Towards Variability Design as Decision Boundary Placement

Catalin Bidian, Eric S.K. Yu
Faculty of Information Studies, University of Toronto, Canada
{catalin.bidian, eric.yu}@utoronto.ca

Abstract

Complex information systems have numerous design variables that are systematically decided upon during the design process. In high-variability systems, some of these decisions are left open and deferred to later stages. For example, in product line architectures, some decision variables are used to generate families of products with variations in features. In user-adaptive systems, the behavior of the system is determined at runtime, based on user characteristics and preferences. In this paper, we propose to characterize variability in terms of boundaries in design decision graphs which depict the space of alternatives. A design decision about variability, such as what choices should be left to the user and which ones should be fixed at which stage in the design process, is then a question of where to place that decision boundary along some path in the relevant decision graph.

1. Introduction

Designing the architecture for a product family means to create the structures that enable the system to achieve its quality goals [1]. Software architects typically plan for changes and put various supporting mechanisms in the architecture. The architectural documentation, however, does not always reflect the architects' thoughts and efforts to achieve support for variability and understanding these situations where change has been planned for is not done explicitly.

Complex information systems have numerous design variables that are systematically decided upon during the design process. In high-variability systems, some of these decisions are left open and deferred to later stages. For example, in product line architectures, some decision variables are used to generate families of products with variations in features. In user-adaptive systems, the behavior of the system is determined at runtime, based on user characteristics and preferences.

Analyzing and documenting variability is essential, particularly if the architecture is used for many product versions over a long period of time, or in a product line where the same architecture is used to build different products.

2. Related Work

The explicit representation of variability in various phases of the product development has been suggested by recent literature. However, to our knowledge, there have been no attempts to provide a uniform representation of variability across all phases and characterize it in terms of boundaries in decision graphs which depict the space of alternatives. Variability within system architecture is typically considered during the design phase and when the product line architecture includes different alternatives for dealing with the variation in features among products [1].

Liaskos et al [2] have introduced a variability-intensive approach to goal decomposition, tailored to support requirements identification for highly customizable software. The approach is based on the semantic characterization of *OR-decomposition* of goals. The variability in solution must reflect the variability of the problem and therefore the identification of the latter is regarded as early requirements engineering problem [2]. Considering variability at the stakeholder level, in terms of their goals, characteristics, and contexts, before drafting a solution, allows us to increase the chances for the resulting product to feature an appropriate set of variation points.

An important part of domain analysis is the commonality and variability analysis, in which the common and varying elements of a domain are identified. The result is formulated as a feature model, which represents admissible combinations of user-visible characteristics of the system-to-be in a concise hierarchical manner [2]. In goal models it is possible to represent variability in stakeholder goals, through the *OR-decomposition* of goals. When a parent goal is *OR-decomposed* into subgoals, fulfillment of any of the latter implies fulfillment of the former.

Bachmann et al [3] have proposed a conceptual meta-model of variation, based on variation points, variants, assets, and rationale. The key point of the meta-model is the separation of the representation of variability from the representation of the various assets developed, with focus on the traceability between various assets.

Softgoals variability is an equally important aspect. Gonzalez-Baixauli, Leite, and Mylopoulos [4] propose a visual variability analysis technique. The model is a restricted version of the general NFR goal model, and is divided into two sub-models: a functional goal model and a softgoal one. Each *OR-path* in the functional goal sub-model represents a possible variant for the software-to-be [4].

A similar approach is the decision-making process [5] to create a generic software design, capable of accommodating the space of alternative functionalities, each of which able to fulfill stakeholders' goals. The authors generate feature models where features have sub-features of a single type and cannot have more than one set of *Alternative* or *OR-features*. In general, there is no one-to-one correspondence between goals and features, and while high-level goals can be mapped directly into grouping features in the initial feature model, leaf-level goals may be mapped into a single feature or multiple features [5].

3. Our Approach

We analyze variability at different stages in the product development process and consider that stakeholders, including designers, have various roles throughout this cycle; their goals may differ at each stage. Goals vary not only from one stakeholder to another, but also within the role boundary, affecting other goals as well as the availability of product features.

To depict variability in goals and features we use the ***i**** multi-agent modeling framework [6] and the *OR-decomposition* of goals [2], along with feature models [7] and we introduce decision boundaries in models, represented using dashed lines, to augment the space of user alternatives by depicting goals which correspond to product features whose usage will be deferred to a later stage.

Feature models are used to describe variable and common properties of products in a family of products, and to derive and validate configurations of software systems. They represent features that will be supported by the actual product. Goal models analyze the desires of users; they are intentional, describing the intended state of affairs, explaining why, or why not, processes and requirements exist, what are the alternatives that have been considered, and what criteria was used to decide among alternatives.

To highlight the variation points that relate user decisions to system features we use the *Task* notation from the ***i**** framework [6].

