Exploring the Dimensions of Variability: a Requirements Engineering Perspective

Sotirios Liaskos[1] Lei Jiang[1] Alexei Lapouchnian[1] Yiqiao Wang[1]
Yijun Yu[2] Julio Cesar Sampaio do Prado Leite[3] John Mylopoulos[1]

[1]Dept. of Computer Science, University of Toronto, Canada,
{liaskos,leijiang,alexei,yw,jm}@cs.utoronto.ca
[2]Dept. of Computing, Open University, United Kingdom, y.yu@open.ac.uk
[3]Dept. of Computer Science, PUC-Rio, Brazil, julio@inf.puc-rio.br

Abstract

Goal models have been found to be effective for representing and analyzing variability at the early requirements level, by comprehensibly representing all alternative ways by which stakeholders may wish to achieve their goals. Our study of goal models as instruments for acquiring and analyzing variability has showed that, focusing on one particular variability dimension, which in our case was the intentional one, allows better understanding of the identified variability and offers more opportunities for analysis and evaluation of alternatives. In this paper we explore other dimensions of variability that have emerged in our study of several areas, including autonomic computing, business process design and database design, and discuss how variability can be modeled and analyzed in each of these dimensions. Then we show how we can manage the variability space that emerges by putting together such dimensions and discuss the role of fitness criteria for identifying alternatives of interest.

1. Introduction

Software variability is an increasingly important research topic in Software Engineering, and a key concept for the study of software product lines and software adaptability. Modeling of variability is known to be essential for analyzing reuse strategies, understanding customization options and communicating these options to stakeholders. Furthermore, variability modeling has recently been found to be a useful instrument for both variability acquisition, that is discovery of variation points in a problem, and variability analysis, that is evaluation of the applicability of each identified variant in a given context and situation. These functions of variability modeling require knowledge of the origin and the meaning of variability, in a way that also allows understanding of the implication of elementary choices to overall properties of the system and, consequently, enables the specification of *fitness criteria* for selecting desired alternatives.

The need to explore and analyze variability at a higher level of abstraction is driven by the great amount of features that are observed in modern software systems, and the difficulty in understanding how the technical details of individual choices affect stakeholder intentions about the system. Direct manipulation of variability under such circumstances may prove difficult, unless the modeling process is focused on the particular *dimensions* that define the space of possible designs for a software system. Each variability dimension comes with a set of concerns to be addressed, a set of relevant concepts, its own modeling practices as well as its own definition of variation types and their semantics. Thus, modeling variability in its particular dimension leads to richer representations that are amenable to reasoning and allow high-level leveraging of variation using dimension-specific terms.

In this paper, we summarize the work we have done so far on variability acquisition, representation and analysis at the level of stakeholder goals, and use it as an example of how focusing on a particular variability dimension supports both the purpose of systematic variability identification and the goal of selecting alternatives that best solve particular problems. Then, focusing on our current work, we discuss other dimensions of variability and for each of them we describe what particular concerns they pose, what modeling challenges they introduce and how the fitness criteria for selecting alternatives can be derived.

We organize our presentation as follows. Section 2 explores how the current research tackles the problem of vari-

ability modeling along some of the dimensions in which it occurs. We also discuss efforts that attempt a categorization of variability. In Section 3, we reflect on our earlier study of intentional variability and make some observations that may apply for the study of other variability dimensions. In Section 4, we explore such dimensions and focus on concerns that are particular to them. In Section 5 we discuss how the dimensions form a variability space, how we can potentially represent constraints that span across dimensions, and how we can use fitness criteria to isolate points of interest. We conclude in Section 6.

2. Understanding Variability

Variability identification and representation has extensively been investigated in the context of domain analysis ([44, 41]). In that context, *commonality and variability* analysis is aimed at identifying common and varying characteristics among systems that belong to the domain under investigation. The result of this process is commonly formulated as a *feature model* ([28, 13]). Features are user-visible characteristics of software products, while feature models represent admissible combinations of features. Feature models are generally understood as tools for modeling the configurability aspect of high-variability systems ([13]). This approach assumes that the feature model plays the exclusive role of gathering and representing all variability aspects of a system (or a family thereof) in one central variability view ([6]). This, however, implies that, since they must potentially accommodate a wide diversity of feature types (such as behavioral, structural or data), feature models necessarily have little to say about the meaning of the variability they represent.

Thus, researchers have been examining ways by which variability pertaining to a specific view of a problem can be implicitly accommodated or explicitly represented in the models corresponding to that view. To model *structural* variability of a problem, Gomaa [21] or America et al. [2], for example, suggest the use of existing modeling constructs to represent variability in class diagrams, by possibly allowing special use of existing constructs, such as stereotypes. In [23], on the other hand, explicit use of variation points within use-case diagrams is suggested, while [27], [26] or [53] propose parameterization of use cases or requirements documents; these can be seen as methods for representing *functional* variability ([23]).

Several options have also been introduced for modeling variability in *behaviors*. Conditions on statechart transitions, which are true or false depending on the variant that has been chosen have been proposed in [28] and [39]. In a similar spirit, a method for generating SCR representations in which rows in the transition tables can be optional is proposed in [17]. In [50], message sequence charts with variable parts are proposed, while [7] and [1] suggest the use of generic storyboards and generic activity diagrams, respectively. Further, in [18] the notion of modal transition systems is discussed, where transitions can be either required or optional ("maybe") depending on the requirements for the particular product instance.

