Towards a Conceptual Model for Unifying Variability in Space and Time

Sofia Ananieva
FZI Research Center for Information Technology
Berlin, Germany
ananieva@fzi.de

Timo Kehrer
Humboldt University of Berlin
Berlin, Germany
timo.kehrer@informatik.hu-berlin.de

Heiko Klare
Karlsruhe Institute of Technology
Karlsruhe, Germany
heiko.klare@kit.edu

Anne Koziolék
Karlsruhe Institute of Technology
Karlsruhe, Germany
koziolék@kit.edu

Henrik Lönn
Volvo Group Trucks Technology
Gothenburg, Sweden
Henrik.Lonn@volvo.com

S. Ramesh
General Motors Global R & D
Warren, Michigan, USA
ramesh.s@gm.com

Andreas Burger
ABB Corporate Research Center
Germany
Ladenburg, Germany
andreas.burger@de.abb.com

Gabriele Taentzer
University of Marburg
Marburg, Germany
taentzer@mathematik.uni-marburg.de

Bernhard Westfechtel
University of Bayreuth
Bayreuth, Germany
bernhard.westfechtel@uni-bayreuth.de

ABSTRACT
Effectively managing variability in space and time is among the main challenges of developing and maintaining large-scale yet long-living software-intensive systems. Over the last decades, two large research fields, Software Configuration Management (SCM) and Software Product Line Engineering (SPLE), have focused on version management and the systematic handling of variability, respectively. However, neither research community has been successful in producing unified management techniques that are effective in practice, and both communities have developed largely independently of each other. As a step towards overcoming this unfortunate situation, in this paper, we report on ongoing work on conceiving a conceptual yet integrated model of SCM and SPLE concepts, originating from a recent Dagstuhl seminar on the unification of version and variant management. Our goal is to provide discussion grounds for a wider exploration of a unified methodology supporting software evolution in both space and time.

CCS CONCEPTS
• Software and its engineering → Software configuration management and version control systems; Software product lines; Software version control: Abstraction, modeling and modularity.

KEYWORDS
revision management, product lines, variability, version control

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:

1 INTRODUCTION
Complex software-intensive systems often need to exist in many variants in order to accommodate different requirements. At the same time, each of these variants is subject to continuous change and heavily evolves during all stages of software development and maintenance. Yet, software versions—resulting from evolution in time—and variants—resulting from evolution in space—are managed radically differently.

Version management, i.e., managing evolution in time, typically relies on a version control system, which provides basic storage services and supports intuitive workflows through a set of additional operations. The versioning of software artifacts has been extensively studied by the Software Configuration Management (SCM) research community since the advent of the first version control systems such as SCCS [22] and RCS [29] in the 1970s and 1980s. Research in this field has focused on versioning models, which define the artifacts to be versioned as well as the way in which these artifacts are organized, identified and composed to configurations [5]. Nowadays version control systems such as Subversion [20] or Git [15] are file-based, organizing versions of files in a directed acyclic version graph. Variants of a software artifact or an entire software system are represented by parallel development branches, where each of these branches has its own chronological evolution. While this strategy is simple, it does not scale in the case of multidimensional variability [5], and alternative version space organizations which tackle that problem never made it into mainstream [8]. Another issue is that the granularity provided, files,
is not always adequate when fine-grained development artifacts such as model elements or code sections are managed.

Next to traditional version management, the need for software mass-customization has been recognized within research on program families in the 1970s [17]. The field later evolved into software product line engineering (SPLE) [4, 7], which can nowadays be seen as the most successful approach to handling multidimensional variability in space, scaling up to several thousands of variants (a.k.a. products) of a software-intensive system. Instead of managing products as clones in parallel branches, SPL advocates to create a product-line platform that integrates all the product features and contains explicit variation points realized using variability mechanisms such as conditional compilation or element exclusion. However, evolving product-line platforms over time is substantially more complex than evolving single variants [13].

In summary, SCM and SPL are two widely established yet actively researched software engineering disciplines offering a variety of concepts to deal with software variability in time and space. However, neither research community has been successful in producing unified management techniques that are effective in practice, and both communities have developed largely independently of each other.