4. Variability Design

Defining variation points is essential for creating a product family that is flexible enough to accommodate always-changing customer needs. Variability needs to be captured from the early stages of requirements definition, further analyzed during architecture and detailed design, and identified for runtime and user decisions.

We introduce decision boundaries at each stage, discussing their placement along with some of the benefits and tradeoffs.

4.1. Decision Boundaries

Increasingly the variability in systems is moved from mechanics and hardware to software. The product family architecture must be designed in such way to support different products, particularly when special configurations of the software components are required. However, because of the high cost of reversing design decisions, software engineers attempt to postpone such decisions to the latest phase in the product development. Over the last few years many organizations have identified conflicts in their software development because of the constantly increasing number of features a product must have to better service the various market segments [8].

Some of the features available in the final product are left open for users' decision. Modeling these decisions into goals and translating them into subsequent features gives no clear demarcation of the magnitude of variability in user decisions. We introduce decision boundaries (dashed line in Figure 1) to augment the space of user alternatives by depicting goals which correspond to product features whose usage will be deferred to a later stage and ultimately determined by the user.

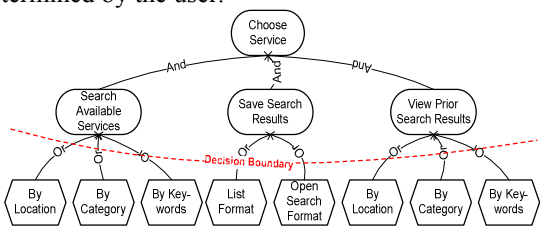

Figure 1. Decision boundary

The decision boundaries will assist in analyzing and establishing the point where detailed design decisions and constraints are loosened and the user can make a selection from an already available range of features.

For instance, in a healthcare scenario [9], patients have requested messaging communication with their physicians. Sending messages in this case proved to be a viable alternative to satisfy the requirement.

Sending messages (Figure 2) depends on the *Type* of message, which can be *Instant* if the communication is done on-line or *Deferred*, when the communication is done off-line. It also depends on the sender's *Location*, which can be *Static* or *Dynamic*, depending on whether or not the user is in motion.

The corresponding feature model (Figure 3) outlines two main alternatives in the final product: *On-line* or *Off-line* communication.

However, the study [9] mentions a conflict of preferences among stakeholders. Patients opted for *On-line*, being more concerned with obtaining real-time response, yet physicians' preferred *Off-line* because it would allow them to read and respond to patients' questions without being interrupted from other consultations.

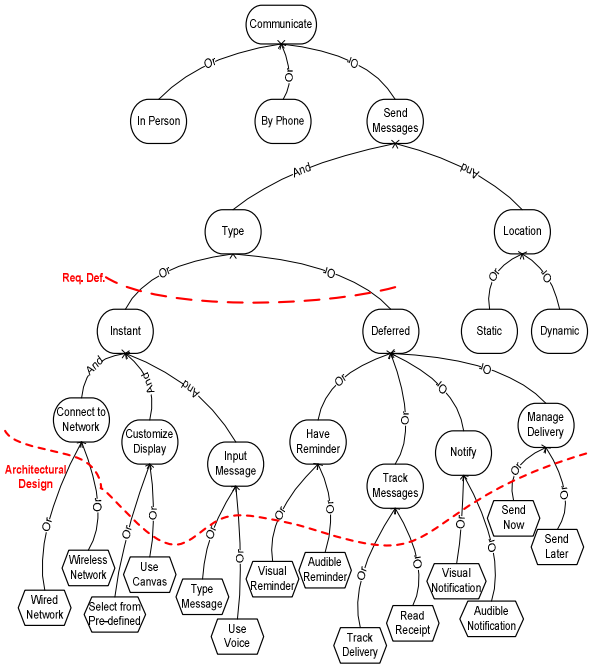

Figure 2. Decision boundaries in goal models

In Figure 2 we note the two places where a boundary was introduced. At requirements definition the boundary signifies that the final product will have both *On-line* and *Off-line* features, and all the sub-features will be left open for user's decision. This will subsequently increase production costs, which may not be fully justifiable for all markets. In contrast, if only one feature will be made available, for instance *Off-line*, it may diminish the ability of the product to satisfy various markets.

Goals determined during architectural design, such as having the ability to *Connect to Network*, or *Have Reminder* and *Track Messages*, actually translate to an entire set of features in the final product, such as the existence of wired or wireless network capabilities, a *Text* or *Audible* reminder, and tracking the *Delivery* and/or *Message Read*, when the message was delivered to and/or read by the recipient.

Therefore some decisions may be deferred from the requirements definition to a later stage (*Architectural Design* dashed line in Figure 2). In this case only a smaller set of features in the final product will be left open (*Architectural Design* dashed line in Figure 3).