Methods for capturing *intentional* variability have recently been proposed. Constructing AND/OR hierarchies of goals has been shown to be an effective way to model intentional variability. Each solution of the AND/OR goal-tree provides an alternative way by which stakeholders can achieve their goals, while soft-goals are used to model qualitative characteristics of alternatives. We elaborate on intentional variability in the next section.

The above proposals are mostly suitable for describing the variability of the problem. Similar work has been done for accommodating and representing variability in models that describe aspects of the solution. At an architectural and design level, in the context of modeling *object-oriented* designs, inheritance, aggregation and parameterization have been found to be useful constructs for accommodating variability (e.g. [2, 21] discussed above). In modeling *component-connector views* of architectures, notable is the Koala extension to Darwin architecture description language ([55]), which introduces the notion of the switch, whereby alternative bindings of interfaces can be represented. An additional technique for representing variability in *module views* of architectures is presented in [5]. Further, at an even lower level lies the problem of variability implementation. Numerous techniques have been proposed; [19] and [22] constitute comprehensive surveys on the matter. Perry also provides a set of options for implementing architectural variability ([42]). Observe that, at this level, the presence of alternative variability implementation and architecting techniques constitutes variability on its own.

Furthermore, variability is relevant to human computer interaction (HCI) design in the context of adaptable and adaptive interfaces. In [38], for example, the alternative configuration of function availability in the main screen (through visual elements such as menus and buttons) of a popular text editor versus the complexity of the resulting interface are studied, while in [37] the problem of customizing such applications is investigated. Explicit modeling of variability in user interfaces is usually not in the scope of these efforts. Nevertheless, in the adaptiveness literature, variability is implicitly modeled by modeling the decision making mechanism, e.g. Markovian processes in [25].

The study of variability in its particular context, such as function, behavior, stakeholder intention, architectural view or user interface offers the opportunity of identifying the particular considerations that apply to each such dimension which in turn may again help the construction of systematic frameworks for variability identification and

management. The literature is already rich of proposals on what important dimensions of variability should be. Thus, in [23], essential variability, that is variability from a user point of view, is distinguished from technical variability, which concerns implementation issues. Essential variability can pertain to functionality (including behavior), system environment, quality, data and others. FORM ([29]) proposes a more coarse grained categorization into four layers, namely capability (e.g. available services to the users), operating environment (i.e the underlying infrastructure), domain technology and implementation technique. In [52], variability in behavior and hardware configuration are given as examples of two distinct variability dimensions. In [1] the accommodation of variability in customer, application, functional, conceptual and realization views is discussed, while in [33] the identification of variability dimensions in requirements is seen as an issue of separation of concerns specific to the particular problem that is being studied.

The definition of variability dimensions facilitates the discovery of variation points and allows understanding of its meaning, origin and rationale. This, in turn, facilitates the process of selecting the appropriate variant. For this to be possible, apart from the identification of the dimensions per se, a modeling framework for both accommodating variability and allowing the selection of variants needs to be introduced for each dimension. This constitutes a significant finding of our earlier work on intentional variability identification and analysis through goal models, which we summarize in the next section.

3. Reflections on the Study of Intentional Variability

Goal modeling has been found to be an effective way for identifying requirements of software systems by focusing on understanding the intentions of the involved stakeholders ([14, 3, 57]). Central to goal modeling is the idea of constructing hierarchies of AND- and OR-decompositions of high-level stakeholder goals into subgoals and then, recursively, into low-level subgoals and tasks that lead to requirements of the system-to-be. When a goal is AND-decomposed into subgoals, all of them must be satisfied for the parent goal to be satisfied. When a goal is OR-decomposed, the satisfaction of one of the OR-subgoals suffices for the satisfaction of the parent goal.

Thanks to the existence of OR-decompositions, the resulting hierarchy already constitutes a representation of variability in stakeholder goals. In Figure 1, oval shaped elements are goals forming such a hierarchy. The model shows how the goal `Schedule a Meeting` is analyzed into subgoals. Normally the decomposition continues until tasks that can fulfill the goals in a known way can be defined; for the interest of space we focus on the higher levels of the tree. For the meeting to be scheduled the individual constraints must be collected and some time slot must be selected. Thus, we AND-decompose the root goal accordingly. Constraints can then be collected by a person (e.g. a secretary) or the system to be; we express this option through an OR-decomposition. The system has several options to collect the constraints. It can unobtrusively check the invitees' on-line calendars or it can send them a request. The invitees, on the other hand, can be given several choices as to whether or not they want to broadcast their constraints. And then, different rules can apply for the selection of the right time slot. In all, it can be observed that the root goal can be achieved in 10 different ways, as many as the solutions of the AND/OR tree. We call these *alternatives* for fulfilling the root goal.