As a step towards overcoming this unfortunate situation, a recent Dagstuhl seminar on the unification of managing software evolution in time and space brought together leading practitioners and researchers from both disciplines to discuss each other’s challenges, solutions, and experiences. In this paper, we report on one of the results of the seminar, namely ongoing work on conceiving a conceptual yet integrated model of SCM and SPL concepts. Clearly, both disciplines share a set of common concepts, notably the idea of composing a system from fragments which serve as units of versioning and as re-usable assets, respectively. These common concepts of system descriptions serve as a starting point for our conceptual model, before we will explore those concepts which we consider to be specific to one of the disciplines and give an idea of how those concepts could be combined for managing variability in time and space. Our goal is to provide discussion grounds for a wider exploration of a unified methodology supporting software evolution in both time and space. The value and possible usage scenarios of a conceptual model are twofold. It may be instantiated to characterize and classify existing approaches, to structure the state-of-the-art and to map and align both communities’ core concepts. It may also pinpoint open issues and serve as a vehicle for evaluating different integration strategies on a high-level of abstraction.

2 CONCEPTUAL MODEL

In this section, we explain and illustrate basic design decisions of a conceptual model for unifying underlying concepts of SCM and SPL. We assume that a software-intensive system is described as a set of different types of models. This includes all source code artifacts which are also considered as models of a particular type in the context of this paper.

In Figure 1, we present a basic conceptual model of variability in time and space. It is composed of three differently colored parts corresponding to (i) concepts for variability in time (blue), (ii) concepts for variability in space (green), and (iii) concepts common to both (red). Figure 2 represents an extension to the introduced model and depicts the proposed integration of variability in space and time.

2.1 Common Concepts of System Descriptions

The common concepts of system descriptions represent the most fundamental intersection between basic concepts of SCM and SPL. Our System Space of discourse is the set of all possible Fragments describing a software-intensive system, as depicted in Figure 1. Fragments are the essential concepts for defining a system description. A fragment can either be an Atom or an Aggregate. Depending on the concrete realization, such an atom can be on different levels of granularity, for example, a single character or a single file. An aggregate contains fragments on its own and may represent, for instance, the node of an abstract syntax tree. For the representation of fragments, we rely on a generalization of the composite design pattern by replacing the actual composite relationship with an aggregate one. Consequently, we do not enforce a hierarchy of containments but consider fragments to be composed to various combinations. The use of an aggregated representation allows us to form a complex structure of fragments serving as a description of a software-intensive system. The fragments contain enough information to allow the system to compose meaningful artifacts, for example a package structure that organizes a set of related classes.

2.2 Concepts for Variability in Time

The proposed model contains elements necessary for capturing the concepts for variability in time. In a general versioning approach, every element of the software system is under revision control and hence the concept of Revision is applied to each fragment in our model, as illustrated in Figure 1. Each revision references a particular fragment and explicitly represents the dynamic character of the fragment. A revision is intended to supersede its predecessor, e.g., due to a bug fix or refactoring. Thus, a sequence of revisions represents the chronological evolution of a fragment. To represent

---

1. https://www.dagstuhl.de/19191

---

Figure 1: A Basic Conceptual Model of Variability in Time (blue), Variability in Space (green) and Shared Concepts (red)
branching (which we consider a temporary divergence for concurrent development) along with merging, multiple (direct) successors and predecessors relate to a revision. This relation gives rise to a revision graph, which is a directed acyclic graph where each node represents a unique revision. The Versioned System is composed of revisions and represents the configurable space of a software system regarding temporal variability. The Revision Space is the set of all possible systems under revision control conceptually corresponding to the system space.

2.3 Concepts for Variability in Space

Next, we describe concepts for variability in space of the conceptual model. The entry point for this is the Product Line, depicted in Figure 1. The product line acts analogously to the Versioned System representing the configurable space of a system (or family of systems) regarding spatial variability. The Variant Space represents a set of all possible product lines. Product lines aim to systematically express the variability of its products in terms of an associated set of Variation Points. To allow reuse of variation points in multiple product lines, we consider the product line an aggregate for variation points. Each variation point has a set of configuration options where each option is realized by concrete fragments. A Product is considered fully specified in space if all existing variation points in the product line are bound to fragments, hence composing a complete product. A partial product, however, does not enforce the binding of every variation point. A ternary association represents the configuration of a product from a product line, which covers the selection of fragments that realize variation points. The association therefore describes the relationship between one product line and a selection of variation points and fragments. The product resulting from that configuration is denoted as an association class.