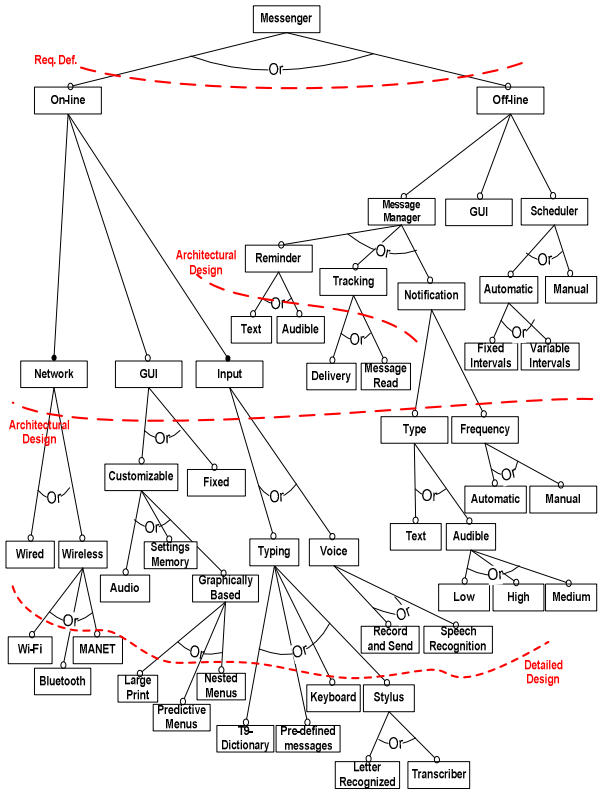

Figure 3. Decision boundaries in feature models

Subsequently, the production costs will not be as high as when the boundary was placed at the requirements definition.

This is not to say that placing decision boundaries as late as possible in the product development and

subsequently leaving fewer features open is recommended. Rather understanding where to place these boundaries, what are the consequences and trade-offs, and how the open features will influence the ability of the final product to satisfy various markets, is the key point in introducing decision boundaries in variability analysis.

In the following sections of the paper we depict and discuss decision boundaries at various stages in product development.

Decision boundaries can be introduced as early as the requirements definition stage.

4.2. Decision Boundaries at Requirements Definition Stage

Requirements are not stable; new or variations of the existing requirements are continuously added. The final product becomes efficient if it is designed to accommodate the possible variations in requirements. Documenting variability and making decisions at this stage is necessary due to numerous requirements rising from multiple stakeholders and their preferences, products, releases, quality constraints, and other specific criteria, such as country or region restrictions.

For instance, in the healthcare scenario [9] previously introduced, because of the conflict of preferences among stakeholders, whereby patients preferred on-line communication for real-time response and physicians opted for off-line, not to be interrupted from other consultations, we can place the decision boundary after the message *Type*, allowing the user to determine whether he/she will be conducting instant or deferred communication (Figure 4).

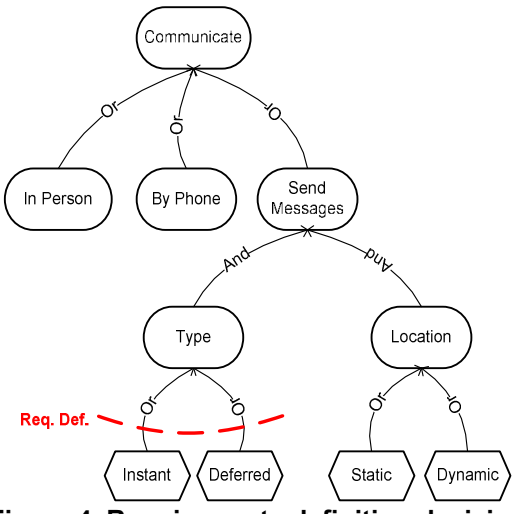

Figure 4. Requirements definition decision boundary – goal model

This reflects in the feature model (Figure 5), allowing us to note that all the features necessary for communicating online or offline will be left open.

If the final product will have the *Messenger On-line*, all the features will have to be made available to the user and this will influence subsequent architectural and design decisions. For example, the *Typing* (Figure 3), done by the use of *Stylus* or *Keyboard*, affects how instantaneous communication is conducted, because users may only be accustomed with one of the features. As well, the various display sizes are hardware variants, each of them requiring a specific display driver [10]. In other words, the software in the final product may have to incorporate variation points to support different display sizes.

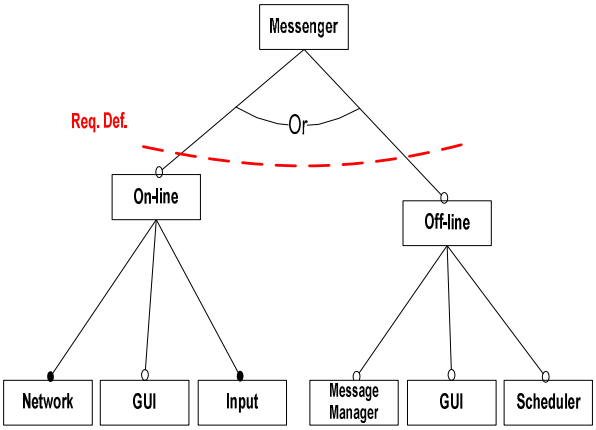

Figure 5. Requirements definition decision boundary – feature model

Accurate identification of decision boundaries during the requirements gathering phase influences subsequent architectural and detailed design decisions.