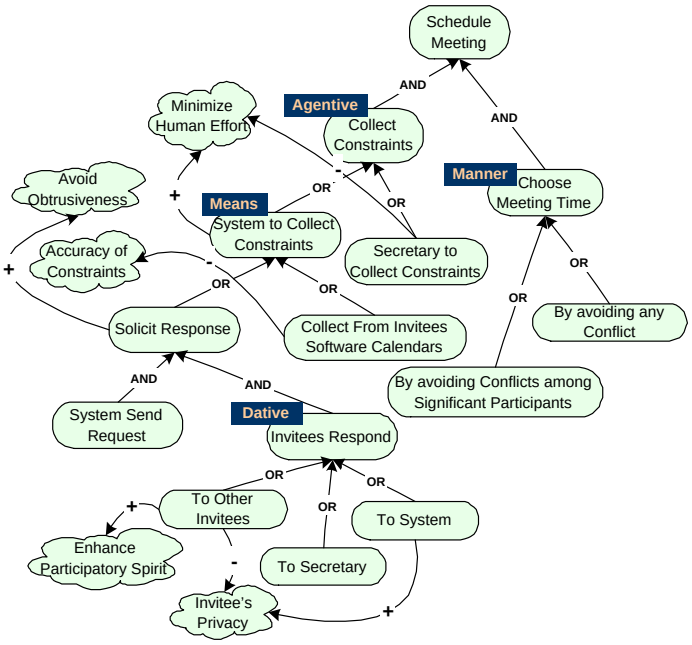

Figure 1. A goal model

Thus, expression of variability in goal models is possible by using OR-decompositions. But what is the meaning of OR-decomposition of goals? In *i** ([56]), a notation widely used for constructing goal models, OR-decompositions are simply understood as alternative *means* (subgoals) by which certain *ends* (parent goal) can be achieved. Nevertheless, in [48] and [47], Rolland et al. propose the use of a set of drivers that can guide the OR-decomposition of goals, and, thus, implicitly associate the meaning of OR-decompositions with the meaning of these drivers. The drivers are a universal set of parameters that are understood as potential parts of the definition of any goal. The acceptance of different values for different parameters introduces

variability in goal specification. This idea, originally proposed in the context of scenario identification, was later brought in a variability acquisition and modeling context ([36]). In that work, we introduced a method for identifying such decomposition drivers, which we called *variability concerns*, given textual information specific to the domain and the goal. Thus, each goal can be associated with a set of variability concerns, each of which must be addressed through an OR-decomposition in the AND/OR tree that emerges from the analysis of the goal. Returning to our example of Figure 1, each OR-decomposition is annotated with the variability concern that it addresses. For instance, alternative agents that can collect the constraints address the *agentive* variability concern, while different algorithms for picking a suitable slot address the concern *manner*.

By referring to the variability concerns that are relevant to the goal at the time we decompose it, we allow the discovery of variability that might otherwise remain hidden. Thus, the *dative* concern, which refers to the agent that is affected by the achievement of a goal, is used to suggest that different response styles to an invitation (to everybody or only to the solicitor) may need to be accounted.

Therefore, by exclusively focusing on acquiring and modeling variability on one dimension, the intentional one, we were forced to construct a specific semantic framework in which the particular dimension of variability can be expressed. This way, apart from facilitating the discovery and understanding of variability, we also allow the definition of fitness criteria to be used for evaluating alternatives implied by goal models. Our investigation of the intentional dimension of variability has already led us to the consideration of three categories of criteria that can be used for selecting alternatives in goal models: *user skills* and *stakeholder preferences* ([24]) as well as *contextual characteristics* ([36]).

User skills are associated with leaf level goals in goal models, which, as mentioned above, describe tasks to be performed by humans alone or through interaction with the system. Thus, for each such task we construct a list of skills that a user who needs to perform the task must have. Conversely, if we are given the skills of a particular individual we are able to filter out courses of tasks that she cannot perform. In our scheduling example, assume that tasks at the leaf level mention that an invitee responds to the constraint collection by filling up a web form. This requires some sensory (e.g. vision) and motor or speech skills to perform the input, some basic cognitive skills as well as language and computer skills. Individuals that lack any of the required skills (e.g. elderly, people with cognitive or other impairments, illiterate) must be provided a different alternative whose tasks they can actually perform. This may even imply that a completely different strategy for collecting constraints must be selected at the higher level.

A similar approach is followed for contextual characteristics: tasks can only be performed under specific circumstances. In our example of Figure 1, constraints can be selected from individual calendars only if the invitee has made them public and there is a means to access them through a network. From a modeling point of view, contextual characteristics are a generalization of skills, as they are also used as conditions for the performance of leaf level tasks.

User preferences on the other hand are specifications of priorities over *soft-goals* that alternatives must satisfy. As opposed to hard-goals we discussed so far, soft-goals are goals for which there is no clear criterion that can be used for deciding whether they are satisfied or not. Thus, soft-goals are *satisficed* (versus satisfied) to a degree and based on relevant evidence. In Figure 1, soft-goals are represented as cloud-shaped elements. Contribution links, annotated with "+" and "-" symbols in the figure, allow us to represent the fact that satisfaction of hard-goals affects positively or negatively the satisfacing of soft-goals. Thus, in Figure 1, the choice to broadcast one's constraints may hurt one's privacy, but it helps building a participatory spirit. Conversely, if we state that building a participatory spirit is more important than privacy of individual invitees, we implicitly suggest that the particular response style should be used. Thus, by specifying the relative importance of soft-goals we implicitly bind low level options. Note that, in practice, goal models can be more complicated networks of soft-goals, requiring advanced reasoning techniques for understanding how satisfaction and denial is propagated from goal to goal. Such reasoning techniques are discussed in [20] and [49].

Notice that these types of fitness criteria are semantically correlated with concepts of the goal models, and this is why they naturally emerged throughout our investigation. Tasks, for example, that lay at the leaf level of goal models, represent courses of actions that are performed by actors in a particular time and place and have an estimated impact on high-level objectives of stakeholders. Thus, by referring to the meaning of tasks, we already refer to a set of concerns (actor characteristics, times, locations, and actor preferences respectively) that may influence variability identification and analysis. If the building blocks of our models were more abstract concepts, such as features, we perhaps wouldn't have been able to detect these additional concerns, cleanly express their relationship with our main concepts, and use them as fitness conditions.

