traceability. Support for Evolution of Embedded Systems*. PhD Thesis, Fraunhofer IRB Verlag, Stuttgart.

WEILER, T. 2003. Modelling Architectural Variability for Software Product Lines. In *Proceedings of Software Variability Management ICSE2003 Workshop*, SVM 2003, Groningen, IWI 2003-7-01, The Netherlands, 5-12.

WEISS, D. M. AND LAI, C. T. R. 1999. *Software Product-Line Engineering. A Family-Based Software Development Process*. Addison-Wesley.