4.3. Decision Boundaries in Architecture Design

A family of products structures its architecture around major commonalities that can be implemented as prefabricated components. Each product is derived from the product family architecture, by using a set of generic components and introducing product-specific code [11].

The variability determined at the requirements gathering phase maps into variability that can be defined at the time of architecture design. At this stage, identifying decision boundaries assists in outlining different possible architectures and supports impact assessment from the perspective of changes that are required in the product family.

For instance, hemodialysis systems have two communication ways [12]: *Push* connection – the hemodialysis machine sends updates to the external system; and *Pull* connection – the system requests updates from the hemodialysis machine. Figure 6 and 7 depict the goal and feature models.

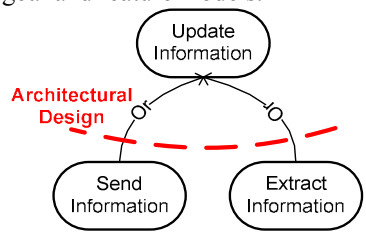

Figure 6. Update info – goal model

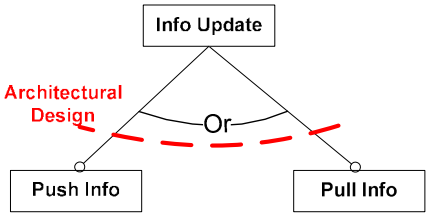

Figure 7. Update info – feature model

Either approach will satisfice the goal of updating treatment information, yet there are trade-offs. If the information is pushed, the hemodialysis machines may require complex modifications, considering the wide range of products on the market. If the information is pulled, software components have to be developed for the system to extract the information without exposing the patient to any hazards.

While it may be easier to define a single architecture for a product, in the case of product families each variation point indicates a possible change in the final architecture.

4.4. Decision boundaries in Detailed Design

Variability analysis and decision boundaries at this stage enable designers to establish alternatives in product specifications and identify possible solutions up-front, without sacrificing any of the qualities of the final product. It also provides the opportunity of reusing components, saving costs and time, when creating new products. Leaving design decisions open allows for future requirements to be met without starting from scratch.

For example, in Figure 3 we note that the final product may have *Wired* or *Wireless* network connectivity. The type of wireless connection, *Wi-Fi*, *Bluetooth*, or *MANET*, can be determined at detailed design time. The decision boundary identifies which product features are left open to meet various requirements.

Another example is an *Alarm* feature which can be enhanced at detailed design time with built-in intelligent decision-making algorithms. Based on parameters such as history of events, *Notification Type*, or *Alarm Intensity*/*Urgency*, the algorithms will 'decide' the *Notification Subject*, *Notification Location*, and/or *Notification Process* that will require immediate or delayed attention (Figure 8).

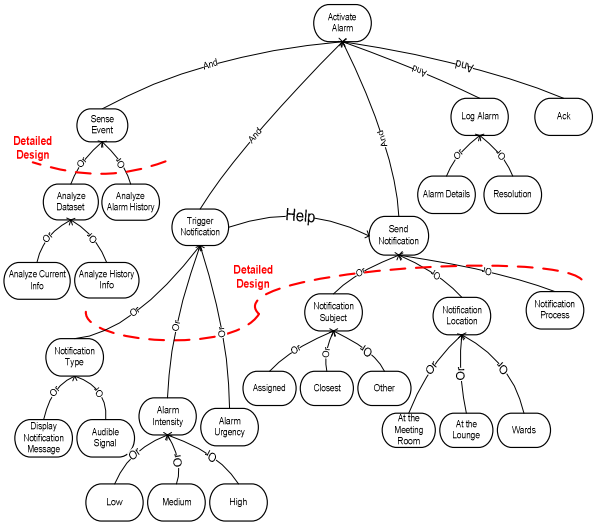

Figure 8. Activate alarm – goal model

The higher-level variation points entail more influence on the detailed design than the lower-level alternatives. For instance, the *Alarm Intensity* can be *Low*, *Medium*, or *High* yet the final design does not significantly change, whereas how the notification is triggered, through the selection of *Notification Type* and alarm *Intensity* and *Urgency*, will have a stronger impact on the final product (Figure 9).