2.4 Combining Variability in Space and Time

So far, we have introduced generic concepts for variability in space and time along with shared concepts of system descriptions generally applied in SCM and SPLE. In the conceptual model depicted in

Figure 2: An Extended Conceptual Model (regarding Figure 1) for Combining Concepts of Variability in Space and Time

Figure 3: Exemplary Revision Model with two Revisions of a System and one Revision for each of the two Components

In this section, we discuss some applications of the presented conceptual model for variability in space and time. We start with a small exemplary scenario that demonstrates how the model could be instantiated to represent revisions of a system. Next, since the model should be adapted and refined to be applied in actual approaches, we present different realization options. Finally, we discuss how appropriateness and expressiveness of the model can evaluated by applying it to existing approaches.

3 APPLING THE MODEL

In this section, we discuss some applications of the presented conceptual model for variability in space and time. We start with a small exemplary scenario that demonstrates how the model could be instantiated to represent revisions of a system. Next, since the model should be adapted and refined to be applied in actual approaches, we present different realization options. Finally, we discuss how appropriateness and expressiveness of the model can evaluated by applying it to existing approaches.

3.1 An Exemplary Scenario

Let us assume a simple scenario, in which an architectural model of software components is developed. Figure 3 shows the first two revisions of the system. In a first revision, a developer creates a composite fragment system, which contains an atomic fragment component1 that represents an initial component. In a second revision, he or she adds a second atomic fragment component2 that represents another component of the system, resulting in a modified system fragment system’. In consequence, both atomic fragments exist in their initial revision, as atomic fragments may not be changed. The composite system fragment, in contrast, was updated, such that its second revision contains a revised representation of the initial revision referenced by the predecessor relation.

3.2 Realization Options

The presented conceptual model (Figure 1, Figure 2) is supposed to express the essential concepts of variability in space and time. It serves as a reference model and thus has to be adapted and refined when it shall be used to realize an approach for SPLE, SCM or a
combination of them. This means that the elements of the conceptual model can be mapped to different realization options. For example, the fragments referenced by revisions to describe the temporal development of a system may be realized as snapshots of the system (or parts of it), or they may be represented as deltas, which only describe differences between revisions. Such design decisions especially affect how a system can be derived from its variability and revision fragments, because delta-based approaches require to apply the complete history of deltas to derive a system state, whereas the system states are explicitly contained in a snapshot-based approach. Regarding variability in space, variable parts can be represented in terms of variation points, like in the orthogonal variability model (OVM) [21], or in terms of a hierarchic structure of features in a feature model [10].

As a general comment, some elements of the conceptual model may not be relevant when realizing an actual approach (such as for instance the System Space) but they however contribute to a complete conceptual system description.

3.3 Evaluation Plan
We can evaluate the expressiveness and appropriateness of our conceptual model by applying it to actual approaches that realize variability in space, in time or both. For example, transferring the revision concepts to the elements of a revision graph in Git and other revision control systems gives an indicator for the appropriateness of our revision concepts. This, for example, includes a mapping of fragments to delta descriptions and revisions to commits with a reference to the previous revision, commit message and a hash code for identification. For investigating the appropriateness of our revision concepts, we plan to transfer the description to Subversion, Git and EMFStore [12] as representatives of state-of-the-art approaches.

A common description of variability in space is achieved with feature models, which can, for example, be defined in FeatureIDE [11]. In the last years, several approaches that combine variability in space and time have been proposed, such as Ecoco [9] and SuperMod [24], whereas others build on a delta-based representation of revisions and variability in space, such as DeltaEcore [25], SiPL [18, 19] and VaVe [1], or describe the evolution of SPLs systematically using model transformation rules [27]. If this conceptual model can be applied to several approaches that reflect the state-of-the-art for managing variability in space and time, we can assume generality of the model with high evidence. For that reason, apart from pure SCM or SPL approaches, we will especially apply the conceptual model to the approaches that combine both.

