Abstract

One approach to improve the reusability of business process-based solutions is to represent the business process as a combination of autonomous services via a service-oriented architecture (SOA) centric style. Such an approach would represent variations at service and business process level in terms of two key constructs – variation points (the placeholders in the solution model where variations can be introduced) and variation features (the actual variations that can be introduced at a subset of the listed variation points for that solution).

One key research issue that has arisen out of the VOE project is how to actually identify and subsequently justify these variation points and variation features based on changed requirements. In this paper we propose our approach called Variation-Oriented Requirements Analysis (VORA) that addresses this issue. VORA is based on a traceability model that maps requirements to use cases, sequence diagrams, business processes and finally service specifications. We show how this mapping can also be used to (semi-)automatically map requirement changes down to variation points and variation features at service and business process level. We also demonstrate VORA via a simple yet realistic running example.

1. Introduction and Motivation

Software service organizations developing custom business solutions are being faced with the increased need to arrest commoditization and enhance productivity which directly contributes to profitability. One way of doing this is via asset-based reuse. The emergence of service oriented architecture (SOA) [5], with its emphasis on loose coupling and dynamic binding, promises to enable more effective asset reuse by packaging assets as reusable services accessible only via their interfaces. However, one obstacle against the realization of this vision is the cost involved in designing and generating variants of a service in an ad-hoc manner to meet varied customer requirements, resulting in the current labor-based situation in the software services industry today.

In order to address this issue, we developed our Variation-Oriented Engineering (VOE) approach [7]. VOE is an end-to-end approach spanning business processes to their SOA implementation to formally model and develop these variants, so that the reuse of solutions with variants can be facilitated. Our approach consists of three major steps: Variation-Oriented Analysis (VOA), which is concerned with analyzing the solution with respect to its static and changing parts; Variation-Oriented Design (VOD), where a variation model for the solution design is instantiated based on the results of VOA; and Variation-Oriented Implementation (VOI), which is responsible for producing an implementation based on the results of VOD.

The two key constructs in our VOE approach are variation points (points in the solution where variations can be introduced) and variation features (the variations to be introduced at the variation points). Engineering variants into a solution therefore consists of applying variation features at selected variation points.

When we presented the results of our VOE approach in the form of a prototype to practitioners at IBM, one key feedback that we received, was the need to (semi-)automatically derive variation models in a solution based on changed requirements. This feedback from the practitioners is the motivation behind the Variation-Oriented Requirements Analysis (VORA) project, whose initial results are reported in this paper.

The aim of the VORA project therefore is to develop and implement techniques for (semi-)automatically deriving the needed variations in an existing business process-based solution from changed requirements, which are a modified version of the base requirements of the existing solution. The challenge is to evolve a methodology to identify the changes in a systematic and formalized manner and to annotate the base requirements with these new changes. This primarily drives the VOA phase of our VOE methodology in terms of definition and consolidation of variation points and variation features.

This paper is organized as follows. We present some related works on requirements analysis and requirements management in Section 2. Given the highly practical orientation of our VOE approach, we
also highlight the state of the practice in the form of IBM’s Rational RequisitePro product. In Section 3 we introduce our VORA traceability model, which is then used in Section 4 to present how traceability from requirements to service specifications and business processes is maintained. Section 5 presents our ongoing work on mapping requirement changes to variation points and features, based on our traceability model. The paper concludes in Section 6 with suggestions for future work.

2. Related Work

Ever since Eric Yu’s work on i* [10], the field of requirements engineering has become a major research area. However, most research in this area has focused on topics such as developing semi-formal methodologies for requirements-driven analysis and design [6], incorporation of aspect-oriented techniques for requirements engineering [1]. As eloquently detailed in [8], however, most requirements engineering and analysis techniques tend to be informal and lacking in precision. In particular, [8] focuses on providing a precise semantics for use case charts and sequence diagrams, which we have leveraged in this paper. A related paper [9] also discusses how to (semi-) automatically determine state machines from use case charts; this paper also forms one of the motivations behind VORA.

