Managing Variability with Traceability in Product and Service Families

Kannan Mohan and Balasubramaniam Ramesh
Department of Computer Information Systems
Georgia State University
{kmohan|bramesh}@gsu.edu

Abstract
Service family architectures are based on a set of basic building blocks that can be configured appropriately to build different services based on the same set of assets. These building blocks should be designed to provide variability so as to facilitate configuration and assembly in order to produce a family of products/services. Such an architectural design for a family of services is recognized as a highly knowledge intensive process. Documenting design decisions associated with these various configurations leading to variability and the capability to trace the life of a variation in both forward and backward directions are key to a flexible development of a service family. In this paper, we discuss the key role played by a traceability-based knowledge management system in documenting such design decisions and in tracing variability. Using a case study, we illustrate the importance of using such a knowledge management system in the design and development of service families.

1 Introduction
The development of e-services, especially for information products, is a very knowledge intensive activity. Service industries have high knowledge content [5] and e-services are prime candidates for knowledge management-centric customization. Applying the concepts of mass customization from manufacturing to knowledge industries allows service providers to enhance the flexibility of products and services, lower costs, increase responsiveness, and reduce cycle time. Service families are based on common architectures that are used to derive the architectures of its members [6]. Such a design using a common architecture or a service platform is preferred as it facilitates customization. Service variety is achieved by careful design and development of a service platform and then plugging in different modules depending on the customer-specific requirements. This approach is associated with extensive configuration design, which involves determining relationships among various components and modules [7]. Efficient development of e-services can benefit immensely by capturing knowledge about architectural design decisions associated with such service platforms, as these decisions have cascading effects on the variety of services that are synthesized from the platform. Specifically, this knowledge will be very helpful in rapidly reconfiguring e-services to suit the varied requirements of different user segments. A basic premise of our work is that traceability, defined as the ability to follow the life of an artifact, provides a powerful approach to managing knowledge in the development of reusable assets produced during domain engineering and configuration of customizable e-services families.

Traceability plays a key role in the development of software systems by confirming the compliance of a system with its requirements. Standards and guidelines for system and software engineering identify requirements traceability as a major concern in achieving quality [1]. It is identified to be central to program maintainability as it is a prerequisite for effective system maintenance and consistent change integration [2]. Prior research also suggests that maintaining traceability leads to less errors and improves the acceptance of the software system. Pohl [3] argues that pre-requirements traceability (tracing requirements to their origins) lays a foundation for managing system evolution when requirements and the context in which systems are expected to operate evolve.

In this paper, we discuss a knowledge management system, which is based on a traceability framework, to capture the various design decisions associated with e-service family development. Using a case study, we illustrate the key role played by the knowledge
management system in the design and development of an e-service family.

2 Traceability

Traceability refers to the ability to describe and follow the life of a conceptual or physical artifact, in both forward and backward directions (i.e., from its origins, through its development, to its subsequent deployment and use, and through periods of on-going refinement and iteration in any of these phases) [1]. Pohl [3] further classifies traceability as pre and post traceability. For example, post requirements traceability is defined as that of refinement, deployment and use of a requirement, while pre-requirements traceability is defined as the traceability of a requirement back to its origin [3].

Ramesh and Jarke [8] summarize the various dimensions of traceability and emphasize the importance of capturing and maintaining trace information by describing the various applications of such information along the various phases of software development life cycle. They argue that traceability facilitates communication among the various stakeholders of the project. The ability to link requirements to their sources as well as the various artifacts developed during the various phases of the system development life cycle can be used to demonstrate how each requirement has been satisfied in the ensuing phases of the development life cycle. This repository of information aids in tracking what is to be changed in an artifact when a change request arises. Further, it can also aid in estimating costs for evolutions/changes.

3 E-Service families

Services that are delivered electronically, typically through the Internet, wireless or land-based telecommunication systems are referred to as e-services. They are defined as internet-based applications that fulfill service needs by seamlessly bringing together distributed, specialized resources to enable complex transactions [9]. The transformation from product-oriented approach to a service-oriented approach necessitated by customer needs for results rather than products, demands the application of development processes associated with services rather than products [10]. Apart from demanding results, customers also care about distinctiveness of services, thereby driving the need for customization. In order to cater to this demand for distinctiveness, companies have to move away from developing software services from scratch. Instead they need to focus on exploiting the commonality among various products, thereby capturing those in a service family architecture [11]. Kahn [12] argues that building a standardized platform and adding customer specific attributes on top of this platform reduces development costs. A service platform is defined as the common set of design variables around which a family of services can be developed [7]. It is a common technological base from which a service family is derived through modification and instantiation of the platform to target specific market niches [13-15]. A clear platform strategy leverages the resulting products, enabling them to be deployed rapidly and consistently [14]. Since costs are driven by commonality among the various products [16], application of product family engineering concepts in customized e-service development would yield considerable benefits in terms of cost and time by facilitating component-reuse.