Interestingly, fitness criteria themselves can vary and, thus, they are also subjects for variability analysis. For example, evaluating each of the supported means of transportation a stakeholder can use to `Travel from Home to Work` may depend on the weather conditions. But weather conditions themselves can vary preventing the selection of certain alternatives for fulfilling our goal, in different ways. For example, rain or cold may trivially pre-

vent the use of a bicycle, but, more rare phenomena, such as sleet, may even prevent the use of any street vehicle. In other words, by systematically analyzing all potential states the weather can take, forces us to consider more alternatives for fulfilling the goal `Travel from Home to Work`.

In order to systematize this, in the goal oriented context we found it helpful to separate *core* variability from *background* variability. While the former is the variability we are trying to model, the latter refers to variability of factors that influence the fitness evaluation of core alternatives and, in many cases, necessitate the introduction of more alternatives. We expect that a similar distinction can be made in other variability dimensions as well. For example in the behavioral dimension, alternative behaviors constitute core variability, while alternative conditions on admissibility of states, transitions or courses thereof may constitute background variability. Similarly, the choice of a particular variability implementation policy ([22]) among a set of options (core) may depend on factors pertaining, for instance, to the device where the software is to be deployed, the expertise of the developers or characteristics of the project (background). The variability analyst develops the core variability model having in mind those background circumstances and how their change may indicate a change to a different variability implementation strategy.

4. Dimensions of Variability

In this Section, we present some observations from our investigation of analysis and design methods in four different areas of software development: business process design, autonomic computing, database design as well as architectural design for pervasive systems. Based on our early evidence, for each of these areas, we explore potential variability dimensions and discuss criteria that can be used for evaluating alternatives.

4.1. Business Processes Design

The design of business processes (BPs) and the supporting IT infrastructure is an important software development area. BPs are characterized by the high level of human involvement in their execution and thus, in addition to the variability normally present in the usual software development process, there are some new variability dimensions to be considered during business process design. Also, prominent among the fitness criteria for selecting appropriate alternatives in these variability dimensions (besides the usual ones such as cost and resource utilization) are social aspects such as rewarding and challenging work environment.

There are many definitions of what a business process is, but in general a BP is seen as a sequence of activities that achieves some business purpose. Since the notion of purpose is present in BPs, they can be modeled using goal-oriented notations (e.g., [30]). Thus, a lot of the discussion on intentional variability included in this paper can be applied to these processes. However, in practice BPs are usually modeled as workflows, which represent activities arranged in sequence or in parallel, each taking some input and producing some output. Common notations for workflow specification are BPMN ([10]) and Petri Nets ([54]). When designing BPs, analysts need to determine which parts of the BP can be represented as well-defined workflows and which – usually highly unstructured creative subprocesses – need to be left underspecified (perhaps, aside from the definition of inputs and outputs). While modeling BPs through workflows, we are presented with several types of variability that need to be represented and analyzed. For the unspecified parts, the variability will be dealt with at runtime by participating employees.

First, we have to deal with the temporal ordering of activities in a workflow. Obviously, there are constraints on that ordering: some activities require inputs that are produced by other activities and this limits their ordering variations. However, this still leaves a lot of choices of how to sequence/parallelize activities. Important criteria to consider here is the performance of the BP (parallelism generally means increased performance), the availability of resources, as well as whether inputs/outputs of activities can be duplicated (for parallel execution).

Another important variability consideration in BP design is the assignment of workflow activities to human/computer resources. Part of the task is to determine which portions of a process can be automated and which should be left for human execution. This can be viewed as defining the boundary for the IT support system. While some tedious and mechanical activities can be easily automated, BPs usually require certain amount of human creativity and problem solving. The criteria here are the availability of human/computing resources, performance, scalability, utilization as well as the desire to keep employees challenged and interested in their jobs. The other part of the task is to determine which particular employees, roles or teams as well as systems, components, or services are going to be responsible for executing workflow activities. Examples of important considerations here are resource cost, and capabilities of the involved employees.

The measurement of BP performance requires the setup of an efficient, non-disruptive, and cost-effective monitoring framework [46]. Key Performance Indicators (KPIs) are measures commonly used to track the critical success factors in BPs from a business perspective. The choices of the appropriate KPIs for processes and their refinement into sets of monitorable and measurable parameters represent another variability dimension in BP design.

There are other BP design activities that give rise to ad-

ditional variability concerns. For example, dealing with failures in BPs involves the selection of error notification (whom to notify, through what means), diagnosis (e.g., automated vs. human intervention) and compensation mechanisms (such as wait, re-execute the activity, re-assign it to another actor, or cancel).

4.2. Autonomic Systems Design

Autonomic Computing (AC) is an example of a software engineering field where we see a great need to model and analyze various aspects of variability during the development process. AC is a rapidly growing research area that aims at reducing software maintenance cost and management complexity. AC promises to move most of this complexity from humans to the software itself by endowing a software system with capabilities to self-configure (to adapt to dynamically changing environment and requirements), self-optimize (to improve its performance and/or other characteristics), self-heal (to monitor, diagnose, and recover from its failures), and self-protect (monitor, detect, and protect itself from attacks and other malicious behaviour) [31]. Collectively, these are referred to *self-** capabilities.

There are a number of possibilities for designing AC software systems. For example, one is to use software agents capable of planning and social interaction to create self-managing systems ([31]). The problems in this approach are the computational complexity of multi-agent systems and the inherent difficulty of predicting and analyzing emergent behavior in them.