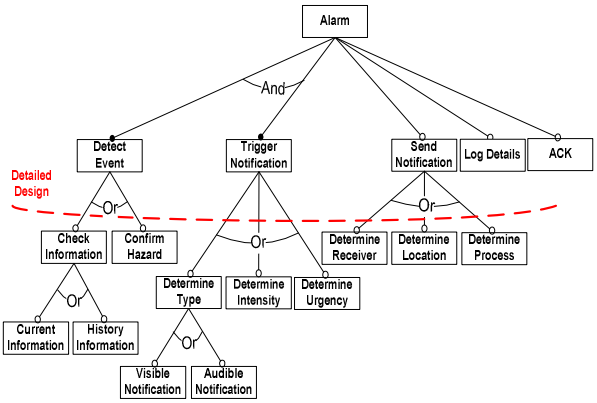

Figure 9. Activate alarm – feature model

During requirements definition and architecture design it may also be necessary to establish contingency and/or emergency plans. The resulting decisions will bring further goal refinements and subsequent changes to the detailed design.

4.5. Decision Boundaries at Runtime

One of the main contributions of a family of products is its focus on domain-specific architectures, providing a systematic derivation of a tailored approach, suited to the capabilities and objectives of a type of corporation. However, many domains impose requirements for domain-specific customization that can hardly be implemented with variation points that are bound before runtime [11].

In user-adaptive systems, the behavior of the system is determined at runtime, based on user characteristics and preferences. These are domain-specific variation points that can be dynamically actualized through ad-hoc system customization done by domain experts. To reach their goals, it is important for domain experts to be able to understand the product variability and make decisions without having to learn about other parts of the architecture.

Goedicke et al [11] discuss the situation of a large warehousing organization with multiple different media shop types, such as web shops, interactive television shops, and m-commerce shops. In this context we have multiple requirements for runtime variability. For instance, *Personalize Shopping* – the website can be personalized via a form-based or graphical interface (Figure 10).

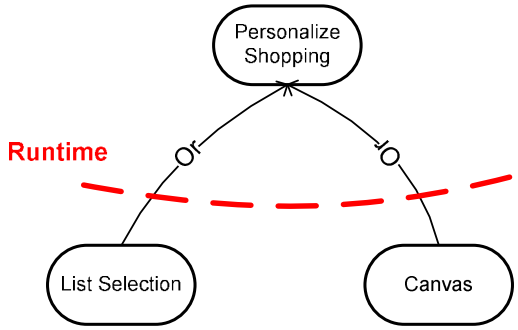

Figure 10. Personalize shopping – goal model

Domain experts can select from a list of available options – predictive menus, nested menus, etc – or use a 'canvas' to paint the website to match market preferences. The personalization information is then stored on the server which in turn generates personalized pages for different content formats (Figure 11).

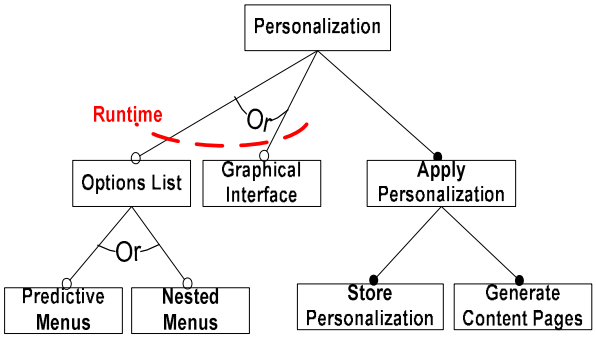

Figure 11. Personalize shopping – feature model

A similar example of runtime variability is the customization of interactive applications which is done by domain experts and content providers via an interface that generates web forms or applets. The parameters are handled by content editors and the customer should not experience the common product line realization during shopping, rather should have the impression that each shop has an individual appearance.

The main trade-offs are driven from the fact that while domain experts have adequate knowledge about the market, they still have to foresee the users' preferences, which may differ from one context to another. This, in turn, entails the availability of more/less features in the final product to meet the users' preferences. Placing decision boundaries helps identifying which features should be left open and which should be fixed, thus eliminating redundancy.

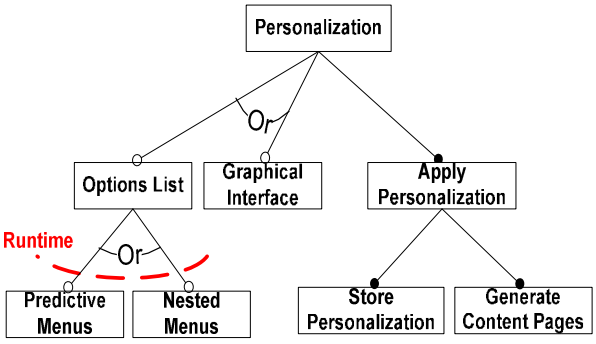

Figure 12. Moving the decision boundary

If the decision boundary is lowered such that only *Predictive Menus* and *Nested Menus* will be left open (Figure 12), then the higher level feature *Graphical Interface* becomes fixed, disallowing customization of the website color, fonts, and other such attributes, or even completely removed, with possible subsequent reduction of development costs.