We define a service family as a set of services that share certain common aspects and have predicted variabilities. Services are clustered as a family based on their commonality, as it is easier to analyze, design and manage the family as a related set of elements rather than concentrating on each member of the family separately [17]. A family-based approach emphasizes both on the commonality among family members and differences between them. Coplien et al. [18] argue commonality and variability analyses helps software engineers achieve a systematic way of thinking about and identifying the family of services/products that they are developing.

3.1 Commonality and variability

Apart from the fact that building a standardized e-service platform and adding customer specific attributes on top of this platform reduces development costs [12], there is evidence to support delaying the points of differentiation in a production process [19]. Postponement, as it is called, is found to be an effective strategy for increasing service variety. Effective platform-based development depends on exploiting the commonality and variability in customer requirements. Recent literature on commonality and differentiation indices alludes to the advantages of using product platforms to proliferate product variety [20]. Commonality is defined using the commonality index [20], which is a measure of how well the design utilizes standardized modules. A higher commonality index is advocated, as it would indicate that product varieties are achieved with fewer unique parts. Differentiation index, which measures where differentiation occurs within the process flow, defines differentiation. This index provides quantitative support for postponement of differentiation.
4 Related research

Bosch [11] discusses the various problems and issues faced by organizations following a product line approach to software development. One of the primary issues identified by Bosch is the need for software engineers to have considerable knowledge about the rationale and concepts underlying the product line architecture and the structure of the reusable assets that are part of the architecture. He argues that usage of product line architectures demands even more architectural and design decision knowledge than for that of general reuse-based software engineering. Further, knowledge about conflicting quality attributes for functionality required is of paramount importance so as to identify the need for implementing different components for the same functionality satisfying these quality requirements. It is critical that software engineers are provided with the capability to trace to the source that demands such conflicting quality attributes so that they can justify the need for multiple components to implement the same functionality in different contexts. This necessitates the maintenance of traceability information in both forward and backward directions. The necessity for traceability maintenance is justified by the need to capture knowledge about architectural decisions associated with service platforms and those that describe reconfigurability of various components. Dependencies among various components that are to be plugged in to the platform should also be captured so as to facilitate reuse under various contexts. Bosch [11] identifies the decision to separate a service/product from a product line or to merge the same to a product line as a critical issue. Therefore, we argue that it is also important to capture decisions related to points of differentiation.

Recent research has identified two common approaches to product family design. In a top-down approach, a company strategically manages and develops a family of products based on a product platform. In a bottom-up approach, a group of distinct products are consolidated or redesigned by standardizing the components used in order to improve economies of scale and reduce inventory [7]. The former approach is advocated for products that are designed from scratch, while for those that are already in production, a redesigning bottom-up strategy is suggested. Since the top-down approach is based on a product platform, upfront capture and analysis of specific customer requirements, which would provide a handle on the level of variety required, is of paramount importance, as this is the parameter based on which the level of differentiation postponement can be determined. Apart from just capturing these common as well as customer-specific requirements, it is essential that designers should be invested with the capability to traverse from the common requirements to its corresponding implementations in the platform, and from the variable, customer-specific requirements to the configurable and pluggable modules of the service.

5 Knowledge management-centric approach

The focus of our work is the development of a framework and a knowledge management system for managing traceability knowledge in the design and development of e-service families. Using a case study, we illustrate the key role played by traceability in managing the highly knowledge intensive process of addressing variability in e-service families.

5.1 Case study

A warehouse management system (WMS), which constitutes one link of a supply chain management system, is described, in order to illustrate the usage of our knowledge management system as a traceability tool in the design and development of an e-service family. Tiwana and Ramesh [21], in their taxonomy of e-services, classify supply chain management systems as a physical B2B service. The WMS is used to handle inventory sent to and from warehouses depending on orders from customers. The software development firm (WMSCo) that is deploying this WMS had developed a ‘base’ version of the product, which encompasses most of the functionality required by its entire customer base. Initially WMSCo started with a minimal base with a few modules and gradually expanded the base product based on requirements prevalent in the industry. WMSCo handles customers from a variety of vertical industries, and within each, a number of customers with diverse functional requirements. For every customer, the base product is customized based on the functional requirements and delivered. Most common customizations required are associated with material handling equipment communication. As WMSCo has a global market, the base product has to be customized to support multi-lingual operation. Since the development team’s ultimate aim is to reduce customization costs as much as possible, most of the adaptations required for various vertical industries are achieved through configuring the base product to operate under the particular context. Teams working on diverse domains frequently meet in order to discuss the possibilities of escalating the most common functional requirements to a feature of the base product. Various optional modules like order management system, workflow management system and transportation management system can be plugged in depending on customer-specific needs.
5.2 Our approach

We describe a knowledge management system based on a traceability framework and its key role played in tracing commonality and variability in customer requirements to their corresponding design artifacts. We contextualize and illustrate the trace and design decision capture using sample scenarios from our case study.