Since, in a nutshell, an AC system is a flexible software system that changes its behavior in a purposeful way while achieving its goal, in [34] we presented an alternative approach that aims at producing systems supporting a space of possible behaviors which are realized through the isomorphic space of possible system configurations. While designing AC systems, a number of variability dimensions must be represented and analyzed. Obviously, the intentional perspective must be represented. Thus, the approach in [34] uses high-variability requirements goal models to capture variability in the problem domain of autonomic systems. These models represent multiple ways the goals of AC systems can be attained as well as the characteristics of these alternatives in terms of the important quality criteria (softgoals). These models are then enriched with control information capturing temporal ordering constraints on goals to support variability-preserving mapping into design-level notations such as statecharts (that represent variability in the behavior of the system). AC systems are then designed to support all (or some) of the alternatives captured in their corresponding goal models and are then augmented with capabilities to switch from one alternative to another at runtime, thus exhibiting adaptive behavior.

In the AC literature (e.g., [31]), self-managing systems are viewed as networks of Autonomic Elements (AEs) that are capable of tuning/modifying their resources or processes and that are furnished with the feedback loop consisting of activities that *monitor* the system and its environment, *analyze* the monitored data (i.e. diagnose), plan a course of action if an intervention is needed, and execute the plan. All of these activities provide a wealth of alternatives that need to be systematically represented and analyzed.

For example, in case of the monitoring, the paramount problem is the selection of data that needs to be captured in order for an AC system to be able to perform the self-* activities. For example, for self-optimization, the system needs to evaluate the currently executing alternative in terms of how it meets quality constraints such as performance to determine if a switch to another option is warranted. For self-healing, data on successes/failures of its constituent components are needed. For self-protection, data on, for example, failed login attempts, must be captured. Similarly, there are several parameters that need to be decided on when designing a monitoring framework for an AC system. The characteristics of the individual monitors themselves need to be defined. Monitors can be on-line or off-line, intrusive or non-intrusive, adaptive or maladaptive. The specifics of the measurement process such as sampling rates and data storage constitute additional variability concerns. Then, the specific monitoring policy in relationship with the monitored system needs to be established. We are currently working towards constructing a richer categorization of such concerns. We also explore ways by which we can construct models that facilitate both understanding of the available options in a monitoring problem and making the appropriate decisions.

The aspect of diagnosis gives rise to another variability space with a wealth of options and concerns (e.g., how eager the system is to generate diagnosis vs. to wait and collect more data). Similarly, the system's adaptivity strategy can be aggressive (possibly resulting in frequent oscillation among behavior alternatives), conservative (possibly resulting in missed opportunities for optimization, but providing a more stable behavior) or can depend on the diagnosis (e.g., very aggressive in case of security concerns/failures and conservative in case of self-optimization). Variability manifests itself in many other aspects of AC systems design, such as the selection of components for achieving the leaf-level goals in the goal models. Component's characteristics such as cost or performance play a role in this.

4.3. Information Modeling and Database Design

Information modeling is concerned with the construction of computer-based symbol structures (i.e., information

bases) which model some part of the real world (i.e., the application domain) [40]. In the case of databases, *data modeling* focuses on the static aspect of the application domain and uses modeling constructs that denote particular individuals, generic concepts and associations that are part of the domain. Classic approaches [4, 45, 12] divide the overall database design into three phases: conceptual, logical and physical design. At conceptual level, the designer collects, analyzes, structures and formalizes relevant *domain concepts*, referring to a conceptual data modeling language (e.g. Entity-Relationship models). Furthermore, data *abstraction mechanisms* are essential components in conceptual data modeling and an important part of the relevant modeling languages. These abstraction mechanisms include the classic ones such as classification, generalization and aggregation as well as less known ones such as contextualization [51], materialization [43] and parameterization [16].

To understand how variability occurs in conceptual design we need to understand the process with which such designs are developed. Conceptual database design is, to all intents and purposes, an engineering process. It consists of a series of decision-making steps and is guided by well-defined *design strategies*. For example, in the classic conceptual database design methodology [4], the top-down strategy produces a conceptual schema by a series of successive refinements. In this approach, starting from an initial schema that describes all the data requirements by means of a few highly abstract concepts and then gradually refining the schema by transforming these concepts into ones with more complex structure, capable of describing the original concepts in more detail. For each design strategy, a set of *transformation primitives* is proposed to guide the transformation process. For example, one top-down primitive may transform an entity into two entities connected by a relationship while another may instead transform the entity into a generalization hierarchy. In addition, *conceptual rules* ([4]) may help designers make strategy-independent decisions, such as whether to model a concept as an entity, relationship or attribute. The application of alternative transformation primitives and conceptual rules lead to alternative designs, in a way similar to how alternative variability implementation techniques are chosen in [22].

The notion of *relevance* can be considered as the general criterion for selecting among alternatives. For example, the specialization of a general concept into more concrete subconcepts relies on the determination of the "discriminator" [9], which is the property of the concept that is "more relevant" to the problem at hand. Consider the concept *Material*. It can be specialized into sub-concepts *Book* and *Journal* on the basis of the *format*, or into *Paper*, *Audio* or *Video* on the basis of the *medium*. Thus, as with variability concerns in the intentional case, such discriminators both guide the identification of alternative specializations and allow us assess the relevance (fitness) of particular specializations through reference to such discriminating characteristics.