4.6. User Decision Boundaries

Building complex architectures for product families raises the issue of specific customizations that have to be incorporated to satisfy various markets. For each customization a certain programming effort is required. Often, initially found variation points do not match the reality of users' preferences. If these variation points are not resolved, it may become difficult to customize products that require rapid changeability.

Identifying and modeling variability at the time of architecture and detailed design does not define which features in the final product are left open for user's selection and how they relate to the higher level goals. While architectural and detailed design variations are considered in the early phases of product development, some alternatives will eventually be left for users to consider. These alternatives are features which have been included in the final product yet their usage will be determined by the user.

For example, from a user perspective, a mobile device should allow selection of services, such as networks, servers, or other mobile devices. Hence, the user should be able to search for services by *Location*, *Category*, or *Key-words* [13] (Figure 13).

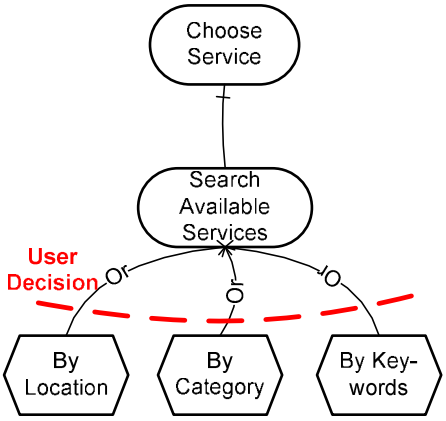

Figure 13. Selection of services

The ability to choose a service is the user's high level goal and from a system perspective this translates into *Search Available Services*. The tasks *By Location*, *By Category*, and *By Key-words* depict the alternatives provided by the system, allowing the user to choose his/her preferred way.

Expanding this requirement further, reveals the necessity to customize the search and save the search results into a list format, or other formats, such as open-search, that can be viewed at any time, either by *Location*, *Category*, or *Key-words* (Figure 14).

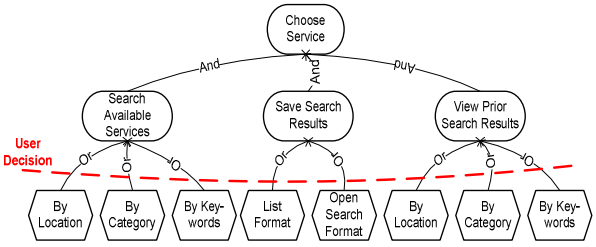

Figure 14. Selection of services – refined

An additional requirement is to have the ability to transfer the search lists to either a server or another mobile device [13]. To allow the user to perform these functions, the system will provide a graphic interface through which the user will input the search parameters, view and customize search requests, as well as store and transfer the search results. The system will provide access to the requested services when the mobile device and the appropriate service are co-localized – when a connection is possible [13]. This implies the ability to search for services as well as the existence of a certification mechanism to support security requirements, such as authentication and authorization.

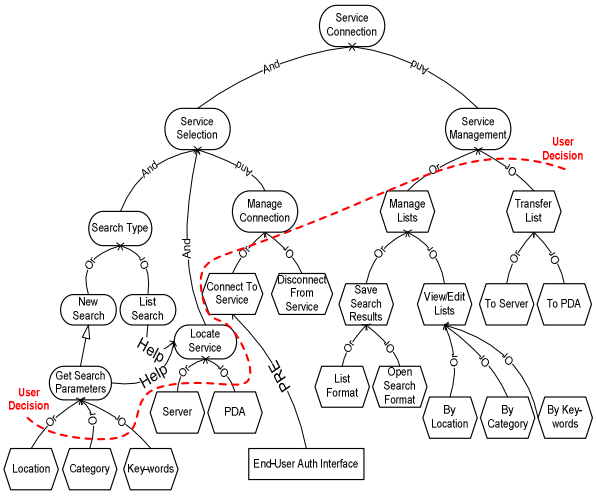

Figure 15. Service connection

The security services, authentication and authorization, are provided within the *Service Selection*. The system architecture in this case would have an *End-User Authentication Interface* responsible for access control and identity management. Therefore, *Service Selection* will communicate with this interface for connection to other services, such as networks, servers, and PDAs. As well, management of lists should be done in a separate module, because of the variation of services and connection types – servers,

PDAs, other mobile devices, Bluetooth, Wi-Fi, MANET, etc.

The system must support interaction between different services and should keep track of all services to which the user's mobile device has been connected [13]. Keeping track of connections can be accomplished either by introducing functionality in *Service Management* or by having a separate mechanism for audit. The latter, however, would significantly modify the design. Most system designs take the approach of activities logging and audit as a separate functionality. Therefore, the system architecture may encompass an *Audit Manager* feature.

The decision boundary can also be placed as depicted in Figure 16.