Currently, one of the best tools for requirements management is Rational RequisitePro¹. It is a comprehensive tool for storing and maintaining requirements written in Word format, and provides the following features:

- Integration of requirements in Word documents with a database for requirements capture and tracking
- Requirements typing and attribution for easy location of requirements
- Traceability from high-level requirements to detailed software requirement specifications
- Graphical display of modified requirements as a result of requirement modification elsewhere in the project
- Detailed reporting for standards compliance
- Integration with modeling tools such as Rational Software Architect (RSA)², an associated IBM product for software design modeling

As can be seen from the above, RequisitePro is an excellent requirements management tool. However, it does not currently support the following three key requirements for VORA:

- **Formal specification of requirements in a machine-readable manner** – that is, in the form of use case diagrams or interaction diagrams as described in [8].
- **Automatically deriving business and service definitions from requirements** - either by derivation from (perhaps multi-level) use cases, and/or by mapping them against existing business process and service assets in a repository. Some transformations are available for RSA – these transformations convert a use case model from RequisitePro to an interaction diagram in RSA. However, these transformations are mainly superficial and syntactic in nature, and are not sufficient for precisely representing the semantics of the use cases as per [8].
- **Mapping requirement changes against variation points and variation features** - in order to select the appropriate variants to meet the modified requirements. In other words, RequisitePro does not support a product line approach towards solution modeling based on changed requirements. Indeed, this is the focus of this paper.

3. VORA Traceability model

The VORA traceability model is based on the development process that derives business process and service models from customer requirements, and is depicted in Figure 1.

Figure 1: VORA Traceability model

A customer requirement is a textual description of some functionality that the solution needs to meet. In our running example, one requirement could be “The solution should provide a facility for verifying the insurance claim submitted by a customer”. Each requirement is realized by a set of use cases. Each use case is realized by one or more interacting sequence diagrams. The dynamic behavior displayed in each sequence diagram becomes an input to determining the services needed to realize the requirement expressed via the requirement.

The above refinement of a requirement is typically implemented by a software architect in conjunction with a team of software engineers. In parallel, the business analyst analyzes the requirements, and synthesizes them into a high-level business process model; this model provides an abstract description of the functionality from a business perspective. One of the tasks of the software team is to ensure that the sequence diagrams that they generate are collectively able to match the business process developed by the business analyst.

Our running example depicts the fulfillment of a typical claims verification requirement for insurance claims processing. In our running example, the use cases corresponding to the requirement are: determine liability, check for potential fraud, do detailed claim investigation. Figure 2 depicts the Determine Liability use case, which depicts (at a high level) two actors; the insurance investigator determines the exact liability for the customer, based on his/her claim, and the claims department receives the processes the results of the liability determination activity.

Figure 2: Use case

Each use case is realized by one or more interaction diagrams. Each interaction diagram represents a subset of the dynamic behavior of the solution, and depicts a collaborative scenario among a set of collaborating entities. This is depicted in Figure 3 for the Determine Liability use case.
As per the \textit{VORA} traceability model, the sequence diagram provides the primary inputs for identifying the services that implement sequence diagrams. As per the (optional) annotation shown in Figure 3, we can extract two services from the sequence diagram – a service that calculates the liability, and a service that sends claim details of a customer to the insurance investigator on demand. In other words, specific combinations of messages in the sequence diagram are coalesced into a service definition.

In parallel, the business analyst would develop the business process model as a representation of the overall solution requirements. Each business process is a combination of requirements, as per the \textit{VORA} traceability model.

The Verify Claims business process is depicted in Figure 4. It is a high-level pictorial description of the business process that is supposed to meet the requirement of claims verification. We also see that the DetermineLiability task in the business process matches the functionality of the sequence diagram depicted in Figure 3.

Based on Figures 3 and 4, the description of DetermineLiability service, the service that calculates the liability of the customer, is shown in Figure 5. Other services in the solution can also be defined similarly.
4. Traceability Maintenance

Maintaining traceability among the different artifacts in the \textit{VORA} model requires an explicit definition of the metadata to be maintained for each artifact. The metadata are essential for specifying the detailed syntax and semantics of each artifact, and will be used to define the traceability relations between the artifacts as per the traceability model depicted in Figure 1. As we will see later in this paper, these rules becomes useful in identifying variation points and variation features within sequence diagrams, services and business processes, when a requirement changes.