Furthermore, more concrete variability concerns and fitness criteria can be used for particular purposes. For example, *temporality* specifies whether the temporal aspect of an entity in the application domain is of interest in the context of a particular problem. In a hospital management system, for example, temporality questions whether the historical data of patient medical profile need to be kept in the database. On the other hand, *accuracy* is concerned with the degree by which details of the relevant concepts are included in the model (e.g. what configuration of attributes to use to model properties of a concept or how to separate units and values for measurement concepts). In both cases, an obvious fitness criterion is the accommodation of data queries that are already known to be of interest, or are conjectured to be of potential interest. As a last example, *access* refers to alternative ways by which external agents can read and manipulate the data entities. Here the degree of trust amongst various stakeholders may be an example of a criterion for selecting among alternative permission schemes.

4.4. General Architectural Design

Several frameworks for architectural modeling based on views have been proposed, including Kruchten's 4+1 view framework ([32]), as well as Clements et al. approach ([11]) based on viewtypes. Independent of the views one chooses to represent an architecture, each such view becomes a variability dimension that needs to be addressed. Further, each view is associated with one or more modeling notations. In Section 2 we already discussed several existing approaches in modeling variability in architectural designs. These include accommodating variability in use cases, class diagrams, component-connector views, statecharts and interaction diagrams. The question is whether existing efforts are sufficient for representing variability in every essential architectural view.

Our experience in analyzing problems that called for solutions of ubiquitous nature (e.g. advanced nurse notification systems for the health care domain), showed the extensive need of variability representation mechanisms for *deployment views*. Such views describe the assignment of software elements to environmental entities (i.e. computing hardware). Likewise, variability occurs within the nodes in terms of their capabilities (e.g. a PC vs. a PDA), which can often dynamically change (e.g. network connectivity of a mobile device). Interestingly, dynamic change of capabilities may depend on pure environmental factors. Thus, network connectivity or power supply may need to be modeled as properties of a location, rather than properties of devices.

From there, an initial allocation of software components

to devices requires modeling of both what software components assume about their environment and what capabilities nodes (e.g. devices) actually offer. In addition to the hard-constraints implied by the capability matching problem, soft-constraints related to quality attributes, such as usability, performance and cost, need to be set for further evaluating the admissible assignments of components to devices.

5. Managing the Design Space

In every dimension, variability is present in the form of *variation points*. A variation point can be explicit by appearing as a separate construct of the modeling language (as e.g. variation point elements in use case diagrams – [23]) or implicit, through the use of existing constructs (e.g. alternative specializations in class diagrams – [21]). In both cases, the variation point offers alternative instantiations of a high-variability model, by being bound appropriately. Putting all dimensions together, the resulting *design space* is a cross-product of the alternatives found in each dimension. Thus, since the design space encompasses *all* possible combinations of alternatives, it can be vast. This constitutes a problem when e.g. in a product derivation context ([15]) a point in this space is sought that best solves a problem.

A first step towards coping with large design spaces is the use of cross-dimensional *interdependency links* between elements and decisions involved in variation points. For example, in a study of alternative event notification designs for nurses in hospitals, a leaf level task `Send Audio Notification` in the intentional dimension, a component `audioNotifier` in a structural dimension, and a state `Audio Notification Being Sent` in a behavioral dimension can all be associated through such interdependency links. Then, when they appear as alternative selections in the context of a variation point (which in our case may concern alternative notification modes such as visual or haptic), binding of a variation point in one dimension automatically constraints the way others are bound.

This way, we can greatly reduce the size of the design space through a small number of decisions. In [58], we introduce a set of patterns that can guide the derivation of interdependency links between the intentional variability dimension at the problem level and other dimensions at the solution level, namely the behavioral, modeled through statecharts, and the component interconnection dimension, modeled through component-connector diagrams.

Furthermore, the fact that variability is modeled and understood across dimensions, offers the opportunity to leverage the design space by specifying preferred characteristics of the alternatives of interest, instead of directly handling variation points. This is exactly where the fitness criteria identified in each dimension play a major role. In our investigation of the intentional dimension, fitness criteria such as the ones discussed in Section 3 were used for restricting the design space implied by the AND/OR-hierarchy of the hard goals. In one of the applications ([35]) we showed that it is possible to configure an e-mail client by only referring to desired satisficing of soft-goals, which was the primary fitness criterion. For example, a stakeholder's desire to maximize `Privacy` while receiving e-mail, implies the selection of the subgoal `Use Encrypted Communication` (as opposed to not using an encrypted one) which, by following a pre-established interdependency link to configuration details of the software system, enables secure sockets layer or secure authentication. Thus, fitness criteria specified for one dimension, automatically pose constraints to other dimensions through the interdependency links. The use of fitness criteria in other dimensions can have the same effect. For example, a reference to discriminators allows the leverage of variability in static structures, which again may have consequences on behavioral views (e.g. remove states related to objects that are absent from the reduced static structure).

Note that the criteria specification approach to variability leverage is not distant from the notion of domain specific languages (DSLs - [8, 13]), since DSLs allow the use of domain specific terms to describe a particular solution. One of the differences, however, may be that fitness criteria specification, the way we envision them, have the form of *hard* and *soft-constraints* that *prune* a design space that is already defined, instead of languages for *generating* members of that space.

On the other hand, handling the design space that emerges from the combination of individual dimensions is greatly facilitated if the space is of *propositional* nature. If the variation points of each dimension can be modeled in AND/OR trees with lateral constraints amongst their nodes as well as between their nodes and the nodes of the respective trees of the other views, then the reasoning problem is one of satisfiability of the underlying propositional theory. However, in order to enjoy the benefits of dimension-specific criteria specification, an extra step must precede to convert such criteria into actual bindings of variation points expressed in a propositional form.