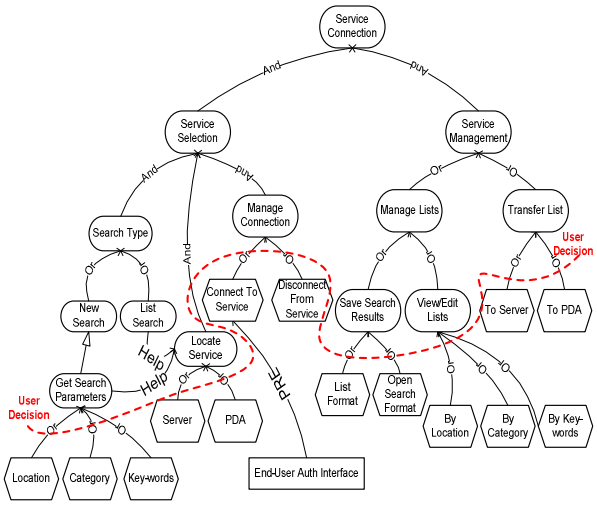

Figure 16. Service connection – different placement of the decision boundary

In this case, some of the features that were previously left open for the user are now fixed. For instance, *Manage Lists* and *Transfer List* are no longer determined by the user. Only how the lists are managed and where they are transferred remains open.

Lowering the decision boundary gives less opportunity for users to determine how and when the product features will be utilized and at the same time decreases the flexibility of the product and its ability to fully satisfy various markets.

4.7. Decision Boundaries and Softgoals

Gonzales-Baixauli, Leite, and Mylopoulos [4] state that goal analysis has to combine quantitative with qualitative techniques to account for subject matters that are not readily quantifiable. The analysis of the interaction of non-functional softgoals and functional goals at a high level of abstraction is an important element, especially when designing variability.

We propose the analysis of how decision boundaries and the variability of goals influence softgoals and their achievement, allowing early identification of variation points in the product architecture.

For example, Liu, Yu, and Mylopoulos [14] state that security issues are about relationships between social actors – stakeholders, users, etc. – and the software acting on their behalf. Let us consider *Security*, *Accountability*, and *Anonymity* as softgoals that need to be satisficed through the introduction of *Protection Mechanisms* (Figure 17).

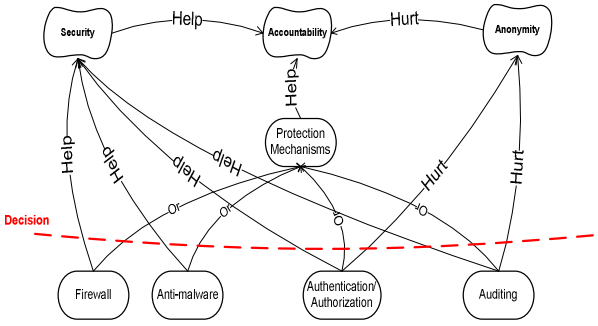

Figure 17. Protection mechanisms

Some subgoals can be *Firewall*, *Anti-malware*, *Authentication/Authorization*, and *Auditing*. If we place the decision boundary as shown in Figure 17, we note that the subgoals *Help* and/or *Hurt* the softgoals. For instance, the presence of the *Auditing* and the *Authentication/Authorization* mechanisms, while helping *Security*, hurts *Anonymity*.

However, if we further refine the subgoals and move the decision boundary, we note that there are alternatives to satisfice both *Security* and *Anonymity* (Figure 18).

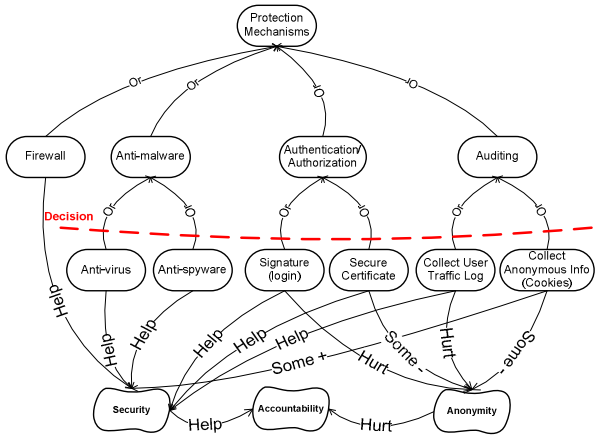

Figure 18. Protection mechanisms – refined

Using *Secure Certificate*(s) for user authentication can have only *Some*– (negative) impact on *Anonymity*, or virtually no impact if third party certificates are used, for instance public and/or private keys. As well, collecting only anonymous information through light usage of cookies has only *Some*– impact and therefore the softgoal of *Anonymity* can be satisficed. Using cookies however has only a small contribution (*Some*+) to *Security*. Nevertheless, considering that the other goals have a significant contribution – *Help* – satisficing *Security*, then the usage of cookies might be an acceptable tradeoff.