A requirement \( R = \langle R-ID, \text{Desc}, \text{ID-R} \rangle \), where:
- \( R-ID \): identifies the requirement
- \( \text{Desc} \): provides a brief description of the requirement
- \( \text{ID-R} \): identifies the set of requirements related to this requirement; in our example, related requirements could be Analyze Claim and Claim Analysis & Report requirements

A Use Case \( UC = \langle U-ID, \text{DR-ID}, I, O, A, S, M, M_{seq}, M_{par}, M_{alt}, M_{neg} \rangle \), where:
- \( U-ID \): identifies a Use Case
- \( \text{DR-ID} \): the ID of the requirement from which this use case is derived
- \( I \): identifies the inputs to the Use Case
- \( O \): identifies the outputs to the Use Case
- \( A \): identifies the actors involved in executing the Use Case
- \( S \): identifies the (automated) systems involved in executing the Use Case
- \( M \): identifies the list of messages sent and received by the actors and subsystems in the use case
- \text{Semantics}: as defined in [8], this defines the semantics of the messages that make up the use case, and consist of the following:
  - \( M_{seq} \): identify messages that execute in a sequence
  - \( M_{par} \): identify messages that execute in parallel
  - \( M_{alt} \): identify messages that execute based on the evaluation of a Boolean condition, i.e., these identify alternate paths taken in the use case based on certain conditions that could evaluate to true
  - \( M_{neg} \): these identify a negative outcome (i.e., failure) of the execution of the use case

A Sequence Diagram \( SD = \langle SD-ID, \text{DUC-ID}, \text{BP-ID}, I, O, E, M, M_{seq}, M_{par}, M_{alt}, M_{neg} \rangle \), where:
- \( SD-ID \): identifies a sequence diagram
- \( \text{DUC-ID} \): the ID of the use case from which this sequence diagram is derived
- \( \text{BP-ID} \): the ID of the business process associated to this sequence diagram
- \( I \): identifies the inputs to the sequence diagram
- \( O \): identifies the outputs of the sequence diagram
- \( E \): these are the entities involved in executing the sequence diagram, and are derived from the actors and subsystems identified in the respective use case
- \( M \): identifies the list of messages sent and received by the entities in the sequence diagram
Semantics: again, as defined in [8], this defines the semantics of the messages that make up the sequence diagram, and consist of the following:
- $M_{seq}$: identify messages that execute in a sequence
- $M_{par}$: identify messages that execute in parallel
- $M_{alt}$: identify messages that execute based on the evaluation of a Boolean condition, i.e., these identify alternate paths taken in the sequence diagram based on certain conditions that could evaluate to true
- $M_{neg}$: these identify a negative outcome (i.e., failure) of the execution of the use case

A service $S = <S-ID, DSD-ID, BP-ID, T-ID, I, O, Inf, Int>$, where:
- $S-ID$: identifies the service
- $DSD-ID$: identifies the sequence diagrams from which this service is derived. Please note that the functionality of a service could be an amalgamation of functionality depicted in several (possibly interacting) sequence diagrams within a use case.
- $BP-ID$: identifies the business process associated with this service
  - $T-ID$: identifies the task(s) in the business process where the service is executed
- $I$: identifies the inputs of the service
- $O$: identifies the outputs of the service
- $Inf$: identifies the operations at the service’s interface
- $Int$: identifies the internal operations of the service, which are not exposed to the outside world

A Business Process $BP = <BP-ID, R-ID, I, O, T-ID, Desc, I, O, Pred, Succ>$, where:
- The metadata associated with a business process are:
  - $BP-ID$: identifies the business process
  - $R-ID$: identifies the requirements met by this business process
  - $I$: identifies the inputs to the business process
  - $O$: identifies the outputs to the business process
  - $T-ID$: identifies the tasks in the business process. For each task, the following data is also captured:
    - $Desc$: describes the task
    - $I$: identifies the inputs to the task
    - $O$: identifies the outputs of the task
    - $Pred$: identifies the predecessor tasks
    - $Succ$: identifies the successor tasks

Please note that the association between a sequence diagram (or service) with a business process is one-way, to the business process. This is because the business process is the business analyst’s view of the business process model as derived from requirements, and needs to be a self-contained entity, against which the other artifacts – use cases, sequence diagrams, service specifications – need to be adapted.

5. Variation Points and Features Derivation from Modified Requirements

The above traceability model and associated metadata will help us derive variation points and features within services and the business process based on modified requirements. For this, we first need to specify the different types of changes possible at each step of our traceability model.

At requirement level, three changes are possible: add requirement, modify existing requirement, delete requirement. In this paper our focus is on modifications to existing requirements. These are expressed via changes to the textual description of the requirement.