6. Conclusions

Putting variability modeling and analysis in the context of the particular dimensions in which variability appears, allows better understanding of its meaning and offers several analysis and evaluation opportunities. Throughout our work on several domains we have acquired an initial understanding of potential variability dimensions and the special concerns each of these introduces. However, our work is still at an early stage. In the future, we expect to have an

exact picture of what these dimensions are and a systematic framework for defining interdependencies among them.

We also intend to focus on modeling aspects of these ideas. We saw that the literature is already rich of proposals for accommodating or explicitly representing variability in a number of different modeling notations. One question is whether and how variation points appearing in models of each dimension can form dependency hierarchies that can be translated to propositional formulae. This, together with a framework for establishing interdependencies across dimensions (again based on propositions), would provide a convenient representation of the design space amenable to well known reasoning techniques. Furthermore, for the variability space to be accessible, the translation of fitness criteria into part of the propositional representation poses an additional challenge.

References

[1] P. America, E. Rommes, and J. H. Obbink. Multi-view variation modeling for scenario analysis. In *Fifth International Workshop on Product Family Engineering (PFE)*, pages 44–65, 2003.

[2] P. America and J. van Wijgerden. Requirements modeling for families of complex systems. In *IW-SAPF-3: Proceedings of the International Workshop on Software Architectures for Product Families*, pages 199–209, London, UK, 2000. Springer-Verlag.

[3] A. I. Anón and C. Potts. The use of goals to surface requirements for evolving systems. In *Proceedings of the 20th International Conference on Software Engineering*, 1998.

[4] P. Atzeni, S. Ceri, S. Paraboschi, and R. Torlone. *Database Systems - Concepts, Languages and Architectures*. McGraw-Hill Book Company, 1999.

[5] F. Bachmann and L. Bass. Managing variability in software architectures. In *Proceedings of the 2001 Symposium on Software Reusability (SSR '01)*, pages 126–132. ACM Press, 2001.

[6] F. Bachmann, M. Goedicke, J. Leite, R. Nord, K. Pohl, B. Ramesch, and A. Vilbig. A Meta-model for Representing Variability in Product Family Development. In *Proceedings of the 5th International Workshop on Software Product-Family Engineering (PFE5)*, Siena, Italy, 2003.

[7] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud. PuLSE: A methodology to develop software product lines. In *Proceedings of the Symposium on Software Reuse (SSR'99)*, 1999.

[8] J. Bentley. Programming pearls: little languages. *Communications of the ACM*, 29(8):711–721, 1986.

[9] A. Borgida and R. J. Brachman. Conceptual modeling with description logics. In *The description logic handbook: theory, implementation, and applications*, pages 349–372. Cambridge University Press, New York, NY, USA, 2003.

[10] Business process modeling notation, version 1.0. www.bpmi.org.

[11] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford. *Documenting Software Architectures: Views and Beyond*. Addison Wesley Professional, 2003.

[12] T. M. Connolly and C. E. Begg. *Database Solutions: A step by step guide to building databases*. Addison Wesley, 2nd edition, 2003.

[13] K. Czarnecki and U. W. Eisenecker. *Generative Programming - Methods, Tools, and Applications*. Addison-Wesley, June 2000.

[14] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):3–50, 1993.

[15] S. Deelstra, M. Sinnema, and J. Bosch. Product derivation in software product families: a case study. *Journal of Systems and Software*, 74(2):173–194, 2005.

[16] E. Dubois, P. D. Bois, and A. Rifaut. Elaborating, structuring and expressing formal requirements of composite systems. In *International Conference on Advanced Information Systems Engineering (CAiSE'92)*, pages 327–347, 1992.

[17] S. R. Faulk. Product-line requirements specification (PRS): An approach and case study. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering (RE'01)*, pages 48–55, 2001.

[18] D. Fischbein, S. Uchitel, and V. Braberman. A foundation for behavioural conformance in software product line architectures. In *Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis (ROSATEA'06)*, pages 39–48, New York, NY, USA, 2006.

[19] C. Gacek and M. Anastasopoulos. Implementing product line variabilities. *SIGSOFT Softw. Eng. Notes*, 26(3):109–117, 2001.

[20] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Reasoning with goal models. In *Proceedings of the 21st International Conference on Conceptual Modeling (ER'02)*, pages 167–181, London, UK, 2002.

[21] H. Gomaa. Object oriented analysis and modeling for families of systems with UML. In *ICSR-6: Proceedings of the 6th International Conference on Software Reuse*, pages 89–99, London, UK, 2000. Springer-Verlag.

[22] J. V. Gurp, J. Bosch, and M. Svahnberg. On the notion of variability in software product lines. In *WICSA '01: Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01)*, page 45, Washington, DC, USA, 2001. IEEE Computer Society.

[23] G. Halmans and K. Pohl. Communicating the variability of a software-product family to customers. *Software and System Modeling*, 2(1):15–36, 2003.

[24] B. Hui, S. Liaskos, and J. Mylopoulos. Requirements analysis for customizable software: A goals-skills-preferences framework. In *Proceedings of the 11th IEEE International Requirements Engineering Conference (RE'03)*, 2003.

[25] A. Jameson, B. Großmann-Hutter, L. March, R. Rummer, T. Bohnenberger, and F. Wittig. When actions have consequences: Empirically based decision making for intelligent user interfaces. *Knowledge-Based Systems*, 14:75–92, 2001.