The primary task in designing a product-line architecture is analyzing the commonality and variability in functional as well as non-functional requirements of a family of products [22]. Our framework can be used to capture deliberations that lead to identification of commonality and variability of requirements and trace the same to their corresponding implementations. Also, the rationale behind various key architectural decisions taken based on the outcomes of these deliberations can be recorded. It is important to capture these architectural decisions, as these are key to understanding future configurations and customizations on the platform that had been created.

5.3 Framework for traceability

We have developed an initial framework for representing traceability among various information objects produced during complex system development activities such as the customization of an e-service. This framework provides primitives to represent the agents, inputs and outputs of the system development process as well as the linkages among them. It can be used to represent various dimensions of traceability information.

The framework is based on reference models for traceability developed in the context of complex system engineering activities [8]. The reference models have been applied in a variety of domains including new product development, concurrent engineering and software development.

Various salient attributes or characteristics of information to be captured can be represented using our framework.

![Conceptual Model (adapted from [8])](image-url)
For instance, links among the inputs and outputs of the customization process such as Requirements, Assumptions, Design objects, System components, Decisions, Alternatives, Critical Success Factors, Configurations, Constraints, etc. must be maintained. These represent the major conceptual elements among which traceability is maintained during the various life cycle stages. Using the conceptual bases of traceability, these input/output combinations can be customized to any specific service domain.

Details about the various stakeholders who play different roles in the creation, maintenance and use of various conceptual objects and traceability links across them need to be represented using this framework. For example, a customer who is a source of requirements for a given configuration needs to be linked to the requirement. In the context of service families, maintaining such linkages between customers and their specific requirements is useful, as these requirements are the identification points of variability and commonality.

Details about the sources that document traceability information need to be captured. Examples of sources of traceability information include requirement specification documents, meeting minutes, design documents, design artifacts, memoranda, telephone calls as well as references to various stakeholders using their phone numbers, e-mail addresses, etc. Stakeholders manage the sources; i.e., they create, maintain, and use them.

Traceability information can be captured using formal as well as informal means. The rationale behind the creation, modification, and evolution of various conceptual objects need to be maintained. This information may include issues, alternatives, and arguments supporting and opposing various architectural, design and implementation decisions. The capture of these various chunks of information can be time-stamped so that the various stakeholders can know when this process-related information was captured, modified and evolved. This framework addresses the different information views (organizational, functional, data, control and output) proposed by Scheer [23] for the Architecture of Integrated Information Systems (ARIS) approach.

Figure 1 shows our initial conceptual framework, adapted from recent work on reference models for requirements traceability [8]. Here, we focus primarily on deliberations that are oriented towards identifying common and variable requirements, those that deal with architectural design decisions that affect the structure of the platform or the base product, those that are raised by functional customizations/configurations required to be done on the base product, and on tracing these common and variable requirements both backward and forward in the development life cycle. The model includes primitives critical to e-services development such as architectural decisions, configurations that result from the design objects / components created during development. Here, requirements refer to customer specific functional or non-functional requirements that would demand customization of the base product as well as common requirements demanded by a wide customer base that would be reflected as an implementation which would be part of the service platform.

5.4 KMS Characteristics

Our Knowledge Management System (KMS) is based on the REMAP (Representation and Maintenance of Process Knowledge) environment that is capable of supporting the capture and reuse of process knowledge [24]. REMAP provides a Web-enabled tool for participants in a service customization team to capture and maintain traceability information representing the various components described in previous section, thereby instantiating our conceptual framework. This software tool is capable of representing this knowledge both formally as well as informally using multimedia chunks. A truth maintenance system supports automated reasoning with captured process knowledge such as the ability to synthesize configurations that are based on a specific set of requirements and assumptions.