A modification to a requirement would require the business analyst to modify the existing business process accordingly. Let us assume the following modifications to the Verify Claim requirement in order to address the needs of high-priority customers not previously addressed before:
- Improved speed of processing
- Inclusion of extent of liability as an additional input for fraud checking
Let us assume that the business analyst has modified the business process as shown in Figure 6. In order to meet the modifications to the original requirement, the business analyst has recommended the following changes:
- DetermineLiability and PotentialFraudCheck tasks to be executed sequentially
- Both of these tasks to be part of a new LiabilityAndFraudChecks sub-process that monitors these tasks and sends their output to ClaimInvestigation task

![Figure 6: Modified Business Process Model](image)

A modification to a requirement would involve either modifications to the associated use cases as per the \textit{VORA} traceability model, or the addition of new use cases, with the modified business process as a guideline. For our running example, the following changes are needed to the use case corresponding to Verify Claim requirement:
- DetermineLiability use case needs to be modified to account for added output, i.e., LiabilityInfo
- PotentialFraudCheck use case needs to be modified to account for added input
- New LiabilityAndFraudChecks use case needs to be added

Due to space restrictions, let us focus on the DetermineLiability use case only. The modification of this use case would involve enriching the interaction between the Insurance Investigator and the insurance company’s Information System with customer and claim information, which requires an additional output action. This output action will be translated in the modified sequence diagram as additional messages that send customer and claim information to the company Information System. Finally, in the service specification, this modification in the sequence diagram leads to enhancement of the service functionality (see Figure 5) to incorporate customer information and claim information as additional outputs of the DetermineLiability service. Hence the variation point identified is the interface operation of DetermineLiability service that outputs liability information. The associated variation feature is the addition of customer information and claim information as output arguments to the interface operation in question.

The above can be more concisely represented via the following artifact traceability relations represented via the appropriate artifact metadata:

- Requirement description → Use Case Output
- Use Case Output → Sequence Diagram Message
- Use Case Output → Sequence Diagram Inputs
- Use Case Output → Sequence Diagram Outputs
- Sequence Diagram Message → Service Input
- Sequence Diagram Message → Service Output

Hence in order for such a result to be obtained, we need to develop mapping rules among the various artifacts and their associated metadata. We detail these rules below.
Requirement modification rule:
R modified \(\Rightarrow\) Add input actions and output actions to use case

Since requirements are expressed textually, the above rule is of necessity informal, and can only be implemented manually. The remaining rules can be implemented in a semi-automatic manner; i.e., they can be initially implemented via tool support, and their output can then be manually refined by the user.

Use Case Modification Rules:
Actions added to use case \(\Rightarrow\) additional messages added to sequence diagram
New use case added \(\Rightarrow\) new sequence diagram added

Sequence Diagram Modification Rules:
Additional messages added to sequence diagram \(\Rightarrow\) (additional functionality added to existing services) OR (new service functionality to be created)
New sequence diagram added \(\Rightarrow\) (additional functionality added to existing services) OR (new service functionality to be created)

Service Modification Rules:
Additional functionality added to existing service \(\Rightarrow\) (variation features to be added to existing service functionality) AND (variation points identified based on additional messages)
New service functionality added \(\Rightarrow\) (variation features identified to business process) AND (associated variation points identified in business process) AND (mapping to appropriate service-level variation points and features for adding functionality to existing services, if needed)

We are currently formalizing these rules via RuleML\(^3\); later, we will be developing tool support in the form of an Eclipse-based plugin that will bridge RequisitePro and Rational Software Architect. We will also be augmenting our traceability model using ideas on formalizing traceability and model-driven transformation rules from [4, 2].

6. Conclusions and Future Work

In this paper, we have presented our ongoing work on VORA, our project for developing an approach for variation-oriented requirements analysis. VORA investigates the problem of (semi-)automatically identify variations in services that are to be implemented based on modified requirements. In this paper we have made an initial contribution towards solving this problem by proposing a traceability model that maps business process requirements to use cases, sequence diagrams and business process models, and finally down to the services that implement the needed functionality. We show how this model can be used to map – via rules that can be formally specified in languages such as RuleML - requirement changes down to the appropriate variations in service functionality needed to realize these changes. We illustrate our approach throughout our paper with a simple yet realistic running example.

Since our work is still ongoing, future work would involve detailing and formally specifying all the possible mapping rules in RuleML, in order to develop derivation techniques similar to those reported in [3, 9]. We will then be using this to provide tool support for our approach, so that it can be tested on larger examples.

Acknowledgements

The authors wish to thank Ahamed Jalaldeen, Sandeep Kohli and Biplav Srivastava for their feedback.

References


\(^3\) [http://www.ruleml.org/](http://www.ruleml.org/)