4 RELATED WORK
There has been a considerable amount of work on concepts and terminology by both research disciplines of SCM and SPLE. For SCM [5, 16, 20], a prominent conceptual model to capture variability in time is represented by the version model describing diverse revision concepts such as the specification of objects to be versioned, revision identification or the supported graph topology, i.e., whether revisions are only structured in a linear sequence or if they form a directed acyclic graph that represents temporary development branches and merges of them. For SPL [3, 21], a common conceptual model to capture variability in space is represented by the variability model which defines the variability of a product line, i.e., by introducing the concept of variation points and defining types of variation for a particular variation point. Compared to the conceptual model presented in this paper, there is however not yet a fundamental conceptual approach for variability in space and time that aims to integrate established concepts of both research disciplines (e.g., revisions and variation points) that is (i) declarative in nature by means of describing systems with variability in space and time and (ii) independent of realization by abstracting from the specification or tooling of systems with variability in space and time.

Conradi and Westfechtel [5] extend the notion of version models to represent the interplay between variability in space and time but concentrate on identifying several degrees of freedom for realizing both dimensions. Apart from the representation of fragments, this, for example, concerns versioning granularities or delta-based realization options such as directed deltas or symmetric deltas. The Uniform Version Model (UVM), introduced by Westfechtel et al. [30], serves as a common model for basic SCM and SPLE concepts but is intertwined with realization aspects as it relies, for instance, on propositional logic and selective deltas (corresponding to version identifiers in SCM or presence conditions in SPL in order to control visibility of fragments). Schwägerl [23] extends and specifies the UVM, among other things removing the concept of fragments and considering each element a versioned item which corresponds to our definition of it. In conclusion, we argue that conceptual models exist for either SCM or SPL but not for both combining concepts for variability in space and time. If they do, however, they are intertwined with aspects of specification and thus are not declarative in the sense of the introduced conceptual model.

Variation Control Systems (VarCS) are actual approaches that integrate concepts of SCM and SPL at different levels of granularity. Linsbauer et al. [14] provide a classification of VarCS and compare selected systems such as SuperMod and Eco. In subsection 3.3, we refer to some of the actual solutions which potentially represent realizations of the conceptual model. As depicted in the evaluation plan, this requires future investigation.

Finally, following the same goal but having a focus different from ours, another paper which emerged from the same Dagstuhl seminar 19191 studies potential synergies and combinations of product-line analyses (i.e., analyses to cope with variability in space) and regression analyses (i.e., analyses to cope with variability in time) [28].

5 DISCUSSION & FUTURE WORK
In this paper, we have proposed a conceptual model that describes the essential concepts of modeling variability of a software system in space and time. We have also presented an extended model that unifies those concepts to represent revisions of variable system parts. To validate that our model is general and appropriate in the
sense that we are able to map its elements to actual approaches for describing such variability, we will apply the model to existing approaches, such as Ecc, SuperMod or DeltaEcore in future work.

Several design decisions in the conceptual model were subject to intensive discussion and may be validated when transferring the model to actual approaches. One central subject of discussion is whether branches in revision control systems are a concept of variability in time to support temporary divergence for concurrent development, or whether they represent a realization of variability in space, as they support the existence of products at the same point in time. For the time being, we chose to follow the former notion and allow branches in the revision concepts, but appropriateness of that decision has to be validated in future work. Another subject of discussion which requires future validation is whether or not to consider the product a subclass of the versioned item. According to Antkiewicz et al. [2], product derivation is either fully automated or followed by manual post-processing (corresponding to the so-called governance levels L5 and L6). In the case of fully automated product derivation (L6), a product represents a fully derived artifact for which revision control becomes superfluous since the product line is already put under revision control in the extended model. When manual post-processing takes place (L5), a product does not represent a fully derived artifact anymore for which revision control becomes reasonable again. Additionally, the semantics of several concepts is only defined through the mechanisms that operate on them. For example, the configuration of a product from a product line, variation points and fragments is expressed in our model, but constraints that define which variation points and fragments may be selected have to be ensured by a configuration mechanism. The same applies to the unifying concept of our extended model. To define what the relations between revisions of product lines, variation points and fragments are, a mechanism that defines how they can be combined has to be defined. Designing such a mechanism, based on the presented model, should be the next step towards a unifying concept for variability in space and time.

ACKNOWLEDGMENTS
We thank all participants of Dagstuhl-Seminar 19191 (Software Evolution in Time and Space: Unifying Version and Variability Management) for the discussions and especially Thorsten Berger and Lukas Linsbauer for their contributions in the breakout group.

REFERENCES