5.5 Scenarios on KMS use

We discuss the support for managing traceability knowledge in supporting e-service development using three sample scenarios drawn from our case study. The first is concerned about the need to explicitly represent the reasons behind changes in the base configuration of the e-service due to changing requirements from various customer segments. In this scenario, various developers handling requests from customers belonging to diverse vertical industries have frequent meetings to decide on escalation of certain commonly requested requirements to a typical feature offered in the base product. These requests are consolidated by a separate team that analyzes the need for including the new requests in the base product. An instance of such a deliberation is shown in Figure 2. A change in the type of the label printer had been requested frequently that developers argue that the base product should be configured such that this new label printer is the default option. The argument that supports this configuration change in the base product asserts that the new printer is more popular in the industry, apart from WMSCo’s current customers.
This argument is based on the assumption that the new printer has been used as a standard printer in at least half of the vertical industries handled by WMSCo. The validity of this assumption makes the argument that is based on this assumption, valid. This leads to the decision to change the default label printer type in the base product. This decision leads to the ultimate configuration of the printer module to support the new printer and to have this as the default option.

Our system is intended to help make decisions about commonality and hence the base configuration of the product in a number of ways. Figure 3 shows a scenario involving an architectural decision to illustrate such capability. WMS is responsible for communication with an order management system. It should be capable of handling order routing if the customer does not have his/her own order routing system. Order routing is added as a feature depending on customer-specific request. Frequent requests for order routing raises an issue and adding order routing as a standard feature to the base product could be an alternative that would respond to this issue. An argument that this component should be added to the base product as it is a standard feature requested by most customers is based on the assumption that at least some (x %) of the customer base requests for the order routing feature. Initially, the number of customers requesting such a feature may be very low and therefore, the component is not included in the base product. However, the validity of the assumption can be ascertained by the system by monitoring the functionalities requested by different customers. When the percentage of customers that require this feature exceeds the threshold (say, 40%), this system can trigger a variety of actions. First, it can simply notify the design team of the changed scenario with respect to this commonality decision suggesting the redesign of the product. On the other hand, if the architecture of the system allows, the system can initiate automated synthesis of the base configuration to include the common functionality. The degree of automation in this feature will depend on a number of factors, including the formality with which traceability information is maintained, the architecture of the product configurations, the desirability of such rapid reconfigurations in the application domain etc.

Figure 2: Capture of deliberation to make a configuration decision
Finally, we illustrate the generation of a configuration satisfying a specific customer requirement. In this scenario, specific customer requirements affect architectural decisions. Among them is a decision about the changing the inner pack number for pellets used in the warehouse. The current base product is not capable of supporting configurations for inner pack numbers. As the percentage of customers that require this feature is below the threshold, it is considered a variability point in the design process. On the basis of the requirements and assumptions specified by the customer, the specific configuration created for the customer includes this component. The scenario depicts the capture of assumptions behind the decision to componentize the inner pack number configuration feature. The assumption that separation and addition of such a component does not conflict with any other component that is added as a result of a customer-specific variability supports the modularization of this particular feature. Similarly, our system evaluates the parameters of various functional and non-functional requirements specified by the customers and suggests the selection of appropriate architectural components at the variation points. Another issue of importance is the maintenance of service configuration information that would aid in tracking the various versions of the services offered. The capability to trace to the various service versions from customer-specific requirements is of importance because of the suitability of specific versions to specific sets of customer requirements. For instance, a component added to satisfy a customer-specific requirement, which is considered as a variability point could cause performance hits if used for a different customer who might need a version satisfying a primitive form of the same feature but might demand drastically different performance requirements. Such cases could cause conflicts among variable requirements, which should be captured along with version information that is suitable for specific set of requirements.

The capabilities of the system illustrated here can be used in managing both product families as well as e-service families. However, with e-services, the need to not only create, but also support various configurations is likely to accentuate the importance of a knowledge based approach such as the one proposed here.
6 Discussion

We have illustrated, using a case study, the importance of establishing and maintaining requirements traceability for e-service families. We argue that a traceability based knowledge management system would aid in the design process of imparting variability to a service platform by providing the capability to trace from a customer-specific requirement that necessitates variability, back to the stakeholder and forward to its corresponding implementations. Capturing the rationale behind the various design decisions that are associated with reconfigurability of a service platform so as to achieve proliferation of service variety enhances our ability to make changes to the services and their architectures according to the rapidly changing customer requirements. Such knowledge about decisions associated with certain configurations of the service platform would prove to be invaluable in handling identical configurations in the future. Capture of such rationale behind platform-oriented and architectural decisions aids in identifying the requirements that were most common among the various customers, thereby facilitating additions to the service platform and reducing future customization expenses. Tracing architectural design decisions to their sources and their implementations results in reduction of customization costs. Such a traceability capture helps us in avoiding mistakes committed during past implementations. The most common problem in making design decisions is due to failure to explicitly state assumptions. Since the framework described above demands at least a semi-formal representation of the assumptions behind the decisions made, change in validity of these assumptions prevents similar mistakes from occurring in the future.

7 References