5. Conclusions and Future Work

When considering different options, stakeholders need to understand how each option will impact their work and pursuit of the project and personal goals [15]. This will help them choose the design that better meets their needs and interests. Decision boundaries permit in-depth analysis of the stakeholder goals and product features. They depict the variability magnitude and allow easier identification of goals and corresponding features that will be deferred to later stages in the product development

We have introduced and briefly discussed decision boundaries in variability analysis, at various stages in the product development cycle, and sketched the analysis of decision boundaries' influence on softgoals. Our paper complements earlier works on the goal-model based approach to variability. We use the *OR-decomposition* of goals [2], adapted to depict variability in highly customizable software, and the decision-making process for software design [5], which allows identification of alternative functionalities that will satisfice stakeholders' goals, and enhance them with decision boundaries for a clear depiction of the extent of variability.

Further work is required to refine the decision boundaries and provide a better understanding of their placement to support product analysis and design, including design trade-offs and influences on softgoals variability. We endeavor to establish precise rules for setting a boundary and how far a boundary can go, examine the connections between boundaries, and define a meta-model and the associated procedures for using it. We will also explore further how the decision boundary concept can be used to aid in the derivation of feature models from goal models.

As well, due to the potential growth of models, there will be a need to develop tools to manage the large scale models, such as visualization and navigation aids that will provide selective views of portions of the overall graph and the queries to generate such views from an underlying formal representation.

Furthermore, requirements conflict is an important aspect in requirements engineering research [16]. Stakeholders often have divergent interests that can drive the final product design in different directions and multiple points of view lead to overlapping requirements that do not always agree [16]. Decision boundaries can play an important role in conflict analysis.

6. References

[1] F. Bachmann and L. Bass, "Managing variability in software architectures", *Symp on Software Reusability: Putting Software Reuse in Context*, 2001, pp. 126-132.

[2] S. Liaskos, A. Lapouchnian, Y. Yu, E. Yu, and J. Mylopoulos, "On goal-based variability acquisition and analysis", *14th IEEE Int'l Requirements Eng Conf*, 2006.

[3] F. Bachmann, M. Goedicke, J. Leite, R. Nord, K. Pohl, B. Ramesh, and A. Vidbig, "A meta-model for representing variability in product family development", *Springer-Verlag LNCS 3014*, 2004, pp. 66-80.

[4] B. Gonzalez-Baixauli, J. Leite, and J. Mylopoulos, "Visual variability analysis for goal models", *12th IEEE Int'l Requirements Eng Conf*, 2004, pp. 198-207.

[5] Y. Yu, J. Mylopoulos, A. Lapouchnian, S. Liaskos, and J. Leite, "From stakeholder goals to high-variability software design", *Tech. Rep. CSRG-509*, Canada, Ontario: University of Toronto, 2005.

[6] E. Yu, "Towards modelling and reasoning support for early-phase requirements engineering", *3rd IEEE Int'l Symp. on Requirements Eng*, 1997, pp. 226-235.

[7]. K. Czarnecki, and S. Helsen, "Classification of Model Transformation Approaches", *OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, 2003.

[8] M. Svahnberg, J. van Gurp, and J. Bosch, "A taxonomy of variability realization techniques", *Software – Practice & Experience*, *35*(8), 2005, pp. 705-754.

[9] A. Tjora, T. Tran, and A. Faxvaag, "Privacy vs. usability: a qualitative exploration of patients' experiences with secure internet communication with their general practitioner", *Journal of Medical Internet Research*, 7(2), 2005, e15.

[10] M. Jaring, and J. Bosch, "Variability dependencies in product family engineering", *Springer-Verlag*, LNCS *3014*, 2004, pp. 81-97.

[11] M. Goedicke, K. Pohl, and U. Zdun, "Domain specific runtime variability in product line architectures", *8th Int'l Conf on Object-Oriented Information Systems*, 2002, pp. 384-396.

[12] P. Bengtsson, and J. Bosch, "Haemo dialysis software architecture design experiences", *21st Int'l Conf on Software Eng*, 1999, pp. 516-525.

[13] V. Bryl, P. Giorgini, and S. Fante, "ToothAgent: a multi-agent system for virtual communities support", 2005.

[14] L. Liu, E. Yu, and J. Mylopoulos, "Analyzing security requirements as relationships among strategic actors", *2nd Symp on Requirements Eng for Info Security*, 2002, pp. 1-14.

[15] E. Yu, and J. Mylopoulos, "Understanding 'Why' in software process modeling, analysis, and design", *16th Int'l Conf on Software Eng*, 1994, pp. 159-168.

[16] E. Yu, and J. Mylopoulos, "Why goal-oriented requirements engineering", *4th Int'l Workshop on Requirements Eng: Foundations of Software Quality*, 1998, pp. 15-22.