[26] S. Jarzabek, W. C. Ong, and H. Zhang. Handling variant requirements in domain modeling. *Journal of Systems and Software*, 68(3):171–182, 2003.

[27] I. John and D. Muthig. Tailoring use cases for product line modeling. In *International Workshop on Requirements Engineering for Product Lines (REPL'02)*, 2002.

[28] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, November 1990.

[29] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. FORM: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5:143–168, 1998.

[30] V. Kavakli and P. Loucopoulos. Goal-driven business process analysis application in electricity deregulation. *Information Systems*, 24(3):187–207, 1999.

[31] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[32] P. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6):42–50, 1995.

[33] C. Kuloor and A. Eberlein. Aspect-oriented requirements engineering for software product lines. In *10th IEEE International Conference on Engineering of Computer-Based Systems (ECBS)*, pages 98–107, 2003.

[34] A. Lapouchnian, Y. Yu, S. Liaskos, and J. Mylopoulos. Requirements-driven design of autonomic application software. In *Proc. 16th Annual International Conference on Computer Science and Software Engineering (CASCON 2006)*, October 2006.

[35] S. Liaskos, A. Lapouchnian, Y. Wang, Y. Yu, and S. Easterbrook. Configuring common personal software: a requirements-driven approach. In *13th IEEE International Conference on Requirements Engineering*, 2005.

[36] S. Liaskos, A. Lapouchnian, Y. Yu, E. Yu, and J. Mylopoulos. On goal-based variability acquisition and analysis. In *Proceedings of the 14th IEEE International Conference on Requirements Engineering (RE'06)*, Minneapolis, Minnesota, September 2006. IEEE Computer Society.

[37] W. E. Mackay. Triggers and barriers to customizing software. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 153–160, New York, NY, USA, 1991. ACM Press.

[38] J. McGrenere, R. M. Baecker, and K. S. Booth. An evaluation of a multiple interface design solution for bloated software. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 164–170, New York, NY, USA, 2002. ACM Press.

[39] H. Muccini and A. Bucchiarone. Formal behavioral specification of a product line architecture. Technical Report Technical Report TRCS 014/2004, Univ. of L'Aquila, 2004.

[40] J. Mylopoulos. Information modeling in the time of the revolution. *Inf. Syst.*, 23(3-4):127–155, 1998.

[41] J. M. Neighbors. Draco: a method for engineering reusable software systems. *Software reusability: vol. 1, concepts and models*, pages 295–319, 1989.

[42] D. E. Perry. Generic architecture descriptions for product lines. In *Proceedings of the Second International ESPRIT ARES Workshop on Development and Evolution of Software Architectures for Product Families*, pages 51–56. Springer-Verlag, 1998.

[43] A. Pirotte, E. Zimányi, D. Massart, and T. Yakusheva. Materialization: A powerful and ubiquitous abstraction pattern. In *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*, pages 630–641, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers.

[44] R. Prieto-Díaz. Domain analysis: an introduction. *SIGSOFT Software Engineering Notes*, 15(2):47–54, 1990.

[45] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill Science/Engineering/Math, 2002.

[46] I. Robson. From process measurement to performance improvement. *Business Process Management Journal*, 10(5):510–521, 2004.

[47] C. Rolland, G. Grosz, and R. Kla. Experience with goal-scenario coupling in requirements engineering. In *RE '99: Proceedings of the 4th IEEE International Symposium on Requirements Engineering*, page 74, Washington, DC, USA, 1999. IEEE Computer Society.

[48] C. Rolland, C. Souveyet, and C. B. Achour. Guiding goal modeling using scenarios. *IEEE Transactions on Software Engineering*, 24(12):1055–1071, 1998.

[49] R. Sebastiani, P. Giorgini, and J. Mylopoulos. Simple and minimum-cost satisfiability for goal models. In *Proceedings of the 16th Conference On Advanced Information Systems Engineering (CAiSE'04)*., 2004.

[50] T. Ziadi and L. Hélouët and J.-M. Jézéquel. Modeling behaviors in product lines. In *International Workshop on Requirements Engineering for Product Lines (REPL)*, pages 33–38, September 2002.

[51] M. Theodorakis, A. Analyti, P. Constantopoulos, and N. Spyratos. Contextualization as an abstraction mechanism for conceptual modelling. In *ER '99: Proceedings of the 18th International Conference on Conceptual Modeling*, pages 475–489, London, UK, 1999. Springer-Verlag.

[52] J. M. Thompson and M. P. E. Heimdahl. Extending the product family approach to support n-dimensional and hierarchical product lines. In *5th IEEE International Symposium on Requirements Engineering (RE'01)*, pages 56–65. IEEE Computer Society, 2001.

[53] W. Tracz. DSSA (domain-specific software architecture): pedagogical example. *SIGSOFT Software Engineering Notes*, 20(3):49–62, 1995.

[54] W. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[55] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee. The Koala component model for consumer electronics software. *Computer*, 33(3):78–85, 2000.

[56] E. S. K. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE Int. Symposium on Requirements Engineering (RE'97)*, Washington D.C., USA, January 1997.

[57] E. S. K. Yu and J. Mylopoulos. Understanding "why" in software process modelling, analysis, and design. In *Proceedings of the Sixteenth International Conference on Software Engineering (ICSE'94)*, pages 159–168, 1994.

[58] Y. Yu, J. Mylopoulos, A. Lapouchnian, S. Liaskos, and J. C. Leite. From stakeholder goals to high-variability software design. Technical report, University of Toronto, 2005.